UNIVERSIDADE DE SÃO PAULO
INSTITUTO DE FÍSICA DE SÃO CARLOS

Leonardo Felipe dos Santos Scabini

Patterns and randomness in networks for computer vision: from graphs to neural networks

São Carlos

2023

Leonardo Felipe dos Santos Scabini

Patterns and randomness in networks for computer vision: from graphs to neural networks

Thesis presented to the Graduate Program in Physics at the Instituto de Física de São Carlos, Universidade de São Paulo, to obtain the degree of Doctor in Science.

Concentration area: Applied Physics

Option: Computational Physics

Advisor: Prof. Dr. Odemir Martinez Bruno

Coadvisor: Prof. Dr. Bernard De Baets

Original version

São Carlos

2023

I AUTHORIZE THE REPRODUCTION AND DISSEMINATION OF TOTAL OR PARTIAL COPIES OF THIS DOCUMENT, BY CONVENTIONAL OR ELECTRONIC MEDIA FOR STUDY OR RESEARCH PURPOSE, SINCE IT IS REFERENCED.

Scabini, Leonardo Felipe dos Santos
Patterns and randomness in networks for computer vision: from graphs to neural networks / Leonardo Felipe dos Santos Scabini; advisor Odemir Martinez Bruno; co-advisor Bernard De Baets -- São Carlos 2023.
233 p.

Thesis (Doctorate - Graduate Program in Computational Physics) -- Instituto de Física de São Carlos, Universidade de São Paulo - Brasil , 2023.

1. Computer vision. 2. Artificial neural networks. 3. Network science. 4. Machine learning. 5. Deep learning. I. Bruno, Odemir Martinez, advisor. II. De Baets, Bernard, co-advisor. III. Title.

FOLHA DE APROVAÇÃO

Leonardo Felipe dos Santos Scabini

Dissertação apresentada ao Instituto de Física de São Carlos da Universidade de São Paulo para obtenção do título de Doutor em Ciências. Área de Concentração: Física Aplicada.

Aprovado (a) em: 24/07/2023

Comissão Julgadora

Dr(a). Odemir Martinez Bruno

Instituição: (IFSC/USP)

Dr(a): Gilberto Medeiros Nakamura

Instituição: (sem vínculo atual)

Dr(a). Eraldo Ribeiro Junior

Instituição: (Florida Institute of Technology/Estados Unidos)

Dr(a): Antoine Manzanera

Instituição: (Paristech/França)

Dr(a): Francisco Aparecido Rodrigues

Instituição: (ICMC/USP)

A sacrifice of myself unto myself.

ACKNOWLEDGEMENTS

To the universe and its greatness, for the possibility of existing, to learn, question, and evolve. And to the old gods, Óðinn, Þórr, Týr, and Ullr, for their primordial wisdom and presence.

To my parents, brothers, grandparents, uncles, and cousins for the familial love and occasional support.

To Marilia for her love, companionship, and support, and for all the moments we shared through our journey together.

To my advisor, Odemir Martinez Bruno, for the guidance, support, trust, and collaboration for the past seven years. Working together has been an incredible journey and a fundamental part of my professional development.

To my co-advisor, Bernard De Baets, firstly for the supervision during my stay at Ghent University, and also for becoming part of the development of this thesis as a whole. It was a great pleasure meeting and working with such an esteemed scientist.

Special thanks to my cats, Puck, Juanita, and Gwyn, for their active participation in developing this thesis by constantly walking over the keyboard. They deserve all the credit for any lost characters or weird typos in the text.

To all colleagues, professors, and workers from the São Carlos Institute of Physics (IFSC-USP), the Institute of Mathematics and Computer Science (ICMC-USP), the Faculty of Bioscience Engineering (UGent), and the Scientific Computing Group (SCG), who contributed, directly or indirectly, to my academic education and the development of this thesis. I am deeply grateful for the discussions, knowledge exchange, and casual moments that were part of my life during the Ph.D.

To the National Council for Scientific and Technological Development (CNPq) (grant #142438/2018-9) and the São Paulo Research Foundation (FAPESP) (grant #2019/07811-0) for the Ph.D. scholarships, and to FAPESP (grant #2021/09163-6) for the scholarship abroad, which made the development of this thesis possible.

The opinions, hypotheses, conclusions and recommendations expressed in this material are the responsibility of the authors and do not necessarily reflect the views of FAPESP.

"One met the dark with learning. But in the end, learned his knowledge was wanting.
The world began without knowledge, and without knowledge will it end.
Fear not the dark, my friend. And let the feast begin."
Locust Preacher

ABSTRACT

SCABINI, L. F. S. **Patterns and randomness in networks for computer vision:** from graphs to neural networks. 2023. 233p. Thesis (Doctor in Science) - Instituto de Física de São Carlos, Universidade de São Paulo, São Carlos, 2023.

Complex networks pervade various aspects of nature, society, and science. One of the most discussed types of network is a neural network, particularly in the last decade with the advances in artificial intelligence (AI). However, little is known about the structure of artificial neural networks (ANNs) in terms of topology and dynamics, from a network science point of view. Moreover, these models are known for being black-box systems, and may also exhibit unexpected behavior. This thesis focuses on AI networks, neural or not, for computer vision (CV) - the sub-field of AI that deals with visual information. Our study explores several aspects of network structure, patterns, and randomness, and their potential for understanding, enhancing, and developing new network-based CV systems. Firstly, we propose a novel network science-based framework for examining ANNs, focusing on their neuronal centrality. This approach, named Bag-of-Neurons (BoN), uncovers the relationship between structural patterns within trained ANNs and their performance and proved to be a promising approach for understanding these systems. These findings led to a new ANN random initialization technique, named Preferential Attachment Rewiring (PARw), which enhances performance and accelerates the training process of shallow and deep ANNs. By leveraging network science, we also develop CV techniques for texture analysis problems, ranging from pure texture images to texture in the wild. We introduce the Spatio-Spectral Network (SSN), a method for image modeling using a graph of pixels, which achieves state-of-the-art (SOTA) results in benchmark datasets. Another new proposal (SSR) couples SSNs with small randomized neural networks, improving performance without a substantial increase in computational costs. The thesis further presents the Randomized encoding of Aggregated Deep Activation Maps (RADAM), a transfer learning method for deep convolutional networks (CNNs) that offers remarkable performance in all CV tasks that were evaluated. We also explore the potential of fully randomized deep CNNs (FR-DCNN) coupled with PARw and RADAM, which prove to be robust texture feature extractors. Lastly, we apply our methods to real-world tasks, including prostate cancer and COVID-19 diagnosis, environmental biosensors using plants, and plant species identification. Our methods, particularly RADAM, consistently achieved SOTA results in these tasks. In conclusion, this thesis introduces six network-based methods for ANNs and CV, and the results show that they consistently improve the SOTA on various image classification problems.

Keywords: Computer vision. Artificial neural networks. Network science. Machine learning. Deep learning.

RESUMO

SCABINI, L. F. S. **Padrões e aleatoriedade em redes para visão computacional:** de grafos à redes neurais. 2023. 233p. Tese (Doutorado em Ciências) - Instituto de Física de São Carlos, Universidade de São Paulo, São Carlos, 2023.

As redes complexas permeiam vários aspectos da natureza, da sociedade e da ciência. Um dos tipos de rede mais discutidos é a rede neural, principalmente na última década, com os avanços da inteligência artificial (AI). No entanto, pouco se sabe sobre a estrutura de redes neurais artificiais (ANNs) em termos de topologia e dinâmica, do ponto de vista de ciência das redes. Além disso, esses modelos são conhecidos por serem sistemas caixa-preta e também podem apresentar comportamentos inesperados. Esta tese foca em redes de AI, neurais ou não, para visão computacional (CV) - o subcampo da AI que lida com informações visuais. Nosso estudo explora vários aspectos da estrutura, padrões e aleatoriedade em redes, e seu potencial para entender, aprimorar e desenvolver novos sistemas de CV. Em primeiro lugar, propomos uma nova metodologia baseada em ciência das redes para examinar ANNs, com foco em sua centralidade neuronal. Esta abordagem (BoN) revela a relação entre padrões estruturais dentro de ANNs treinadas e seu desempenho, provando ser promissora para entender esses sistemas. Essas descobertas levaram a uma nova técnica de inicialização aleatória de ANN (PARw), que melhora o desempenho e acelera o processo de treinamento de redes rasas e profundas. Também desenvolvemos técnicas de CV para problemas de análise de textura utilizando ciência das redes, focando entre imagens de textura pura ou sem controle. Apresentamos um método (SSN) de modelagem de imagem usando um grafo de pixels, que alcança resultados estado-da-arte em bases de dados de referência. Outra nova proposta (SSR) acopla SSNs à pequenas redes neurais aleatórias, melhorando o desempenho sem um aumento substancial nos custos computacionais. A tese apresenta ainda RADAM, um método de transferência de aprendizado para redes convolucionais profundas que também demonstra desempenho notável em CV. Também exploramos o potencial de redes profundas totalmente randomizadas (FR-DCNN), que combinadas com PARw e RADAM provaram ser extratores robustos de características. Por fim, aplicamos nossos métodos em tarefas reais, incluindo diagnóstico de câncer de próstata e de COVID-19, uso folhas como biossensores, e identificação de espécies de planta. Nossos métodos, particularmente o RADAM, alcançaram os melhores resultados nessas tarefas. Em conclusão, esta tese apresenta seis métodos baseados em redes para ANNs e CV, e os resultados mostram que eles melhoram consistentemente o estado-da-arte em vários problemas de classificação de imagens.

Palavras-chave: Visão computacional. Redes neurais artificiais. Ciência das redes. Aprendizado de máquina. Aprendizagem profunda.

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS AND ACRONYMS

AI	Artificial Intelligence
ANN	Artificial Neural Network
BoN	Bag of neurons
CF	Current Flow
CLBP	Complete Local Binary Patterns
CN	Complex Network
CNTD	Complex Network-Traditional
CV	Computer Vision
CNN	Convolutional Neural Network
DCNN	Deep Convolutional Neural Network
FC	Fully connected
GAP	Global Average Pooling
GLCM	Gray Level Co-occurrence Matrices
GPU	Graphic Processing Unit
KNN	K-Nearest Neighbors
LBP	Local Binary Patterns
LCG	Linear Congruential Generator
LDA	Linear Discriminant Analysis
ML	Machine Learning
MLP	Multilayer Perceptron
NN	Neural Network
PARw	Preferential Attachment Rewiring
RAE	Randomized autoencoder
RADAM	Randomized encoding of Aggregated Deep Activation Maps

RNN	Randomized Neural Network
SGD	Stochastic Gradient Descent
SOTA	State-of-the-art
SSN	Spatio-spectral network
SSR	Spatio-spectral representation
SVM	Support Vector Machine
ViT	Vision Transformer

CONTENTS

1 INTRODUCTION

Networks are everywhere and are one of the most common structures governing the functioning of a multitude of systems. From the quantum world to the macro-scale of the universe, atoms and galaxies behave collectively in networks that govern nature all around us. In a simplistic definition, a network, which is also referred to as a graph, is a model composed of two main elements: nodes (or vertices, actors) and connections (or edges, links, interactions). While many networks are physical manifestations, with palpable nodes and wiring such as a computer network, most are invisible or an abstract concept, like a social network. As some examples, we can mention the networks of protein interaction that are essential for the biological functioning of life in general, social networks that represent the way we humans interact with each other, such as how infectious diseases spread, and even the network of galaxies, where gravity governs the interaction between celestial bodies. Even when the structure of nodes and connections is just abstract in such systems, we can imagine, model, and analyze them as networks. The interaction between its nodes produces complex patterns and dynamics that represent the system functioning. We illustrate the general concept of networks in Figure 1, but the reaching of network theory goes much beyond the few examples we give.

1.1 Background and Motivation

Despite the expressiveness of networks, a dedicated empirical field of study for them emerged only in the late 20th century. (1, 2) One of the fuels for this event was a phenomenon of modern society: the advent of computers and computer science. Technology provided a capacity for the storage of information as never seen before and was followed by the universal adoption of computers by society, culminating in the *big data* phenomenon. It refers to the exponential growth of generated and stored data, which also extends to data from many network-like systems. The processing speed of modern computers then allowed analyzing this information efficiently, with surprising practical results. We can now describe many aspects of real-world networks, such as the fact that any person in the world is connected to any other by an average distance of six social relations, or how the neural network of the brain is different when a person has schizophrenia.

This new research field is usually referred to as Network Science, or Complex Networks (CN), and is truly interdisciplinary as it brings together computer scientists, physicists, and mathematicians. The focus is on analyzing complex systems composed of many elements and interactions, hard to interpret with simple approaches. The modeling of complex systems as networks allows us to examine their topological and functional characteristics deeply, encompassing several potential applications that attract sociolo-

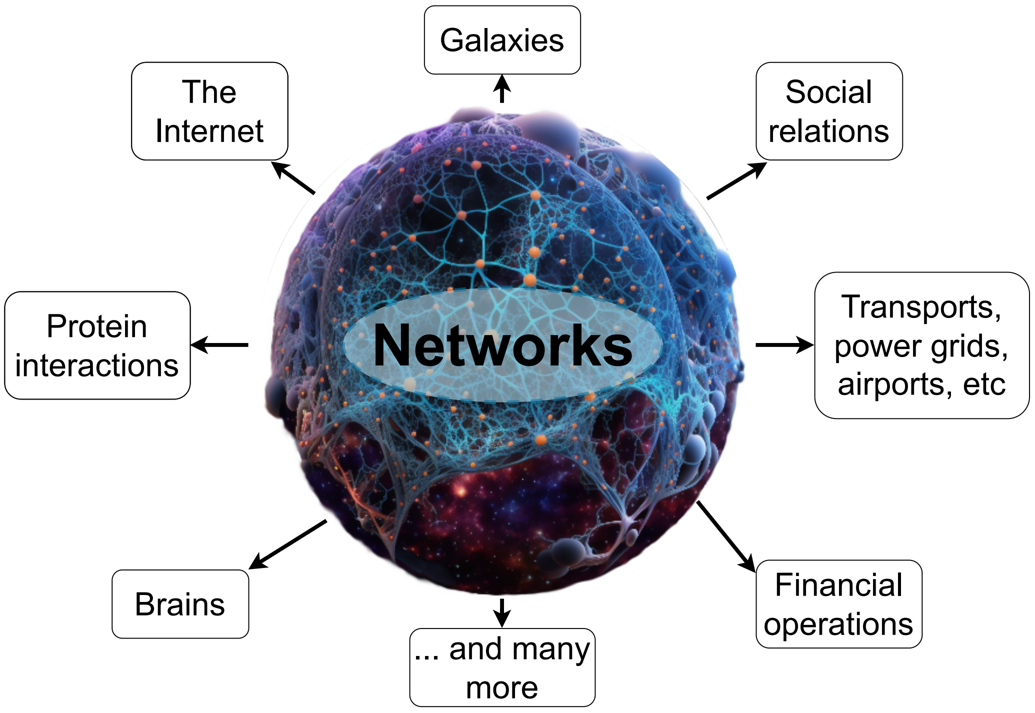


Figure 1 – A conceptual art to illustrate what networks represent. The central picture was generated using Artificial Intelligence (AI) by merging many types of networks such as a galaxy and a neural network.

Source: By the author.

gists, infectious disease specialists, neuroscientists, and many others.

Also in the late 20th century and also powered by *big data*, Artificial Intelligence (AI) has been flourishing especially in the last decade with the popularization of deep learning. (3, 4) AI is a broad field with a multitude of paradigms, but deep learning, specifically, is a synonym for Machine Learning (ML) and deep Artificial Neural Networks (ANNs). These computational models mimic the biological brain structure using artificial neurons that simulate physicochemical processes through mathematical equations. Since 2012, (3) deep models have been overcoming complex problems in pattern recognition in various domains. One such domain, Computer Vision (CV), a field dedicated to enabling machines to understand visual data (images and videos), has been particularly transformed by these advancements. However, many other AI fields have been achieving groundbreaking results due to deep ANNs, such as speech recognition and natural language processing. Moreover, these AI networks provided many successful applications (5) in a broad range of areas such as physics, (6) neuroscience, (7) etc.

The fast advances in the deep-learning literature and hardware technology make it possible to create increasingly deeper and more complex ANNs, leading to a continual search for better performance. Their inherent and increasing complexity and opacity cause the black box problem, (8) which arises due to the intricate interactions between the numerous layers and neurons in a deep ANN, which makes it difficult to interpret the learned representations and decision-making processes. Therefore, as deep ANNs grow to the level of billions and even trillions of parameters (by the beginning of 2023), black box approaches become common in many applications, where the models can achieve impressive results that, on the other hand, cannot be directly explained. This discrepancy happens because little is known regarding these deep ANNs' internal functioning, and their fast growth complicates this understanding even more.

One crucial aspect of ANNs is their construction and training. Randomness plays an important role here since the strength of the connections between the artificial neurons is randomly initialized before training starts. Various strange and interesting effects have been observed related to the randomness of ANNs. Some specific random weight subsets (9) may have critical implications on the performance of an ANN, and these weights might not even need further training, (10, 11) leading models with completely random weights to achieve impressive performance. (12) It was also shown that trained ANNs often converge to weight neighborhoods near their initial setup. (13, 14) However, many works overlook these aspects and the process of randomly initializing an ANN.

Another important effect of deep ANNs is their superior performance itself, or their ability to approximate complex functions. Despite their empirical success, a comprehensive and fundamental understanding of the reasons behind their high performance remains a challenge. The study of deep ANNs has revealed some of their key properties, such as the ability to learn hierarchical representations throughout their depth with varying degrees of abstraction. However, a more holistic understanding from a structural point of view is still lacking. The relationship between these properties, the emergence of knowledge from a stochastic training procedure, and the outstanding performance of deep ANNs in practice is not fully understood. Their complexity is not only rooted in their depth and a large number of artificial neurons but also in the intricate ways these neurons interact with each other. These interactions emerge and evolve during the ANN construction and training, where randomness is a key factor, giving rise to a complex network that is responsible for their impressive performance. Therefore, understanding the functioning of deep ANNs requires the study of these networks, their structure, and their dynamics. This gap in our understanding limits the ability to design better and more efficient ANNs and hampers the development of a solid theoretical foundation for deep learning.

Network Science can also provide fresh insights into the structure and functioning of deep ANNs by approaching the artificial neurons and their connections as a complex network. While the idea of representing a neural network as a graph is not new (after all, the term "neural network" itself suggests a network structure), the application of specific network analysis techniques is a novel and promising direction. This approach allows for the use of tools and concepts from Network Science to analyze and understand the structure and dynamics of ANNs. For instance, it could help in deciphering the organization of artificial neurons, the function of different network modules, and the role of specific connections in the network. This may help to explain the expressive performance of deep ANNs and, furthermore, how to design even better ones.

However, applying network science to deep ANNs presents several challenges. There are many types of ANN architectures and training mechanisms, and they can focus on many types of data (text, images, audio, etc). On the other hand, the scale and size of deep networks often render traditional network analysis techniques computationally infeasible. Furthermore, considering the perspective of a common Network Science analysis, most of the ANNs show at first glance a rather simple graph structure, such as a fully connected network. In this sense, a better and more general representation is necessary for studying existing ANN architectures. Future research should focus on developing a comprehensive framework that integrates Network Science and deep ANNs, addressing the challenges posed by their unique characteristics. This will involve devising novel network analysis techniques, investigating the role of network properties in the performance of deep ANNs, and exploring the application of Network Science principles to improve their design, interpretability, and efficiency. The interdisciplinary nature of this research area offers a promising avenue for understanding ANNs and advancing AI and ML.

On the other hand, the most common pattern recognition approach before the popularization of deep learning was the development of hand-engineered techniques, which continues a viable approach in some scenarios. These are methods where features are manually designed rather than learned from data, *i.e.*, they are based on a priori knowledge about the problem domain and are typically easy to interpret and understand. For instance, Network Science-based CV methods (15–19) that model images as graphs have achieved promising performance on vision tasks such as texture analysis, particularly when the image acquisition is controlled. However, they are often limited when dealing with complex and uncontrolled real-world data, for instance when processing large amounts of images from the internet, as they are not as adaptable as ML-based features. This dichotomy between ANNs and hand-engineered techniques presents a significant challenge, but also a unique opportunity. For instance, a hybrid system that combines the strengths of both approaches could be a potential candidate for many applications.

1.2 Objectives

This thesis dives into the dual aspects of structure and randomness in network systems, and how these can be harnessed in the field of CV. For that purpose, we explore Network Science, ANNs, and their combination, with the goal of learning, characterizing, and analyzing images. We aim to elucidate the internal organization of deep ANNs, identify the network characteristics that are crucial for their performance in CV, and derive new design principles based on these insights. For instance, we investigate the role of connectivity patterns and centrality measures for network performance on different CV tasks. We study these properties before and after training and how they can be influenced by tasks with increasing difficulty, and different initialization methods. We argue that this network-centric perspective sheds light on the elusive aspects of deep ANNs, such as their generalization ability, or their surprising success in learning useful representations from data. Furthermore, we consider our findings to improve the initialization and training of deep ANNs to alleviate common issues such as vanishing/exploding gradients.

Afterwards, we approach the task of texture analysis and build upon previous literature on Network-Science-based image modeling. In contrast to ANNs, this is a hand-engineered approach that is explainable and less complex in terms of computational budget. We also explore randomized shallow and deep ANNs, and how they can be used for effective feature extraction by being combined with our techniques. This approach results in several new proposals ranging from hand-engineered to ANN-based, and also a hybrid method that combines the strengths of the two approaches. We focus both on general vision tasks and also on texture analysis and apply our methods in challenging and relevant real applications. In this context, our objectives are three-fold:

1. **Analyzing and improving ANNs using Network Science:** This component of our study involves a comprehensive analysis of the structure of ANNs using the principles of Network Science. We aim to understand the interplay between network structure and performance in CV tasks, focusing on how the network topology and its initialization can impact the overall performance. We hypothesize that it is possible to predict the performance of a network by analyzing its structure, providing valuable insights into the identification of more efficient network topologies. This analysis resulted in two main contributions: a new Network Science methodology to analyze ANNs (BoN) and a new random initialization method (PARw) based on these findings;

2. **Devising new texture analysis techniques using Network Science and ANNs:** The second objective changes the focus from general vision tasks to texture analysis, according to the demands of specific applications (see objective 3). We employ our findings together with previous literature on Network Science and ANNs for texture analysis, where four new methods are proposed. Firstly, a new hand-engineered technique is developed (SSN) using Network Science to model and characterize images. Secondly, a hybrid approach is proposed (SSR) by coupling the first method with randomized ANNs. The third method (RADAM) is a new feature aggregation and encoding technique based on randomized ANNs that harnesses the power of pre-trained deep ANNs. And lastly, we employ various of the previous find-

ings in another method based on fully randomized deep ANNs (FR-DCNN). The ultimate goal is to create a new class of models that can handle complex visual data more efficiently, with varying computational budgets;

3. **Applications of the developed methods:** Our methods are employed in four specific applications: prostate cancer and COVID-19 diagnosis based on genosensor images, plant biosensing using images from Jacaranda Caroba leaves, and plant species detection based on leaf midrib images.

In conclusion, as we further explore the intersection of Network Science and ANNs in CV, we uncover new ways to understand and improve deep learning networks, and to better model and characterize images. The possibilities are vast, as well as the potential impact on both fields and their applications. The thesis resulted in various new proposals and results, which we summarize in Figure 2. We hope that our work contributes to better integration of AI and Network Science. While these two fields have developed mostly independently, we believe that they have much to gain from each other. The complex, dynamic, and often mysterious nature of deep ANNs makes them an exciting subject of study for Network Science, which can help demystify the “black box” nature of deep ANNs and guide their future development.

1.3 Structure of the text

This thesis is organized into seven chapters and two appendices. We suggest the reader to take a closer look at the appendix (A.1 and B) since they contain descriptions of common topics and datasets in the field. We decided to organize the document this way to make the main chapters easier to follow and to control the length of the text. Considering the main chapters, we start with an ample review and discussion about Network Science, ANNs, and CV, which is given in Chapter 2.

Afterward, Chapter 3 introduces our proposals for the Network Science-based analysis and improvement of ANNs. The chapters contain all the fundamental descriptions and discussion about two of the proposed methods, as well as a thorough experimental evaluation of them and comparisons with other deep-learning techniques. Chapter 4, which we call a network-based approach to texture, introduces four new CV methods focusing on texture analysis. The methods are based on Network Science and ANNs, and we give all the fundamental descriptions about them, followed by an experimental evaluation of their properties and parameters.

In Chapter 5 we present a systematic experimental evaluation of texture analysis using a variety of hand-engineered and deep-learning-based methods from the literature, followed by comparisons with our proposed methods described in the previous chapters. This chapter covers various aspects such as different texture analysis approaches, a variety

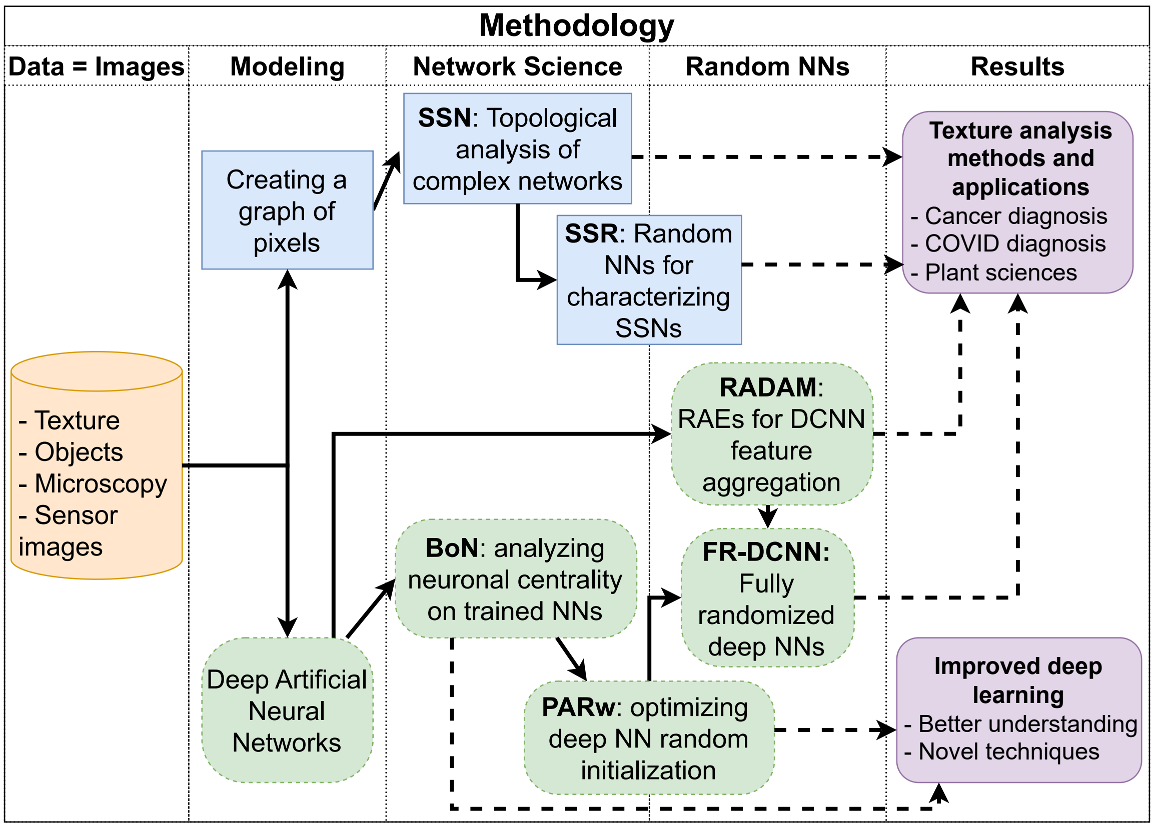

Methodology
Data = Images
Modeling
Network Science
Random NNs
Results
- Texture
- Objects
- Microscopy
- Sensor images
Creating a graph of pixels
Deep Artificial Neural Networks
SSN: Topological analysis of complex networks
SSR: Random NNs for characterizing SSNs
BoN: analyzing neuronal centrality on trained NNs
PARw: optimizing deep NN random initialization
RADAM: RAEs for DCNN feature aggregation
FR-DCNN: Fully randomized deep NNs
Texture analysis methods and applications
- Cancer diagnosis
- COVID diagnosis
- Plant sciences
Improved deep learning
- Better understanding
- Novel techniques

Figure 2 – A workflow of our methodology, representing the different approaches (objectives 1 and 2) and their combinations into new methods. The dotted arrows represent the final contributions of each approach, and the colors (blue and green, or upper and lower boxes, also highlighted by the shape of the boxes) represent the two parallel lines of investigation, respectively: Network Science for approaching neural networks, or images. Both lines of research culminate in the use of randomized ANNs. The work results in several new methods including ANN understanding and improvement, texture analysis methods, and their applications in various CV tasks.

Source: By the author.

of benchmark datasets, and the efficiency of the methods. Similarly, Chapter 6 evaluates a variety of methods on four real tasks, which involve prostate cancer and COVID-19 diagnosis through microscopy images from genosensors, as well as analyzing plants for biosensing and detecting plant species. We compare various methods from the literature with the proposed methods and discuss the potential implications of the results. Finally, Chapter 7 concludes the work with the most important results and remarks, guidelines for future works, and a summary of the publications resulting from our investigations.

2 BACKGROUND AND FUNDAMENTALS

In this chapter, we introduce and discuss the main concepts explored throughout the development of our research. More specifically, we give the fundamentals about Network Science and graphs, ANNs, and CV.

2.1 Network science

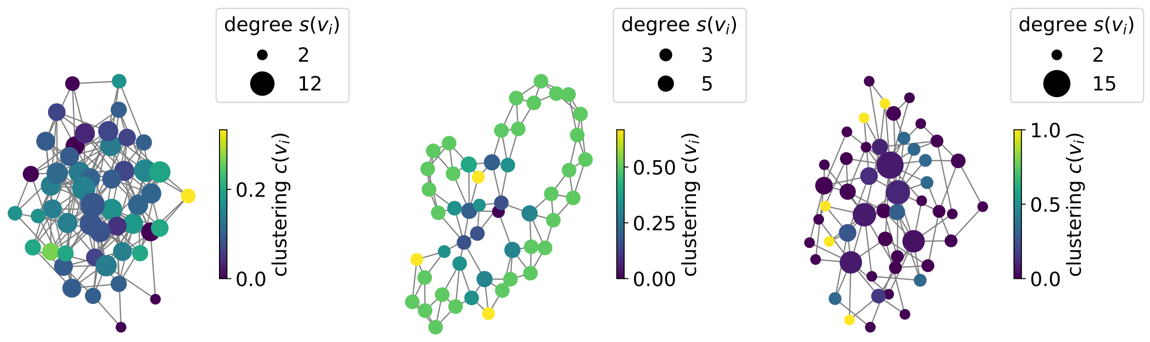
Complex network (CN) research, or Network Science, emerged from the interdisciplinary interaction between graph theory, physics, and statistics. Its primary aim is to study large networks derived from complex natural processes, such as the internet, cell compound interaction, traffic flow, social networks, and many other network-like systems. These systems are challenging to analyze due to the intricacy of deducing their collective behavior from the knowledge of individual system components. (20)

2.1.1 Complex network models

A network can be mathematically defined as a tuple $G = (V, E)$, where $V = \{v_1, ..., v_n\}$ represents a set of vertices and the set $E = \{e(v_i, v_j)\}$ denotes the connections between vertex pairs. An adjacency matrix $A \in \mathbb{R}^{n \times n}$ can be used to represent a network, with $A_{i,j} = 1$ indicating the existence of a link between v_i and v_j, while 0 signifies no connection. There are many types of networks and the nomenclature depends on the edge and node structure. In weighted networks, edges contain weights $e(v_i, v_j) \in \mathbb{R}$ (instead of a binary relation) that could represent a multitude of relational properties such as similarity, distance, etc. This definition can also be extended to the adjacency matrix, which simply means that $A_{i,j} = e(v_i, v_j)$ instead of a binary indication of the existence of connections.

The edges may represent a bidirectional relation, meaning that $e(v_i, v_j) = e(v_j, v_i)$, which represents an undirected network. The opposite holds for directed networks, *i.e.*, if v_i points to v_j, it does not mean that v_j also points to v_i. The nodes may also contain additional characteristics which are denominated as node features and can represent properties such as the age of a person in a social network, the coordinates of a city in a power grid, and many more. Analyzing node features can be crucial to discriminate between the essential and negligible characteristics of nodes for the structure of the network. (21)

Initially, researchers believed that most networks from natural complex systems exhibited random topology, from which derived the Erdös–Renyi model. (22,23) However, it was found later that complex structural patterns emerge in real networks, leading to the development of improved models to describe them. The most prevalent models include the small-world (Watts–Strogatz) (1) and scale-free (Barabasi–Albert) (2) networks. Figure 3 illustrates these three models, where their structural differences become clear (later on, we describe the measures in the figure). These discoveries opened new avenues for pattern recognition, with complex networks being employed as tools for modeling and characterizing natural phenomena across various domains. (24) To give a few examples, we can mention applications in biology, such as cellular (25) and metabolic (26) networks, neuroscience (brain networks), (2, 27) sociology (social networks), (28, 29) epidemiology, (30) etc.



(a) Erdös–Renyi (random). (b) Watts–Strogatz (small-world). (c) Barabasi–Albert (scale-free).

Figure 3 – Example of classic CN models and some of their respective vertex properties (degree, or strength, and clustering coefficient).

Source: By the author.

2.1.2 Vertex centrality

From the vertices and edges, various metrics can be calculated to quantify a CN topology. (31) One of the most common approaches is to estimate vertex centrality, which quantifies the relative importance or influence of individual vertices of a network. By evaluating the centrality of vertices, it is possible to uncover underlying patterns, gain insights into the structural properties of networks, and develop targeted strategies for intervention or control aiming at specific key vertices. For instance, in social network analysis, central vertices may represent influential individuals or organizations; in transportation networks, they may correspond to critical hubs or junctions. Various centrality measures have then been proposed in the literature to emphasize different aspects of a vertex's importance in a network. The degree is the most common measure and is computed by simply counting the number of edges linked to the vertex. For a more broad definition, we consider:

$$s(v_i) = \sum_{v_j \in V} e(v_i, v_j), \tag{2.1}$$

which is referred to as node strength or weighted degree in weighted networks, since it takes into account the real values corresponding to each edge weight, while for unweighted networks it just counts the number of edges. It can also be calculated over the adjacency matrix by simply assuming the row corresponding to the node ($\sum_j A_{i,j}$). For undirected networks, $s(v_i) = s_{in}(v_i) = s_{out}(v_i)$, while s_{in} and s_{out} can be used to quantify input and output connections separately in directed networks (or k_{in} and k_{out} when referring to unweighted networks). It is possible to notice in Figure 3 how vertex degree helps to distinguish between different network structures.

The degree serves as the foundation for describing several network properties, such as the scale-free phenomenon. (2) In these networks, the degree distribution follows a power-law distribution $p(s) \approx s^{-\gamma}$, where γ varies according to the network structure. This topological pattern highlights the existence of hubs, *i.e.*, some vertices have a very high degree and play a critical role in the network's internal functioning, while most other vertices have a low degree. Numerous studies have found the scale-free property in real-world networks from various domains, typically with $\gamma \approx 3$.

It is also possible to calculate second-order measures considering the properties of the neighboring nodes. This approach is based on the assortativity principle, which states that nodes tend to connect to other similar nodes. For instance, the average neighbor strength (for weighted networks) (32) of a node v_i considers the mean strength of other nodes connected to it:

$$s^w_{nn}(v_i) = \frac{1}{s(v_i)} \sum_{v_j \in N(v_i)} e(v_i, v_j) s(v_j)\,, \tag{2.2}$$

where $N(v_i) = \{v_j \mid e(v_i, v_j) \in E\}$ define the set of nodes connected to v_i. Another more complex measure following the assortativity principle is the second-order centrality, (33) which computes the standard deviation of the return times to a given node v_i during a perpetual random walk on the network. In practice, it can be solved analytically, *i.e.*, without the simulation of a random walk:

$$s_o(v_i) = \sqrt{2 \sum_{v_j \in V} M(v_j, v_i) - n(n+1)}\,, \tag{2.3}$$

where a classical discrete time Markov chain M is employed (further details can be consulted on (33)).

Another walk-based measure is the subgraph centrality, (34) which considers all weighted closed walks, of all sizes, starting and ending at a given vertex. Each closed walk is associated with a connected subgraph, and the weights decrease with path length. This measure can be computed spectrally using the eigenvalues and eigenvectors of the adjacency matrix:

$$s_g(v_i) = \sum_{j}^{n} (u_j^{v_i})^2 e^{\lambda_j}\,, \tag{2.4}$$

where u_j is an eigenvector corresponding to the eigenvalue λ_j of A.

In a similar fashion, it is possible to find graph cliques, (35, 36) *i.e.*, a subset of nodes that are all adjacent. These objects can also be interpreted as a complete subgraph of the network. A maximum clique is the most extensive clique under this condition. It is possible to compute the number of maximum cliques in which a vertex participates:

$$m_c(v_i) = \sum_k \begin{cases} 1, & \text{if } v_i \in M_k \\ 0, & \text{otherwise} \end{cases}, \tag{2.5}$$

where M_k are sets of nodes representing each maximum clique k. Finding maximum cliques in a graph is an NP-complete problem. The search is usually performed by depth-first search algorithms and pruning the search tree, resorting to well-chosen pivoting nodes. (36)

Small-world networks (1) are models whose principal property is the facility for information spreading. In other words, these networks have a short average shortest-path distance where vertices can be reached easily (through a few steps). Another essential property of these networks is their high clustering coefficient, which describes the level of interconnectivity between neighboring vertices. This measure counts the number of triangles (fully connected triples) that occur between a vertex and its neighbors by:

$$c(v_i) = \frac{2 \bigtriangledown (v_i)}{s(v_i)(s(v_i) - 1)}, \tag{2.6}$$

where $\bigtriangledown(v_i)$ is the number of triangles in which v_i participates, or simply the number of edges between the neighbors of v_i. Note that this is the formulation for unweighted networks, *i.e.*, it works for $s(v_i) \in \mathbb{Z}$. The equation is normalized by the maximum number of possible triangles that could be formed between v_i and its neighbors, *i.e.*, $c(v_i) = 1$ when all neighbors of v_i are interconnected. Figure 3 shows the clustering coefficient of different types of networks, where it is possible to identify the small-world model by its higher average clustering.

The clustering coefficient can also be computed for bipartite networks, also referred to as bipartite local clustering: (37)

$$b_c(v_i) = \frac{\sum_{v_j \in N(N(v_i))} p(v_i, v_j)}{|N(N(v_i))|}, \tag{2.7}$$

where $N(N(v_i))$ are the second-order neighbors of v_i (neighbors of the neighbors, excluding v_i), and $p(v_i, v_j) = \frac{|N(v_i) \cap N(v_j)|}{max(|N(v_i)|, |N(v_j)|}$ is the pairwise cluster coefficient between nodes v_i and v_j.

Path-based measures are also important for characterizing information flow, such as how fast a node communicates to others. The harmonic centrality (38) considers this principle and computes the sum of the reciprocal of the shortest path distances from all the vertices to a given vertex:

$$h_c(v_i) = \sum_{j \in V} \frac{1}{d(v_j, v_i)}\,, \tag{2.8}$$

where $d(v_j, v_i)$ is the result of Dijkstra's algorithm (39) for computing the shortest path between two nodes v_i and v_j, considering the edge values for weighting the calculation. However, this computation ignores all other than the shortest paths. In most scenarios, information flows throughout all the network paths, such as in a power grid. Therefore, some measures relax the shortest-path definition by including all paths and weighting according to their length. For instance, the current flow (CF) approach, (40) inspired by the electric flow on circuits. The idea is to consider the network as a resistor network where edges are resistors and vertices are junctions between resistors. The electric network analogy is achieved by computing the inverse Laplacian matrix (see (40) for further details). In this scenario, a possible measure is the CF closeness:

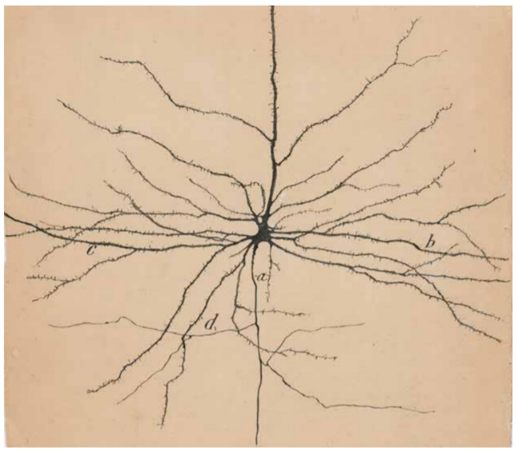
$$c_{CF}(v_i) = \frac{n-1}{\sum\limits_{v_j \in V} p_{v_i,v_j}(v_i) - p_{v_i,v_j}(v_j)}\,, \tag{2.9}$$

where $p_{v_i,v_j}(v_i) - p_{v_i,v_j}(v_j)$ corresponds to the effective resistance, *i.e.*, the alternative distance metric between the two nodes which consider all possible paths on the electric network.

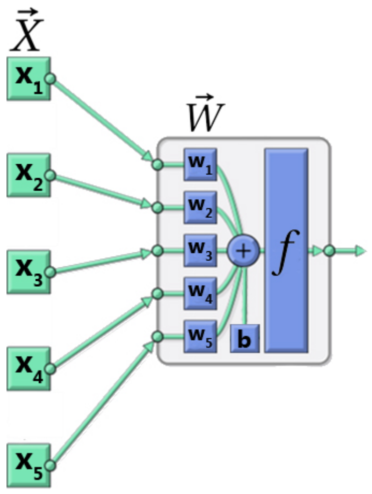
Beyond the CN properties and measures we describe here, the literature is rich and heterogeneous on other approaches, and we refer the reader to (31) for a more in-depth review.

2.2 Artificial Neural Networks

Another network-like system is a neural network (biological or artificial), which has been a constant topic of research in both neuroscience and AI. Humans always tried to understand the brain's functioning, one of the critical elements of our evolution. However, it was only in the last century that science was able to show that it is composed of a highly interconnected network of billions of tiny cells, called neurons. Its dense neuronal communication (synapses) makes possible the realization of the most diverse and complex tasks by the brain. This conception emerged as a result of the endeavors of Spanish scientist Santiago Ramón y Cajal, (41) the 1906 Nobel laureate in medicine and often regarded as the father of modern neuroscience. This groundbreaking work allowed for the first-time visualization of the neuronal structure from various regions of the cortex. Figure 4 (a) depicts one of the earliest drawings created by Ramón y Cajal to illustrate a pyramidal neuron, which plays a crucial role in the functioning of the cerebral cortex (additional illustrations can be found in (42)). In this illustration, one can observe the



(a) One of the first illustrations of a neuron, conceived by Ramón y Cajal in the early 1900s.



(b) Design of the Perceptron, Rosenblatt's first artificial neuron proposed in 1958.

Figure 4 – Illustrative representations of a natural (a) and an artificial (b) neuron. The dendrites of the natural neuron (smaller sparse connections) send information to its cell body (nucleus), which processes it and propagates it down the axon (strongest connection, in the north direction). Analogously, the input connections of the artificial neuron send data to its nucleus, which are weighted by weights w, which the activation function processes and propagates through the output connection.

Source: (a) Adapted from CAJAL (41); PANDYA *et al.* (42); (b) By the author.

central region of the neuron, the nucleus, connected by numerous dendrites and an axon, which establish its relationship with other neurons.

It is the dense connectivity among the billions of neurons and their intricate interaction mechanisms that enable the execution of a wide range of complex tasks performed by the brain. However, a complete understanding of how the brain functions remains elusive. Age-old debates entangle disciplines such as philosophy, medicine, and neuroscience in the quest for explanations of phenomena like consciousness, and whether it is or not a brain process. (43) On the other hand, today we know of specific aspects of neural processing, including their corresponding cortical regions. For instance, it is known that a region within the occipital lobe, termed the visual cortex, is responsible for receiving and processing the initial stages of vision; however, beyond a certain point, its connections branch out to various distinct regions of the brain in a highly complex manner.

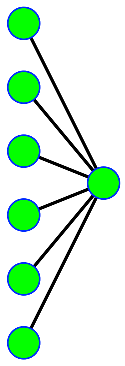
After the first neuroscience discoveries about the brain, and with the dawn of Computer Science, scientists started to study artificial neural models. Around 1943, from the work "*A logical calculus of the ideas immanent in nervous tissue*", (44) the first ideas of artificial neurons are established. In 1958, Rosenblatt introduced the Perceptron, (45) the first artificial neuron capable of performing pattern recognition tasks after training. It can be defined as a function $f(b+\sum x_i w_i) = y$, where the training process consists of adjusting each weight w_i in a supervised fashion, *i.e.*, the desired output for the training set is known beforehand. Given a training sample, the new weight is adjusted as $w_i^{t+1} = w_i^t - \alpha(\bar{y} - y)x_i$, where $\bar{y}$ is the expected output for the input x and α is the learning rate, which controls the update magnitude. By iteratively repeating this process for a given training set, the Perceptron tends to find a solution for the problem if it is linearly separable.

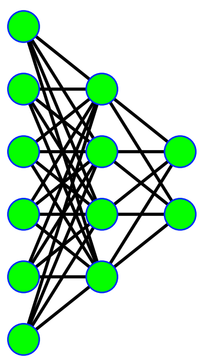
Rosenblatt's proposals caused excitement; nevertheless, many researchers were not satisfied with the performance of the Perceptron. Its fundamentals were criticized in the book "*Perceptrons: an introduction to computational geometry*", (46) which showed its lack of capacity in specific tasks. However, the next works revealed that the model's true potential is achieved when many neurons are combined, which led to the emergence of Artificial Neural Networks (ANNs). ANNs, in turn, are composed of a chain of connected neurons usually organized in layers. Theoretically, a network large enough could fit any function (linear or not), which attracted the scientific community's attention. Since then, a multitude of multi-neuron/multilayer networks have been proposed for various types of data, learning paradigms, computational budgets, etc.

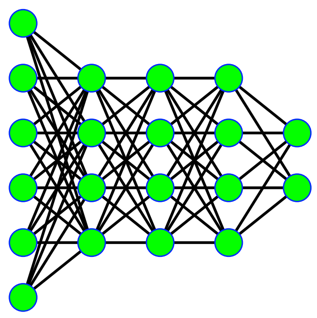
A simple illustration of some of the types of ANNs approached throughout this thesis is shown in Figure 5, and their structure can be freely adjusted by including or removing neurons, layers, connections, activation functions, etc. One of the most traditional ANN models is the Multilayer Perceptron (MLP) (47) (Figure 5 (b)), usually composed of 3 layers: input, hidden layer, and output. These layers are fully connected with the next and previous layers, and no connections exist between neurons of the same layer. Between the decades of 1970 and 1980, (48) advances in these networks' training through backpropagation helped consolidate the multilayer approach. This training technique is prevalent today and consists of performing weight updates by propagating the last layer output error backward until the input layer.

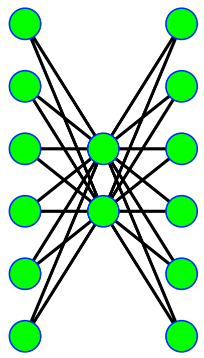
2.2.1 Deep learning

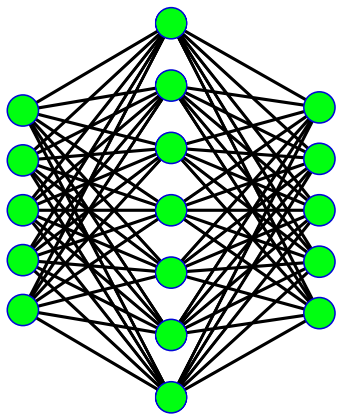
Although the theory behind ANNs supports the hypothesis of an ideal model for any pattern recognition problem, training the model shares similarities with some NP-hard optimization problems in the sense that finding the exact global minimum (optimal weights) can be computationally intractable. The loss function is typically non-convex in the high-dimensional parameter (weight) space, meaning that there might be multiple local minima, and the optimization process could get stuck in a local minimum instead of finding the global minimum. In this sense, many difficulties were faced in the early days of multilayer ANNs, and their performance remained similar to other statistical and logical Machine Learning (ML) methods between 1990 and 2000. This stagnation

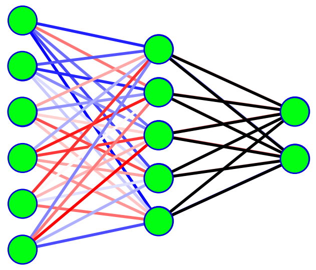

(a) Perceptron.


(b) Multilayer Perceptron (MLP).


(c) Deep feedforward.


(d) Autoencoder.


(e) Sparse autoencoder.


(f) Randomized.

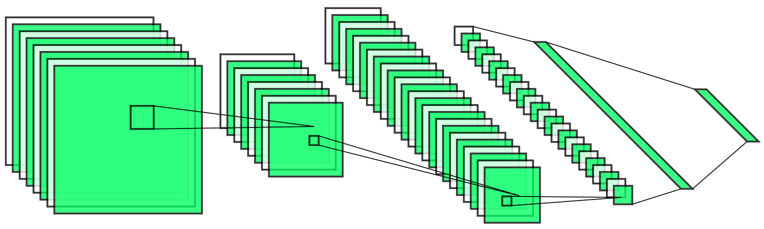

(g) Convolutional network.

Figure 5 – Some examples of ANN architectures.

Source: By the author.

happened due to the lack of efficient training techniques for deeper networks. This training approach, widely known as deep learning, faced difficulties such as the natural occurrence of overfitting* due to the vast number of parameters on deep networks. Moreover, the limited computational power and the lack of annotated data were unfeasible for performing more complex training experiments.

The most common deep ANN model is the deep feedforward network (Figure 5 (c)), due to its connections that propagate information in a single direction, which is similar to

* Overfitting: Training problem in ANNs, which denominates models addicted to the training data while achieving poor performance on different data. In other words, we say that the network memorized the training set but is incapable of generalizing for different cases.

early ANNs such as the MLP, but it consists of multiple hidden layers. This model gained attention again in 2006, (49) where authors showed that the unsupervised pre-training of layers as a way of weight initialization could facilitate the use of deeper models. This technique became known as fine-tuning, where a network with predetermined weights (non-random) is re-utilized. This approach is also known as transfer learning when the inherited weights are ported to another task since they bring previous knowledge learned in a different domain. It is essential to notice that deeper networks also gained more attention due to the massive parallel processing power of graphic processing units (GPU) (50) for simple floating point operations. In various linear algebra tasks, like matrix operations which is the core of ANNs, GPU is superior to CPU processing, allowing significantly faster training of more significant amounts of neurons and layers. In summary, today we are able to effectively train huge deep networks thanks to both a wide collection of training techniques proposed throughout the years and the high computational power of computers. These models are able to achieve great power and flexibility by representing the world as a nested hierarchy of concepts, with each concept defined in relation to simpler concepts, and more abstract representations computed in terms of less abstract ones. (4)

The types of ANNs discussed so far are composed of traditional fully-connected layers. However, there are many other kinds of architectures, and one of them that attracted notorious visibility is the convolutional neural network (CNN, Figure 5 (g)). The ideas that converged into the current convolutional models date back to 1990. (51) This kind of model focuses on multidimensional data with spatial/temporal relations such as time series, images, and videos. It has an additional biological inspiration, where layers follow the classical notion of simple and complex cells from discoveries on the visual cortex of cats. (52) The convolutional architecture also resembles the hierarchy LGN-V1-V2-V4-TI in the visual cortex. (53) Despite its emergence in the 90s, CNNs received little attention until 2012, when the deep CNN (DCNN) AlexNet (3) was introduced. This network was trained with 1.2 million images and had 650 thousand neurons composing 60 million parameters. This model demonstrated high discriminatory power on the ImageNet challenge, (54, 55) which focuses on object recognition.

A convolutional layer is a fundamental building block of CNNs. The primary purpose of a convolutional layer is to learn local patterns and spatial features in input images (or activation maps, for deep layers) through the application of convolutional filters or kernels. These filters are small matrices with learnable weights that are applied to local regions of the input image through a sliding window operation. The mathematical formulation of the convolutional layer can be expressed as:

$$L_{i,j,k} = b_k + \sum_{m=1}^{M} \sum_{n=1}^{N} \sum_{c=1}^{Z} X_{i+m-1,j+n-1,c} \cdot \mathcal{W}_{m,n,c,k} \, , \tag{2.10}$$

where:

- $L_{i,j,k}$ is the output activation (or feature) map at position (i, j) and channel k.
- b_k is the bias term associated with the k-th filter.
- $X_{i+m-1,j+n-1,c}$ is the input image, or feature map, at position $(i+m-1, j+n-1)$ and channel c.
- $\mathcal{W}_{m,n,c,k}$ is the weight of the filter at position (m, n), channel c, and filter k.
- M and N are the spatial dimensions of the filter.
- Z is the number of input channels.

The convolution operation is performed for each filter across all input channels, resulting in a set of output feature maps that capture the learned spatial features. In other words, a convolutional layer is a set of k filters, with weights $\mathcal{W}$, that are applied in parallel to the same input, and the output of the layer, L, or the activation map, is a set of k matrices representing the result of each filtering. Usually, a nonlinear activation function $\phi(L)$ is applied over the output of a convolutional layer. The result is then usually pooled (reduced), flattened, processed again by another convolutional layer, batch normalized, etc. The choice of these operations, and how they are organized (sequentially or not), is what defines the architecture of not only a CNN but any kind of ANN. The literature is vast on this subject, and we refer the reader to references (56–58) for a broad review of ANN architectures.

Since 2012's ImageNet results, CNNs became the predominant approach to computer vision, and several DCNNs have been proposed. (56) The Inception architecture, (59) also known as GoogLeNet, was designed to minimize the computational cost of previous models by using inception modules consisting of parallel convolutional layers. The architecture includes varied filter sizes and 1x1 convolutions for dimensionality reduction, allowing for increased depth with reduced computational costs compared to previous architectures like AlexNet. (3) The model evolved over time, with versions such as InceptionV3 (60) introducing regularization and label-smoothing. These architectures are usually composed of a set of layer blocks, which contain multiple convolutional or other types of layers organized in a specific way (not necessarily sequentially). Figure 6 (a) shows an example of an InceptionV3 block, where the sizes of the layers roughly illustrate their dimension (filter sizes, number of filters, etc). Subsequently, the ResNet (61) architecture was introduced to address certain challenges faced by prior deep models. ResNet introduces residual connections, or skip connections, which enhance signal flow and mitigate vanishing or exploding gradients during training, thus optimizing the training process. Figure 6 (b) illustrates a typical ResNet block.

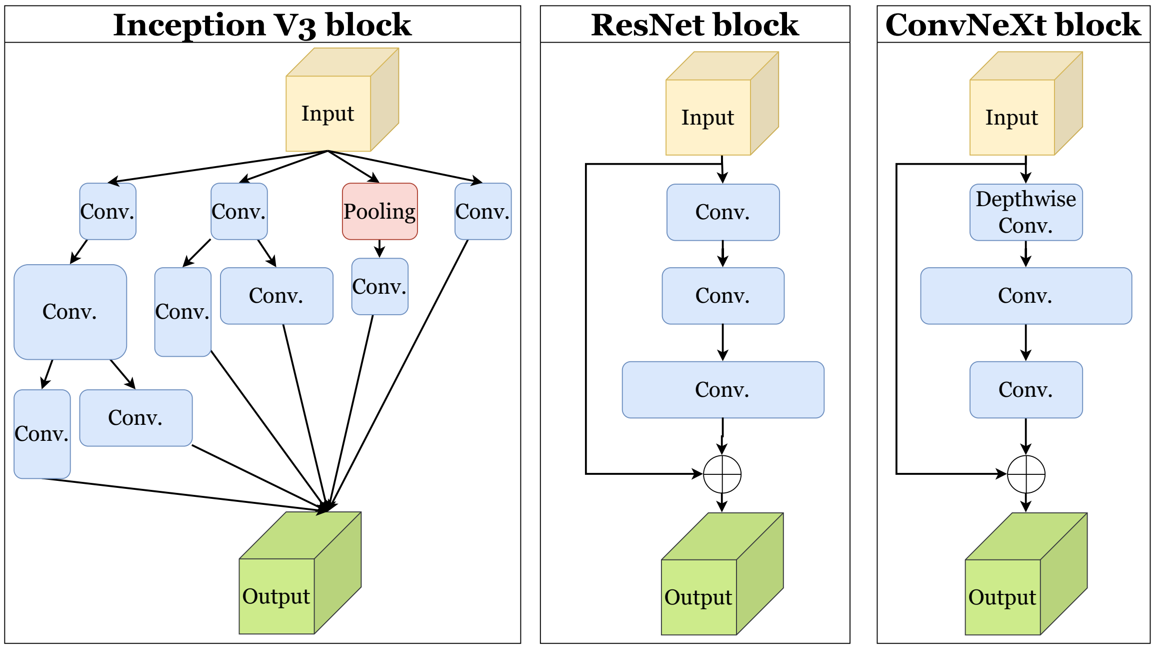


Figure 6 – Illustration of DCNN layer blocks that are used in the building of different architectures. The sizes of the rectangles roughly represent the dimensions of a layer (filter sizes, number of filters, etc), and the arrows represent the signal flow between layers. The symbol $\oplus$ represents the sum of the signals. The process starts at an input source (an image, or the output of a previous block), and ends at the output of the block.

Source: By the author.

The growing performance of DCNN models has motivated many successful applications. (5) However, almost all modern CNNs still contain some fully connected parts resembling the early MLP. Moreover, it has been shown recently that pure MLP models (with additional tricks) can be employed in deep learning tasks with state-of-the-art (SOTA) performance comparable to CNNs. (62, 63) Another recent architecture, the Vision Transformer (ViT), (64, 65) has been dominating the CV literature, also challenging CNNs. It is based on the transformer architecture, (66) which is an effective deep-learning mechanism for machine translation tasks. The ViT presents an efficient way to adapt the transformer architecture to images. On the other hand, this architecture does not contain the common inductive biases that, for instance, CNNs naturally do, missing inherent translation equivariance and locality. To overcome these limitations, most works on ViTs have proposed strong pre-training on large datasets and also new approaches for spatial embeddings/encoding of the input image. More recently, the ConvNeXt architecture was proposed (67) which, while still fully convolutional, can compete with ViTs. ConvNeXts blend standard ResNets with ViT concepts, incorporating design elements such as patching over the input, inverted bottleneck, depthwise convolution, larger filter size, and improved layer normalization. Some of them can be seen in the illustration of Figure 6 (c). Although these CNN design choices have been previously explored individually, their collective implementation in ConvNeXt was motivated by the emergence of ViTs.

2.2.2 Backpropagation training

ANNs are complex models with a wide range of construction blocks, such as activation functions, (68) synapse (weight) initializers, (69) layers, architectural blocks, etc. For neuron activation functions, most methods usually intend to project the inputs in a non-linear space. We can mention the Rectifier-Linear-Unit (68) $\phi(x) = max(0, x)$, a simple yet effective activation function that became popular after the popularization of deep models. Moreover, beyond the ANNs construction stage, there are many training particularities, such as data preparation/normalization, optimizers, learning parameters, and stopping criteria. Current vision models usually normalize data according to the ImageNet standards, (54) which consists of a z-score normalization that centers the data on 0 (subtracting the ImageNet pixel average) and makes the variance equals 1 (dividing by the ImageNet standard deviation). These choices usually depend on the problem addressed or the architectural style, but they are defined mostly by empirical tests and following good practices from previous works.

The training process of multilayer neural networks in a supervised learning setting aims to find the optimal weights that minimize a given loss function. Basically, the loss measures the discrepancy between the network predictions and the expected output, or true target values. Firstly, consider a neural network $\aleph = \{L_0, ..., L_d\}$, composed of a set of $d + 1$ sequential layers where L_0 is simply the propagation of a given input X. There is a wide range of approaches (56–58) for organizing the layers of $\aleph$, referred to as the model architecture. Here we consider the simplest case, *i.e.*, a feed-forward network with a linearly organized and sequential set of layers. Let the output of neuron j (or convolutional filter) in layer $l - 1$ be denoted by $L_j^{(l-1)}$. Then, for each neuron in layer l, we obtain the forward propagation by:

$$L_i^{(l)} = \phi\left(\sum_{j=1}^{n_{l-1}} \mathcal{W}_{ij}^{(l-1)} L_j^{(l-1)} + b_i^{(l-1)}\right), \tag{2.11}$$

where n_l is the number of neurons in layer l, $\phi(\cdot)$ is the activation function, $\mathcal{W}_{ij}^{(l-1)}$ is the weight connecting neuron j in layer $l - 1$ to neuron i in layer l, and $b_i^{(l-1)}$ is the bias term for neuron i in layer l. This process is then repeated for all d layers until obtaining L_d.

After the forward pass, we can compute the loss function $L(\mathbf{y}, \hat{\mathbf{y}})$, where $\mathbf{y}$ is the true expected target value, and $\hat{\mathbf{y}}$ is the output value from the last layer L_d. By backpropagating this error, we can compute the gradient of the loss with respect to each weight and bias value using the chain rule of differentiation. Finally, it is possible to adjust the weights and biases using the gradients coupled with an optimization algorithm such as Stochastic Gradient Descent (SGD) (70) or its variants like AdamW, (71) ADADELTA,

(72) and others. The update rule for a weight $\mathcal{W}_{ij}^{(l)}$ is given by:

$$\mathcal{W}_{ij}^{(l)} \leftarrow \mathcal{W}_{ij}^{(l)} - \alpha \frac{\partial L}{\partial \mathcal{W}_{ij}^{(l)}}, \tag{2.12}$$

where α is a learning rate, and $\frac{\partial L}{\partial \mathcal{W}_{ij}^{(l)}}$ is the gradient of the loss function with respect to the weight $\mathcal{W}_{ij}^{(l)}$. Note that the derivatives will depend on the activation functions employed in the layers of the model. By iteratively repeating this procedure for a number of epochs in a training dataset, the model tends to converge to a solution to the problem. However, notice that this is a simple formulation of the procedure, while in practice various other factors are usually involved. The training of modern deep ANNs has been extensively studied, and numerous improvements and variations have been proposed, such as the use of adaptive learning rates, momentum, and regularization techniques like dropout, weight decay, etc.

In short, ANN development requires extensive architecture engineering and empirical tests that also is hardware-demanding. However, this search is not random since there are good practices in the field, but they do not ensure the model will work as expected in specific scenarios. In contrast, there are techniques for automatic architecture selection, a.k.a. neural architecture search, (73, 74) which focus on searching good models for given datasets, but they require considerable computational resources since these methods are basically random searches. Moreover, other properties may also cause unexpected behaviors such as instability and vulnerability in neural networks, such as adversarial samples. (75) These drawbacks are another motivation for better understanding the internal properties and functioning of ANNs.

2.2.3 Random weight initialization

In the majority of cases, the weights of an ANN are randomly initialized before training starts. These weights are represented by a collection of weight matrices $\mathcal{W}_l \in \mathbb{R}^{n_{l-1} \times n_l}$, each representing the connections between two consecutive layers, where l represents layer indices from the input to the output layer, with n_l neurons each. The initialization of these matrices is an old topic, and here we focus on the initialization of deep models, a trend that became popular after the beginning of this century. The reader may also refer to (76) for a broader review of weight initialization techniques. Early works on deep ANN weight initialization (77, 78) focused on unsupervised pretraining layers, which could improve training speed and generalization performance of standard gradient descent. Later on, (69) proposed to sample weights from a uniform distribution centered on zero with bounds defined by the activation variance of linear neurons. This technique is usually known as Glorot initialization and defines either a uniform or normal distribution from which to sample random weights to initialize a layer. Basically, the method considers the variance of the activation of non-linear functions and the size of a layer to define a lower and an upper bound for a uniform distribution:

$$w_{min} = -\sqrt{\frac{6}{n_{l-1} + n_l}}, w_{max} = \sqrt{\frac{6}{n_{l-1} + n_l}}, \tag{2.13}$$

where the constant 6 is obtained by deriving the variance of the sigmoid nonlinearity. Similarly, parameters for a normal distribution can be obtained:

$$\mu = 0, \sigma = \sqrt{\frac{2}{n_{l-1} + n_l}}. \tag{2.14}$$

However, the Glorot derivation defines just the bounds of values for randomly sampling weights, and it also does not account for most recent non-linear activation functions such as rectified linear units (ReLU). (68) After the popularization of deep learning, many methods focused on more specific techniques for successfully training deeper models. In, (79) the authors extended the Glorot method (69) for ReLU nonlinearities, which consists of varying the bounds for random weight sampling to account for the ReLU variance accordingly. This method is known as Kaiming initialization, sometimes also referred to as "Microsoft weight initialization". (80) The intuition is similar to Glorot's method, where bounds are given for uniform sampling:

$$w_{min} = -\sqrt{\frac{6}{n_{l-1}}}, w_{max} = \sqrt{\frac{6}{n_{l-1}}}, \tag{2.15}$$

or following a normal distribution with parameters:

$$\mu = 0, \sigma = \sqrt{\frac{2}{n_{l-1}}}, \tag{2.16}$$

where the only difference to Glorot's method is the scaling of the values focusing on only one layer, and the new constants derived for ReLU. However, this simple difference is enough to considerably improve the training of deep rectified networks. (79) The Pytorch library, (81) one of the most widespread Python implementations of deep neural networks, uses a similar formulation as default initialization for fully connected (linear) and convolutional layers. They consider a uniform initialization bounded between $[-\sqrt{\frac{1}{n_{l-1}}}, \sqrt{\frac{1}{n_{l-1}}}]$. Another approach is to truncate the values for the normal versions of both Glorot and Kaiming initialization, ensuring more stable tails for the weight distribution by controlling the maximum possible weight magnitude.

Another work (82) suggests that the weights should be chosen as a random orthogonal matrix, and show that this has similar effects as pretraining layers, and helps to train deep models faster and better. They achieve this using a mathematical condition for error backpropagation, namely dynamical isometry, where deep random orthogonal linear networks achieve perfect dynamical isometry and achieve depth-independent learning times. This approach was further explored more recently, (83) showing that in deep networks the width needed for efficient convergence with orthogonal initialization is independent of the depth, whereas the width needed for efficient convergence with Gaussian initialization scales linearly with the depth.

Another approach considers random walk initialization, (84) which keeps the log of the norms of the backpropagated errors constant and was shown to make it possible to train networks with depths of up to 200 layers. They achieve a random walk analogy by deriving the estimated backpropagated normalized errors flowing through random layers in a deep network. (85) proposes to scale the random weight bounds by accounting for the variance when using dropout layers. It was also shown that the Hessian matrix can be considered for ANN initialization, (86) e.g., using an approximation of the layers' Hessians via Jacobian products and the chain rule.

Another approach toward ANN initialization is the re-parametrization of random weights. In (87) the authors propose a technique named Layer-sequential unit-variance initialization, which starts with orthonormal weight matrices (82) and then normalizes the variance of the layer output to be one, given training batches. (88) proposes a re-parametrization of the weight vectors by decoupling their length from their direction using statistics computed from a batch of data (weight normalization). However, these techniques depend on the training data and may be sensitive when estimating the initial values. Moreover, these techniques are usually also applied throughout training, and not only at initialization, introducing additional learning costs.

A recent approach to deep ANN pruning called the lottery ticket hypothesis, (9) suggests that there exist random subnetworks ("winning lottery tickets") that can achieve similar or better performance than the whole model in the same number of training steps. After pruning, the remaining parameters can be reset to their random initial state and the subnetwork can be trained again to a similar optimum compared to the original larger network. Therefore, this subnetwork is a subset of the random initial weights and is not related to some organization that emerges during training. Follow-up works on the LTH (10,11,89) propose that random weights that "win the lottery" are highly correlated with each other, and may not even require training to achieve comparable performance to that of the trained models. Additionally, (14) show that the successful training of ANNs via SGD (70) converges to the close neighborhood of the initial configuration of its weights. These findings corroborate that specific random initializations may have great effects on training dynamics and model performance.

2.2.4 Randomized neural networks

Some of the previously mentioned effects on random weights of deep ANNs are related to the properties of Randomized Neural Networks (RNNs) (90–93) (sometimes also referred to as Extreme Learning Machines), which is a class of models specifically designed to use hidden layers with random weights where only the output layer is trained. RNNs gained popularity in recent years due to their simplicity and fast learning algorithm, while maintaining satisfactory performance in pattern recognition tasks. Works on feature representation learning (94, 95) also point out that neural networks with random weights can be used as feature extractors combined with linear classifiers to achieve considerable performance. This effect may be explained by their architecture alone, which usually are inherently frequency selective and translation invariant, even with random weights. (12)

Here, we focus on RNNs in their simplest form (Figure 5 (f)), comprised of a single feed-forward hidden layer with weights given randomly and an output layer learned using a least-squares solution. First, consider $X \in \mathbb{R}^{n\times z}$ as the input training samples with n data points and $z-1$ features, and a constant -1 appended to the features to represent the bias input. The forward pass of the hidden layer for all samples is then obtained as:

$$g = \phi(X\mathcal{W})\,, g \in \mathbb{R}^{n\times q}, \tag{2.17}$$

where $\phi(\cdot)$ is a sigmoid nonlinearity, and $\mathcal{W} \in \mathbb{R}^{z\times q}$ represents the random input weights for $q-1$ neurons and a bias. Evidence suggests that the particular choice of random initialization has little impact once the random weights are fixed. In this sense, a common trend among previous works is the use of the Linear Congruential Generator (LCG), (96) a simple pseudo-random number generator in the form of:

$$x_{k+1} = (ax_k + b) \mod c\,, \tag{2.18}$$

where a, b, c, and x_0 are parameters that tune the equation to produce different distributions of pseudo-random numbers. Nevertheless, the process is simple and deterministic, ensuring that the same distribution of weight values is obtained for the same network if reinitialized or executed in different machines. Another common step is the use of z-score normalization to obtain a random weight sequence with mean zero and unity variance.

Given the desired output labels $Y \in \mathbb{R}^{n\times\overline{z}}$ for supervised learning of multivariate regression with $\overline{z} \geq 1$ variables, the output weights $\Xi \in \mathbb{R}^{\overline{z}\times q}$ of the RNN are obtained as the least-squares solution of a system of linear equations:

$$\Xi = (g^T(gg^T)^{-1})Y\,, \tag{2.19}$$

where $g^T(gg^T)^{-1}$ is the Moore–Penrose generalized pseudo-inverse (97, 98) of matrix g. After that, the output for new samples $\overline{X} \in \mathbb{R}^{\overline{n}\times z}$, at each of the $\overline{z}$ output neurons, is obtained by computing again the hidden activation $g = \phi(\overline{X}\mathcal{W})$, and then $g\Xi^T \in \mathbb{R}^{\overline{n}\times\overline{z}}$.

To evaluate the computational complexity of the RNN, consider a set of n input feature vectors. First, to calculate the hidden layer output g, approximately $t_1 = nzq$ operations are needed (matrix multiplication). The most expensive operation is computing the output weights Ξ since it involves computing the pseudo-inverse. In practice, we solve it with a single call to *torch.linalg.lstsq(g, Y)*, from the PyTorch library, (81) which solves the system of linear equations $g\Xi = Y$ using QR factorization and has an approximate cost of $t_2 = nq^2$. Therefore, the total estimated cost is $t_1 + t_2 = nzq + nq^2$. It is important to stress that in the vast majority of the cases, $n \gg z, \overline{z}$, and q, and $(z, \overline{z}, q)$ do not depend on n. For instance, in some examples during the development of this thesis, n varied between $[16384, 150528]$, while z varied between $[5, 3840]$, $\overline{z}$ between $[1, 3840]$, and q between $[1, 24]$. By removing these lower terms, we then estimate the asymptotic cost of RNNs as $\mathcal{O}(2n)$.

In summary, the RNN architecture is a simple yet powerful technique for various pattern recognition tasks. Its learning algorithm based on the least-squares solution, instead of the common backpropagation training, may result in speedups of up to 150 times (99) to training time while maintaining similar or even superior performance. These gains can be expressive when hundreds of thousands of samples are used to train a model. The RNN can also be used as a randomized autoencoder (RAE) (99) by considering the input feature matrix X as the target output $Y = X$ (Figure 5 (d-e)). In this sense, the model is composed of a random encoder and a least-squares-based decoder that can map the input data. The previous work also suggests the use of random orthogonal weights (82) for the initialization of the random encoder. After training with the aforementioned approach, the output weights will represent the transformation of the projected random space g back into the input data X (output).

2.3 Computer vision

Vision is a vital sense for the vast majority of the living beings on our planet. The eye has evolved over 500 million years to become a naturally advantageous organ. The human eye can perceive millions of colors, intricate details, depth, and focus at different distances. The brain processes the light captured by the retina, starting from the retina's nerve endings, before it travels through neurons and synapses to the visual cortex in the occipital lobe. Our visual system functions as a multi-stage pipeline, ultimately enabling us to comprehend the elements in the visual world. In AI we are also interested in replicating the remarkable capabilities of the visual system, and this line of research is known as Computer Vision (CV). By leveraging computational techniques and algorithms, CV aims to enable machines to interpret and understand the contents of digital images or videos. This technology has vast potential for a wide range of applications, such as robotics, surveillance, medical imaging, and autonomous vehicles, among others.

2.3.1 Texture analysis

CV is an extensive sub-field of AI, particularly within the popularization of deep learning over the past decade. Nonetheless, numerous CV tasks necessitate focusing on particular attributes within images, which often proves sufficient for a wide range of practical applications. This is also corroborated by the way the visual cortex works. For instance, various studies in neuroscience suggest that there are separate neural pathways for processing information about different visual properties such as motion, depth, color, and shape. (100) This means that not all neural pathways need to be active at the same time for performing some tasks. Therefore, in CV tasks it can be beneficial to tackle specific visual properties of images. One such visual property that is often approached is texture.

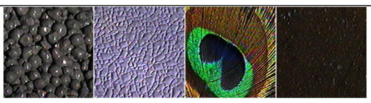
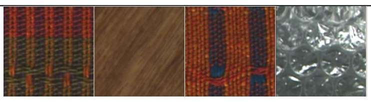
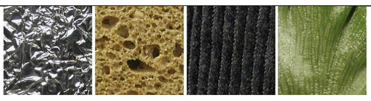
Although an abstract concept, with no widely accepted formal definition, texture refers to the perceived surface properties or structure of objects, which may include roughness, smoothness, coarseness, or fineness. It can also be seen as a pattern of local variations in color, and brightness. The human visual system is adept at recognizing and distinguishing textures, allowing us to differentiate between many things in our environment. In digital images, one definition is that texture emerges from the local intensity constancy and/or variations of pixels producing spatial repetitive patterns, random variations, or specific arrangements of pixel values roughly independently at different scales. The sub-field of texture analysis in CV has its roots in the 1960s. As a result, a variety of methods has been developed over the course of texture analysis research, (101,102) paving the way for potential applications in fields such as industrial inspection, (103) medical imaging, (104) among others.

2.3.2 Textures and benchmark datasets

To better illustrate the nuances of texture recognition, we start by describing two key concepts: texture categories and texture instances. Texture categories represent distinct groups or classes of textures that share common visual properties or characteristics. For example, a texture category could be “wood”, which includes various textures that exhibit patterns and properties associated with wood surfaces, such as grain patterns and color variations. Texture instances, on the other hand, refer to specific examples or occurrences of a texture within a given category. A texture instance represents a particular manifestation of the shared properties of a texture category, such as a specific wood pattern from one species. Texture instances may vary in terms of their appearance, such as color, scale, or orientation, but they still belong to the same overall category. In summary, the main difference between texture categories and texture instances lies in their level of abstraction. In the context of texture recognition, the goal is typically to classify texture categories, although some datasets may also organize images according to texture instances.

Throughout the history of texture analysis, various benchmark datasets have been proposed to capture textures under different conditions and to evaluate CV methods to recognize them. They can be roughly divided into material texture and texture attributes. Material textures refer to the surface properties of objects and materials, which can vary greatly depending on the type of material, such as metal, fabric, or wood, and often exhibit specific patterns, like roughness, or granularity. Material texture datasets (105–109) focus on capturing these properties to enable the recognition and classification of materials based on their surface appearance. Some datasets may also combine materials with texture from objects and elements found daily. (110–113) Describable attributes are subjective properties that depend on the imaged object as well as on human judgments, whereas materials are objective. (114) They can describe texture patterns using human-interpretable semantic words, for instance, when labeling it as “banded”, “striped”, “matted”, etc. Table 1 shows a basic taxonomy of the referenced datasets, which were used throughout this work for evaluating and comparing texture recognition methods. We describe each of them in more detail in Chapter B.

Table 1 – Taxonomy of some texture recognition datasets (employed in this work), along with four of their classes to highlight their differences. “Pure” stands for datasets with pure or homogeneous texture images, and “wild” stands for textures in the wild.

	dataset	pure	wild	material	attributes
	Vistex (110)	×		×	
	CUReT (105)	×		×	
	Outex (106)	×		×	
	KTH-TIPS (108)	×		×	
	USPtex (111)	×		×	
	FMD (109)		×	×	
	DTD (114)		×		×
	GTOS (112, 113)		×	×	

Source: By the author, figures are from each public dataset.

Another aspect of texture datasets is the conditions under which image acquisition was conducted. Most of the early works focused on the construction of datasets by carefully controlling image acquisition conditions, which is usually done in a lab. In these cases, the texture usually covers the whole image, with no multiple objects or backgrounds. We refer to this type of dataset (105–108, 110, 111, 115) as pure texture or homogeneous texture. While this approach makes sense for controlled environments, *e.g.* when analyzing microscopy texture, it may miss realistic conditions which are present in real-world images, for instance in images from the web, captured with mobile phones, etc. In this sense, "in the wild" texture datasets (109, 112–114) consist of images captured in natural, uncontrolled environments, or from the web. These datasets often contain a wide variety of lighting conditions, viewpoints, and backgrounds, making them more challenging for recognition algorithms due to their inherent variability and unpredictability. However, some samples from these datasets may be ambiguous, and the wide variety of artifacts present in the images raises the question of whether they also target general object detection and recognition. For instance, DCNNs pre-trained for object recognition tend to work well in this case, (114) while may struggle in pure texture datasets. (19)

2.3.3 Hand-engineered texture analysis methods

The challenge of texture analysis lies in effectively modeling and characterizing local and global texture patterns while maintaining a balance between performance and complexity (cost). However, many aspects of texture pose significant challenges to be modeled. Furthermore, the inherently unpredictable nature of digital images introduces additional variability, rendering the task even more complex in real-world applications. Therefore, numerous methods have been proposed over the years for the modeling and characterization of textures. Early works focused on carefully designing mathematical techniques to model textural properties from images, then extract a compact representation from it (feature vector). After the popularization of deep learning, various works focused on leveraging neural networks for automatically extracting the features and/or performing the final task (classification, regression, etc). Even so, many methods still consider the former approach since it can surpass neural networks in some cases, while also being more efficient and interpretable. They can be often referred to as "handcrafted" techniques, but we avoid this term here. We will use the terms "mathematical" or "hand-engineered" features/methods.

Mathematical methods for the description of textural patterns usually consider properties such as statistics, (116–118) frequency, (119, 120) complexity/fractality, (19, 111) and others. (121) The first methods primarily focused on two well-established approaches: statistical-based and filtering-based. (101) Statistical methods investigate local measures based on grayscale co-occurrences, with the most widely used being the Gray Level Co-occurrence Matrices (GLCM) (122) and the Local Binary Patterns (LBP). (118) These methods have influenced various subsequent techniques (123) that follow similar principles. For instance, the Complete Local Binary Patterns (CLBP) (124) incorporate information not covered by the original LBP through the local difference sign-magnitude transform. Another approach to texture analysis involves transforming the image into the frequency domain, where various methods have been proposed based on Gabor filters (120) or the Fourier spectrum. (119) Another category of hand-engineered texture analysis examines image complexity and falls within the model-based paradigm, such as using fractal dimension. (125) Most of these methods separate color from texture, so they focus on grayscale images (2-dimensional matrices of pixel intensity).

Color texture analysis involves characterizing the texture of images represented in color spaces, such as the RGB (red, green, and blue) model. Various approaches exist in the literature, with the majority falling into the integrative category, which separates color from texture. Integrative methods typically compute conventional grayscale descriptors for each color channel individually. Consequently, any grayscale method can be employed by combining descriptors from each channel. Another approach, known as pure color, solely considers the first-order distribution of color without taking spatial aspects into account. The most prevalent method in this category utilizes color histograms (126) to achieve a compact summarization of color. However, these methods overlook the spatial interaction of pixels. As a result, a common strategy is to combine pure color with grayscale descriptors in parallel methods. For a comparative analysis of different integrative, pure color, and parallel methods, readers may refer to. (127, 128) Color-texture information can also be obtained by examining the spatial relationships between various colors. In a 1998 publication, (129) a technique employing filter banks was introduced to characterize color texture based on monochromatic and opponent channels, derived from the output of Gabor filters. Another work (130) considers the fractal dimension for tackling the mutual interference of color channels.

2.3.3.1 Network-Science-based texture modeling and characterization

Some methods based on image complexity focused on Network Science (16–19) for modeling images as networks and utilizing their topological properties for texture characterization. We describe this approach in more depth, since two of the new proposals in this thesis fit into this paradigm, building upon previous works. Initially, each image pixel is treated as a vertex in an undirected and weighted network. Given a grayscale image I with wh pixels and intensity levels $p(i)$ ranging between $[0, L]$ (L represents the maximum possible pixel intensity), a network $N = V, E$ is constructed by obtaining the vertex set $V = v_1, ..., v_{wh}$ and edge set E based on a specified connection rule. Certain studies (15, 131) utilize the absolute intensity difference between pixels to define connection weight. The intensity difference is unaffected by changes in the average image illumination. (132)

By introducing a radius r to define a spatial window, a new network N^r is generated, wherein each vertex is connected to its neighbors. The first works proposed a connection weight of $\frac{|p(v_i)-p(v_j)|}{L}$ (normalized absolute intensity difference) if $d(v_i, v_j) \leq r$, where $d(v_i, v_j)$ denotes the Euclidean distance between pixels. This connection weight inversely corresponds to pixel similarity, with lower values indicating higher similarity. However, this expression does not incorporate spatial information in the connection weight, prompting subsequent research to propose alternative rules. In (16) the term d is incorporated into the edge weight calculation, assigning equal significance to both the pixel intensity difference and its spatial position within the connection neighborhood ($\frac{1}{2r}(d(v_i, v_j) + r\frac{|p(v_i)-p(v_j)|}{L})$). A distinct approach is presented in (133) where intensity and distance are directly proportional ($\frac{|p(v_i)-p(v_j)|d(v_i,v_j)}{Lr}$). The inclusion of spatial information addresses the limitation of earlier methods in which connection weights for pixels with equal intensities remained constant, regardless of their distance.

The process outlined thus far yields a network N^r with a connection density proportional to r. However, this results in a regular network, as all vertices contain the same number of connections (except for border vertices). Consequently, a transformation is required to derive a network containing pertinent complex topological patterns. In most methods, this is achieved through connection thresholding, introducing an additional parameter t. A new network $N^{r,t}$ is derived by modifying its edge set $E = \{a(v_i, v_j) | a(v_i, v_j) \leq t\}$. The new network maintains connections between similar pixels, where the parameter t governs the level of similarity. The resulting topology is directly influenced by parameters r and t. However, this allows for a comprehensive analysis of the network's dynamic evolution by adjusting the parameters from smaller to larger neighborhood sets and various degrees of pixel similarity. The texture characterization is then conducted using topological measures from these CNs, such as using vertex centrality.

The concepts of CN applied to texture analysis have been explored and improved in more recent works. In (134) a new technique is proposed to build a vocabulary learned from CN properties and characterize the detected key points through CN, exploring the relevance of various topological measures. In (19) a new multilayer model is introduced for colored images, where each network layer represents an image color channel, and its topology contains within-between channel connections spatially. This work also proposes a new method for estimating optimal thresholding, and the use of the vertex clustering coefficient is also introduced for the network characterization, achieving promising results. On the other hand, the use of thresholding and a common CN modeling of the pixel relations are still subject to improvement.

2.3.4 Texture Analysis with Artificial Neural Networks

At the turn of the last century and during the 2000s, texton-based learning approaches such as Bag of Textons (135) and Bag of Words (136) were proposed. In these approaches, a dictionary of textons is obtained based on local descriptors, and images are represented by statistics over the texton dictionary. The demand for features invariant to scale, rotation, illumination, and viewpoint stimulated the development of local invariant descriptors, such as the scale-invariant feature transform (SIFT). (137) More recently, learning techniques have gained popularity in integrating texture with object recognition. (138) Consequently, contemporary research on textures and image analysis has shifted its focus from hand-engineered features toward deep learning approaches, which we discuss in the following.

The rapid advance of deep learning-based models for general vision tasks has created a common sense that they are universal solvers for CV tasks. This revolution in the field started with CNNs, (3,51,139) a powerful neural architecture that still dominates many CV areas. However, usual CNN architectures that perform well for general vision tasks may fail to achieve SOTA performance in texture recognition tasks in comparison to hand-engineered approaches. (19) In this sense, current works focus on adapting and adjusting them specifically for texture analysis.

There have been various studies involving deep CNNs for texture recognition, and here we review them according to two approaches: feature extraction or end-to-end fine-tuning. Some studies explore CNNs only for texture feature extraction and use a dedicated classifier apart from the model architecture. Cimpoi *et al.* (140) proposed one of the first works on the subject, where they compare the efficiency of two different CNN architectures for feature extraction: FC-CNN, which uses a fully connected (FC) layer, and FV-CNN, which uses a Fisher vector (FV) (114) as a pooling method. They demonstrated that, in general, FC features are not that efficient because their output is highly correlated with the spatial order of the pixels. Later on, Condori and Bruno (141) developed a model, called RankGP-3M-CNN, which performs multilayer feature aggregation employing Global Average Pooling (GAP) to extract the feature vectors of activation maps at different depths of three combined CNNs (VGG-19, Inception-V3, and ResNet50). They propose a ranking technique to select the best activation maps given a training dataset, achieving promising results in some cases but at the cost of increased computational load, since three pre-trained DCNN backbones are needed. Lyra *et al.* (142) also proposes feature aggregation from multiple convolutional layers, but pooling is performed using an FV-based approach (Multilayer-FV).

Another approach consists of new end-to-end architectures that enable training new texture-specific modules/layers along with fine-tuning pre-trained DCNN backbones. Zhang *et al.* (143) proposed an orderless encoding layer on top of a DCNN, called Deep Texture Encoding Network (Deep-TEN), which allows for images of arbitrary size. Xue *et al.* (112) introduces a Deep Encoding Pooling Network (DEPNet), which combines features from the texture encoding layer from Deep-TEN and a global average pooling (GAP) to explore both the local appearance and global context of the images. These features are further processed by a bilinear pooling layer. (144) In another work, Xue *et al.* (113) also combined features from differential images with the features of DEPNet into a new architecture.

Using a different approach, Zhai *et al.* (145) proposed the Multiple-Attribute-Perceived Network (MAP-Net), which incorporated visual texture attributes in a multi-branch architecture that aggregates features of different layers. Later on (146) they explored the spatial dependency among texture primitives for capturing structural information of the images by using a model called Deep Structure-Revealed Network (DSRNet). Chen *et al.* (147) introduced the Cross-Layer Aggregation of a Statistical Self-similarity Network (CLASSNet). This CNN feature aggregation module uses a differential box-counting pooling layer that characterizes the statistical self-similarity of texture images. More recently, Yang *et al.* (148) proposed DFAEN (Double-order Knowledge Fusion and Attentional Encoding Network), which takes advantage of attention mechanisms to aggregate first- and second-order information for encoding texture features. Fine-tuning is employed in these methods to adapt the backbone to the new architecture along with a new classification layer.

More recently, Vision Transformers (ViTs) have been dominating the CV literature and challenging CNNs, especially on image classification. (64, 65) Some works have briefly explored their potential for texture analysis through the Describable Textures Dataset (DTD), achieving SOTA results. Firstly, ViTs achieve competitive results compared to CNNs, but they lack the typical convolutional inductive biases, which usually imply needing more training data to learn better representations of the spatial relations between the pixels. Therefore, to achieve general vision capabilities, ViTs need strong pre-training on huge image datasets, such as ImageNet-21k (54) and Bamboo. (149) Another approach is to optimize the construction of multitask large-scale ViTs such as proposed by Gesmundo (150) with the μ2Net+ method. Another promising alternative is to use attention mechanisms to learn directly from text descriptions about images, *e.g.* using Contrastive Language Image Pre-training (CLIP). (151) Although these approaches usually improve a ViT's general performance, they do not ensure spatial invariance, which raises more questions about the performance of such models for texture recognition. Moreover, texture datasets and texture recognition applications usually have limited training data, increasing the challenge for data-hungry models such as ViTs. Therefore, little is known regarding the applicability and capabilities of ViTs on texture analysis.

Some works have investigated RNNs to learn texture features from images, which is a middle ground between hand-engineered and deep-learning-based methods. Sá Junior *et al.* (152, 153) used small local regions of one image as inputs to an RNN, and the central pixel of the region as the target, optimizing them for regression. Since one

RNN is trained for each image, extracting local windows and applying the RNN to each of them individually is, in fact, similar to applying a convolutional layer with random weights. The trained weights of the output layer for each image are then used as a texture representation. Another approach is the use of Network Science to model images and, consequently, use randomized ANNs to learn their structural properties applied for texture analysis. (154) However, this work is limited to grayscale images. Going in a different direction, (155) proposed texture feature extraction by using CNs for modeling the output of pre-trained deep CNNs. On the other hand, this approach falls into the transfer learning paradigm since it requires a pre-trained DCNN backbone.

3 THE STRUCTURE AND PERFORMANCE OF ARTIFICIAL NEURAL NETWORKS IN COMPUTER VISION

In this chapter, we discuss several aspects of Artificial Neural Networks (ANNs) and propose two new methods, which are each described in a section of this chapter. ANNs are in a non-stopping ascension since the introduction of deep learning, and many successful applications have been developed. (5) The successful training of deep models is usually possible through a careful architecture design, high model complexity, and large-scale datasets. Innovative neural architectures, training mechanisms, and big data collections constantly push forward the SOTA on various applications. The fast advances in these techniques and hardware technology make it possible to create deeper and more complex models. However, this phenomenon leads to black-box approaches becoming usual in many applications, as the study of ANN fundamentals does not follow the same rhythm. For instance, the lack of understanding of its internal details (156) causes some intriguing inexplicable properties to be observed in deep convolutional networks, (157) one of the most popular ANN architectures for Computer Vision (CV). Experiments have shown that imperceptible input perturbations can drastically impact performance, a phenomenon also known as adversarial examples. (75) It is also possible to create entirely unrecognizable images, but that convolutional networks strongly believe are recognizable objects. (157) Some works (158,159) also discuss the impacts of stochastic and random hyperparameters (different random seeds) used during the construction and training of deep ANNs, and also the expected uncertainty in this process. (160) One important finding is that although the performance variance caused by these processes can be relatively small, outliers are easily found, *i.e.*, models with a performance much above or below the average.

The aforementioned effects can happen not only in CV models but in any kind of ANN since one of their properties is sharp decision boundaries that emerge during training with common backpropagation mechanisms. In this sense, small perturbations in the input can push the output of the model outside these boundaries, leading to drastic changes in prediction, unstable confidence, etc. These performance instabilities limit research not only on neural network functioning, but also on network optimization such as better synapse initialization, construction, architecture search, and improved fitting mechanisms. Therefore, understanding the functioning of ANNs is recently one of the main topics within the field, which has become an interdisciplinary effort. On the other hand, although a lot of research has been done in this aspect, understanding the broad impacts of the ANN's degrees of freedom in their structure and dynamics is still a challenge since modern deep models hold millions or even billions of parameters.

3.1 The complex network properties of fully connected neural networks

Several aspects of ANNs are compatible with complex systems, which are usually non-linear, with a hard-to-predict behavior, and sometimes sensitive to small input/parameter perturbations. One interesting approach then to studying such systems is through Complex Networks (CN), or Network Science, which has an extensive literature on the analysis of biological neural networks. For instance, the existence of the small-world CN phenomenon was shown in the neural network of the *Caenorhabditis elegans* worm. (161) Various works then found such topological patterns also in human neural networks, like from magnetoencephalography recordings, (162) analysis of cortical connections (163) and of the brain stem medial reticular formation. (164) In this context, CN principles have helped to understand the origin of dynamic brain activity in anatomical circuits at various levels of scale. (165)

3.1.1 Correlated works

In the case of ANN, few works have approached them from CN principles, especially if we consider the period before the research of this thesis started (before the second semester of 2018). In (166) Hopfield networks are simulated as scale-free CNs, reducing the number of neuron connections and reducing the size and computational cost while keeping tolerable performance. A work from 2004 (167) shows that attractor networks with scale-free topology have a greater capacity to store and retrieve binary patterns than highly random Hopfield networks with the same number of synapses. The relative performance of the scale-free system increases with increasing values of its power-law exponent. Small-worldness in ANNs is achieved in (168) by including shortcut connections between layers in small feed-forward models (up to 225 neurons), showing that it can improve performance and training time. After the deep learning diffusion, a work (169) proposed a small-world ANN that has better performance than the traditional structure on the task of a diabetes diagnosis. The concept of weight sparsity was explored by (170) to obtain small-world properties on ANNs, allowing for the training of wider networks without incurring a quadratic increase in the number of parameters.

After 2018, more works started to emerge on this topic, and some of them actually reference our research. Since this is a fast-developing field, some of these works in this period were published in parallel to ours and also corroborate our findings. A 2019 work (171) explores random ANN architecture generation based on Network Science models by considering layers as nodes and shows that they can achieve competitive performance compared to optimized hand-crafted architectures. Similarly, in a 2020 work (172) the authors propose to analyze the graph structure of ANNs to generate new random architectures for computer vision and analyze their complex network properties with performance. They suggest that performance is approximately a smooth function of the clustering coefficient and average path length (considering their relational graph modeling). In another 2020 work (173) the authors show that increasing the smallworldness of feed-forward ANNs may reduce overfitting. Basically, this is the effect caused by the residual connections in ResNet models. (61) Also in 2020 (174) CNs are employed for the analysis of 5-layer deep-belief networks, and results suggest correlations between topological properties and functioning. In (175) the authors explore the emergence of network motifs, *i.e.*, a group of neurons with a specific connection pattern, on a small Multilayer Perceptron (MLP) ANN. They consider different numbers of neurons in this architecture applied to small synthetic data and conclude that specific motifs emerge during training. They also argue that the network topology can be strongly biased by choosing an appropriate weight initialization. Similar findings (176) also suggest that the structure and performance of ANNs are related. Going in a different direction, (155) considers the modeling of the output of vision architectures using CNs, and achieves improved performance for texture analysis. In this particular case, the methodology needs trained models and works as some sort of transfer-learning to specific domains.

Although these works suggest CN properties within ANNs, much still needs to be done to uncover their properties. For instance, a common trend among the mentioned works is that they do not consider a reliable sample size to account for the variance caused by the degrees of freedom during ANN construction and training. It is widely known that different weight initializations can cause drastic changes in neural network behavior, such as some specific distributions which may make training particularly effective. (9) In this sense, many of the findings of these previous works could be questioned from a statistical point of view, and are also hard to replicate in different scenarios. Moreover, some of the works lack direct connections to efficient practical tools that could be integrated with SOTA ANN construction, training, optimization, etc. Therefore, further studies are required to improve the understanding of the CN properties of ANNs and also in line with the final applicability of these findings. This approach can help in achieving improved interpretability of ANNs, identification of bottlenecks in the network, guidance for model pruning and compression, and insights into the development of more efficient architectures.

3.1.2 Building a neural network dataset

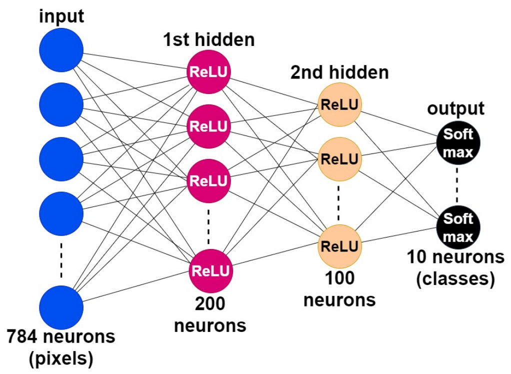
We propose an analysis of the internal CN properties of fully connected neural networks and their correlation to classification performance on vision tasks. This architecture is considered one of the most diffused models since early neural network studies, and it is still popular among modern deep methods. Its number of parameters (synapses) grows exponentially with the number of neurons and layers. To uncover topological properties emerging in such systems, this ample parameter space needs to be explored in more depth. However, it becomes impractical to compute every possible combination even for small architectures. Therefore, we propose constructing a neural network population with different properties within the most common practical scenarios in the field. This approach allows for the comparison and statistical evaluation of a wide range of neural networks, which is one of the main differences from previous works. Our efforts described in this Section resulted in a publication (177) in the journal Physica A: Statistical Mechanics and its Applications, from Elsevier.

We propose a new neural network dataset composed of a wide range of synapse configurations. The idea is to use a fixed architecture (number of neurons and synapses) and training hyperparameters, then train networks of different initial weights. For that, we vary the seed of the pseudo-random number generator that produces the synapse distributions (discussed later on). In this sense, it is possible to widely explore the synapse space of random initial networks and analyze how it impacts the training dynamics and final behavior. After training, even networks of similar performance have different initial weight configurations, allowing us to analyze recurrent CN properties emerging between them. It is important to notice that the idea for building our dataset is different from previous works on Neural Architecture Search (178). In these works, the goal is to search and optimize architectures for better performance, while here we are storing the entire network for tracking the properties which explain their performance.

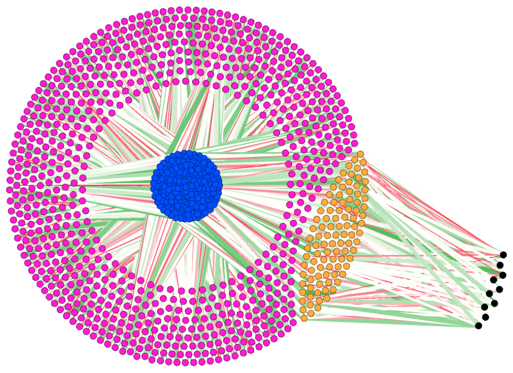
Firstly we define the architecture for our study. This choice depends on the problem one needs to address; we focused on the supervised classification for vision tasks. An MLP-like architecture is considered, where the input patterns are grayscale images of sizes 28x28, which are fed into a flattened layer of 784 neurons. Two hidden layers, with 200 and 100 neurons ($n = 300$), respectively, employ the ReLU activation function. The output layer comprises ten neurons with a softmax activation function, representing a 10-class classification task. Here we do not consider the bias term for simplicity. This architecture yields 177,800 trainable parameters. Figure 7 (a) shows a summary of the model, which we implemented using the Keras 2.2.4 library (179) [*]. In Figure 7 (b), we show all the model's neurons and synapses using a network visualization technique that clusters vertices according to their degree. Neuron colors represent its corresponding layer, while edge colors refer to the synapse signal (green for positive and red for negative).

To achieve different networks from this same architecture, we vary the weight initialization. Usual initialization techniques, such as the Glorot (69) or Kaiming (79) weights, optimize the distribution by sampling from a normal function around 0 with a small standard deviation. In practice, this usually results in more stable training and better average performance (though not necessarily implying stable performance (158,180)). However, our focus here is not on better performance but on heterogeneity. Therefore, we use an initialization technique that allows a broader range of possible weight configurations. We consider a uniform distribution in the interval $[-0.9, 0.9]$. We found empirically

[*] https://github.com/keras-team/keras/releases/tag/2.2.4



(a) Fully connected neural network.



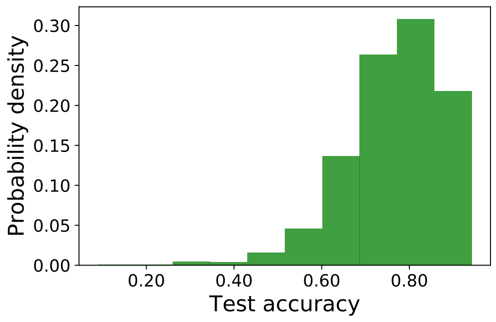
(b) Visualization of all its neurons and synapses.

Figure 7 – The architecture used in this work with its corresponding configuration (a), and a visualization of all its neurons (1,094 nodes) and synapses (177,800 edges) as a weighted and undirected network (b).

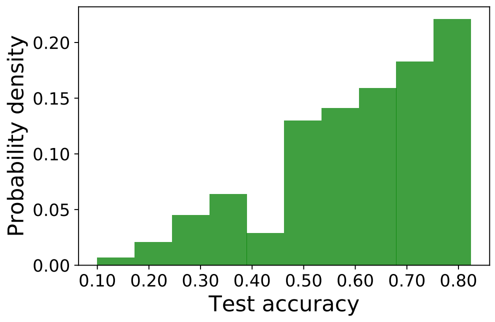
Source: SCABINI; BRUNO. (177)

that for our architecture, using a wider range such as $[-1, 1]$ tends to produce poorer models, while compressing it tends to produce more similar models and better average performance. The considered interval then generates more diverse networks at both ends regarding performance in different vision benchmarks, as Figure 8 shows. Notice, however, that average performance varies significantly between tasks, which is a reflection of their difficulty rather than hyperparameter choice. The most challenging task for this architecture is texture recognition (KTH-TIPS), where the models are barely superior to random guessing (10% accuracy).

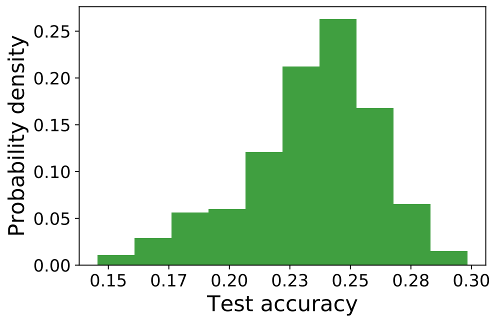
For the training, we considered the following: SGD optimizer on a categorical cross-entropy loss function, learning rate $\alpha = 0.01$, and batch size 100. We observed that the training usually stabilized around 15 to 20 epochs, then a total of 30 training epochs were considered. The networks are evaluated by their classification accuracy, especially the test accuracy, *i.e.*, the final performance on the hold-out testing set after training is finished. With the described architecture and training procedure, we address the following four vision benchmarks of image classification: MNIST, (139) Fashion MNIST, (181) CIFAR-10, (182) and KTH-TIPS. (107,108) These are different visual recognition tasks where the model needs to classify 10 thousand images into 10 classes while using 60 thousand images for training (we use grayscale images). MNIST is composed of handwritten digits (from 0 to 9), Fashion MNIST contains a variety of fashion items, and CIFAR-10 represents objects. The version of the KTH-TIPS we use is composed of 28×28 grayscale texture patches cropped from the original KTH from 10 different materials (classes), and randomly sampled to compose the training and testing splits. For more details about validation and the image datasets, we refer the reader to Chapters A and B, respectively. For each



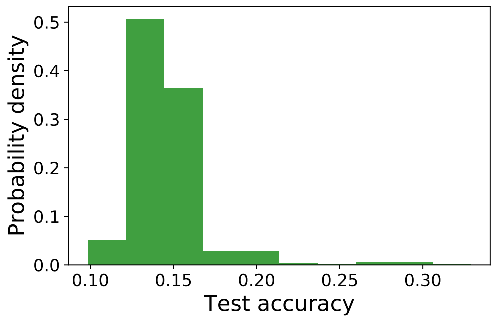
(a) MNIST, average train accuracy 80.6(±12.5) and test accuracy 78.8(±12.1).



(b) Fashion MNIST, average train accuracy 62.2(±16.9) and test accuracy 60.7(±16.3).



(c) CIFAR-10, average train accuracy 23.8(±2.8) and test accuracy 23.4(±2.7).



(d) KTH-TIPS, average train accuracy 14.8(±2.4) and test accuracy 14.6(±2.4).

Figure 8 – The distribution of classification accuracy of the neural network population considering the test set of each benchmark. It contains 1000 samples of fully connected and 4-layer deep neural networks trained on each benchmark, totaling 4000 samples.

Source: SCABINI; BRUNO. (177)

dataset, we train a population of 1000 neural networks, each one with a different random seed for weight initialization. The remaining configuration, such as training algorithms and batch order, are kept fixed for all networks. The result is a fairly distributed population of 4000 models in terms of the final performance, as Figure 8 shows.

3.1.3 Neuronal complex network characterization

Considering the neural network population, we compute CN centrality measures from its neurons. Each network is approached as a weighted, undirected, and multipartite graph. Consider a neural network $\aleph = \{L_0, ..., L_d\}$ with depth $|\aleph| = d + 1$ (number of layers), where $L_i = [n_1, ..., n_l]$ represents its ith layer with l neurons and a weight matrix $\mathcal{W}_i$, while L_0 and L_d are its input and output layers, respectively. We then construct a weighted graph $G(\aleph) = (V, E)$, where $V = [L_0, ..., L_d]$ is its set of vertices representing all neurons, and $E = [\mathcal{W}_0, ..., \mathcal{W}_{d-1}]$ their synapses matrices where rows indicate the source of a connection and columns indicate the target neuron in the following layer. The graph structure is built upon neurons and synapses and does not consider the activation function or the direction of connections.

Given the new graph representation G, centrality measures can be computed to describe each neuron n_i ($f(n_i)$). This is a classic approach in CN research, where methods try to estimate the vertex centrality, or importance, for the system structure and dynamics. Vertex centrality is, therefore, an important property of CNs and has helped to identify critical properties in a wide range of real-world systems. (24) Considering that the literature is rich on such methods (31) it is possible then to characterize neurons from different perspectives, as different measures have different topological meanings. For that, we compute centrality from hidden neurons (layers $[L_1, ..., L_{d-1}]$), *i.e.*, as our samples are 4-layers deep, only neurons from layers L_1 and L_2 are considered. The input and output layers are not considered as they are structurally different (connections in just one direction), and the first and last hidden layers already include their connections. In other words, we chose to analyze only nodes from the hidden layers, but the graph is not modified, it is still a multipartite network from all neurons and layers of the ANN.

Considering a hidden neuron n_i in the graph representation, it is possible to calculate a given measure $f(n_i)$ to characterize it. We narrowed down a set of specific measures considering different approaches/paradigms and some practical characteristics such as complexity/computation time and availability in known libraries (for easy reproduction). It is also important to notice that some measures cannot provide useful information for regular/fully connected graphs. Therefore, we employ some threshold techniques (described below).

The NetworkX 2.4 library (183) [†] is employed for computing network properties, which is one of the most complete and diffused frameworks in Python for network analysis. NetworkX and Python are used also considering other important properties such as open source code and the ease to combine it with neural network libraries. The reader can find the formulation and more details about each measure in Chapter 2, Section 2.1. Below, we give a short description of each CN measure and its interpretation in the ANN context:

- **Weighted degree, or strength** (**s**), Eq. (2.1): Represents the neuron strength, the rate at which it lets information flow. High positive strength means an excitatory neuron that gives higher importance to both its incoming and outgoing connections. Strength near zero means an average neuron, a neutral neuron (connections near zero), or a neuron whose connection spectrum is half positive and half negative, configuring an antagonistic neuron. Negative strength is the predominance of negative connections, which means inhibitory neurons. This measure is computed over

[†] https://networkx.org/documentation/networkx-2.4/

the original neural network, with all its edges and weights.

- **Average neighbor strength** ($\mathbf{s^w_{nn}}$), Eq. (2.2): Similar to the neuron strength, but focuses on the first-order neighbors of the neuron. In a neural network, neuron neighbors are the neurons on the previous and the next layer. It means that the strength of i depends on the strength of its neighbors. Therefore, if they have high strength, i probably also has high strength. This measure is computed over the original neural network, with all its edges and weights.

- **Second order centrality** (**so**), Eq. (2.3): This measure is based on the concept of random walks, and is computed over a thresholded version of the original neural network, where we keep only positive connections. It represents the standard deviation of the return times to a given neuron of a perpetual random walk on the neural network.

- **Subgraph centrality** (**sg**), Eq. (2.4): Represents the sum of weighted closed walks of all lengths starting and ending at a given neuron. A closed walk is associated with a connected subgraph, *i.e.*, a subset of interconnected neurons within the ANN. This measure is computed using a spectral decomposition of the adjacency matrix over a thresholded version of the original neural network, where we keep only positive connections.

- **Number of maximum cliques** (**mc**), Eq. (2.5): In a neural network, a clique is a subset of neurons such that every two neurons are adjacent. Considering that the unweighted neural network is regular, a thresholded version is used (only positive connections). In this scenario, a maximum clique is a path between excitatory synapses joining the largest number of neurons. Therefore, this measure counts the number of such structures in which a neuron participates, representing its importance for input stimulation.

- **Bipartite local clustering** (**bc**), Eq. (2.7): As the neural network is a multipartite graph, we consider this measure, which estimates the local neuron density according to second-order neighbors. The traditional clustering coefficient cannot be computed for our neural networks, as there are no triangles between neurons. This version then relaxes the original clustering coefficient for such structures. This measure is calculated over a thresholded and unweighted version of the original neural network, where we keep only positive connections.

- **Harmonic centrality** (**hc**), Eq. (2.8): A path-based measure of centrality, which represents the sum of the reciprocal of the shortest path distances from all other neurons to a given neuron. This measure is computed over a thresholded version of the original neural network, where we keep only positive connections.

- **Current flow closeness** ($\mathbf{CF_c}$), Eq. (2.9): This is a path-based approach, inspired by the electric flow on circuits. Each neuron pair is considered a source-target pair, where the current is injected at the source and drained at the target. The measure then quantifies the importance of neurons for the electric flow throughout all the neural network paths, *i.e.*, the synapse flow. This approach is more realistic in a neural network than the other centrality measures which assume that information flows exclusively through the shortest paths. This measure is computed over the original neural network, with all its edges and weights.

Considering these centrality measures, we propose local neuron descriptors or global average features from each hidden layer. Given a hidden layer L_i with n neurons, and a given measure f, a feature vector $v_f(L_i) = [f(v_1), ..., f(v_n)]$, for all $v \in L_i$ can be obtained by concatenating the centrality measure of each neuron. The vector v_f preserves neuron order on the layer. However, this order is not meaningful as the network is fully connected, and it also depends on random initialization. To remove this spatial information we compute the layer average:

$$v_{L_i,f} = \frac{1}{|L_i|} \sum v_f(L_i). \tag{3.1}$$

For local characterization, it is possible to compute a set of descriptors for a given neuron by combining different measures. A local feature vector is then expressed as:

$$\psi(v_i) = [f_j(n_i)], \tag{3.2}$$

where f_j is a CN measure ($f_j \in \{s,\ s^w_{nn},\ CF_c,\ bc,\ sg,\ hc,\ so,\ mc\}$). In this sense, each neuron is a sample to be characterized and not the whole layer or neural network. By analyzing the local descriptors $\psi(v_i)$ from all neurons, it is then possible to characterize the local topological properties of the neural network.

We employ a non-supervised approach to find neuronal signatures by grouping local descriptors with similar features. This approach has been employed for image recognition and is usually called bag-of-visual-words. (136) It derives from the bag-of-words method, which considers the word frequency (features) from a given vocabulary of k types. In the case of visual words, features are real-valued vectors representing local image properties. This approach has also been explored for building CN vocabularies using local descriptors based on centrality, computed from networks modeled from images. (133) Here, our features are neuronal topological signatures expressed through the CN measures of each neuron. Given a set of neural network graphs $P = [G_a]$, a matrix is built by stacking neuron descriptors from all neural networks in P:

$$D = [\psi(v_i)], \text{for all } v_i \in G_a, \text{for all } G_a \in P, \tag{3.3}$$

and $D \in \mathbb{R}^{|P| \cdot |G_a| \times m}$, where $|P|$ is the number of graphs in P, $|G_a|$ the number of hidden neurons in each neural network, and m the number of measures. We consider P as the set of neural networks trained for a given vision benchmark (1000 networks), $|G_a| = 300$ (see the architecture description), and m is the number of considered CN measures (further discussed in Section 3.1.4).

Matrix D represents all the CN neuronal properties from a set of networks P, which is considered to build the vocabulary of the k most relevant features or neuron groups. We call this method bag-of-neurons (BoN), where each vocabulary element represents a neuron type that usually occurs in these neural networks. To find them, the k-means clustering (or Lloyd's algorithm) is applied to obtain group centers in a non-supervised fashion. It finds k centroids (group centers) by first starting with random positions and then iteratively averaging its distance to the nearest sample points (D rows). The $k++$ method (184) is considered for initialization (rather than regular random sampling). We also consider relative tolerance 10^{-3} with regard to the Frobenius norm of the difference in the cluster centers of two consecutive iterations to declare convergence (we refer the reader to Chapter A for more details about k-means). The matrix D is feature-wise normalized by the largest value to avoid disproportion in the Euclidean distance calculation. A total of 100 repetitions is performed each time we run the k-means algorithm, and the best result based on inertia is used. This result is a set of m-dimensional centroids $C = [\psi_1, ..., \psi_k]$ representing the average features of the k most common neuron types. It is important to notice that even when using $k++$ and performing 100 repetitions, the position of the centroids may still slightly vary by some decimal places, as the process is still randomly initialized.

3.1.4 Analysis and results

We first analyze qualitatively the correlation between CN measures and the neural network performance by computing vectors $\upsilon_{L_i,f}$ for each hidden layer, given a measure f (see Eq. (3.1)). It summarises all the neural information into two values (1 for each hidden layer) for each measure. For qualitative analysis, the information is visualized in two axes where $x = \upsilon_{L_1,f}$ and $y = \upsilon_{L_2,f}$, representing the 2D spatial distribution of each neural network. Results are shown in Figure 9 and contain the best 100 and worst 100 samples in terms of test accuracy on the Fashion MNIST dataset, and color represents the test accuracy. It is possible to notice that the projections of some measures are practically linearly separable considering the two performance groups of networks, more specifically the strength (s), number of cliques (mc), second-order centrality (so), and the bipartite local clustering (bc). The subgraph centrality (sg) shows a total correlation between the first and second hidden layers, with a small difference in scale probably related to the different number of neurons and synapses it contains. Nonetheless, it is still possible to distinguish the two groups of networks even if considering the information from a single axis.

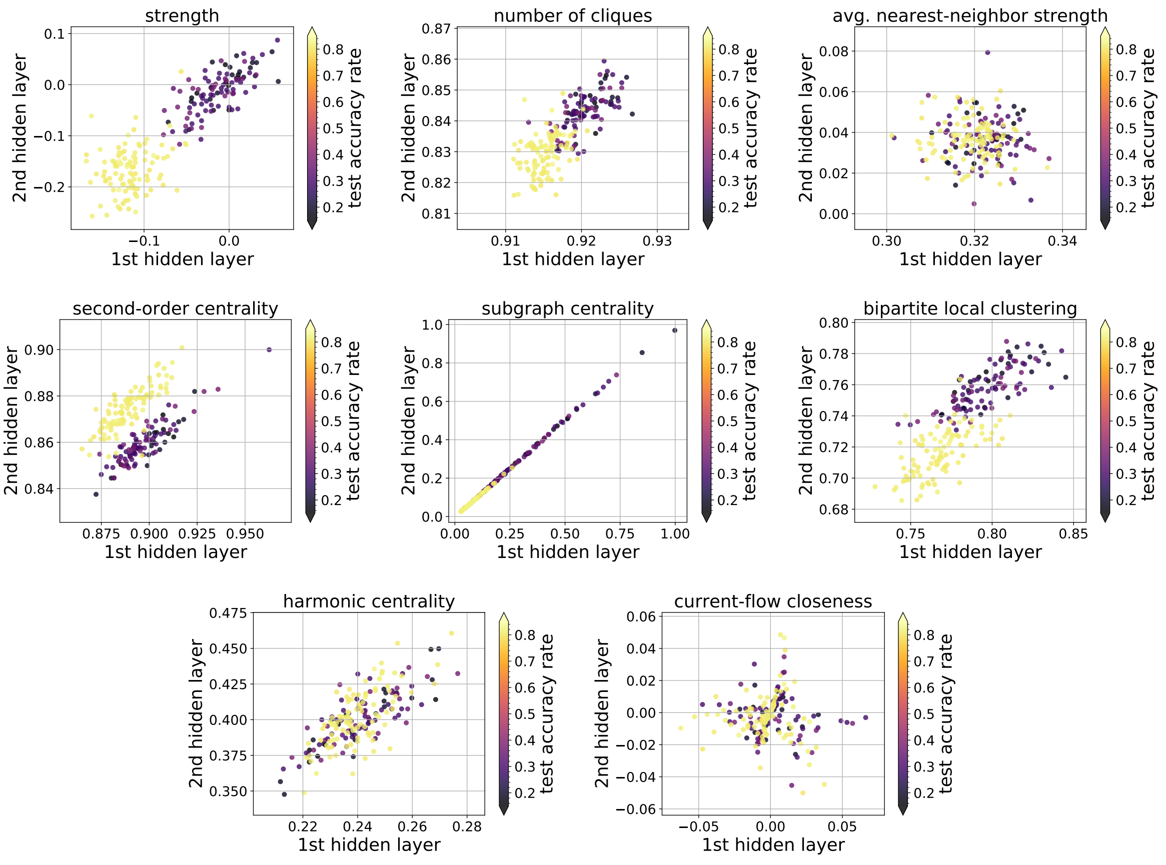


Figure 9 – Spatial distribution of trained neural networks according to the average of their hidden neuron's topological measures ($\upsilon_{L_i,f}$). We show the best and worst 100 samples, and colors represent their respective final test accuracy considering the Fashion MNIST dataset.

Source: SCABINI; BRUNO. (177)

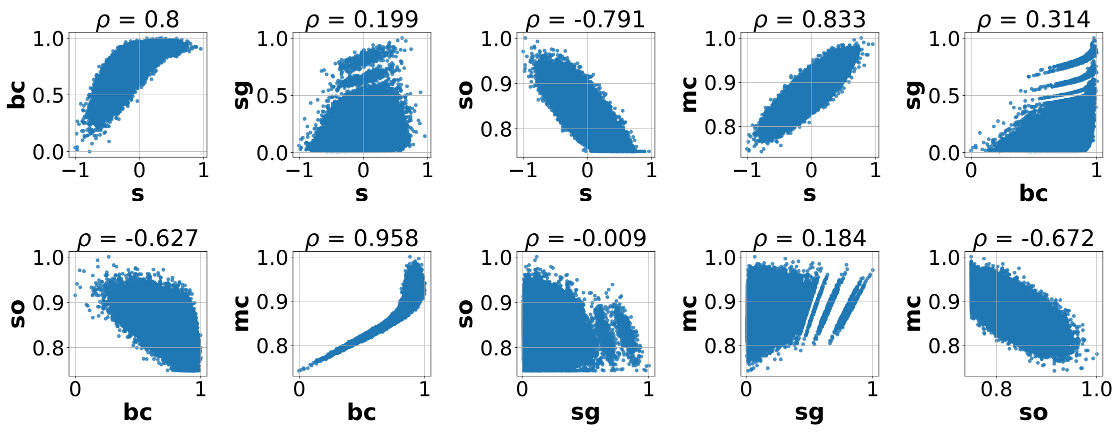
On the other hand, results suggest that the average nearest-neighbor strength (s^w_{nn}), harmonic centrality (hc), and current-flow closeness (cfc) are not correlated to performance. It is not easy to directly determine why they fail to tackle the correlation, as they represent different aspects of the graph. For instance, in (174) authors have shown that the average nearest-neighbor degree may be correlated to the functioning of some receptive fields, but they do not discuss it in depth. However, it is not intuitive to believe that the assortativity principle takes place on fully connected feedforward neural networks in terms of CN properties. For instance, considering neurons from the first hidden layer its neighbors are, respectively, the input and second hidden layer, which most probably have a different role in the network dynamics. The same accounts for the neighbors of neurons from the second hidden layer, *i.e.*, the output and first hidden layers. Therefore, considering the 4-layer architecture we analyze, our results suggest the absence of correlation between assortativity and performance. On the other hand, this assumption may not hold for deeper networks and broader vision tasks, as larger models may contain more redundancy.

Results also suggest that the CF closeness (cfc) and harmonic centrality (hc) are not directly related to neural network performance. Considering that both are path-based measures, we conclude that this kind of information may not correlate directly to the model's accuracy. Therefore, as our goal in this work is to understand structure and performance, they are not considered for the next analyses (discarded along with s^w_{nn}). Nonetheless, further studies on different scenarios would be interesting.

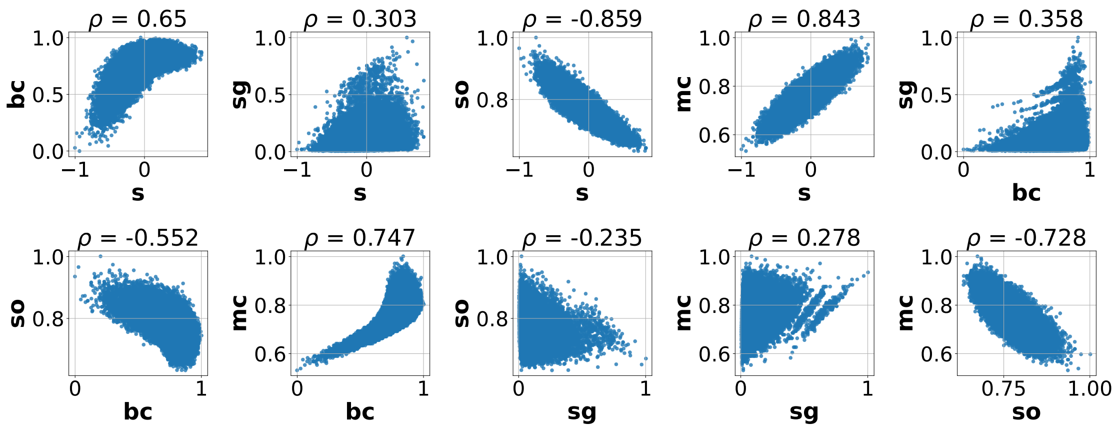
Different centrality measures may have considerable correlation even in networks from diverse domains. (185) For instance, random walks may be attracted to nodes with high degrees, reflecting the degree distribution. Therefore, we consider the Spearman correlation between measure pairs to discard redundant information. The local descriptor $\psi(v_i)$ of each neuron (see Equation 3.2) is computed considering each measure individually, resulting in a single value. We then cross these values for each measure pair, for each hidden layer separately. Each plot on Figure 10 shows the measure pairs for all neurons from the 1000 neural networks (trained on Fashion MNIST), along with the corresponding Spearman's correlation ρ. If a pair of measures have $|\rho| > 0.8$, we remove one of them to reduce redundancy and keep the simplest measure (lowest computer cost). The strength has a high correlation with the second-order centrality and maximum cliques; thus, we keep the former and remove the other two. The removed measures (so and mc) also present a considerable correlation between themselves.

3.1.5 Properties of the Bag-of-Neurons (BoN)

The remaining 3 measures are considered for neural network characterization by building the local neuron descriptor $\psi(v_i) = [s(v_i), bc(v_i), sg(v_i)]$ and combining them to obtain matrix D (see Eq. (3.3)). Our BoN approach is then performed to obtain neuron groups in a non-supervised fashion using k-means. One of the drawbacks of k-means is that the parameter k must be given beforehand, *i.e.*, the number of data groups must be known, which is one of the main problems in data clustering regardless of the chosen algorithm. In our case, we cannot assume the number of possible neuron types in neural networks according to CN principles, nor did any previous work. Therefore, we use the elbow method for estimating the optimal number of groups which is a heuristic based on the distortion measure obtained from various k values. This measure sums squared distances from each sample point to the nearest centroid found by k-means. Normally, increasing the number of clusters will naturally reduce the distortion since there are more clusters to use, but at some point, it is overfitting. The elbow method then estimates the point of inflection on the curve so that adding another cluster doesn't give much better modeling of the data. We tested k values from 2 to 18, results are shown in Figure 11. It is



(a) First hidden layer.



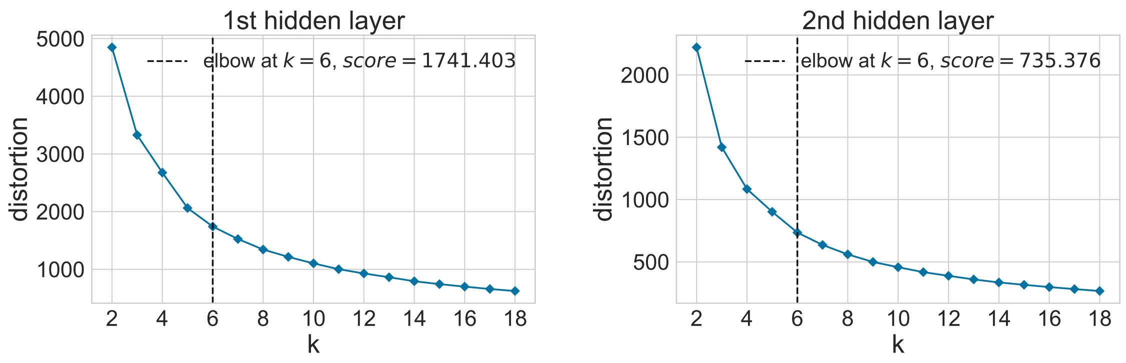
(b) Second hidden layer.

Figure 10 – Correlation between pairs of different CN measures ($f_j(v_i)$) from hidden neurons of networks trained on Fashion MNIST. Considering the first hidden layer, there are 200 thousand neurons, and considering the second hidden layer, there are 100 thousand. ρ is the Spearman correlation between measure pairs.

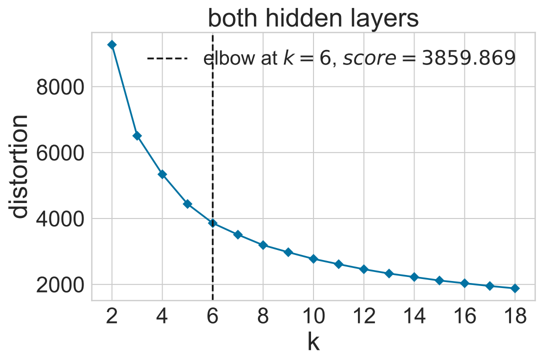
Source: SCABINI; BRUNO. (177)

possible to observe that the elbow method indicates optimal k around 6 in any case, either when grouping neurons from the first or second hidden layer separately or combined. It also happens independently of the dataset on which the neural networks were trained. This indicates a universal behavior of six topological signatures emerging on neurons of these networks, in different scenarios.

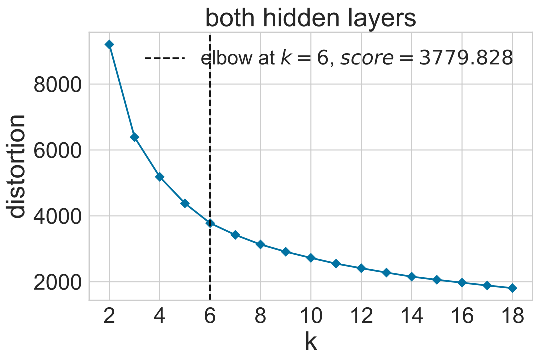
Considering the k-means results using $k = 6$ over matrix D with all hidden neurons, we show a three-dimensional visualization of the clusters in Figure 12, where the axis are the top-3 centrality measures (s, bc, and sg). We also show the obtained centroids ψ_i on each dataset in Table 2, sorted by strength (s). It is important to reiterate that



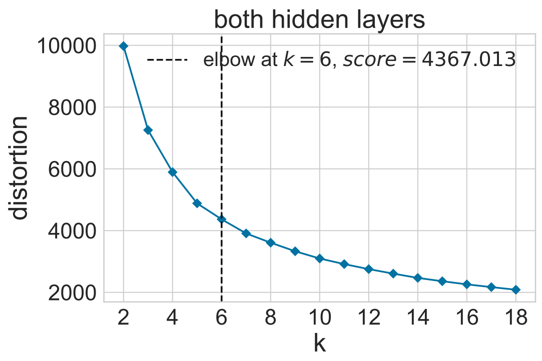
(a) Fashion MNIST, individual hidden layers.



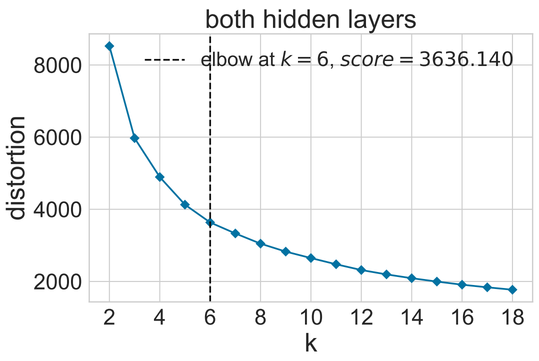
(b) MNIST.



(c) Fashion MNIST.



(d) CIFAR10.



(e) KTH-TIPS.

Figure 11 – Finding the number of neuronal groups according to the Elbow method. Clustering is performed over hidden neuron's CN measures considering 1000 neural networks trained on each benchmark. Six neuron types are found independently of the considered domain or hidden layer.

Source: SCABINI; BRUNO. (177)

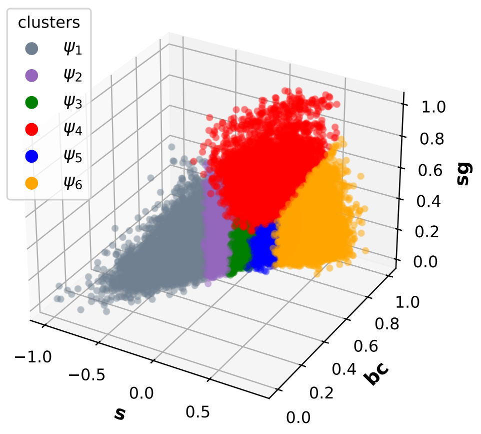
each measure is normalized by the maximum absolute value, for each dataset. Another interesting pattern is observed here. The six neuron types detected are very similar across tasks, mainly when looking at strength and bipartite clustering. There are some particularities where measures may vary, for instance, the strength and subgraph centrality on CIFAR10 at neuron type ψ_6, but most of the remaining properties are considerably similar. Differences are expected in some cases, mainly on the first hidden layer, due to the natural differences in the input image patterns. For instance, MNIST represents digits covering a fraction of the image, while the remaining pixels are the background. Fashion MNIST has a similar structure, with multiple background pixels. On the other hand, CIFAR10 and KTHTIPS have grayscale pixels covering the whole image (in most cases). The amount of background pixels then causes neurons of the first hidden layer to become more or less inhibitory. It directly influences the number of synapses connected to background information. Nevertheless, this seems to have a small impact on the six neuron types we found in the hidden layers since most of them have similar topological properties across the different tasks. This behavior also points to global topological patterns emerging independently in neural networks with different initialization and trained on different domains.

Table 2 – The properties of the six neuronal groups found on the hidden layers of neural networks trained on the corresponding benchmarks.

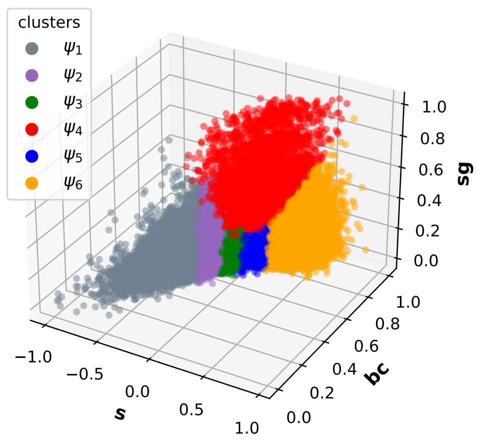
MNIST	ψ_1	ψ_2	ψ_3	ψ_4	ψ_5	ψ_6
s	-0.40	-0.21	-0.05	0.05	0.12	0.32
bc	0.57	0.73	0.82	0.87	0.86	0.89
sg	0.11	0.13	0.13	0.36	0.13	0.17
Fashion MNIST	ψ_1	ψ_2	ψ_3	ψ_4	ψ_5	ψ_6
s	-0.42	-0.24	-0.08	0.04	0.10	0.31
bc	0.56	0.72	0.81	0.86	0.85	0.88
sg	0.07	0.08	0.08	0.35	0.09	0.13
CIFAR10	ψ_1	ψ_2	ψ_3	ψ_4	ψ_5	ψ_6
s	-0.43	-0.26	-0.11	-0.06	0.05	0.27
bc	0.57	0.73	0.82	0.85	0.85	0.89
sg	0.16	0.17	0.16	0.40	0.17	0.24
KTH-TIPS	ψ_1	ψ_2	ψ_3	ψ_4	ψ_5	ψ_6
s	-0.42	-0.25	-0.11	-0.04	0.04	0.26
bc	0.56	0.73	0.82	0.87	0.86	0.89
sg	0.13	0.14	0.13	0.33	0.14	0.19

Source: SCABINI; BRUNO. (177)

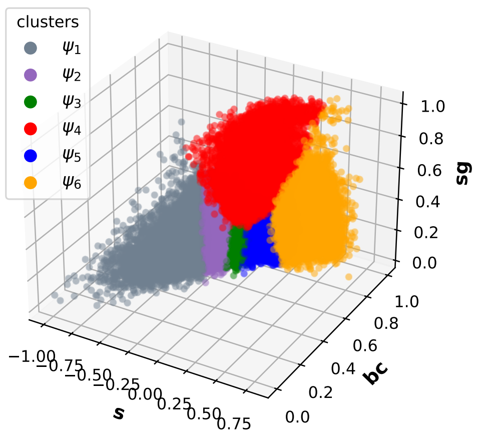
Using the obtained BoN we can find neurons that match one of the six groups by simple nearest-neighbor association to its local descriptor. To better compare the average structure of networks with different performances, we consider models from 3 accuracy ranges and compare their differences in neuron type occurrence. Figure 13 shows the distribution of neuron types for networks trained on each benchmark. They are grouped according to the colors: red represents the 100 networks with lower accuracy, blue is the 100 around the median, and green is the top 100. We can see some similarities across different



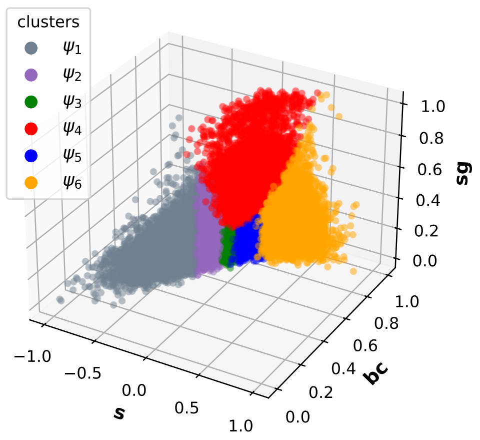
(a) MNIST.



(b) Fashion MNIST.



(c) CIFAR10.



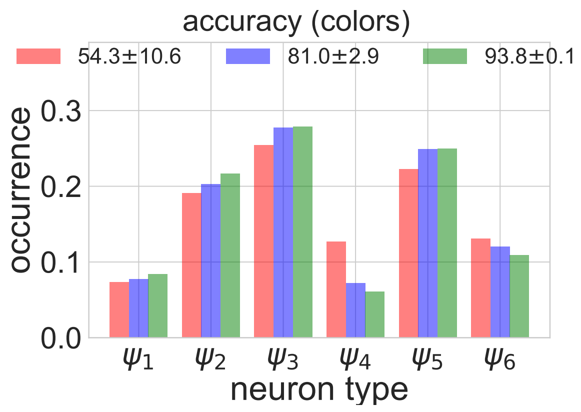
(d) KTH-TIPS.

Figure 12 – Visualization of 1.2 million hidden neurons, corresponding to 1000 neural networks (300 neurons each) trained on each benchmark, and the corresponding clustering groups (colors) found with k-means over its CN centrality measures in each case.

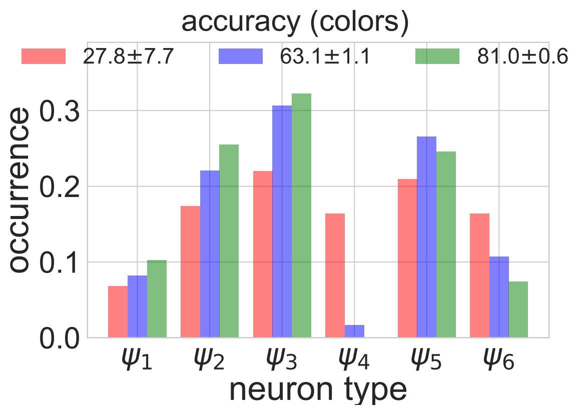
Source: SCABINI; BRUNO. (177)

tasks again. For instance, the occurrence of most neuron types follows a similar shape. These findings align with the results of (174) where authors have shown that receptive fields (neurons) with similar functions develop similar topological properties in different target domains (they considered two benchmarks). It is widely known that receptive fields in deep neural networks tend to progressively extract higher-level features from the input. In CV, our study domain, the first layers would then detect simple visual cues such as lines and circles, while deeper layers combine them into more detailed features. The simple features are usually universal and independent from the target domain.

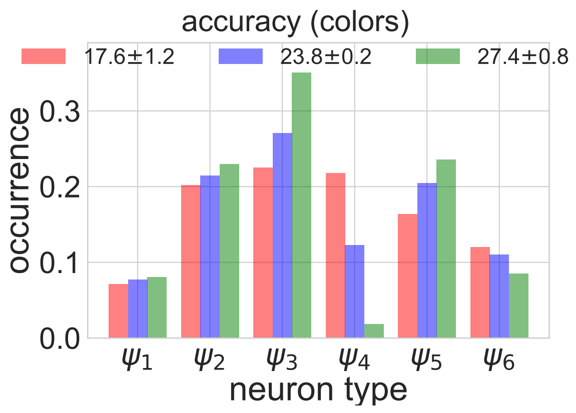
The most interesting property we can observe within the neuron types occurrence is how the distributions change according to the neural network performance. The oc-



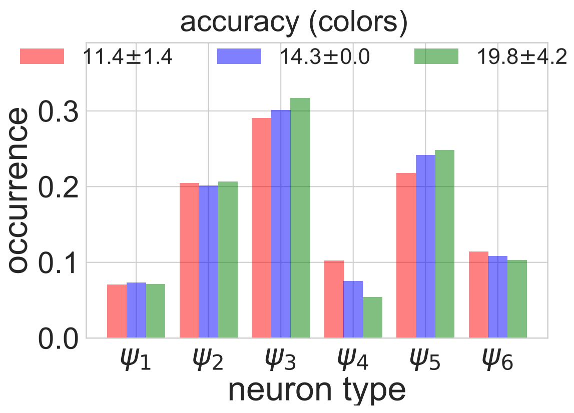
(a) MNIST.



(b) Fashion MNIST.



(c) CIFAR10.



(d) KTH-TIPS.

Figure 13 – Occurrence of neuron types (ψ_i) on neural networks trained on different vision benchmarks. Colors represent the 100 networks with the worst test accuracy (red), the 100 around the median (blue), and the top 100 (green).

Source: SCABINI; BRUNO. (177)

currence of most neuron types is different for each accuracy range (colors). Comparing the results in Figures 12 and 13, it is possible to notice how the different neuron types are distributed and their correlation to performance. For instance, neuron types ψ_2, ψ_3, and ψ_5 are more common among the best-performing networks and have a similar structure (their clusters are spatially close). Neuron type ψ_3 is the most common among the best neural networks, having the highest occurrence and also the highest occurrence difference to the lower performance networks. This group of neurons has a slight negative strength, closer to 0 compared to other neuron types. This phenomenon explains why weight initializers with normal distribution in small ranges around zero tend to yield better results than widely uniform initializers. Combining with the other properties of this neuron type, we can assume that there is a strong tendency for neural networks with higher accuracy to contain around 30% of hidden neurons with a slight negative strength $-0.1 \leq s \leq -0.05$, a considerably high clustering coefficient $bc \approx 0.8$, and subgraph-centrality $0.08 \leq sg \leq 0.16$.

On the other hand, neuron type ψ_4 seems to be less critical for better-performing networks, and it appears more frequently among those with worse performance. These neurons have a small strength magnitude and one of the highest clustering coefficients. However, their most interesting property is the highest subgraph centrality of all other types, by a significant margin, and this is also observed throughout different benchmarks. It is a notable contrast and highlights the difference between structure and performance on these networks. We then conclude that neural networks with poor performance tend to have considerably more neurons with higher subgraph centrality ($sg > 0.3$) when compared to other networks. This property may be highlighting neuron saturation, usually caused by fading/exploding gradients during gradient descent, which would explain why these networks fail to achieve higher performance.

We also calculate the similarities in neuron type occurrence when obtaining the BoN in different benchmarks. The Jensen–Shannon Divergence (JSD) is considered, which provides a smoothed and normalized version of the Kullback–Leibler Divergence (KLD). (186) This metric quantifies the difference (or similarity) between two probability distributions, with scores between 0 (identical) and 1 (maximally different). We compute neuron types occurrence on networks for BoNs obtained in the same benchmark the model was trained (ψ), or in a different benchmark ($\bar{\psi}$). We then calculate $\text{KLD}(\psi, \bar{\psi}) = \sum_i p(\psi_i) \log(\frac{p(\psi_i)}{p(\bar{\psi}_i)})$, which is used for computing JSD$= \frac{1}{2}(\text{KLD}(\psi, \frac{\psi+\bar{\psi}}{2}) + \text{KLD}(\bar{\psi}, \frac{\psi+\bar{\psi}}{2}))$. The JSD is then averaged over frequency pairs of all 1000 networks, for all possible combinations between benchmarks for the construction and application of the BoN. As we can see in the results in Table 3, the average divergence values are relatively low, around 0.21 and 0.33, which indicates significant similarity between neuron types. Nevertheless, these results suggest an average behavior of similarity emerging within neuron types from BoNs obtained with neural networks trained on different benchmarks.

Table 3 – The Jensen–Shannon Divergence between the neuron types from BoN vocabularies obtained and applied in different benchmarks, averaged over all 1000 networks of each case.

		target			
		MNIST	Fashion MNIST	CIFAR10	KTH-TIPS
vocabulary	MNIST	0	0.21 ± 0.16	0.32 ± 0.17	0.27 ± 0.14
	Fashion MNIST	0.24 ± 0.17	0	0.33 ± 0.2	0.28 ± 0.17
	CIFAR10	0.29 ± 0.16	0.25 ± 0.18	0	0.24 ± 0.19
	KTH-TIPS	0.27 ± 0.15	0.24 ± 0.16	0.28 ± 0.19	0

Source: SCABINI; BRUNO. (177)

3.1.6 Conclusion

With Network Science, we explored the correlations between the structure and performance of fully connected neural networks on CV tasks. Our first contribution is a new dataset with four thousand neural networks, each with different initial random synapses, applied to known vision benchmarks (MNIST, Fashion MNIST, CIFAR10, and KTH-TIPS). The use of an ample ANN sample size and various CN measures results in a more complete and robust statistical analysis and is another advantage over previous works. We then consider each neural network as a weighted and undirected graph and focus on analyzing local neuron properties. Eight CN centrality measures are computed from hidden neurons, which are also available in the dataset. Our analysis shows that some of them (average nearest neighbor strength, current-flow closeness, and harmonic centrality) do not present relevant information regarding the model's performance. We also find redundancy in some cases, where correlations were high (second-order centrality and the number of cliques). In the end, we found that three centrality measures contain the most relevant information (strength, bipartite clustering, and subgraph centrality) and show interesting neural network properties.

Our intuition is to quantify the emerged CN structure of an ANN after training (with a fixed architecture) caused by different random initializations. In other words, we also want to show if networks with different learned weights but similar performances could exhibit similar structural properties. Our results are sufficient to confirm these assumptions for the fully connected architecture we studied. We found a strong correlation between neuronal centrality and neural network performance. In some cases, it is possible to linearly separate the top and worst 100 neural networks by simply taking the hidden layers average of a single CN measure. We also propose a local descriptor for hidden neurons by concatenating the three best measures and found that they are usually grouped into 6 neuron types throughout the neural network population. Each group has a distinct topological signature and shares similarities even between the different tasks the neural networks are employed. Some neuron types are more common than others, and we show that their distribution pattern is also directly related to performance. We were able to highlight the CN properties related to this phenomenon, with distinct differences between different performing networks. For instance, neurons with a combination of a high clustering coefficient, and, most importantly, a much higher subgraph centrality tend to emerge more frequently in lower-performing networks. We believe our findings can also be extended to different data types (such as audio and video) and tasks, considering natural signals and normally distributed datasets. On the other hand, further investigation is needed for different optimization algorithms and loss functions to understand their impacts on emerging neuronal centrality.

Network science is a strong candidate to understand the internal properties of neural networks. Our results strongly suggest that the structure and performance of fully connected models are directly related. Our findings are important for understanding the functioning of neurons, their learning dynamics, and their impacts on the whole system's performance.

3.2 Improving randomly initialized neural networks through network science

The proposed methods and analyses presented in the previous section shed new light on the understanding of the CN properties of ANNs after training. While this is a pioneering and innovative work on the topic, one of the drawbacks of employing that full methodology in practice for larger networks is the cost to compute various centrality measures for each neuron. Moreover, the analyses focused on the models after training in order to understand their final structure, which would be costly to apply for larger and deeper architectures. On the other hand, one of the measures we studied, neuron strength, is extremely fast to compute and was able to capture the correlation between the structure and performance of the ANNs. In this sense, we now direct our efforts towards a more practical approach for ANN optimization that considers only this measure and which can generalize to larger and deeper networks. This work was developed during the author's stay at Ghent University, from January 2022 to January 2023, under the direct supervision of the co-advisor Bernard De Baets. It resulted in a finished paper (187) which we submitted for peer review in an international journal of the field.

Firstly, it is important to notice that the only degree of freedom causing the correlation between structure and performance to emerge in our previous experiments was the random weight organization. This happens because we used a fixed training procedure, so all the stochastic events of the training pipeline are the same for all networks, and, more importantly, a single weight initializer was used. This means that the weight distribution of all networks follows the same shape (a uniform distribution bounded in $[-0.9, 0.9]$), and the only difference is the way these weights are organized. Therefore, we raise the new hypothesis that the organization of random weights is an important factor in building better networks that could train to better solutions.

3.2.1 The properties of randomly initialized ANNs

Some intriguing properties have been observed in neural network initialization, for instance, specific subsets of random weights that make training of sparse ANNs particularly effective. (9) More surprisingly, these random weights may not even require additional training. (10, 11) It has also been shown that successfully trained ANNs usually converge to a neighborhood of weights close to their initial configuration. (13, 14) These works corroborate the importance of initial weights and also point toward the existence of particular random structures related to better initial models. Nevertheless, most works initialize ANNs with simple methods that only define bounds for random weight sampling. Moreover, the effects caused by the randomness of these methods are most of the time ignored, *i.e.*, researchers arbitrarily choose to consider either a single random trial or to select the one that yields the better performance within a set of trials, which is a way to over-tune and overestimate the model capabilities.

The randomness involved in ANN initialization raises some questions. For instance, what are the structural properties among a large set of randomly initialized networks? Is there any meaningful pattern on random weights that could help to improve initial models, such as the “winning lottery tickets”? (9) In this sense, we study the structural properties of random ANNs through the lens of Network Science. We focus on the most common weight initialization approaches, which rely solely on the information of a layer to produce its random weights. The proposed method is based only on the weight matrices of the model; therefore, we represent ANNs as collections of real matrices $\theta = \left(\mathcal{W}_{n_l,n_{l+1}}\right)_l$. Each matrix $\mathcal{W}$ is handled individually, and represents an undirected, weighted, and bipartite network:

$$G(\mathcal{W}_{n_l,n_{l+1}}) = (V_l \times V_{l+1}, E)\,, \tag{3.4}$$

where V_l and V_{l+1} are the sets of nodes representing the input and output neurons, respectively. The set of edges can be simply represented as the weight matrix:

$$E = \mathcal{W}_{n_l,n_{l+1}}\,, \tag{3.5}$$

where $E(i, x)$ denotes the connection weight between neuron $i \in V_l$ and $x \in V_{l+1}$.

This notation naturally covers fully connected (FC) layers, and can also be extended to convolutional layers $\mathcal{W}_{w,h,z,o}$ by expressing them as 2-dimensional matrices $\mathcal{W}_{n_l,n_{l+1}}$ with shape $n_l = whz$ by $n_{l+1} = o$, where w and h are the spatial dimensions (width and height) of a filter, z the number of input channels, and o the number of filters (output channels). In the graph analogy, this means that the filter dimensions represent the input neurons, and the number of filters represents the output neurons. This can also be extended to convolutional layers with 3 or more dimensions by flattening the additional (trailing) dimensions accordingly. In other words, we can express weight matrices in two dimensions for most types of ANN layers, even newer ones such as those present in transformers (*e.g.*, self-attention), meaning that our method can be applied in any such case. In fact, most weight initializers from the literature approach weight matrices in this fashion. Our analysis based on Network Science then considers ANN computational graphs as a set of 2-dimensional matrices or a set of input-output neuron pairs as complete bipartite graphs.

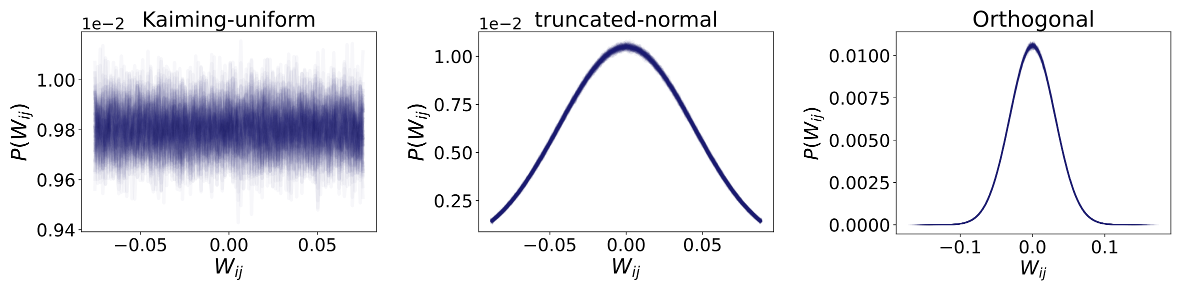
Given $G(\mathcal{W})$ for a given weight matrix, one possible approach to analyze its structure and dynamics is through node centrality. Since the network is a complete bipartite graph, this means that most of the centrality measures cannot provide meaningful information, and also that cost increases significantly if it depends on the number of edges. In this sense, and also following our previous findings, we resort to one of the simplest centrality measures: the (weighted) node degree, *i.e.*, the node strength ($s(i)$ in Eq. (2.1)). Our findings suggest that the strength of trained neurons is related to the performance of ANNs. This measure can be considered as a direct estimate of the average firing rate of artificial neurons $f_i(\sum_j \mathcal{W}_{i,j} X_j)$ if we consider a linear activation function, no bias, and i.i.d. inputs.

We consider randomly initialized networks, meaning that for each layer the corresponding matrix $\mathcal{W}$ is generated by a given random weight initializer. The strength of an artificial neuron s in such network $G(\mathcal{W})$ is obtained by summing the random weights of the edges connected to it. The central limit theorem guarantees that the sum of independent random variables is normally distributed, even if the original sample space is not normally distributed. This means that the strength distribution $P(s)$ of layers in randomly initialized networks will approximately follow a normal distribution. This characteristic is to be expected for random graphs. (22) We can also estimate the variance of s for a layer l through the law of total variance by:

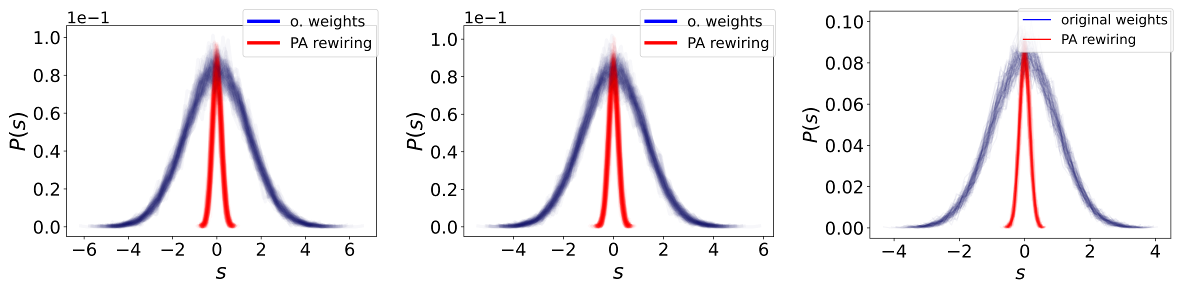
$$\sigma^2(s) = \sigma^2(\mathcal{W}) n_l\,, \tag{3.6}$$

i.e., the variance of the sum of independent random weights (strength) is the sum of the variances of the weights. Since all weights have the same variance, given by a weight initializer, we can simply multiply this variance by the number of neurons.

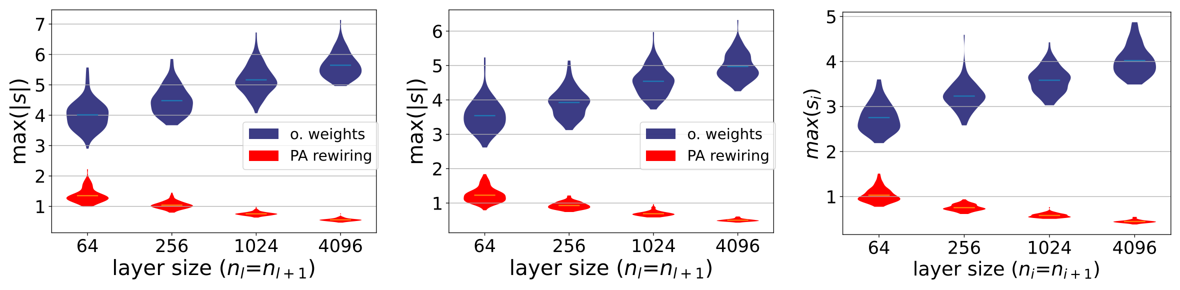
Although we can estimate $\sigma^2(s)$, the bounds for the tails of the distribution $P(s)$ are indefinite (tend to $\pm\infty$), independently from the weight distribution shape (uniform, normal, truncated, etc). This means that even if the weights are sampled from a truncated distribution, outlier neurons will emerge in the strength distribution. To illustrate this, we generate several random layers with the Kaiming method (79) using both the uniform and a truncated version of the normal distribution (truncating at $\pm 3\sigma$), and the Orthogonal initializer. (82) Results for the different methods and layer sizes are shown in Figure 14. We first show 100 weight distributions for an FC layer of sizes $n_l = n_{l+1} = 1024$ and strength distribution $P(s)$ (a-b, blue lines). One can notice the resulting long-tailed strength distribution for literature initializers. Figure 14 (c, blue) also shows that the maximum absolute strength produced by literature methods increases with layer size, even though these methods consider this size to define the sampling interval. Moreover, the maximum distribution (c, blue) is also long-tailed, asymmetric, and has a considerably high variance. This means that neurons with highly negative and positive strengths emerge during random initialization in a complex way. Intuitively, these effects are also expected to happen with other common initializers (*e.g.*, Glorot, and the original Kaiming-normal method).



(a) Weight distributions generated by initialization methods (for sizes $n_l = n_{l+1} = 1024$).



(b) Strength distributions for the above cases (a), including the result after applying the proposed PA rewiring.



(c) Maximum strength distributions for different layer sizes.

Figure 14 – Properties of random weights generated with literature methods (original weights, blue) or modified by the proposed PA rewiring method (red). We show the generated weights (100 trials) (a), their strength distributions before and after PA rewiring (b), and the distribution of maximum absolute strength observed in each case for different layer sizes (c).

Source: By the author.

The results shown above are not entirely unexpected, since they are well-known in probability theory as the result of summing an increasingly higher number of random variables. Moreover, outlier nodes with extreme connectivity are a common property of real-world networks with the scale-free characteristic, (2) *i.e.*, the presence of hubs causing the strength distribution to follow a power law. However, there is no evidence that such a structure is beneficial for the random initialization of ANNs. We argue that hub neurons at initialization may harm the training performance and gradient flow, since they may dominate the layer's signal both at forward and backward passes. This is intuitive to imagine if we consider a ReLU network, where a neuron with a highly positive strength may generate signals with a high magnitude, while a highly-negative strength may cancel a significant part of the signal. Such cases may cause problems of neuronal saturation, exploding/vanishing gradients, or dying ReLU units. In this sense, one would expect the initial strength of neurons to be correctly scaled for signals to propagate uniformly, and for more neurons to be able to learn collectively. In the following, we discuss our proposed method based on these observations.

3.2.2 Rewiring through Preferential Attachment (PARw)

We hypothesize that the high neuronal strength observed with literature initializers may not be ideal for ANN initialization. To modify their strength distribution, we consider a well-known model from Network Science, the scale-free network. (2) The algorithm to generate scale-free networks is based on a strength preferential attachment (PA) rule, which consists of increasingly including new nodes and edges into the network giving higher priority to hubs ($P(i) = \frac{s(i)}{\sum_j s(j)}$) when wiring them. This greedy iterative process causes a small fraction of nodes to receive a large portion of the network connections, creating the so-called hubs, while the majority of nodes keep a small strength.

For ANN initialization, we propose to avoid the scale-free structure usually present in real-world networks. First, consider that randomly initialized layers may contain negative strengths and connections (the most common approach for ANN initialization). Given a weight matrix $\mathcal{W}_{n_l,n_{l+1}}$, initialized by some random initializer, consider i as indices for neurons in the input layer l and x as indices for neurons in the output layer $l+1$. Let us start by assuming that only one of the neurons x in layer $l+1$ is connected to all other neurons in layer l, which we will refer to as the first neuron $x = 1$. In this sense, we can compute the current temporary strength $s_{t=1}$ of a neuron i in the first layer, at the initial iteration $t = 1$, by $s_{t=1}(i) = \mathcal{W}_{i,1}$, since each neuron i will contain only a single connection coming from x. We then propose the following PA probability for these nodes to receive "new" connections:

$$P(i) = \frac{s_t(i) + |\min(s_t)| + 1}{\sum_j s_t(j) + |\min(s_t)| + 1}\,, \tag{3.7}$$

which means that we shift the strength distribution to the positive side, and add 1 to avoid null probability. In this sense, the probability ranges from the lowest negative strength (smaller probability) to the highest positive strength (higher probability). Now we cycle through $t \in [2, n_{l+1}]$ iterations visiting a neuron x at each iteration. The weights $\mathcal{W}_{i,x}$, $i = 1, \ldots, n_l$, in ascending order, are then rewired to nodes in layer l according to the probabilities $P(i)$. In other words, neurons with negative strength have more chance to receive new positive connections, while the same goes for positive strength and negative edges. At each iteration t, a new temporary strength is calculated after the last rewiring done in the previous iteration $t-1$:

$$s_t(i) = \sum_{x=1}^{t-1} \mathcal{W}_{i,x} = s_{t-1}(i) + \mathcal{W}_{i,t-1}\,, \tag{3.8}$$

along with the new probabilities $P(i)$. A new rewiring w.r.t. the input neurons can then be performed with the weights of a new neuron $x = t$. After rewiring all neurons in layer $l+1$, this procedure will reorganize ANN weights so that the neuronal strength of input neurons is closer to zero, as we show in Fig. 14 (b-c, red). The whole process is also described in Algorithm 1. Note that we assume that $\mathcal{W}$ is already filled with random values, which is needed for computing the strengths, and no weights are created in practice as we simply assume that existing weights are reorganized.

Algoritmo 1: A python-like pseudo-code of the proposed PARw method.

```
1  function PA_rewiring ($\mathcal{W}$);
   Input  : An $n_l \times n_{l+1}$ weight matrix $\mathcal{W}$ filled by some weight initializer
   Output: $\mathcal{W}$ after rewiring
2  $s_1$ = zeros($n_l$)
3  for $t$ from 2 to $n_{l+1}$, do
4      $s_t = s_{t-1} + \mathcal{W}[:, t-1]$
5      $P = s_t + \text{abs}(\min(s_t)) + 1$
6      $P = P/\text{sum}(P)$
7      targets = random_choice($[i \in [1, ..., n_l]]$, prob=$P$, size = $n_l$)
8      new_edges = argsort($\mathcal{W}[:, t]$)
9      $\mathcal{W}[\text{targets}, t] = \mathcal{W}[\text{new\_edges}, t]$
10 end
11 return $\mathcal{W}$
```

The PARw process described performs the following action: for a layer and its corresponding weight matrix, rewire its connections so that the strength distribution of the input neurons tends to zero. This procedure does not affect the strength of the neurons in the output layer, since we are rewiring just the end-point of the connection pointing to the input layer, for each output neuron individually. To achieve the same for the output neurons, the rewiring process can be repeated by simply reversing the targets: rewiring connections from each input neuron to the output neurons, making the strength of the output neurons also tend to zero. In fact, the distributions that are shown in Figure 14 (b-c) after rewiring already consider this two-step procedure for layers with sizes $n_l = n_{l+1}$. The whole procedure is illustrated in Figure 15, where one can observe how an initial long-tailed strength distribution, generated by a traditional initializer, vanishes when connections are reorganized with PARw. Note that the two rewiring rounds are needed to effectively reduce both input and output strengths. In this sense, we are considering both signals to be equally important for ANN functioning. However, it is important to stress that we did not analyze the individual impacts of the input and output strengths, for instance by testing the rewiring individually for each of them.

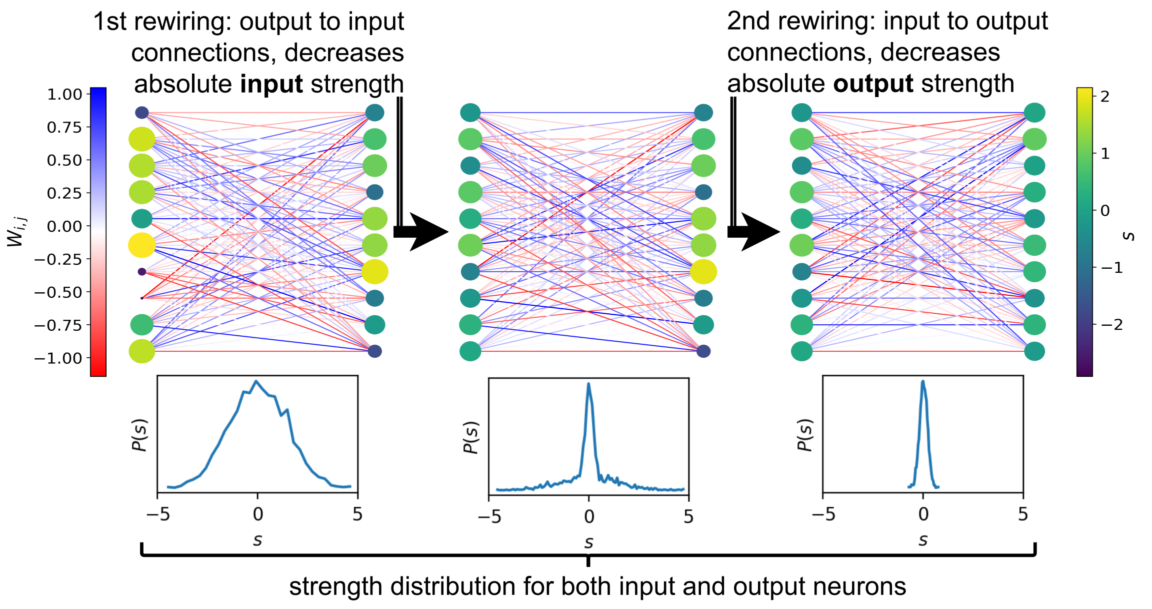

1st rewiring: output to input connections, decreases absolute **input** strength
2nd rewiring: input to output connections, decreases absolute **output** strength
$w_{i,j}$
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
2
1
0
−1
−2
s
P(s)
−5
0
5
s
strength distribution for both input and output neurons

Figure 15 – Illustration of the functioning of PARw at reorganizing weights to decrease absolute neuronal strength. The size of the neurons represents their strength, from the most negative (smaller) to the most positive. We show a small layer with 10 neurons and 10 inputs, while the strength distributions are calculated for a layer with 1024 neurons and 1024 inputs.

Source: By the author.

3.2.3 Computational complexity

The cost of the PARw algorithm mainly depends on the size of the layer one needs to rewire, *i.e.*, the numbers of input and output neurons n_l and n_{l+1}. The main loop iterates over all except one of the output neurons x in layer $l+1$, in a total of $n_{l+1}-1$ iterations. Let us then estimate the cost of the operations inside the loop. The computation of the temporary strength s_t (line 4) adds an array of n_l connections to the previously stored array of strengths, *i.e.*, a cost of $\mathcal{O}(n_l)$. To compute the PA probabilities, we need to find the minimum strength ($\mathcal{O}(n_{l+1})$), add it to s_t ($\mathcal{O}(n_l)$), and divide it by the sum of s_t ($\mathcal{O}(n_l)$). Therefore, the computation of $P(i)$ entails a total cost of $\mathcal{O}(3n_l)$. The cost to select the target nodes (line 7) also depends on the number of neurons in the layer ($\mathcal{O}(n_l)$). Finally, the most expensive operation inside the loop is the sorting of the connections between x and all other neurons in layer l, yielding $\mathcal{O}(n_l log(n_l))$. There is an additional cost of $\mathcal{O}(n_l)$ to effectively rewire the connections, *i.e.*, to swap the weight matrix positions (line 9). The total cost for the operations inside the loop is then $\mathcal{O}(6n_l + n_l log(n_l))$. By nesting these operations with the outer loop and removing the constant, the total cost is:

$$\mathcal{O}(n_{l+1}(n_l + n_l \log(n_l))), \tag{3.9}$$

or, if we consider that $n_l \approx n_{l+1} = n$ (average number of neurons per layer):

$$\mathcal{O}(n^2 \log(n))\,. \tag{3.10}$$

It is important to note that since our graph modeling approach depends on the number of weights $|\mathcal{W}|$, n will be considerably smaller for convolutional layers compared to dense (FC) layers.

3.2.4 Experimental setup

In this section, we give a detailed description of the experimental setup and the results obtained. We focus on a more robust statistical evaluation rather than performing a single trial with several models on huge datasets, which is the most common approach in the deep learning literature. Instead, several repetitions are performed by varying random seeds for weight initialization. We freeze all the other seeds responsible for the stochastic/random processes involved in building and training the models and managing the data. A total of 100 repetitions were performed for each case (one case is a single combination of model, initializer, and task) with the first learning schedule, and 10 repetitions for the experiments with deeper models and data augmentation (Section 3.2.7). We believe that this is a more comprehensive and robust validation to assess the individual impacts of random weights, which is our main goal (rather than seeking the highest possible performance). Moreover, some experimental constraints were carried out due to limited computing resources, since we do not have access to enormous clusters with hundreds of GPUs, as required by SOTA deep learning research (usually performed by big techs).

The results presented in this section correspond to two different scenarios: a baseline case with a simple learning schedule that is less computationally expensive, and a more complex scenario (see Section 3.2.7). For the former case, we do not employ any data augmentation, which allows us to allocate all training data into GPU memory for faster computation. We consider SGD with momentum equal to 0.9, batch size 128, and a cosine schedule (188) with an initial learning rate of 0.01 (annealed to 0, no restarting). We found that this configuration was the better overall choice to achieve convergence with the architectures and datasets analyzed in this first scenario. We test different models, initialization methods, and datasets, and also perform a statistical analysis (100 trials) of both model performance and training dynamics.

Datasets: To cover different tasks, we consider MNIST, (139) FMNIST, (181) and CIFAR10. (182) Each dataset contains a fixed train and test set, and we use an additional split: a new validation set randomly sampled from the training set, with the same size as the test set, used for model selection. The test set is held out for evaluation at the end of the training. It is well known that the use of an additional validation data split is highly important to assess the real capabilities of a model and to avoid a biased and overfitted selection. However, this is not usually done in deep learning research since the ultimate goal is to maximize performance on a fixed test set, which is the case of the most common datasets (e.g. the ones used here). In this scenario, there is a danger of overfitting to excessively re-used test sets. (180) Therefore, the use of different validation sets has been discussed in the deep learning literature as a good practice for a more robust experimental setup. (158,159,180) Considering our work, it is important to stress that our goal is not to achieve the highest possible performance but to strictly measure the effects of random weights with a meaningful sample size. For instance, the use of an additional validation set consequently reduces the number of training samples, leading to reduced performance. Moreover, we apply the models on the real test set just once: after finishing training, using the best-performing model on the validation set.

Weight initializers: We considered methods from the literature that do not require external knowledge about the data or architecture itself, and act only locally at each layer. The following methods are considered for comparisons and to be evaluated as the base weights for PARw: both uniform and normal versions of the Kaiming method, (79) a truncated version of the Kaiming-normal method (truncating at $\pm 3\sigma$), and orthogonal weights. (82, 83) It is important to stress that these techniques were selected due to their similarity to the proposed method, *i.e.*, they work locally in a given layer without external knowledge of the data or architecture, except for the layer size and the activation function which are needed to define the variance of the weights. For each initializer, 100 trials are performed, *i.e.*, 100 random seeds are used to initialize each model, in each experiment. The proposed PARw method is applied to the weights generated by these seeds. To quantify the individual effects of different random weights, all other stochastic factors involved in the model construction and training are fixed using a global seed. Data is randomly split into training and validation once, and the same splits are maintained throughout all trials. The training random shuffle, *i.e.*, the order in which the training batches are presented to the model, and the random batch generation are also maintained for all trials.

Statistical evaluation: To obtain statistically meaningful results, we consider the average and standard deviation for all metrics, as well as the median and median absolute deviation. Moreover, when comparing the averages of any two methods, we also perform a t-test considering the null hypothesis that both cases have identical averages. When comparing the medians of any two methods, the Kruskal-Wallis H-test test is performed to check the null hypothesis that their medians are equal. In both cases, we consider a statistically significant difference when $p < 0.05$ (more than 95% confidence). Colored table cells in the results of this section represent whether or not the statistical test detected a significant improvement (green), a significant weakening (red), or if the null hypothesis was confirmed with $p \geq 0.05$ (gray). Statistical tests are always performed w.r.t. a literature initializer compared to some weight operation (such as the PARw method).

3.2.5 The effects caused by the neuronal strength of random networks

To highlight the impact of neuronal strength on ANN performance, we start with a baseline experiment as a sanity check. A LeNet architecture (51) is considered, with four layers (two convolutional and two FC), ReLU neurons, and additional layer sizes of 20 and 50 (convolutional filters), and 800 and 500 (FC layers), respectively. This configuration was employed considering that it has the same number of convolutional and FC layers, and the layer widths were increased to compensate for the lack of one layer (common in the LeNet-5 version). A hundred random LeNet models are then generated, each with a different random initialization. Before training, we compute statistical properties from their strength distribution. The models are then trained on CIFAR10 with everything else fixed: the same training hyperparameters, data splits, etc. Therefore, the only degree of freedom is the random weight organization, while the shape of the distribution is the same since we are considering a single initializer in each experiment. Figure 16 shows the spatial distribution of the networks considering the average variance $\overline{\sigma^2}(s)$ (between all layers) and average fourth central moment $\overline{\mu_4}(s)$ (non-normalized kurtosis) of their strength distributions. Colors represent the final test accuracy after training; r and p are the Pearson correlation coefficient and the corresponding p-value testing the null hypothesis, *i.e.*, whether the correlation coefficient is in fact zero. It is possible to notice a statistically significant ($p < 0.05$) negative correlation of ≈ -0.3 between both measures before training and the model's performance at the end of training. This means that strength distributions with lower variance and tails will likely provide higher performance after training. It is also possible to notice that the Orthogonal initializer produces strength distributions with smaller tails and lower variance, which seems to reflect its improved overall accuracy compared to the Kaiming-normal initializer. The fact that such measurable patterns can be detected in random weights corroborates our hypothesis that the strength distribution is an important factor when initializing ANNs.

To support our hypothesis and the previous findings, we propose to further explore the strength distribution of random ANNs as a way to improve initialization. First, a naive approach is considered as a baseline: given a network and a random weight initializer, for each layer, repeat the initialization process K times and select one layer from this set that satisfies some desired property. The minimum and maximum strength variance ($\sigma^2(s)$) of the layer are considered as the selection criteria. It is important to stress that we are not selecting the tails of the average distribution for the whole model, as shown in Figure 16. Instead, this is a random search applied to each layer of the model independently, producing layers that are extremely unlikely to appear simultaneously in a single model by chance. In addition, we also apply the proposed PARw method to the original weights (same seeds). Figure 17 shows the average training dynamics of LeNet networks by considering the described approaches on the MNIST and CIFAR10 datasets. It is pos-

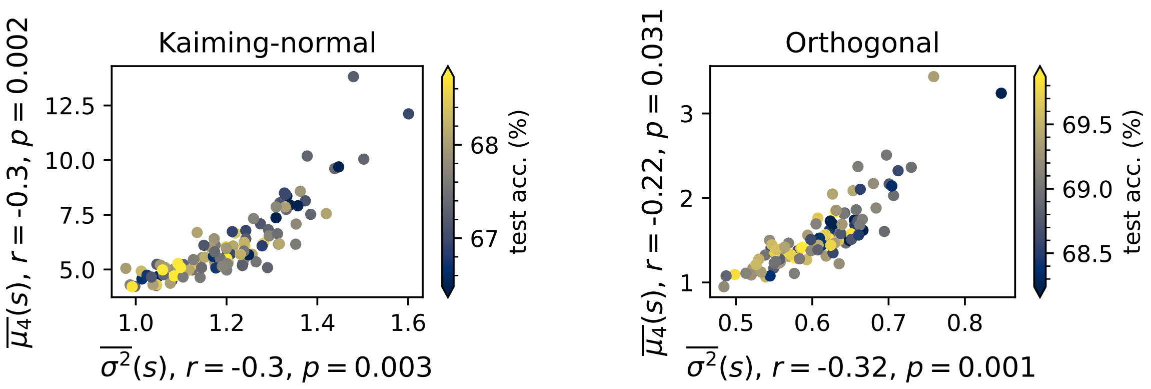


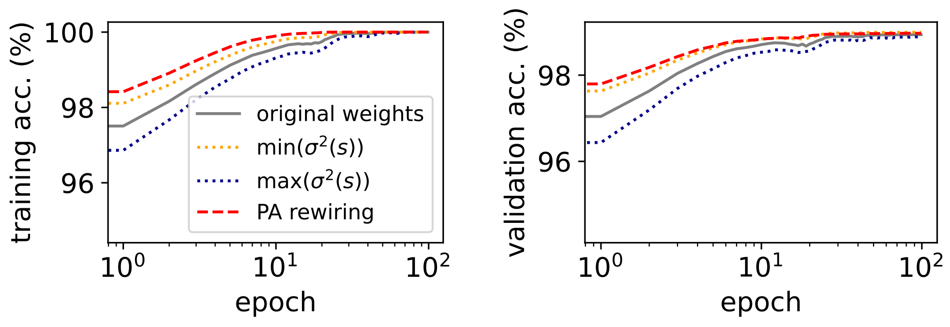
Figure 16 – The statistical properties of the neuronal strength distribution (average variance and fourth central moment) of a set of 100 LeNet models generated with each random weight initializer, trained on CIFAR10. Statistics are computed before training, and colors represent the final test accuracy after training; r and p are the Pearson correlation coefficient and corresponding p-value w.r.t. the test accuracy.

Source: By the author.

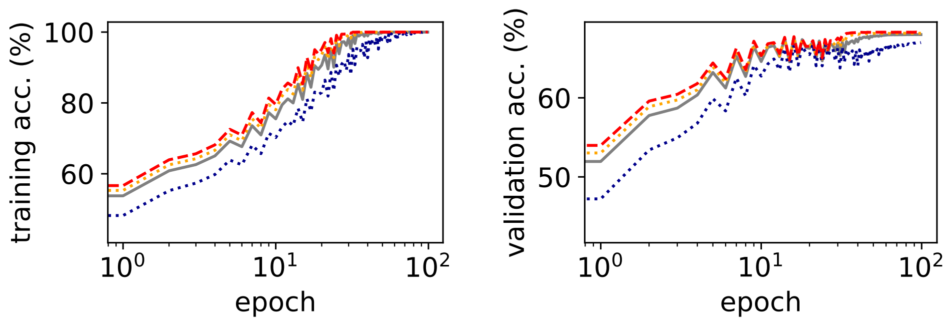
sible to notice a symmetric behavior: while maximizing the strength variance decreases both training and validation accuracy during training, minimizing it increases both metrics, surpassing the original weights. Moreover, the proposed method provides a greater improvement than simply randomly reducing the strength variance. In other words, these procedures create models with a better initial parameter space, as the results at epoch 1 show, and this superiority persists throughout training.

Table 4 shows statistical metrics of the results in Figure 17, and also of the final test accuracy of the models. The final model is selected considering the highest validation accuracy obtained during the 100 training epochs. A considerable statistical significance is achieved when confirming that a higher strength variance decreases model performance, while lower values increase it. It is also possible to observe that it similarly affects the convergence time. Another interesting effect is that the standard deviation of most metrics increases when maximizing the strength variance. On the other hand, when minimizing the strength variance or using the proposed PARw method, the standard deviations decrease, corroborating a more stable scenario. PARw achieves the best results overall, both regarding training performance and convergence time and also for the performance after training. These results corroborate two hypotheses: a lower neuronal strength at random initialization is important for better training dynamics, as random minimization shows; and the organization of the weights is also important since we achieve both properties using PARw and better results.

To better understand the effects of PARw during training, we compute the average



(a) MNIST.



(b) CIFAR10.

Figure 17 – The average training dynamics of 100 LeNet networks on MNIST and CIFAR10 with the Kaiming-normal initialization (grey line) compared to applying different operations on the original weights. The x-axis (epochs) is on a logarithmic scale for better visualization since higher increases happen at the first training epochs.

Source: By the author.

absolute gradient flow $\overline{|\nabla_{\mathcal{W}}|}$ at each epoch, for each layer of the LeNet models, and show the results in Figure 18. Each curve represents the average over the 100 models, with continuous lines related to the original weights and dashed lines for the models using PARw. One can notice that the gradient flow is reduced at the output layer, and this behavior is consistent using both Kaiming-normal and Orthogonal weights, for both datasets. This effect is related to the lower classification loss achieved by the models using PARw. On the other hand, the gradient flow increases in the preceding layers (at the initial epochs) and the models converge faster. This is an important behavior that helps to avoid the vanishing gradient problem. Therefore, the fact that higher gradient values propagate to preceding layers even when the classification loss is lower, corroborates that PARw improves the backward signal flow during training.

Table 4 – Statistical metrics computed for 100 LeNet models trained on MNIST and CIFAR10, with different weight initializations. Colors represent statistical tests ($p < 0.05$) performed w.r.t. the base weights (Kaiming-normal), with green meaning a positive difference, and red representing a negative difference. Classification accuracy is measured by percentage, while convergence is measured by the number of epochs.

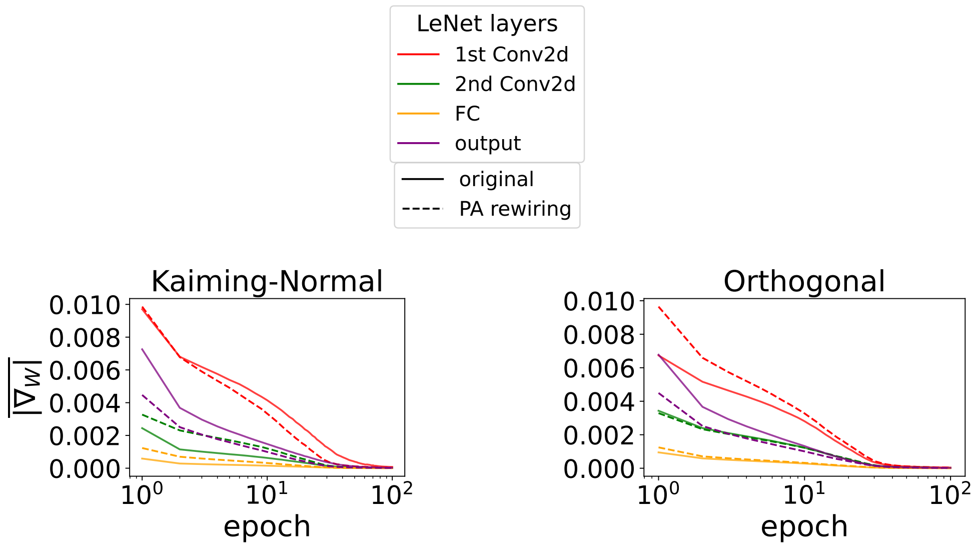
		training metrics			test acc.	
	weights	ep. 1 acc.	ep. 1 val. acc.	converged	average	median
MNIST	Kaiming-normal	95.93±2.31	95.54±2.26	16.10±6.59	99.01±0.13	99.03±0.13
	$\max(\sigma^2(s))$	94.67±2.68	94.35±2.64	20.95±8.42	98.95±0.18	98.98±0.11
	$\min(\sigma^2(s))$	96.93±0.61	96.53±0.62	13.06±4.30	99.10±0.08	99.10±0.06
	PARw	97.37±0.31	96.85±0.32	10.22±2.40	99.10±0.07	99.10±0.05
CIFAR10	Kaiming-normal	50.17±1.60	49.09±1.34	10.95±1.98	67.59±0.65	67.57±0.65
	$\max(\sigma^2(s))$	43.37±6.85	42.95±6.66	15.75±3.23	66.81±0.74	66.76±0.51
	$\min(\sigma^2(s))$	52.08±0.93	50.76±0.91	9.86±1.02	67.77±0.46	67.75±0.34
	PARw	53.30±0.56	51.57±0.58	9.60±0.80	68.03±0.53	68.07±0.37

Source: By the author.

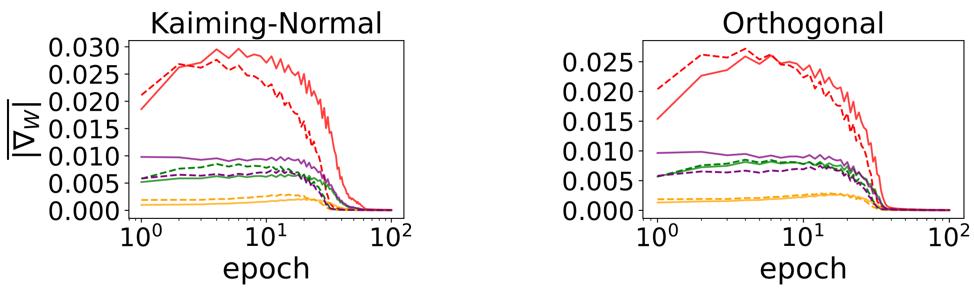
3.2.6 Systematic comparison

A broad empirical evaluation is performed to assess the impact of the proposed PARw method compared to literature weight initializers in different scenarios. We start with a simple model: a fully connected architecture with two hidden layers with 64 ReLU neurons each (referred to as shallow-thin). The intuition is to verify the effects of PARw on a small scale with simple architectures. The model is trained on MNIST and FMNIST datasets, which are composed of grayscale images with simpler visual patterns. The results for this architecture are shown in Table 5. It is possible to observe that PARw improved the training metrics in the majority of cases, and improved final test accuracy for most cases on the MNIST dataset. On the FMNIST dataset, the same is observed regarding training performance, but not after training. In this case, although the proposed method improved training and convergence, in the end, the models reached a similar performance. This could be due to the limitation of this very small architecture for the given tasks.

We consider another FC architecture with increased complexity: eight hidden layers with 2048 ReLU neurons each (deep-wide). The results are shown in Table 6, indicating a similar behavior as observed with the shallow-thin model. Training performance and convergence are improved in most of the cases, and test performance gains were observed when using the Kaiming-Uniform initializer. The performance of the shallow-thin and deep-wide architectures shows that this is not the ideal architecture for these tasks, and also that scaling a traditional FC model, in a traditional training schedule, results in poor performance improvement. Nevertheless, we evaluated the PARw method in such a scenario as a baseline check, and the results show that the training performance is improved



(a) MNIST



(b) CIFAR10

Figure 18 – Average of the absolute gradient $\overline{|\nabla_{\mathcal{W}}|}$ at each layer during the training of 100 LeNet networks on MNIST and CIFAR10, with two different weight initializers. The continuous line represents the behavior using the original weights, the dashed line is the effect after applying the proposed PARw, and the colors represent the corresponding layer. The x-axis (epochs) is on a logarithmic scale for better visualization since higher values appear at the first few training epochs.

Source: By the author.

in most cases. As for the testing performance, the method usually improves performance or keeps it similar to that of the base weights, except for a single case with the deep-wide architecture.

As previously stated, common FC architectures are not the best approach for complex CV tasks. Therefore, we focus on more modern architectures in the following experiments. Table 7 shows the results obtained with the LeNet architecture for different weight initializers on all three datasets (MNIST, FMNIST, and CIFAR10). Significant improvements in training performance are observed in the majority of the cases, while some isolated cases perform similarly to the base weights. Significant test accuracy im-

Table 5 – Comparison of the proposed PARw method applied to several weight initializers for the shallow-thin FC architecture. For each case, 100 models were trained, and colors represent the result of statistical tests performed between the base weights and the rewired version.

		training metrics			test acc.	
	weights	ep. 1 acc.	ep. 1 val. acc.	converged	average	median
MNIST	Kaiming-Uniform	93.05±0.36	92.29±0.44	15.83±4.14	97.43±0.12	97.44±0.12
	PARw	93.38±0.30	92.56±0.34	15.24±4.00	97.47±0.13	97.47±0.08
	Kaiming-Normal	93.05±0.32	92.30±0.35	15.54±3.63	97.41±0.13	97.40±0.13
	PARw	93.27±0.42	92.48±0.44	15.08±2.99	97.45±0.11	97.44±0.07
	truncated-normal	92.99±0.41	92.26±0.41	15.91±3.79	97.42±0.11	97.42±0.11
	PARw	93.33±0.33	92.57±0.34	15.62±4.40	97.47±0.12	97.47±0.08
	Orthogonal	92.34±0.38	91.70±0.42	18.44±5.17	97.58±0.11	97.60±0.11
	PARw	92.61±0.34	91.95±0.36	19.54±5.13	97.59±0.13	97.60±0.09
FMNIST	Kaiming-Uniform	83.31±0.71	82.81±0.75	26.34±6.26	88.28±0.24	88.27±0.24
	PARw	83.60±0.65	83.03±0.64	24.24±7.70	88.32±0.24	88.32±0.16
	Kaiming-Normal	83.18±0.75	82.80±0.78	25.44±7.21	88.19±0.22	88.22±0.22
	PARw	83.41±0.75	83.02±0.77	25.02±7.31	88.22±0.24	88.23±0.15
	truncated-normal	83.22±0.73	82.84±0.75	26.13±6.50	88.22±0.20	88.22±0.20
	PARw	83.46±0.64	83.00±0.65	24.08±7.60	88.24±0.20	88.24±0.13
	Orthogonal	82.77±0.77	82.48±0.80	29.14±2.76	88.43±0.18	88.46±0.18
	PARw	82.94±0.78	82.66±0.81	29.25±2.90	88.43±0.18	88.42±0.13

Source: By the author.

provements are also observed for the majority of the cases, except for Orthogonal weights on FMNIST, the only case with a measurable decrease in performance after applying PARw. In general, the results show that in these tasks and using a convolutional model, PARw provides more significant gains consistently in different scenarios, especially on a more complex task (CIFAR10).

3.2.7 Deeper models and advanced training pipelines

To measure the impact of the proposed PARw method in a more complex scenario, we consider deeper models and more rigorous training schedules. In some cases, we adapted the training schedule to be more suitable to the available computing resources (balancing batch size with VRAM usage and time to train), and to achieve convergence on CIFAR10 (also considering that we split off an additional validation set, resulting in less training samples). The following architectures are analyzed with the CIFAR10 dataset:

- ResNet (18 and 50): (61) For ResNet18, the training configuration is the same as in the previous section. For ResNet50, training is done using the AdamW optimizer (71) for 150 epochs with a cosine learning rate schedule with an initial learning rate of 0.02 (annealed to 0, no restarting), weight decay of 0.005, batch size 512, and

Table 6 – Comparison of the proposed PARw method with different weight initializers for the deep-wide FC architecture, trained on the MNIST and FMNIST datasets.

		training metrics			test acc.	
	weights	ep. 1 acc.	ep. 1 val. acc.	converged	average	median
MNIST	Kaiming-Uniform	95.74±0.69	94.47±0.71	5.94±1.05	98.10±0.07	98.10±0.07
	PARw	95.92±0.72	94.49±0.73	5.66±1.34	98.14±0.08	98.14±0.05
	Kaiming-Normal	93.04±2.52	92.03±2.40	8.22±2.82	98.14±0.10	98.15±0.10
	PARw	95.96±0.79	94.54±0.77	5.55±1.06	98.10±0.08	98.11±0.05
	truncated-normal	95.75±0.72	94.46±0.72	5.79±0.90	98.12±0.10	98.12±0.10
	PARw	96.01±0.81	94.62±0.81	5.79±1.26	98.10±0.08	98.10±0.05
	Orthogonal	93.62±0.69	92.80±0.71	7.44±1.07	98.32±0.06	98.32±0.06
	PARw	93.83±0.81	92.98±0.83	7.48±1.37	98.32±0.08	98.33±0.05
FMNIST	Kaiming-Uniform	85.40±1.09	84.32±1.04	7.47±0.84	89.80±0.21	89.79±0.21
	PARw	86.14±0.94	84.76±0.92	6.84±0.87	89.88±0.25	89.90±0.16
	Kaiming-Normal	85.37±1.04	84.41±1.03	7.29±0.86	89.76±0.20	89.76±0.20
	PARw	86.13±1.03	84.78±0.98	6.92±0.91	89.75±0.18	89.76±0.11
	truncated-normal	85.51±1.09	84.48±1.03	7.26±0.81	89.79±0.22	89.80±0.22
	PARw	86.09±0.95	84.79±0.90	6.89±0.85	89.80±0.21	89.82±0.15
	Orthogonal	82.95±1.18	82.51±1.20	10.15±1.18	90.03±0.18	90.05±0.18
	PARw	83.18±1.31	82.71±1.30	9.82±1.08	90.08±0.20	90.08±0.14

Source: By the author.

gradient norm clipping.

- SL-ViT: (189) Vision Transformer (ViT) architectures have recently demonstrated great potential for CV tasks. (64) The SL-ViT is a variant of the original ViT that is more suitable for training from scratch with smaller datasets. The method is trained as in (189) [‡]: using the AdamW optimizer, 100 epochs, cosine learning rate schedule with an initial learning rate of 0.003 (annealed to 0, no restarting), batch size 128, 10 warmup epochs, and weight decay of 0.05. For regularization, we use label smoothing, (190) stochastic depth, (191) CutMix, (192) and Mixup. (193)

- ConvMixer: (194) A more recent architecture that mixes the spatial and channel locations of patch embeddings using only standard convolutions. We considered patch size 2, kernel size 5, hidden dimension 256, and depth 8. The model is trained similarly as in (194) [§] considering 100 epochs with the AdamW optimizer, triangular learning rate scheduling with a maximum learning rate of 0.1, weight decay of 0.005, batch size 512, and gradient norm clipping.

For the training of all deep models (except ResNet18), the following data augmentation is employed in addition to the procedures already described above: random

‡ www.github.com/aanna0701/SPT_LSA_ViT

§ www.github.com/locuslab/convmixer-cifar10

Table 7 – Comparison of the proposed PARw method applied on different weight initializers when using the LeNet architecture, trained on the MNIST, FMNIST, and CIFAR10 datasets.

		training metrics			test acc.	
	weights	ep. 1 acc.	ep. 1 val. acc.	converged	average	median
MNIST	Kaiming-uniform	96.25±1.44	95.86±1.42	15.56±6.55	99.06±0.12	99.06±0.12
	PARw	97.36±0.31	96.84±0.35	10.54±2.25	99.10±0.07	99.10±0.05
	Kaiming-normal	95.93±2.31	95.54±2.26	16.10±6.59	99.01±0.13	99.03±0.13
	PARw	97.37±0.31	96.85±0.32	10.22±2.40	99.09±0.07	99.09±0.05
	truncated-normal	96.18±1.67	95.81±1.65	15.77±8.30	99.05±0.17	99.08±0.17
	PARw	97.34±0.32	96.81±0.33	10.52±2.37	99.09±0.07	99.09±0.04
	Orthogonal	97.08±0.41	96.74±0.44	12.40±2.37	99.19±0.06	99.19±0.06
	PARw	97.13±0.40	96.72±0.42	11.63±2.28	99.19±0.06	99.18±0.03
FMNIST	Kaiming-uniform	84.57±0.92	84.01±0.85	12.34±1.26	90.54±0.23	90.52±0.23
	PARw	85.62±0.85	84.96±0.81	10.91±1.24	90.63±0.22	90.64±0.16
	Kaiming-normal	84.45±0.89	83.87±0.85	12.77±1.74	90.55±0.23	90.52±0.23
	PARw	85.56±0.86	84.82±0.81	10.91±1.32	90.64±0.19	90.61±0.13
	truncated-normal	84.50±1.04	83.95±0.97	12.64±1.64	90.53±0.24	90.54±0.24
	PARw	85.62±0.87	84.95±0.87	11.11±1.22	90.66±0.20	90.66±0.14
	Orthogonal	83.61±1.04	83.23±1.03	13.05±1.06	91.06±0.19	91.06±0.19
	PARw	83.98±1.10	83.58±1.11	12.94±1.34	90.98±0.20	90.97±0.11
CIFAR10	Kaiming-uniform	49.91±1.67	48.88±1.50	11.09±2.04	67.65±0.61	67.69±0.61
	PARw	52.87±0.47	51.29±0.53	9.74±0.67	68.00±0.56	67.97±0.43
	Kaiming-normal	50.17±1.60	49.09±1.34	10.95±1.98	67.59±0.65	67.57±0.65
	PARw	53.30±0.56	51.57±0.58	9.60±0.80	68.03±0.53	68.07±0.37
	truncated-normal	50.10±2.07	49.06±1.85	10.86±1.97	67.68±0.64	67.68±0.64
	PARw	52.83±0.50	51.32±0.62	9.64±0.77	67.90±0.54	67.98±0.40
	Orthogonal	50.65±0.54	49.84±0.67	9.98±0.20	69.08±0.48	69.08±0.48
	PARw	51.39±0.50	50.42±0.53	9.98±0.20	69.24±0.49	69.26±0.34

Source: By the author.

scaling, RandAugment, (195) random erasing, (196) and color jitter. We keep the same configuration of random seeds as previously described: each repetition changes Pytorch's manual seed responsible for random weight sampling, while the training/validation split is kept, and PARw is applied to the weights generated with the same seeds. We performed the statistical tests for the ResNet18 experiments, since we performed 100 repetitions to each weight initializer with this architecture, while 10 repetitions were done for the other models due to computing constraints. This is done since we are using much bigger models and strong data augmentation, making it unfeasible to store all data in GPU memory for faster computation, and batch generation needs to be done in the CPU. In this scenario, the differences in training time on a 32-core CPU with an RTX 2080 ti GPU were around 30 to 60 times compared to the previous experiments with smaller models without data augmentation. Note again that we keep our three data split policy and experimental constraints to achieve a more robust evaluation of the impacts of weights, rather than to perform a search for the best possible performance (e.g., state-of-the-art results on CIFAR10 (197) may require more than 600 training epochs). Nonetheless, since we isolate all other stochastic and random procedures that could interfere with performance, we believe that 10 repetitions give a good estimate of the impact of the weights alone. Note, however, that we do not perform statistical tests in this case since the sample size is too small. In this case, we show the maximum test accuracy of each model among the repetitions performed, to compare the initialization techniques if used to search for the best possible model.

The results for the experiments with the deeper models are shown in Table 8. As base weights, we considered the Kaiming-normal initialization for ResNets, which is the proposed initialization for this specific architecture, and Orthogonal weights for the other models, since it achieved the best results in our previous experiments. For ResNet18, the proposed PARw method increases the training performance, speeds up convergence, and reaches higher test results at the end, all with statistically significant differences. Note that since no data augmentation is used in this case, the lowest validation loss is reached with around seven epochs only (with a small standard deviation). This does not seem to affect the gains after training (in terms of overfitting), where rewiring improved the average performance and also the highest obtained test accuracy. For the other cases, the method also provides similar effects, improving average performance and the maximum obtained test accuracy. It is important to notice that training performance at just 1 epoch is considerably increased in all architectures, as well as validation performance in most cases. However, the convergence time in this scenario is higher when using any given initial weights, highlighting the effects of strong regularization. In other words, although PARw increases the performance at the beginning of training, this does not mean that models will train faster in this case as stochastic data augmentation prevents the models to fall into local minima. On the other hand, PARw still provides performance improvements after training in all cases, implying a better generalization power. Additionally, it decreases performance variance, meaning more stable weights, and increases the peak performance achieved among the trials.

3.2.8 Discussion

Our results corroborate the findings discussed in the previous section and those of previous works on the complex network properties of ANNs, (172,174–176) reinforcing the importance of these properties. Random initialization was a key factor in most of the mentioned works. Our observations on the strength variance of randomly initialized neurons partially explain this phenomenon, corroborating that the performance differences are caused also by the weight organization at initialization, rather than by the training procedure alone. This is also corroborated by correlating our results with the theoretical findings of (160) and the empirical results of (158,159) (on the variance caused by random

Table 8 – Comparison of weight initializations using deeper architectures and more complex training schedules on the CIFAR10 dataset. ResNet18* was trained according to the simpler schedule (SGD, 100 epochs), and 100 repetitions were conducted (along with statistical tests).

		training metrics			test acc.	
model	weights	ep. 1 acc.	ep. 1 val. acc.	converged	average	max.
ResNet18*	Kaiming-normal	63.48±0.64	57.85±0.62	7.48±0.88	72.94±0.53	74.23
	PARw	64.71±0.60	58.61±0.64	7.08±1.00	73.36±0.49	74.36
ResNet50	Kaiming-normal	13.93±3.24	13.92±3.17	141.10±5.05	92.29±0.35	92.74
	PARw	14.06±2.74	13.91±2.61	139.00±5.71	92.37±0.26	92.75
SL-ViT	Orthogonal	22.64±0.61	22.42±0.68	108.00±0.00	89.10±0.29	89.52
	PARw	24.43±0.94	24.35±0.91	108.00±0.00	89.13±0.26	89.57
ConvMixer	Orthogonal	39.18±1.30	38.54±1.35	95.70±1.79	93.69±0.14	93.84
	PARw	39.91±1.26	39.16±1.31	95.50±2.46	**93.77**±0.13	**93.95**

Source: By the author.

seeds). In this sense, we observe a similar behavior as these previous works by varying only the weight initialization seed, suggesting that random weights are a key factor in causing such variance. It is also relevant to mention the findings of (14) which corroborates some effects caused by specific weight initializations. For instance, they state that ANNs that are "strong learners" converges to the neighborhood of their initial configuration. Considering our proposed method, the obtained parameter space after rewiring seems to be one such special case at initialization, directly impacting the final performance of the model.

Empirical results show that the proposed PARw method causes significant effects on ANN performance. More specifically, performance is considerably improved after a single training epoch, corroborating a better initial parameter space, and the convergence time is decreased in most cases. More surprisingly, it also improves the test performance of final models, leading to a better generalization power. Another effect noticed in most experiments is the decrease in performance variance, considering both the standard deviation and the median absolute deviation. However, it is important to notice that in some specific cases, it may decrease the final performance of the models, as results with a shallow architecture and simple training schedule show. We believe that, in this case, the performance gains at the beginning of training cause models to overfit too fast (see convergence time), thus reaching poor local minima that a simple training schedule is not able to overcome. Therefore, this effect may be also related to the training schedule and not only to the initial weights, since better training seems to overcome this limitation. For instance, when more complex training schedules with strong regularization are employed, the models achieve both higher training and test performance in general. The effects of PARw in this latter case, which is the common scenario in state-of-the-art research with large-scale vision models, highlight the importance of the initial random weights and their organization.

Regarding the combination of PARw with other deep learning techniques, our results show that it works well in conjunction with batch normalization (198) and residual connections, (61) two of the most important modules of state-of-the-art ANNs. It also works well with more complex training schedules, and with more recent architectures, such as ViT and ConvMixer. Regarding other kinds of weight operations, such as weight normalization (88) and other methods that make the learned function invariant to scaling of the weights, the proposed PARw goes in a different direction as we do not change the scale and shape of the weight distribution. In other words, as our approach only reorganizes weights at initialization, it can be used together with any of these methods. Analyzing the impacts of combining our approach with all such methods is beyond the scope of this work, as there is a large number of possibilities. Nevertheless, we believe that the selected methods, architectures, datasets, and statistical analysis employed in this work give a good estimate of the behavior of PARw in practice. We also avoided doing analyses of individual models and neurons as this would be highly influenced by the randomness of the initializers. For instance, we believe that these specific analyses, which are common in the ANN literature, are one of the potential flaws when dealing with several degrees of freedom, causing some findings to be hard to replicate and generalize. Therefore, our analyses focused on distributions and patterns observed collectively on a wide range of random networks, under different scenarios and training dynamics.

3.2.9 Conclusion

A new method to improve weight initialization is proposed for general ANN architectures. To the best of our knowledge, this is the first work to directly explore Network Science concepts in the context of random weight initialization of deep ANNs. First, we showed a measurable correlation with statistical confidence between the neuronal strength at random initialization and the final performance of a LeNet model. We then explored neuronal strength properties to improve initialization, showing that a higher strength variance decreases performance, while a lower variance increases it. These concepts were considered to develop a new heuristic, called PARw, which works in conjunction with any weight initialization method by reorganizing neuronal connections in a way that their strength distribution is smoothed. In other words, we modify the long-tailed strength distribution produced by literature methods, and the experiments show that this simple procedure improves training and test performance while reducing the variance of the results. The method works locally at each ANN layer, without external knowledge from the architecture or domain in which it is being employed, and has a relatively low computational budget since it is applied just once at initialization.

Our empirical evaluation corroborates the importance of the random initialization of ANNs, even when deep models and complex training schedules are employed. The proposed PARw method provides a simple and cheap way to optimize random weights, which can be easily coupled with state-of-the-art deep learning pipelines. Although we only consider computer vision architectures in our experiments, the proposed method is based on generic weight matrices. In this sense, it can be simply coupled with existing initialization policies for different architectures and in different domains.

4 NETWORK-BASED TEXTURE ANALYSIS

Computer vision is a broad research field with many specific applications such as the recognition of objects or scenes, with tasks such as classification, regression, segmentation, etc. The deep learning literature on CV focuses on building and training general vision models, *e.g.* by training them on large datasets with a variety of images from the internet, which imply extreme computational loads. These models can then be ported to several downstream tasks through transfer learning. The proposed methods from the previous chapter focused on this paradigm, exploring how to improve deep ANNs for general visual recognition. However, many CV tasks require approaching only specific properties in images, which are enough for many real-world uses. One such task is texture analysis, which we discussed in greater depth in Chapter 2, Section 2.3. Although large general vision models perform well on various tasks, they may fail to compete at the SOTA level on texture analysis tasks compared to more efficient hand-engineered features. (19,154) In this context, and considering the demands of texture-based applications, we tackle texture analysis by exploring this duality between feature engineering and deep learning-based methods in a variety of scenarios. More specifically, we employ a similar framework as before: the concepts of network science and random networks are explored and combined, which we named a network-based approach to texture analysis. In this chapter, we describe four new texture analysis methods using this approach and a variety of experiments to understand and evaluate their properties.

4.1 Spatio-Spectral Networks (SSN)

Our first proposal for network-based texture analysis goes in the more classical approach, *i.e.*, developing mathematical feature extraction techniques. It focuses on a well-established paradigm: modeling images as graphs. More specifically, we follow a line of research where a complex network structure is obtained by representing pixel similarity (*e.g.*, as in (15, 16, 19, 133)). This is a classic line of research in the Scientific Computing Group where many texture recognition methods have been developed in the last decade. (16, 19, 131, 154, 199–207) We discussed this approach in more detail in Chapter 2, Section 2.3.3.1. Here, we describe a new method focusing on one of the latest developments of the approach, (19) where a multilayer network is built for trichromatic (RGB) images. In this sense, we propose various improvements over the previous CN-based methods for texture analysis. This work was also developed in collaboration with researcher Lucas Correia Ribas, resulting in a publication (204) in the Information Sciences journal from Elsevier.

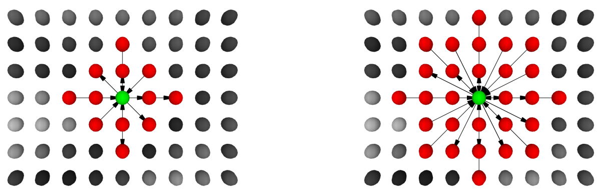
The proposed Spatial-Spectral Network (SSN) (204) is an innovative image analysis approach that leverages network science to capture and represent complex spatial-spectral relationships within images. This method models images as directed networks, incorporating multiscale and radially symmetric neighborhoods to generate more accurate and meaningful characterizations. Utilizing traditional centrality measures such as input degree, input strength, and output strength, the SSN method efficiently extracts topological features that highlight various texture patterns within the images. Subsequently, these features are summarized through statistical measures, resulting in a compact image descriptor that captures the network's structure. The SSN method accommodates both grayscale and RGB images, and its computational cost is considerably lower than DCNNs. By employing network science to analyze images, the SSN method offers a powerful and versatile tool for understanding and characterizing complex image data.

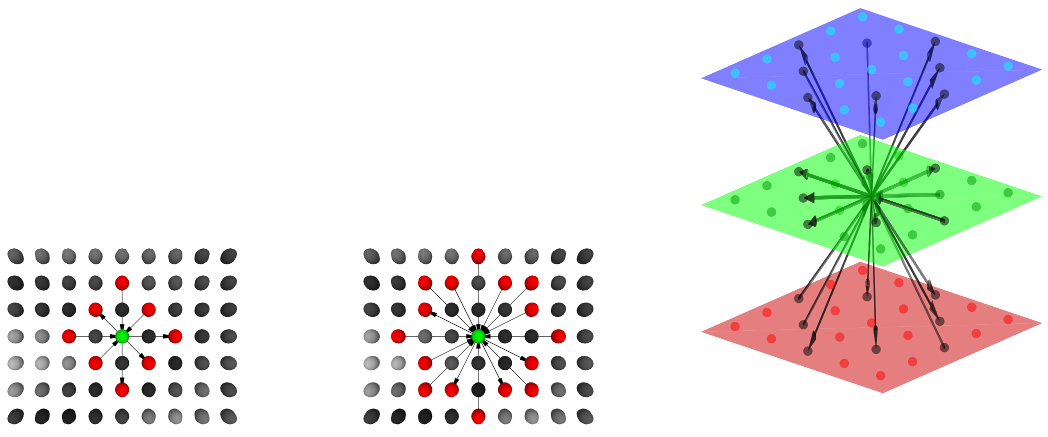
SSNs model the spatial relationship between intra-channel and inter-channel pixels utilizing a directed CN. It takes a color image as its input, containing z channels, and subsequently generates a collection of image descriptors for color-texture characterization. Let us consider a color image I with dimensions whz, signifying that it comprises wh pixels, each with z color channels, with values ranging from $[0, L]$. A network N can be defined as a tuple $N = (V_N, E_N)$, where V_N denotes the network vertices and E_N represents the edges. The vertex set is established as described in (19) where each image pixel i is assigned to a vertex for every color channel, resulting in $V_N = \{v_1, ..., v_{wh*z}\}$. This process yields a multilayer network in which each layer corresponds to a specific color channel of the image. Each vertex v_i possesses a set of node features, including a coordinate pair (x, y) to indicate the position of the represented pixel and a value $p(v_i) \in L$ representing the pixel intensity value within its respective color channel.

Previous works generally used a series of radii $R = \{r_1, ..., r_n\}$ to constrain the vertex neighborhood size, with a set of sliding windows of radius r_k defining a distance limit for vertex connections, ultimately generating a collection of networks N^{r_k}. A vertex v_i is connected to pixel v_j if it resides within the neighborhood $G_{v_i}^{r_k} = \{v_j \in V_N | d(v_i, v_j) \leq r_k\}$, where $d(v_i, v_j)$ denotes the 2-D Euclidean distance between the vertices. In line with (19) this neighborhood encompasses vertices across all color channels since it solely considers the spatial position of the pixels for distance computation. This approach enables the observation of dynamic evolution as r_k increases by examining each network $N^{r_1}, ..., N^{r_n}$ and has proven effective for color texture characterization.

However, it is important to note that $\{G_{v_i}^{r_k}, ..., G_{v_i}^{r_{k-1}}\} \subsetneq G_{v_i}^{r_k}$ for an increasing set of radii $R = \{1, 2, ..., r_n\}$, and $\forall (r_k, r_l)$, $G_{v_i}^{r_k} \neq G_{v_i}^{r_l}$. This implies that each neighborhood contains all smaller neighborhoods, resulting in redundancy between networks $N^{r_1}, ..., N^{r_n}$. In this context, we introduce a radially symmetric modeling criterion by redefining the neighborhood as $G_{v_i}^{r_k} = \{v_j \in V_N | r_{k-1} < d(v_i, v_j) \leq r_k\}$, which only considers vertices within a distance range between the considered radius and the previous radius, thus $\{G_{v_i}^{r_1}, ..., G_{v_i}^{r_{k-1}}\} \nsubseteq G_{v_i}^{r_k}$. It is crucial to note that we include vertices with a distance of 0 in the neighborhood $G_{v_i}^{1}$, which occurs when a pixel connects to itself across different color channels.

In essence, our proposed neighboring approach involves connecting the central pixel to pixels situated at the boundary of a circle with a radius of r_i and then repeating this process to create independent networks for a set of n radii. Since each network covers a specific region based on r_i ($G_{v_j}^{r_i}$), the combination of all networks eventually encompasses all surrounding pixels within radius r_n. Figure 19 illustrates this concept, displaying a CN modeled for a single image channel and highlighting the distinction between the conventional (a) and radially symmetric (b-c) neighboring techniques (we will discuss the connection direction later on). In our proposal, the radially symmetric neighborhood is then extended for all image channels (c). Our experiments, along with the comparison with other CN-based methods, corroborate that this is a better modeling approach, providing more relevant features which lead to better results than the previous techniques.


(a) Standard CN neighboring ($r = 2$ and $r = 3$).


(b) Radially symmetric CN neighboring ($r = 2$ and $r = 3$).

(c) Proposed radially symmetric neighboring in a multilayer CN ($r = 2$).

Figure 19 – The two approaches for creating CN connections in a single-channel image (a, b), and an extension of the symmetric neighborhood for a 3-channel image (c). The connections point to the pixel with the highest intensity.

Source: SCABINI; RIBAS; BRUNO. (204)

The definitions presented thus far describe the pixel neighborhood in which vertices are interconnected. Consequently, the next step consists of establishing these connections for a directed and weighted network. The weight of the connection, denoted as $a(v_i, v_j)$, between pairs of vertices v_i and v_j incorporates their node features, and is determined by their absolute intensity difference proportionally to their Euclidean distance within the image:

$$a(v_i, v_j) = \frac{(|p(v_i) - p(v_j)| + 1)(d + 1) - 1}{(L + 1)(r + 1) - 1}. \tag{4.1}$$

This equation represents a modification of those proposed in (19,133) to circumvent the potential cancellation of one side of the multiplication when the intensity difference or distance is equal to 0 (identical pixels in distinct channels) and to generate uniform values within the range of $[0, 1]$. It is crucial to recognize that, in accordance with the proposed neighboring structure, the formation of connections among all channels implies that the network effectively performs spatial opponent color processing. As such, the SSN incorporates the bio-inspired aspect of Opponent-Process Theory, (208) taking into account the opposing color pairs red versus green, red versus blue, and green versus blue rather than red versus green, and blue versus yellow. If an alternative color space is employed, the corresponding opposing color pairs will be considered in accordance with the specific color space.

As previously noted, the majority of CN-based methodologies employ weighted undirected networks and need thresholding techniques to extract pertinent topological information. This requirement arises due to the fact that a network connecting all pixels within a fixed radius exhibits constant topological measures, including degree, strength, clustering, and others. However, thresholding introduces additional parameters requiring tuning, which has proven to be a drawback in prior research where authors explored computationally expensive techniques for optimal threshold selection. (19,133) To address this issue, we propose a directed approach that eliminates the need for thresholding, reducing the method's parameters solely to the set of radii $R = \{r_1, ..., r_n\}$. This is accomplished by associating the direction of the connection with the gradient direction, specifically towards the pixel exhibiting higher intensity. We developed this concept initially for grayscale texture characterization. (154) Here, we generalize it to multilayer networks of color images, in conjunction with our novel connection weight equation (Equation 4.1).

Given a network $N^{r_k} = (V_N, E_N)$, its set of edges is defined as:

$$E_N = \{a(v_i, v_j) \in E \mid r_{k-1} < d(v_i, v_j) \leqslant r_k \wedge p(v_i) < p(v_j)\}, \tag{4.2}$$

where when $p(v_i) = p(v_j)$, the edge is bidirectional, *i.e.*, $a(v_i, v_j) = a(v_j, v_i)$. This procedure generates a network that encompasses relevant topological information manifested in its directional connection patterns. These patterns can be quantified through directed measures such as the input and output degree and strength of vertices, effectively eliminating the need for connection pruning.

The spatio-spectral characteristics of our proposed network are derived from the connections within and between channels, while the edge directions incorporate image gradient information. The neighborhood constrains a local spatial analysis, while the interactions between channels or bands capture the spectral relationships of pixels. By combining both approaches, the network can effectively model the intricate color-texture properties of the image, which is the rationale behind the term "spatio-spectral". To emphasize this information, we divide the original network $N^{r_k} = (V_N, E_N)$ into two additional networks as previously proposed. (19) The first network, $W^{r_k} = (V_W, E_N)$, has edges that are a subset of N containing within-channel connections. Thus, $E_W = \{a(v_i, v_j) \mid p(v_i, z) = p(v_j, z)\}$, where $p(v_i, z)$ returns the channel or layer of v_i. The second network, $B^{r_k} = (V_B, E_B)$, represents between-channel connections, where $E_B = \{a(v_i, v_j) \mid p(v_i, z) \neq p(v_j, z)\}$. The vertex sets of each network (N, W, and B) are identical ($V_N = V_W = V_B$), as only their edges are divided. Figure 20 illustrates the structure of the three networks for a radius $r = 1$. By quantifying their topology, it is possible to extract rich color-texture information for image characterization, which will be discussed in the subsequent sections.

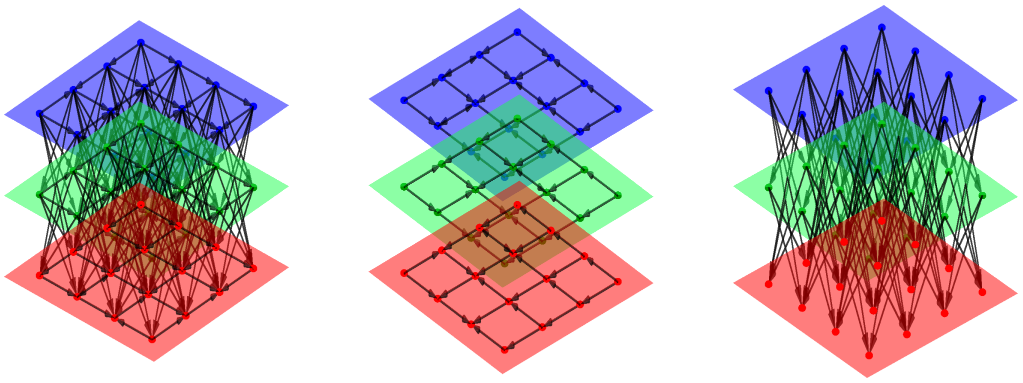

(a) $N^{r=1}$. (b) $W^{r=1}$. (c) $B^{r=1}$.

Figure 20 – The multilayer structures of the spatio-spectral networks are illustrated for $r = 1$.

Source: SCABINI; RIBAS; BRUNO. (204)

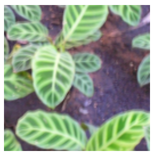
4.1.1 SSN characterization

In order to comprehensively characterize the directed SSN we suggest employing simple/efficient (yet well-established) centrality measures, specifically the input and output degree and strength k_{in}, k_{out}, s_{in}, and s_{out}, respectively (refer to Section 2.1.2 and Eq. (2.1)). The efficiency of these measures lies in the fact that they can be computed during the network modeling process itself, as the calculation necessitates only the examination of the vertex neighbors. Consequently, there is no substantial increase in the computational cost for the network characterization. In comparison, this proposal demonstrates a reduced cost in contrast to the previous method (19) which relies on the vertex clustering coefficient and necessitates the storage of the network in order to examine its clustering structure.

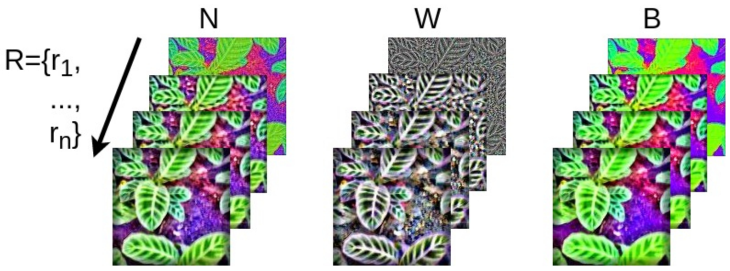
It is important to note that for all $v_i \in V$, $k_{in}(v_i) + k_{out}(v_i) = k^r_{max}$, where k^r_{max} denotes the maximum possible number of connections when using the modeling radius r. This implies that the input and output degrees are linear transformations of each other in relation to the maximum degree of the network. Consequently, we opt to utilize only the input degree. However, the same assumption cannot be made for vertex strengths $s_{in}(v_i)$ and $s_{out}(v_i)$, as it also depends on the distribution of image pixel values within the connection neighborhood. As a result, we incorporate both input and output strengths. In this context, the network characterization is conducted by concatenating these three measures: input degree (k_{in}), input strength (s_{in}), and output strength (s_{out}). Each topological measure accentuates distinct texture patterns within the modeled image, as demonstrated in Figure 21 for each type of complex network. We take into consideration a collection of feature maps (combination of measures for all pixels) encompassing all topological information acquired through the characterization of networks N, W, and B, utilizing the aforementioned three centrality measures in a multiscale manner with a set of increasing radii $R = 1, ..., r_n$, given r_n. In the qualitative analysis depicted in Figure 21, it becomes evident that the feature maps, *i.e.*, the CN metrics, emphasize several aspects of the texture.

The feature maps derived from the characterization of SSNs necessitate summarization to yield a single, compact image descriptor. One feasible approach to achieve this is by obtaining the probability distributions from the three centrality measures, a common method employed for network characterization. We propose computing the distribution for each layer of N, W, and B independently, thereby extracting z distributions from each network. It is reasonable to deduce that conducting separate analyses for each layer would yield more specific information pertaining to patterns present within individual network layers. This technique demonstrates improved results compared to utilizing the entire network, as proposed before. (19)

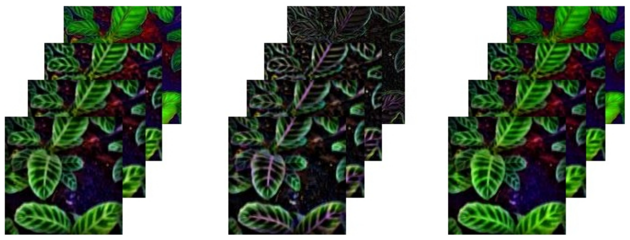
To precisely compute the probability distribution function of each network layer, we set the number of bins for counting the input degree, k_{in}, to the maximum possible degree, k^r_{max}. For the occurrence counting of the strength measure, the bin number is the maximum possible degree multiplied by 10. We define the probability distribution function of each measure by $P^{k_{in}}$, $P^{s_{in}}$, and $P^{s_{out}}$. Figure 22 illustrates each distribution of an SSN $N^{r=4}$, in accordance with the proposed layer-wise analysis, wherein a discernible



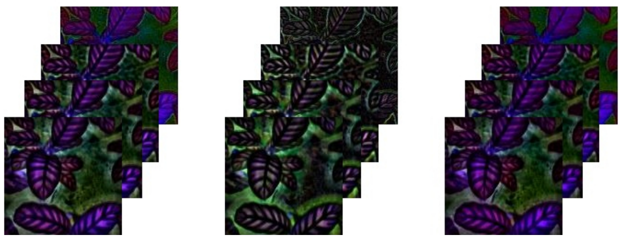
(a) Input texture to be modeled using the proposed SSN with $R = \{r_1, ..., r_n\}$.



(b) k_{in}.



(c) s_{in}.



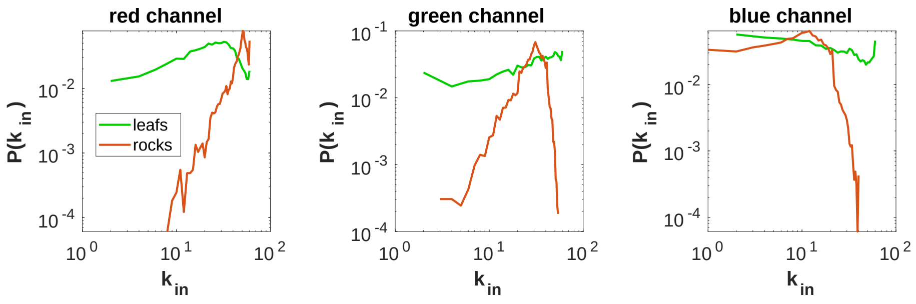
(d) s_{out}.

Figure 21 – Visualization of the centrality properties of SNNs, or its feature maps as we refer to. We show the three directed centrality measures (b-d) obtained from SSNs modeled for the input texture (a) with a set of radii R. The final images are obtained by converting vertex measures in each network layer into an intensity value (normalized to 8bit, *i.e.*, between $[0, 255]$).

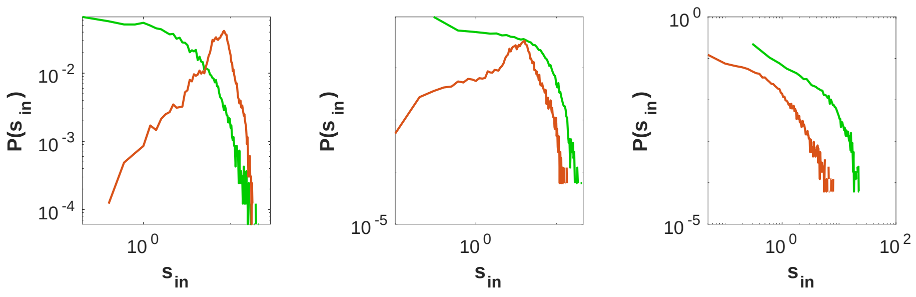
Source: SCABINI; RIBAS; BRUNO. (204)

distinction between the two different input textures can be observed. It is also possible to observe power-law behaviors in various cases, corroborating the complex structure of SSNs correlating to scale-free networks. However, the topological organization of the network varies significantly based on the input texture under examination, which is a desired property for the sake of feature extraction.

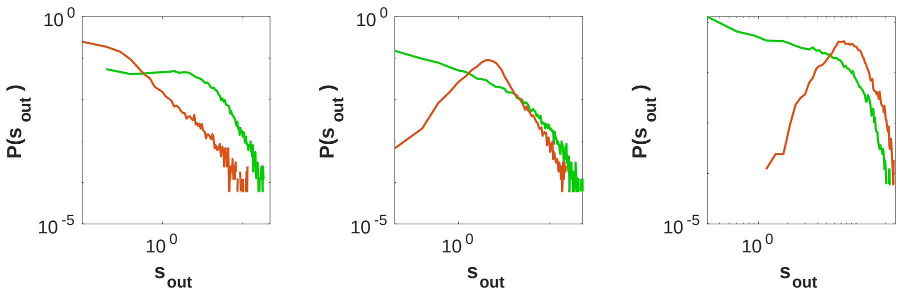
(a) Input textures (leafs and rocks).



(b) Distribution of k_{in} for each layer of $N^{r=4}$.



(c) Distribution of s_{in} for each layer of $N^{r=4}$.



(d) Distribution of s_{out} for each layer of $N^{r=4}$.

Figure 22 – Distribution (P) of centrality measures k_{in}, s_{in} and s_{out}, at each layer of an SSN $N^{r=4}$ (b-d), according to the two input textures (a).

Source: SCABINI; RIBAS; BRUNO. (204)

For the sake of comparison, the multilayer network from the previous work (19) appears to exhibit similar topologies for different image inputs, with variations in the small-world effect and the occurrence of power-law-like degree distributions. Conversely, the directed SSNs in our approach present heterogeneous configurations, wherein the network structure differs substantially between various textures, thereby providing superior topological measures for characterization. This divergence from the previous work can be primarily attributed to the incorporation of connection direction, radially symmetric neighboring, and layer-wise network characterization.

4.1.2 Feature vector

Although the distributions $P^{k_{in}}$, $P^{s_{in}}$, and $P^{s_{out}}$ effectively encapsulate the network topology, their direct application as image descriptors remains impractical. This is because their size increases proportionally with the parameter r. For instance, in the case of a single network $N^{r=4}$, the size would be $(19 + 190 + 190)3 = 1197$, or 3591 for the three networks (including W and B). Consequently, we propose the use of statistical measures to further compact the SSN descriptors. For that, we adopt the four measures presented in the previous work (19) — mean μ, standard deviation σ, energy e, and entropy ϵ — along with the third and fourth statistical moments, namely Skewness and Kurtosis, respectively. The Skewness is defined as:

$$\lambda_f = \frac{\frac{1}{|P^f|}\sum_i (P^f(i) - \mu_f)^3}{\left(\sqrt{\frac{1}{|P^f|}\sum_i (P^f(i) - \mu_f)^2}\right)^3}, \tag{4.3}$$

while the Kurtosis is given by:

$$\kappa_f = \frac{\frac{1}{|P^f|}\sum_i (P^f(i) - \mu_F)^4}{\left(\sqrt{\frac{1}{|P^f|}\sum_i (P^f(i) - \mu_f)^2}\right)^2}. \tag{4.4}$$

By combining the six statistics for each of the three topological measures of every layer, we can construct a comprehensive network descriptor. Given a network $N^r = (V_N, E_N)$ and its set of vertices $V_N = \{V_1, ..., V_z\}$ split for each of the z layers, a vector $\varpi(V_i)_f = [\mu_f, \sigma_f, e_f, \epsilon_f, \lambda_f, \kappa_f]$ represents the six statistics for a specific layer i and measure f. The concatenation of descriptors $\varpi(V_i)_f$ from each layer, for each measure, constitutes the network descriptor:

$$\varpi(N^r) = [\varpi(V_1)_{k_{in}}, \varpi(V_1)_{s_{in}}, \varpi(V_1)_{s_{out}}, ..., \varpi(V_z)_{k_{in}}, \varpi(V_z)_{s_{in}}, \varpi(V_z)_{s_{out}}]. \tag{4.5}$$

Finally, the multiscale analysis is achieved by employing a set of increasing radii, denoted by $R = \{1, 2, ..., r_n\}$, preserving the proportion of each symmetric neighborhood. The final descriptor, which captures the dynamic growth of each network (from smaller to bigger radius), is acquired through concatenation. Consequently, we have $\varphi_N^{r_n} = [\varpi(N^1), ..., \varpi(N^{r_n})]$, $\varphi_W^{r_n} = [\varpi(W^1), ..., \varpi(W^{r_n})]$, and $\varphi_B^{r_n} = [\varpi(B^1), ..., \varpi(B^{r_n})]$. The concatenation of features from networks N, W, and B culminates in a comprehensive representation that encompasses the entire spatio-spectral information:

$$\varphi_{WB}^{r_n} = [\varphi_W^{r_n}, \varphi_B^{r_n}], \tag{4.6}$$

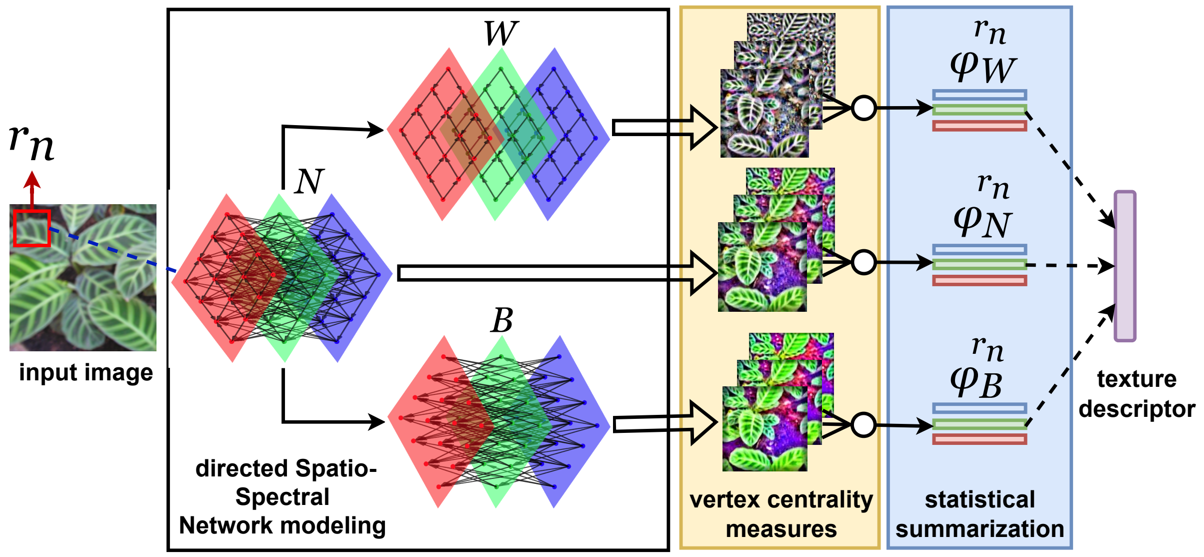


Figure 23 – Illustrating the proposed SSN approach given an input image and the parameter r_n. The first step is the modeling of networks N, W, and B with radially symmetric neighborhoods. The networks are characterized by centrality measures (input degree and input/output strength) from each layer separately. These measures from each network are then statistically summarized and combined to compose an image descriptor.

Source: SCABINI; RIBAS; BRUNO. (204)

$$\varphi^{r_n} = [\varphi_N^{r_n}, \varphi_W^{r_n}, \varphi_B^{r_n}]. \tag{4.7}$$

Figure 23 delineates each step of the entire process of the proposed method, from the image input to the resulting SSN feature vector. It is worth noting that the proposed model can also be applied to grayscale images by setting $z = 1$ and utilizing one intensity matrix. In this scenario, only spatial connections are established, resulting in the edges of network B being empty ($E_B = \emptyset$) and $E_W = E_N$ ($W = N$). In such cases, users should employ either network N or W.

4.1.3 Computational complexity

The computational cost of the proposed SSN is directly proportional to both the image size and the largest radius utilized (r_n), which defines the window size for connection creation. The final radius, r_n, implies that all pixels with a distance $d \leq r_n$ must be visited to construct each neighborhood $G_{v_i}^{r_1}, ..., G_{v_i}^{r_n}$, in accordance with the radially symmetric neighborhood previously defined. Given an image I with dimensions wh and z channels ($|I| = whz$), let $g_{r_n} = (\frac{(2r_n+1)^2-1}{2}z) + z$ represent the number of pixels encompassed within all neighborhoods for a set of radii $R = \{1, ..., r_n\}$. The cost of modeling the image as an

SSN is thus:

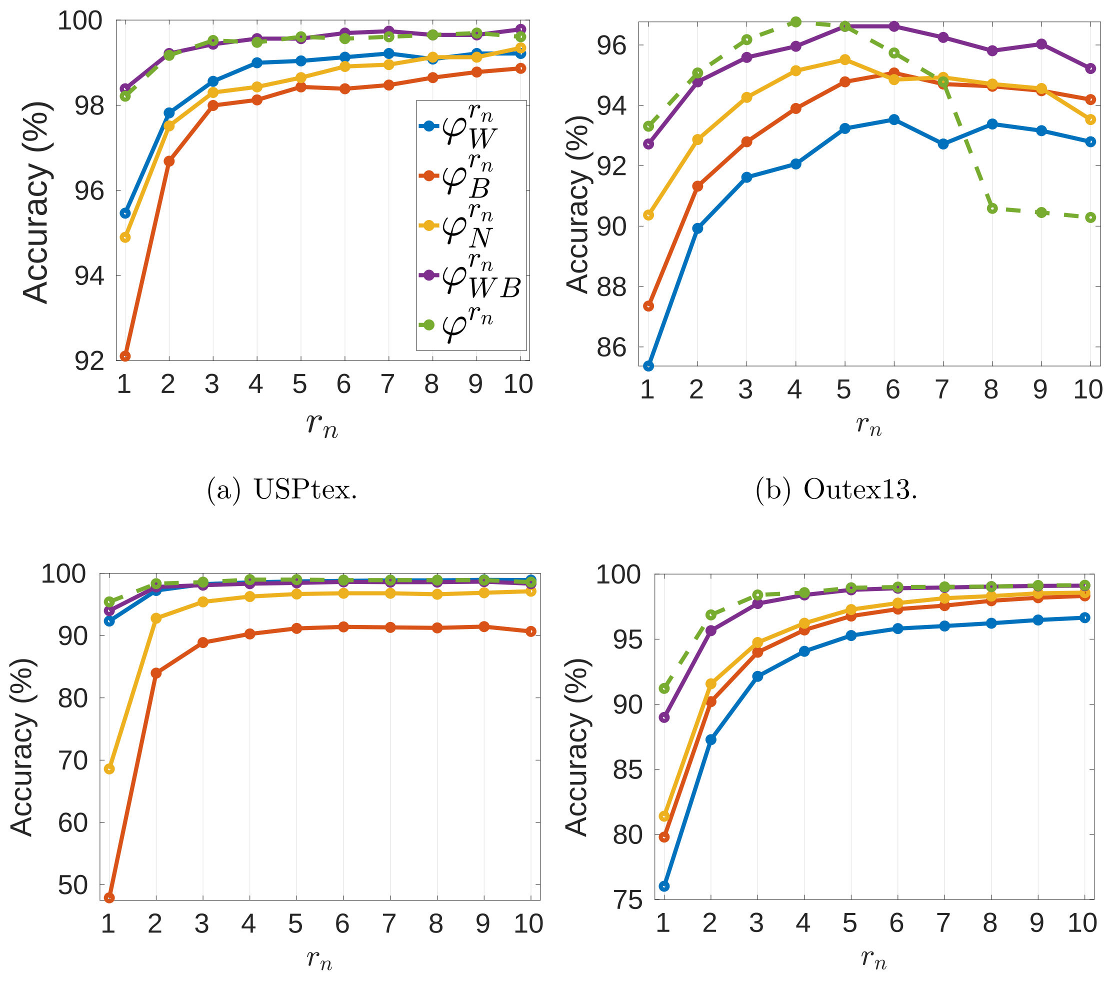
$$\mathcal{O}(|I|g_{r_n}). \tag{4.8}$$

This takes into account the concurrent modeling of networks N, W, and B, as it is only necessary to verify the pixel channel to determine the type of connection. As previously noted, the overall implementation of the proposed method bears resemblance to the prior work (19) (which has a cost of $\mathcal{O}(2|I|g_{r_n})$) but omits the cost associated with computing the clustering coefficient for each vertex. Consequently, the network measures are computed during the modeling step with no additional cost, since there is no need to store the network. The characterization cost is then linked to the computation of each measure's distribution and its corresponding statistics, which is significantly smaller than the modeling cost. As a result, the asymptotic limit of the proposed method aligns with that defined in Eq. (4.8). As the time-consuming experiment detailed in (19) demonstrates, this feature extraction approach is faster than using recent DCNNs.

4.1.4 Analysis of the proposed SSNs

In this section, we present a comprehensive set of experiments conducted to assess the performance of the proposed SSN method under various conditions. To this end, we implement a supervised classification framework utilizing the Linear Discriminant Analysis (LDA) classifier (209, 210) applied over the proposed descriptors. This classifier is selected considering its wide use in previous similar works. The performance of our proposed method is quantified through the accuracy achieved using leave-one-out cross-validation, a robust evaluation technique involving n iterations (where n corresponds to the number of samples in the dataset). In each iteration, one sample is held out for testing while the remaining samples are used for training the classifier. This process is repeated, ensuring that each sample is used as a test instance once. The overall accuracy is calculated as the percentage of samples that are correctly classified during this procedure. This framework is employed in four texture benchmarks: USPtex, (111) Outex13, (106) MBT, (115) and CUReT. (105) For a more complete description of the experimental setups employed throughout this thesis, we refer the reader to Section A.1 for understanding different classifiers and validation procedures, and to Section B.1 for all details concerning the texture benchmarks.

We conduct a thorough analysis of the proposed SSN method, focusing on the influence of the parameter r_n and the effects of employing different feature vectors derived from the combination of networks W^{r_n}, B^{r_n}, and N^{r_n}. Figure 24 illustrates the variation in accuracy rates obtained with each feature vector $\varphi_W^{r_n}$, $\varphi_B^{r_n}$, $\varphi_N^{r_n}$, $\varphi_{WB}^{r_n}$, and φ^{r_n} as r_n increases. We observe that the performance improves up to a certain point before stabilizing, except for Outex13, where it starts to decline after $r_n = 6$. This suggests that smaller r_n values are inadequate for capturing comprehensive multiscale texture characterizations, and excessively high values on Outex13 may lead to feature redundancy due to the increasing number of descriptors.



(a) USPtex.

(b) Outex13.

(c) MBT.

(d) CUReT.

Figure 24 – Performance of SSN descriptors obtained from each network N, W, and B (separated or combined), by varying the radius parameter r^n on different texture datasets.

Source: SCABINI; RIBAS; BRUNO. (204)

Performance peaks are achieved with $r_n = 10$ on USPtex and CUReT datasets but with a marginal performance difference compared to lower r_n values. Consequently, a more stable region is observed around $3 \leq r_n \leq 6$ when considering all datasets. Concerning the feature vectors, the performance of each network individually fluctuates across datasets: $\varphi_W^{r_n}$ demonstrates superior performance on USPtex and MBT, whereas $\varphi_N^{r_n}$ prevails on Outex13 and CUReT. Meanwhile, the combined vectors $\varphi_{WB}^{r_n}$ and φ^{r_n} consistently yield the highest results in most cases, as anticipated, since they incorporate measures from both within and between-channel analyses. Consequently, we recommend utilizing these combinatorial feature vectors to achieve optimal results in various scenarios.

To complement the previous analysis, Table 9 shows the average accuracy (across the four datasets) of the concatenated feature vectors $\varphi_{WB}^{r_n}$ and φ^{r_n} as a function of increasing r_n, along with the corresponding number of descriptors. As previously mentioned, the interval $3 \leq r_n \leq 6$ exhibits a smaller standard deviation, indicating a more stable performance region. Furthermore, the highest average accuracy is achieved with $r_n = 6$ for $\varphi_{WB}^{r_n}$, utilizing 648 descriptors, and with $r_n = 4$ and $r_n = 5$ for φ^{r_n}, employing 648 or 810 descriptors, respectively. In this context, the results suggest that the two alternatives, φ_{WB}^{6} or φ^{4}, exhibit comparable average performance to φ^{5}, but with a reduced number of descriptors. To ensure more robust performance across various scenarios, we recommend employing φ_{WB}^{6} (SSN$_a$), as it incorporates a larger neighborhood, thereby providing a more comprehensive multiscale analysis. Consider φ_{WB}^{10} (SSN$_b$) for an even bigger scale, which can be beneficial in some cases.

Table 9 – Mean classification accuracy (LDA) over four texture recognition tasks (standard deviation in brackets) using different SSN feature vectors. The highest results are highlighted in bold type.

	$\varphi_{WB}^{r_n}$		φ^{r_n}	
r_n	descriptors	mean acc.	descriptors	mean acc.
1	108	93.5±3.4	162	94.5±2.6
2	216	96.9±1.7	324	97.4±1.6
3	324	97.7±1.4	486	98.2±1.2
4	432	98.1±1.3	648	**98.5**±1.0
5	540	98.4±1.1	810	**98.5**±1.1
6	648	**98.5**±1.1	972	98.3±1.5
7	756	98.4±1.3	1134	98.1±1.9
8	864	98.3±1.5	1296	97.0±3.7
9	972	98.4±1.4	1458	97.4±3.2
10	1080	98.1±1.7	1620	97.4±3.0

Source: SCABINI; RIBAS; BRUNO. (204)

4.1.5 Conclusion

We presented a novel image modeling and characterization method for texture analysis using directed and multilayer networks, among other improvements over previous methods, named SSN. A texture descriptor is obtained by quantifying the centrality of SSNs, which can then be used to train an ML model for a given task. We thoroughly evaluated the properties and the performance of SSNs in different texture recognition tasks, reaching a final configuration based on our findings. In Chapter 5, we compare the SSN model to a variety of methods from the literature and also in other public benchmarks.

4.2 SSR: SSNs characterization with RNNs

In the previous section, we presented the SSN image modeling technique coupled with a topological characterization through common network measurements (vertex centrality). Since centrality measures are computed for each vertex (pixel), their statistical summarization is needed in order to obtain a compact image descriptor. This summarization is done empirically, and some statistics may perform better in some specific scenarios, then the network characterization becomes a feature engineering task. In this sense, we propose another model where SSN features are learned automatically using artificial neural networks, which can adapt to SSNs from different types of images. More specifically, we consider Randomized Neural Networks (RNNs), (90–93) also referred to as Extreme Learning Machines (ELM), which are efficient to train. This work was developed in collaboration with researchers Lucas Correia Ribas, Jarbas Joaci de Mesquita Sá Junior, Kallil Zielinski, and Rayner Condori. The research resulted in one publication (154) in the Pattern Recognition journal, from Elsevier, one publication (207) in the 11th International Conference on Image Processing Theory, Tools and Applications (IPTA, 2022), presented by the author in Salzburg, Austria, and two pre-prints (211,212) currently under review. Here, we focus on the latter work (212) which extends the concepts to generalize to color images, specifically considering SSNs. We call the proposed method Spatio-Spectral Representation (SSR), which we describe in the following.

SSRs are obtained by training individual RNNs to each image over its SSN vertex centrality measures (k_{in}, s_{in}, and s_{out}). Firstly, we formulate the problem as a neural network task by reorganizing local vertex measures into feature matrices (the RNN structure and training are discussed later). This is achieved by considering the same multiscale modeling approach as described before, *i.e.*, using a set of radii $R = \{r_1, ..., r_n\}$. Consider first the SSN $N^{r_n} = (V_N, E_N)$, with $|V| = whz$ vertices representing the pixels in all z channels. For each vertex $v_i \in V_N$, a local descriptor $\varpi(v_i) = [f_{r_1}(v_i), ..., f_{r_n}(v_i)]$ is obtained by recording the evolution of a centrality measure f as the radius r, *i.e.*, the SSN neighborhood range, increases. By repeating this procedure for all whz vertices, a matrix $X_f(N^{r_n}) \in \mathbb{R}^{n \times whz}$ is obtained by the concatenation of $\varpi(v_i) \forall v_i \in V_N$. This matrix is used as a set of inputs to an RNN where each column is a sample (a vertex), and each row is a vertex feature (centrality measures at various radii).

The training of RNNs is supervised, so a label is needed for each vertex. In this sense, we consider the original pixel intensity in the image, so a vector of labels is obtained as $Y = [p(v_i)]$, for all $v_i \in V$, with size whz. It is important to stress that the set of vertex labels Y is the same regardless of the centrality measure f considered to build X_f. This procedure is illustrated in Figure 4.2 and can be applied for various cases: by using any centrality measure f and for each of the SSNs N, W, or B. In this sense, given an input image modeled as multiscale SSNs with a maximum radius r_n, we obtain a total of nine matrices following these combinations, respectively: $X_{k_{in}}(N^{r_n})$, $X_{k_{in}}(W^{r_n})$, $X_{k_{in}}(B^{r_n})$, $X_{s_{in}}(N^{r_n})$, $X_{s_{in}}(W^{r_n})$, $X_{s_{in}}(B^{r_n})$, $X_{s_{out}}(N^{r_n})$, $X_{s_{out}}(W^{r_n})$, $X_{s_{out}}(B^{r_n})$.

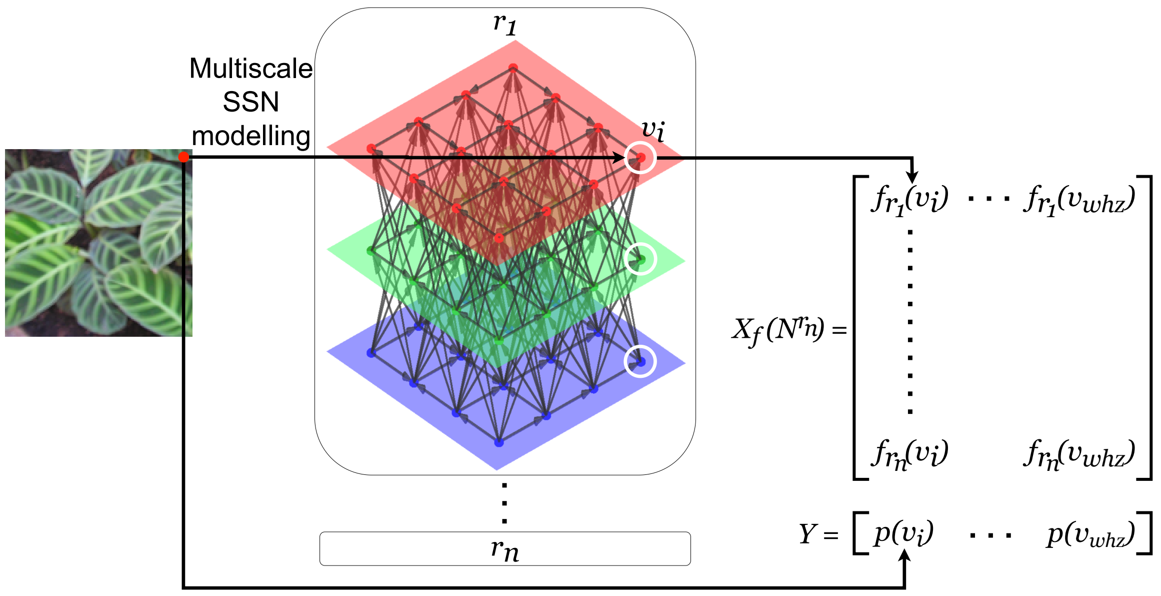

Multiscale SSN modelling
r_1
v_i
r_n
$X_f(N^{r_n}) =$
$f_{r_1}(v_i)$ · · · $f_{r_1}(v_{whz})$
$f_{r_n}(v_i)$ $f_{r_n}(v_{whz})$
$Y = [p(v_i)$ · · · $p(v_{whz})]$

Figure 25 – Illustration of the creation of input feature matrices $X_f(N^{r_n})$ and vertex labels Y used to train RNNs to obtain SSRs. The input image is modeled in a multiscale fashion ($R = \{r_1, ..., r_n\}$) using the SSN framework. A centrality measure $f(v_i)$ is computed at different scales (r) to compose the columns of $X_f(N^{r_n})$, for each vertex v_i, and the intensity (or color) of its corresponding pixel is used to compose the label vector Y.

Source: By the author.

The intuition behind this approach is to train a neural network to correlate the SSN topological evolution of vertices to the intensity value (or color) of the pixel they represent. This is achieved by training one neural network to each image (given one matrix X) so that the learned correlation is related only to the patterns within that image. Afterwards, this knowledge can be used to characterize (or encode) the image, *i.e.*, to compose image descriptors. On the other hand, common neural networks are usually trained using stochastic procedures (such as SGD), which would not be suitable for such a task since the process is not deterministic and would produce different results for the same image if repeated. Moreover, it is also both time-consuming, since it is impossible to predict the number of necessary iterations, and complex to tune considering its many hyperparameters such as learning rate, batch size, and many more depending on the optimizer. In contrast, RNNs propose a simpler solution for training neural networks, instead of the common backpropagation. In its simplest form, it is a single-hidden-layer feed-forward neural network whose input weights are random, while the weights of the output layer are learned by a closed-form solution (one-shot learning). For instance, compared to common neural networks, gains up to 150 times can be observed in the training time of RNNs. (99)

This training procedure is deterministic and well-defined theoretically, and is also easily solvable with common matrix multiplication techniques, implying efficient implementation in common neural network libraries.

Recently, some works have investigated RNNs to learn texture features for image analysis, which is achieved by using networks with fixed random weights trained for each individual image. Sá Junior *et al.* (152) used small local regions of one image as inputs to an RNN, and the central pixel of the region as the target. The trained weights of the output layer for each image are then used as a texture representation. Ribas *et al.* (154) improved the previous approach by incorporating graph theory to model the grayscale texture image. The later work was developed through a collaboration in our research group, in which the author participated (and is a co-author in the referred work). The methodology described here was partially developed during that collaboration, from which we expanded to encompass color textures. In the following, we discuss these mechanisms and how we employ them to obtain SSRs for texture characterization.

Consider $X_f(N^{r_n}) \in \mathbb{R}^{n \times whz}$ as input matrices for an RNN (given a measure f and network N^{r_n}), and $g = \phi(X\mathcal{W})$ as its forward pass of the hidden layer with a sigmoid nonlinearity, where $\mathcal{W} \in \mathbb{R}^{whz \times q+1}$ represents the random input weights for q neurons. Before describing its training, it is important to discuss a crucial aspect of RNNs, which is the generation of the random weights $\mathcal{W}$. We build an LCG (see Equation 2.18) sequence of length $u = q(n+1)$ using the same configuration as previous works, (152, 154) *i.e.*, with $a = u + 2$, $b = u + 3$ and $c = u^2$. The weight matrix is obtained by reshaping the sequence into q rows of length $\overline{z} + 1$ (q neurons and $\overline{z}$ features plus a bias term) and applying zero-center and unity-variance normalization.

We then consider the training of 1-layer RNNs by using a least-squares solution at the output layer. Given the output labels Y, the output weights Ξ are obtained as the solution of a system of linear equations:

$$\Xi = Yg^T(gg^T + \lambda I)^{-1}, \tag{4.9}$$

where $g^T(gg^T)^{-1}$ is the Moore–Penrose pseudo-inverse (97,98) of matrix g, I is the identity matrix, and $\lambda = 10^{-3}$ is a parameter for the Tikhonov regularization (213, 214) used for avoiding inaccurate inverse when the matrices become close to singular. After this operation, the RNN using input random weights $\mathcal{W}$ and output weights Ξ is ready for regression tasks predicting the pixel colors given their local topological properties.

Trained RNNs are used for feature extraction by utilizing the knowledge encoded in their final weights Ξ. To obtain an image descriptor, we propose the concatenation of output weights from a set of nine RNNs, each trained with each of the X_f matrices (combinations of measure and SSN networks). Consider $\Xi_f(N^{r_n})$ as the output weights of an RNN trained with input matrix $X_f(N^{r_n})$. In this sense, by combining RNNs trained in all cases, we obtain a vector:

$$\Upsilon_q^{r_n} = \Big[\Xi_{k_{in}}(N^{r_n}), \Xi_{s_{in}}(N^{r_n}), \Xi_{s_{out}}(N^{r_n}), \Xi_{k_{in}}(W^{r_n}), \Xi_{s_{in}}(W^{r_n}), \Xi_{s_{out}}(W^{r_n}),$$
$$\Xi_{k_{in}}(B^{r_n}), \Xi_{s_{in}}(B^{r_n}), \Xi_{s_{out}}(B^{r_n})\Big]. \tag{4.10}$$

The descriptor $\Upsilon_q^{r_n}$ is obtained for one given SNN modeled with maximum radius r_n and a value of q (size of the RNN's hidden layer). These two parameters directly impact the RNN learning process and, consequently, the obtained features from their output weights. To improve the feature extraction procedure, we propose an additional representation:

$$\bar{\Upsilon}_q^{(r_{n1}, r_{n2})} = \left[\Upsilon_q^{r_1}, \Upsilon_q^{r_2}\right], \tag{4.11}$$

which combines two radii parameters. Similarly, it is possible to combine different RNN sizes into another representation:

$$\varphi_{(q_1,\dots,q_m)}^{(r_{n1}, r_{n2})} = \left[\bar{\Upsilon}_{q_1}^{(r_{n1}, r_{n2})}, \dots, \bar{\Upsilon}_{q_m}^{(r_{n1}, r_{n2})}\right]. \tag{4.12}$$

Figure 4.2 illustrates the overall idea of the SSR pipeline. The multi-step feature concatenation is performed to ensure that a robust representation is obtained. For instance, by using different neighborhood sizes r_n, the RNN is able to learn more complex multiscale patterns. On the other hand, by considering different RNN sizes, we are tuning their learning capacity from smaller neural networks, which encode simpler patterns, to bigger models with an increasing number of parameters and capabilities. Note that when the size of the RNN is increased, its random weight distribution changes according to Eq. (2.18), which also means a different projection of the inputs performed by the hidden layer. When combined, all these properties help in obtaining better image descriptors, as we observed in our experiments. However, it is also important to notice that these parameters should be tuned to achieve a balance between several aspects, such as the characterization performance, the computational cost (size of the descriptor and cost to train the RNNs), and feature redundancy.

4.2.1 Analysis of the proposed SSR

We evaluate the parameters to obtain the SSR feature vector $\varphi(r_{n1}, r_{n2})_{(q_1,\dots,q_m)}$, *i.e.*, the combination of different values of r_n and q. This is performed following the building blocks of the final vector, so we start analyzing vector $\bar{\Upsilon}_q^{(r_{n1}, r_{n2})}$ for $(r_1, r_2) \in \{4, 6, 8, 10, 12\}$. For this case, a fixed number of hidden RNN neurons $q = 4$ is used. Figure 4.2.1 (a) shows the obtained results as the average classification accuracy after evaluation with the LDA classifier over three texture benchmarks (USPtex, Outex13, and MBT) using leave-one-out cross-validation. It is possible to notice that combining two radii is considerably superior to using individual values, and the highest results are

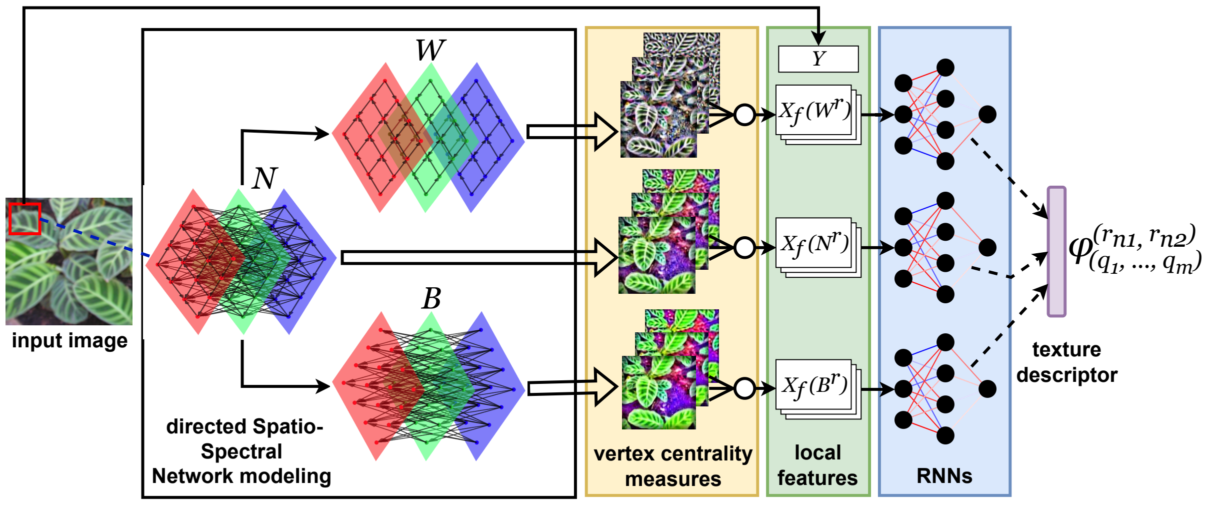
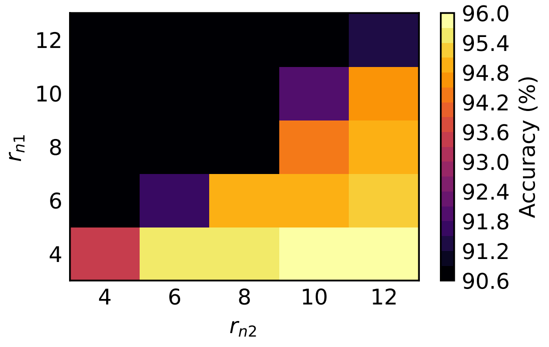

W
N
B
Y
$X_f(W^r)$
$X_f(N^r)$
$X_f(B^r)$
$\varphi^{(r_{n1},\, r_{n2})}_{(q_1,\, \dots,\, q_m)}$
input image
directed Spatio-Spectral Network modeling
vertex centrality measures
local features
RNNs
texture descriptor

Figure 26 – The proposed pipeline for obtaining SSR descriptors. An input image is modeled as SSNs, from which local descriptors are obtained to each vertex as the evolution of centrality measures (f) as the modeling radius r increases. These local features (X_f) from each SSN network are used to train 1-layer RNNs given the corresponding pixel colors as the targets (Y), and the learned output weights compose the final image descriptor.

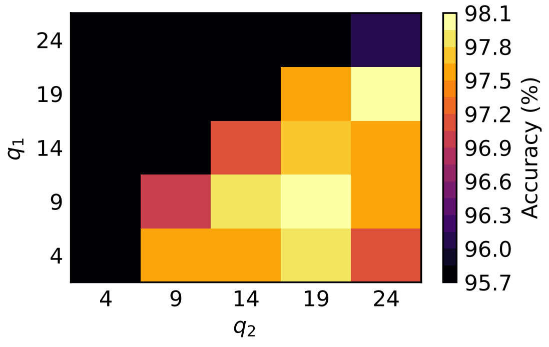
Source: By the author.

obtained by combining $r_{n1} = 4$ to $r_{n2} = 10$ and 12. This means that RNNs are trained in two modeled SSNs, one using independent neighborhoods with radii $R = \{1, 2, 3, 4\}$ and the other using $R = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$ (or up to 12). This behavior is expected as we previously discussed, *i.e.*, it enables the RNNs to learn more complex local multiscale patterns. The utilization of smaller radii facilitates the extraction of local fine-grained texture features, while larger radii incorporate more global features. By incorporating information from varying spatial scales, a more robust and accurate representation of the image can be achieved, which is essential for many advanced image processing and computer vision applications. For the sake of efficiency, we selected $r_{n2} = 10$ since it yields similar results as $r_{n2} = 12$ while being faster to compute. It is also important to notice that in practical terms, only the SSN $N^{r_n=10}$ is built, while the $N^{r_n=4}$ version is obtained during the modeling of the larger one, and the networks W and B are obtained by segmenting their connections during the modeling as well.

A second experiment evaluates the combination of RNNs with varying numbers of hidden neurons (q) while using $r_n = (4, 10)$. Firstly, we evaluate the use of RNNs with a fixed size q or combining two choices q_1 and q_2 to build the final SSR descriptor $\varphi^{(4,10)}_{(q_1,\dots,q_m)}$. The results (average classification accuracy) are shown in Figure 4.2.1 (b) for $q \in \{4, 9, 14, 19, 24\}$, where it is possible to notice again that using a single size (diagonal) is not the best approach. The best results are observed with the combinations $q = (4, 19)$,



(a) Varying r_n, using $q = 4$.



(b) Varying q, using the combination ($r_{n1} = 4$, $r_{n2} = 10$)

Figure 27 – Evaluating parameter combinations for building SSR descriptors. The results are the average classification accuracy of the LDA classifier over three texture benchmarks (Outex13, MBT, and USPtex). The upper diagonal is omitted since it is just a repetition of the combinations in the lower diagonal of the matrices. (a) We first test combinations of two choices of parameter r_n, and the diagonal represents the result when using each individual r_n. (b) The second matrix shows the results for different q values (diagonal) or combining two different choices q_1 and q_2.

Source: By the author.

$(9, 14)$, $(9, 19)$, and $(19, 24)$. A final parameter evaluation is then performed testing combinations of three RNN sizes from these best results, *i.e.*, $q \in \{4, 9, 14, 19, 24\}$. As the results in Table 10 show, the performance is more stable in this scenario (around 98% accuracy) for any combination of q. We also show the final number of descriptors obtained with each configuration, which can be calculated as $\sum_q 18(q_i + 1)$, where 18 indicates the 9 matrices X_f for each of the two radii $r_n = (4, 10)$, and $+1$ represents the bias term of the RNN. We selected as the two best SSR descriptors the vectors $\varphi^{(4,10)}_{(4,9,19)}$ (SSR$_a$) and $\varphi^{(4,10)}_{(9,19,24)}$ (SSR$_b$), which achieve the best results and have a varying number of descriptors which should work in different applications. The asymptotic cost to obtain these SSRs representations is the sum of the cost of SSNs (see Eq. (4.8)) and RNNs (see Section 2.2.4), *i.e.*, $\mathcal{O}(whzg_{r_n} + 2whz)$, where g_{r_n} is the neighborhood size of the largest modeling radius.

4.2.2 Conclusion

In this section, we presented a novel method for texture analysis by coupling the previously described SSN image modeling with an RNN characterization technique, which we named SSR. While SSN employs statistics from the centrality distribution to compose a feature vector, SSR uses the weights of RNNs that learn the descriptors automatically by training locally to each image. We evaluated the impacts of the SSN modeling parameter and the dimensions of the RNNs on the performance of texture recognition using SSRs

Table 10 – Evaluation of three combinations of RNN sizes (q_1, q_2, q_3) for building SSR features $\varphi^{(4,10)}_{(q_1,q_2,q_3)}$. The results are the average and standard deviation of LDA's classification accuracy on three texture benchmarks (Outex13, MBT, and USPtex). The two best results are highlighted in bold type.

(q_1, q_2, q_3)	descriptors	Accuracy (%)
(4, 9, 14)	540	98.1±1.3
(4, 9, 19)	630	**98.2±0.9**
(4, 9, 24)	720	98.1±1.0
(4, 14, 19)	720	98.0±1.2
(4, 14, 24)	810	98.0±1.1
(4, 19, 24)	900	98.0±1.1
(9, 14, 19)	810	98.0±1.1
(9, 14, 24)	900	98.0±1.1
(9, 19, 24)	990	**98.2±0.8**
(14, 19, 24)	1080	97.8±1.0

Source: By the author.

in three public benchmarks. In Chapter 5, we consider these findings for comparing the proposed SSR to other texture analysis techniques on a variety of benchmarks.

4.3 RADAM: Randomized autoencoders for deep feature aggregation

In this chapter, we previously described two new texture analysis methods using a more traditional feature engineering approach, *i.e.*, the image modeling and characterization techniques are carefully designed. In this section, we describe another proposed method that follows a different approach: taking advantage of pre-trained DCNNs. In general, these deep neural networks are trained for generic visual recognition using large datasets with images from the internet. However, the amount of available data for training them for texture recognition is not comparable to these cases, and deep models struggle to avoid overfitting when trained from scratch on some texture datasets. (19) On the other hand, they can be ported to texture analysis through transfer learning by doing fine tuning or feature extraction, but exhibit an average performance in general and fail to compete at the SOTA level with some hand-engineered features.

Therefore, recent deep learning-based approaches to texture recognition (113, 141, 145–148) propose feature aggregation techniques for better employing the knowledge of pre-trained deep CNNs. These transfer-learning approaches combine the general vision capabilities of pre-trained models with dedicated techniques to capture additional texture information, achieving SOTA performance on several texture recognition tasks. Therefore, most of the recent works on deep texture recognition propose to build new modules around a pre-trained deep network (backbone) and to retrain the new architecture for a specific texture analysis task. However, even if the new modules are relatively cheap in terms of computational complexity, the retraining of the backbone itself is usually costly since they are composed of dozens or even hundreds of thousands of parameters.

Here, we propose a new feature encoding module that acts on the backbone only as a feature aggregator and extractor, and a dedicated classifier is applied over the obtained features (no backbone fine-tuning is done). This method was developed during the author's stay at Ghent University, from January 2022 to January 2023, under the direct supervision of the co-advisor Bernard De Baets. It also had the collaboration of researchers Kallil Zielinski, Lucas Ribas, and Wesley Nunes Gonçalves, resulting in a finished paper (215) submitted for revision in an international journal. The main idea of the proposal we called Random encoding of Aggregated Deep Activation Maps (RADAM), is to use multi-depth feature aggregation and randomized pixel-wise encoding to compose a single feature vector, given an input image processed by the backbone.

4.3.1 Deep activation maps

Using a pre-trained DCNN as a feature extractor can provide useful image representations for various tasks even if the task differs significantly from the task originally considered for pre-training the model. (216) To achieve that, the most common approach is to remove the classification head and any possible fully connected layers at the top of the model, and then use the output of a deep convolutional layer, a.k.a. a deep convolutional activation, or deep activation map. A feature vector can be obtained from it by using pooling or other operations, which can then be simply used to train a dedicated classifier (a.k.a. linear classifier probe). The usual approach is to use DCNNs pre-trained in large natural image datasets. Here, we indicate the pre-training dataset used for them using "in", *e.g.* in IN-21K (ImageNet-21K (54) is the complete ImageNet), and ImageNet-1K (2012 version) was used when not stated (most of the cases). We refer the reader to Section B.2 for further details about the ImageNet dataset.

Consider an input image $I \in \mathbb{R}^{w_0 \times h_0 \times 3}$ fed into a DCNN backbone $B = (d_1, ..., d_n)$, consisting of n blocks of convolutional layers. An activation map, *i.e.*, the output of any convolutional block given I, is a 3-dimensional tensor (ignoring the batch dimension, for simplicity) $X_i \in \mathbb{R}^{w_i \times h_i \times z_i}$, that we refer to here as a deep activation map, where i indicates the layer's depth. In the case of convolutional layers, these feature maps are obtained by processing the output of layer $i-1$ with a set of filters and a nonlinear activation function, which we omit here for simplicity. However, it is important to notice that these filters are composed of weights (a.k.a. connections, or synapses in a neural network) which can be random (right after the network initialization), or trained (adjusted after backpropagation in a given task). Moreover, there are vastly more types of layers in a DCNN that also yield deep activation maps, for example, the dropout layer, the concatenation layer, batch normalization, etc. The CNN literature is vast and is not our focus here, so we refer the reader to Section 2.2 for a general discussion of ANNs, and to (56) for a comprehensive review of CNN architectures.

The 3-dimensional matrix X_n represents the output of the last convolutional layer (n) of a DCNN backbone, and it is usually flattened to fit into a fully-connected layer. Another approach is the use of Global Average Pooling (GAP) (217) to reduce the data before passing it to a classification layer:

$$\varphi^n_{GAP}(j) = \frac{\sum\limits_{x=1}^{w_n} \sum\limits_{y=1}^{h_n} X_n(x, y, j)}{w_n h_n}, \tag{4.13}$$

where $X_n(x, y, j)$ represent the value at position (x, y) of the 2-dimensional matrix at position $j \in z_n$, and $w_n h_n$ is the spatial size of this matrix. In other words, each position j of the feature vector φ is obtained by taking the average of each of the z_n 2-dimensional matrices j stacked to compose X_n. This vector can be employed as a final image descriptor, therefore pre-trained DCNN can be used as feature extractors by obtaining φ for a different task from which it was trained.

Feature aggregation can be achieved by combining the GAP feature vectors from different convolutional layers:

$$\varphi_{GAP_{agg}} = [\varphi^a_{GAP}], \text{for all } d_a \in B, \tag{4.14}$$

where a set B of convolutional blocks d_a can be considered. This set can be either all the convolutional layers of the model, which would yield a huge vector $\varphi_{GAP_{agg}}$, or a selection of intermediate layers throughout the backbone. The latter approach is more feasible since the dimension of the feature vector does not grow significantly for deep models, and also considering that two subsequent convolutional layers may contain similar features. This is the reason we discretize the backbone into a set of layer blocks $B = (d_1, ..., d_n)$, where each block d_a is a set of layers rather than a single layer.

4.3.2 Pre-trained deep convolutional networks: backbone selection

The process of feature aggregation consists of combining the outputs of different activation maps at different depths. To that end, we divide the backbone into a fixed number of blocks according to different depths. This division is made to keep a fixed number of blocks for feature extraction, regardless of the total depth of the backbone architecture. Most previous works on deep texture recognition consider pre-trained ResNets (61) as backbones. Here, we consider the output of five blocks of layers according to the ResNet architecture, meaning that five activation maps are considered for feature aggregation. Additionally, we consider the ConvNeXt architecture, (67) a more recent method with promising results in image recognition. For this backbone, we consider the activation maps from the four blocks of layers according to the architecture described in the original work. More specifically, the following ConvNeXt configurations are used, with their corresponding number of channels (z_i) of each block:

- ConvNeXt-nano *: $z_i = (80, 160, 320, 640)$.
- ConvNeXt-T: $z_i = (96, 192, 384, 768)$.
- ConvNeXt-B: $z_i = (128, 256, 512, 1024)$
- ConvNeXt-L: $z_i = (192, 384, 768, 1536)$.
- ConvNeXt-XL: $z_i = (256, 512, 1024, 2048)$.

Later on, we also explore RADAM applied to various other DCNN backbones according to results from the literature, and additional experiments we perform. However, during this section, we focused only on ResNets and ConvNeXts for evaluating and adjusting the RADAM structure.

4.3.3 Deep Activation Map Aggregation

Given each deep activation map X_i obtained by a DCNN backbone, we first apply a depth-wise l_p-normalization ($p = 2$, *i.e.*, Euclidean norm)

$$X_i(:,:,j) = \frac{X_i(:,:,j)}{\max(||X_i(:,:,j)||_2)}, \tag{4.15}$$

where $X_i(:,:,j)$ represents the 2-dimensional activation map at each channel $j \in z_i$ with spatial sizes (w_i, h_i).

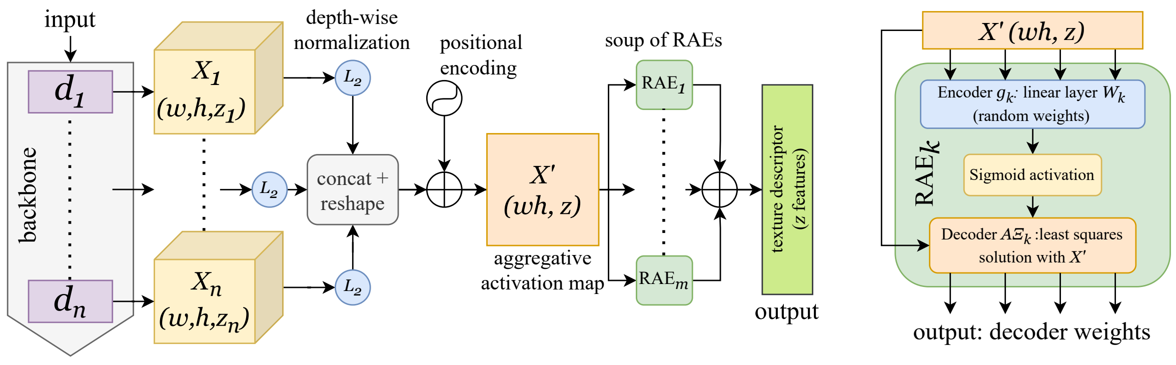
For feature aggregation, we propose to concatenate the activation maps along the third dimension (z_i). However, each map X_i initially has a different spatial dimension w_i and h_i. To overcome this, we simply resize all activation maps with bilinear interpolation using the spatial dimensions of $X_{\frac{n}{2}}$, $(w_{\frac{n}{2}}, h_{\frac{n}{2}})$ as the target sizes. In other words, we consider the spatial dimensions at the middle of the backbone as our anchor size, meaning that some activation maps will require upscaling (if $i > \frac{n}{2}$) and others downscaling (if $i < \frac{n}{2}$). Naturally, the information from deeper activation maps receives higher priority considering that upscaling preserves more information than downscaling. These assumptions consider the most common structure of convolutional architectures where the spatial size decreases with layer depth. Nonetheless, the idea is to keep all activation maps with a fixed spatial dimension. From now on, we will refer to spatial dimensions of all X_i as $w = w_{\frac{n}{2}}$ and $h = h_{\frac{n}{2}}$. For an input size of 224x224, this results in $w = h = 28$ for the

* This variant was not presented in the original paper but is available in the PyTorch Image Models library. (218)

backbones explored in this work. The concatenation of activation maps is then performed as:

$$X' = [X_1; ...; X_n] \in \mathbb{R}^{w \times h \times z} \rightarrow X' \in \mathbb{R}^{wh \times z}, \tag{4.16}$$

where $[.;.]$ denotes the concatenation along the third dimension, and $z = \sum_i z_i$ is the resulting number of channels after concatenation. Considering common convolutional architectures where $z_i < z_{i+1}$, deeper activation maps have a higher influence on the overall z features since they will provide more features for the aggregation procedure. Additionally, the 2-dimensional activation map at each channel z_i is flattened, resulting in the reshaped 2D representation X' with sizes wh-by-z, which we refer to as an aggregated activation map. These steps are illustrated in Figure 28(a), which reports the overall structure of the proposed method.



(a) The proposed feature encoding module (RADAM).

(b) Randomized Auto-encoder (RAE).

Figure 28 – Illustration of the proposed RADAM architecture (a), given an input image to a final descriptor. The RAE is shown in detail (b), which is a simple 1-layer auto-encoder solved through least-squares, where we use the decoder weights as descriptors (summed for m RAEs).

Source: By the author.

4.3.4 Pixel-wise Randomized Encoding

The aggregated activation map of a single image is used to train an RAE considering each spatial point, or pixel (row of X'), as a sample and each channel (column of X') as a feature. In this sense, the method also works with arbitrary input sizes (if accepted by the backbone) since the spatial dimensions only affect the number of training samples for the RAE. Intuitively, larger input sizes would improve the RAE training, but would also increase the backbone cost significantly. Therefore, in this work, we consider only a constant input size of 224x224 (forced resizing), since this is the most common configuration of various backbones. Moreover, considering that the spatial organization of the pixels is lost due to the flattening procedure of X', we add a positional encoding composed of sine and cosine functions of different frequencies (66) with dimension z, extended for 2 spatial dimensions: (219)

$$
\begin{aligned}
&\mathrm{PE}(x,y,2i) = \sin(\frac{x}{10000^{4i/z}}),\\
&\mathrm{PE}(x,y,2i+1) = \cos(\frac{x}{10000^{4i/z}}),\\
&\mathrm{PE}(x,y,2j+z/2) = \sin(\frac{y}{10000^{4j/z}}),\\
&\mathrm{PE}(x,y,2j+1+z/2) = \cos(\frac{y}{10000^{4j/z}}),
\end{aligned}
$$

where $x \in w$ and $y \in h$, which is then added to the aggregated activation map via element-wise sum:

$$X' = X' \oplus \mathrm{PE}. \tag{4.17}$$

After summing the positional encoding to X', the first step of the RAE is to project the inputs using a random fully-connected layer with weights $\mathcal{W}_k \in \mathbb{R}^{z\times q}$, followed by a sigmoid nonlinearity. The weights are generated using the LCG for simplicity and better replicability, followed by standardization (zero-centered, unity variance) and orthogonalization. (82) These configurations were chosen according to previous works. (99, 220) As for the LCG parameters, we use $a = 75$, $b = 74$, and $c = 2^{16}+1$, starting with $x = 0$, which is a classical configuration according to the ZX81 computer from 1981. It is important to notice that this is a different configuration compared to previous works using RNNs since we reevaluated it for the RAE context (experiments are shown later). Moreover, here k works like a seed for random sampling, denoting a starting index inside the LCG space generated with the given configuration. More details on LCG weights are given in the Supplementary Material, such as an ablation on the impacts of different LCG configurations. The forward pass of the encoder $g_k \in \mathbb{R}^{wh\times q}$ for all samples is then obtained as:

$$g_k = \phi(X'\mathcal{W}_k)\,, \tag{4.18}$$

and the decoder weights $A\Xi_k \in \mathbb{R}^{z\times q}$ are obtained as the least-squares solution of the usual RNN, but changing the target Y to X':

$$A\Xi_k = X'^T g_k^T (g_k g_k^T)^{-1}\,. \tag{4.19}$$

The main idea of employing an individual randomized neural network for each image is to use the output weights themselves as a representation. In the case of RAEs, the output layer has the same dimension as the input layer. Therefore, a single hidden neuron ($q = 1$) is considered to preserve the dimensionality. In this sense, the resulting decoder weights are represented by:

$$A\Xi_k = (\nu_1, \ldots, \nu_z)\,, \tag{4.20}$$

where ν_i represents the connection weight between the single hidden neuron and the output i, corresponding to feature $i \in z$.

A single-neuron RAE may be limited in encoding enough information contained in the deep activation maps. Therefore, we propose an ensemble of models or, as recently introduced, (221) a model "soup", which is achieved by combining the weights of m parallel models. Here, each model is an RAE with a different random encoder (using a different LCG seed), and the combination is performed by summing the decoder weights

$$\varphi_m = (\sum_{k=1}^{m} A\Xi_k(\nu_1), \ldots, \sum_{k=1}^{m} A\Xi_k(\nu_z)) \,. \tag{4.21}$$

It is important to note that the encoders g_k of each of the m RAEs have a different random weight initialization. This is achieved by creating an LCG sequence of size mz so that we have z weights for each of the m RAEs. The structure of the RAE is illustrated in Figure 28(b), and following the whole RADAM pipeline shown in Figure 28(a), a texture representation, or feature vector φ_m, is obtained for the input image. The feature vectors φ_m are then used to train a linear classifier for a given texture recognition task (more details on the classifier can be consulted in Section 4.3.6). Our model is implemented using PyTorch (81) (except for the classification step), making it easier to couple RADAM with several deep learning methods implemented in this library. The classification step is performed using Scikit-learn. (222)

4.3.5 Discussion about RADAM

The idea of local random networks trained on a single image, and its weights used as features, is supported by previous works on texture analysis. (152,220) By using the same weights for all images, even if random, all of them will be projected by the same function, and the least-squares solution will try to linearly correlate the projection with the desired output. Here, we extend these concepts to 3-dimensional activation maps obtained with convolutional networks, while also considering multi-depth feature aggregation. Another new approach is the combination of different random networks using a "model soup". (221) The intuition is that since each model contains different random weights, each learns a different encoding of the activation map that when combined improves the representation (as we show in our experiments).

Considering RAEs with a single neuron, the model learns to encode the z features of each pixel into a single value, which is similar to the process of global pooling across the 3rd dimension of the activation map. However, instead of choosing a pooling method or measure, the model will automatically select the best projection that preserves most of the information, averaged over all pixels. This representation is mapped through the decoder weights, which correspond to a function to transform back the 1-dimensional latent space into the original z-dimensional features. Since using the decoder weights itself as a feature representation results in the same size as the 3rd dimension of the activation map, the output is also similar to a global pooling across the 1st and 2nd (spatial) dimensions. The aggregation procedure, on the other hand, extends this concept for a multi-depth and dynamic analysis. In the end, the RAE encodes the evolving features throughout the architecture at individual pixels, giving more importance to deeper features. Like global poolings, the RAE training is also spatially-orderless since each pixel is processed individually. However, we noticed that including positional encoding is beneficial, suggesting that the spatial information present in deep activation maps is useful for texture analysis if better employed.

Lastly, regarding the computational budget of the RADAM module alone (not considering the backbone), the number of FLOPs of a single forward pass mainly falls into the computation of the least-squares solution of the RAE. We implemented this step with *torch.linalg.lstsq*, which uses QR factorization with an approximate number of FLOPs of $\mathcal{O}(2wh + z)$ (see Section 2.2.4). This operation is performed over matrix X' with sizes 784×3840 in the worst case, considering ConvNeXt-XL (784 comes from the 28x28 spatial dimension in the middle of the backbone for 224x224 inputs). These sizes are significantly smaller depending on the backbone, for instance, 784×1200 for ConvNeXt-nano. However, the most significant term for the cost is $wh = 784$, which is constant when using 224x224 inputs in all the architectures we explored (but could be different for other types of networks). Nevertheless, the cost is significantly smaller than the cost of DCNN backbones. As for the number of parameters (not trainable), only the encoder weights are required to be stored since the decoder weights are the outputs of the model. In the worst case scenario during our experiments, *e.g.* using the ConvNeXt-XL backbone and $m = 4$, the number of required weights was 15360. In contrast, ConvNeXt-nano requires only 4800 weights. These parameters are negligible when compared to the backbone sizes of around 350 and 15 million parameters for ConvNeXt-XL and ConvNeXt-nano, respectively.

4.3.6 Analysis of the proposed RADAM

We evaluate two design choices for the RADAM pipeline, the use of positional encoding and the classifier. The method is compared with or without encoding under two different classifiers: Linear Discriminant Analysis (LDA) (209, 210) and Support Vector Machines (SVM). (223) Since the evaluation of positional encoding concerns the spatial properties of texture, we consider the Outex10 benchmark (106) which focuses on rotation invariance and contains a pre-defined training and testing split (see Section B.1 for further details). As the results in Table 11 demonstrate, positional encoding improves or maintains performance in all cases, especially under the LDA classifier. On the other hand, SVM provides the best results in all cases while gaining less improvement from positional encoding. Nevertheless, we keep the positional encoding in our architecture since the additional cost is negligible compared to the potential gains. The SVM is also used as the classifier for the following experiments in this section.

Table 11 – Ablations with positional encoding and different classifiers, using RADAM ($m = 1$) with different backbones. Results are measured by classification accuracy considering the single train/test split of Outex10.

		Outex10	
Encoding	Backbone	LDA	SVM
none	ResNet18	77.5	83.4
positional	ResNet18	79.5	84.7
none	ConvNeXt-nano	82.4	89.6
positional	ConvNeXt-nano	85.9	89.6
none	ResNet50	86.0	87.4
positional	ResNet50	87.4	87.4
none	ConvNeXt-T	87.8	91.1
positional	ConvNeXt-T	89.4	91.1

Source: By the author.

The only free parameter of the proposed RADAM method is the number of RAEs to be combined, *i.e.*, m. We evaluated m ranging from 1 to 32 and the results are shown in Figure 29 for different backbones in the Outex13 dataset (using its original validation split). We observe significant gains for m from 1 to 4, while for larger values performance tends to stabilize. These results indicate that $4 \leq m \leq 8$ is a good approach for a balance between performance and cost since no significant gains are achieved above that. All the following experiments are performed using $m = 4$.

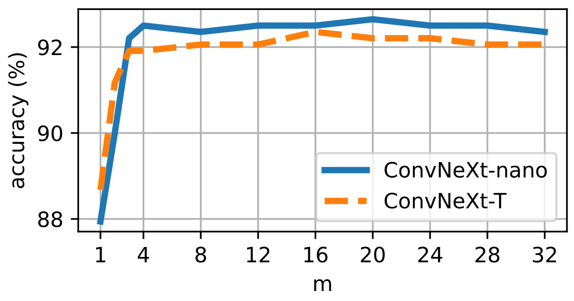


Figure 29 – Impacts of changing the number of RAEs (m) in the proposed method, considering different backbones on the Outex13 dataset. All other experiments in this paper consider $m = 4$.

Source: By the author.

The effects observed when increasing m are expected considering what is usually seen in model ensembles, or “model soups”, (221) where the combination of models trained separately may be beneficial. On the other hand, our encoders are random, and each one has different weights. However, even if each encoder creates a different random projection of the input, the decoders learn to transform the projection back to the same feature space. In other words, the RAEs learn different encoding-decoding functions for the same input that, when combined, provide a better representation in our feature extraction use case.

We also evaluate the performance variance of RADAM caused by changing the LCG configuration (a, b, and c) for weight initialization of the RAE. We selected a set of 10 configurations from the table available in [†], and the results are shown in Table 12. We considered four texture datasets (DTD, FMD, KTH-TIPS2-b (107, 108), and Outex13), which represent different types of textures and tasks. In general, we observe minimal performance variance concerning these parameters, especially with larger backbones, corroborating that the choice of the random weights is not critical. We adopted parameters $a = 75$, $b = 74$, and $c = 2^{16} + 1$ starting with $x = 1$ (first line in the table), as it uses the smallest integers and is also a classical configuration according to the ZX81 computer from 1981. However, small variations may impact the comparison between RADAM and other methods. Nevertheless, it is important to notice that our choice of LCG parameters does not impact the new SOTA results achieved. In fact, carefully selecting specific configurations in Table 12 for each dataset and backbone yields even higher results, but we preferred a single consistent configuration to be used for all cases to avoid overfitting.

To show the gains of RADAM over other pooling approaches, we performed additional experiments using Global Average Pooling (GAP), Global Max Pooling (GMP), Global Fractional Max Pooling (FMP), and Global L-p Pooling (GLpP). We refer the reader to (224) for a review of pooling techniques, including the aforementioned ones. We apply these techniques over pre-trained backbones to obtain a feature vector, given input images, which are then used to train a linear SVM (like RADAM). The obtained results are shown in Table 13 for four backbones and two challenging texture recognition datasets (DTD and FMD). We indicate the pre-training dataset used for the backbone using “in”, *e.g.*in IN-21K (ImageNet-21K (54)), and ImageNet-1K was used when not stated. In summary, the GAP is superior to other pooling techniques from the literature in all cases. On the other hand, RADAM achieves considerably superior results than GAP, with gains above 10% in absolute classification accuracy in some cases. These results corroborate the effectiveness of RADAM for summarizing deep activations into texture descriptors, especially when considering ConvNeXt backbones.

Additionally, we evaluate GAP agg., which aggregates the GAP from each of the n feature blocks and returns an image representation with z features (as RADAM). The results are shown in Table 14, where it is possible to observe that RADAM overcomes GAP by a considerable margin, in all backbones and datasets. GAP agg. usually improves over the standard GAP, which is to be expected since more features are being added.

[†] www.wikipedia.org/wiki/Linear_congruential_generator

Table 12 – The variance on SVM classification accuracy with RADAM features when changing the LCG configuration to generate random weights for the RAE, averaged over various configurations on four texture benchmarks. For the second part of the table (ConvNeXt results), the LCG configurations correspond to the same rows as for ResNet18.

Parameters of LCG				ResNet18			
source	a	b	c	DTD	FMD	KTH	Outex13
ZX81	75	74	$2^{16}+1$	68.1	77.7	84.7	89.0
Numerical Recipes book	1664525	1013904223	2^{32}	68.0	77.9	85.0	90.0
Borland C/C++	22695477	1	2^{32}	66.7	79.7	86.0	88.2
glibc (used by GCC)	1103515245	12345	2^{31}	68.0	78.0	84.7	90.0
Borland Delphi, Virtual Pascal	134775813	1	2^{32}	67.9	79.2	85.6	90.1
Microsoft Visual/Quick C/C++	214013	2531011	2^{32}	68.1	78.2	85.1	89.3
Microsoft Visual Basic	1140671485	12820163	2^{24}	68.1	79.2	85.4	88.7
RtlUniform from Native API	2147483629	2147483587	2^{31}-1	68.0	78.9	85.1	89.6
Java's java.util.Random	25214903917	11	2^{48}	67.9	77.4	85.4	89.7
cc65	16843009	826366247	2^{32}	68.0	78.7	85.6	89.0
			average	67.9±0.4	78.5±0.8	85.3±0.4	89.4±0.6

	ConvNeXt-nano				ConvNeXt-XL in ImageNet-21K			
	DTD	FMD	KTH	Outex13	DTD	FMD	KTH	Outex13
	74.9	87.1	89.6	92.5	83.7	95.2	94.4	92.5
	75.0	86.9	89.2	92.4	83.9	95.7	94.5	91.8
	75.1	86.8	89.5	91.2	83.7	95.3	94.6	92.4
	75.0	86.6	89.8	92.2	83.9	95.4	94.1	92.6
	75.1	86.4	89.5	92.4	83.8	95.6	94.5	92.4
	75.1	87.1	89.5	92.5	83.8	95.5	94.6	92.8
	74.9	87.7	89.4	92.2	83.7	95.3	94.6	92.4
	75.1	86.9	89.6	91.9	83.8	95.2	94.6	91.9
	74.9	86.9	89.3	92.6	83.8	95.5	94.8	92.2
	75.0	87.6	89.0	92.2	83.8	95.5	94.5	92.8
average	75.0±0.1	87.0±0.4	89.4±0.2	92.2±0.4	83.8±0.1	95.4±0.2	94.5±0.2	92.4±0.3

Source: By the author.

However, the standard GAP sometimes performs better than this simple aggregation by concatenation. These results suggest that better aggregation techniques are needed for improving texture feature extraction. In this sense, the proposed RADAM represents a promising alternative, achieving SOTA performance in all benchmarks we considered, as we discuss later on (see Chapter 5). In summary, our evaluation shows that feature extraction with RADAM provides considerably better results than using the backbone alone (coupled with pooling techniques).

Finally, we evaluate the empirical running time of RADAM (with $m = 4$ and SVM) in contrast to the costs of ResNet. Our intuition is to compare the use of RADAM, which requires only a forward pass of the backbone and an SVM, to the approach in previous works of fine-tuning the backbone. We measured the average time (in milliseconds) of 100 runs of ResNet18 and ResNet50 over one 224x224 RGB image considering the RADAM module, the ResNet forward and backward passes, and the corresponding times when fine-tuning the backbone our using RADAM for feature extraction with SVM inference.

Table 13 – Classification accuracy (linear SVM) for features obtained with different pooling techniques in comparison to the proposed RADAM. We consider two challenging texture benchmarks (DTD and FMD).

method	backbone	DTD	FMD
GAP	ResNet18	64.1±1.1	77.1±0.7
GMP	ResNet18	62.3±0.8	75.6±0.5
FMP	ResNet18	62.3±0.8	75.6±0.5
GLpP	ResNet18	63.7±1.1	77.0±0.6
RADAM	ResNet18	68.1±1.0	77.7±0.5
GAP	ConvNeXt-nano	61.9±1.0	73.8±0.9
GMP	ConvNeXt-nano	51.7±0.9	62.4±0.8
FMP	ConvNeXt-nano	51.7±0.9	62.4±0.8
GLpP	ConvNeXt-nano	44.0±0.5	50.4±0.8
RADAM	ConvNeXt-nano	74.9±0.7	87.1±0.4
GAP	ConvNeXt-B	69.2±1.3	83.6±0.5
GMP	ConvNeXt-B	59.5±1.2	77.0±0.8
FMP	ConvNeXt-B	59.5±1.2	77.0±0.8
GLpP	ConvNeXt-B	59.1±1.3	70.6±0.8
RADAM	ConvNeXt-B	76.4±0.9	90.2±0.2
GAP	ConvNeXt-B in IN-21K	79.2±0.6	91.9±0.4
GMP	ConvNeXt-B in IN-21K	70.6±0.8	84.2±0.4
FMP	ConvNeXt-B in IN-21K	70.6±0.8	84.2±0.4
GLpP	ConvNeXt-B in IN-21K	69.7±0.9	72.0±0.6
RADAM	ConvNeXt-B in IN-21K	82.8±0.9	94.0±0.2

Source: By the author.

It is important to notice that the cost of RADAM varies according to the backbone since the number of aggregate features (z) changes, thus impacting the RAE training cost. As an example, consider fine-tuning ResNet50 on the GTOS-M dataset according to the costs shown in Table 15. Using only 10 training epochs with a batch size of 1, the process would take around 40 hours on CPU or 9 hours on GPU. On the other hand, extracting features with RADAM followed by SVM inference on the whole GTOS-M dataset takes around 1.3 hours on CPU or half an hour on GPU. The training of SVM on the whole dataset (on CPU) takes an additional 15 minutes on average, without hyperparameter tuning. The results demonstrate that the RADAM module is considerably faster than the ResNet backbone, both in terms of inference speed and comparing feature extraction followed by SVM with training/fine-tuning. These results extend to ConvNeXt-nano and ConvNeXt-T, considering that the cost is comparable to ResNet18 and ResNet50, respectively, corroborating our claims that RADAM provides both considerable savings in training time and SOTA results at a similar inference cost. Moreover, considering more costly backbones such as ConvNext-L and XL, the gains in training time can be even more expressive.

Table 14 – Classification accuracy (linear SVM) when using image features obtained from the backbones with different techniques: the GAP, the aggregation of GAP from each of the n feature blocks of the backbones (the same way we approach them with RADAM), and the proposed RADAM. We consider two challenging texture benchmarks (DTD and FMD).

method	backbone	DTD	FMD
GAP	ResNet18	64.1$\pm$1.1	77.1$\pm$0.7
GAP agg.	ResNet18	64.6$\pm$1.2	77.2$\pm$0.6
RADAM	ResNet18	68.1$\pm$1.0	77.7$\pm$0.5
GAP	ConvNeXt-nano	61.9$\pm$1.0	73.8$\pm$0.9
GAP agg.	ConvNeXt-nano	71.7$\pm$0.6	84.4$\pm$0.4
RADAM	ConvNeXt-nano	74.9$\pm$0.7	87.1$\pm$0.4
GAP	ResNet50	72.6$\pm$0.9	84.5$\pm$0.6
GAP agg.	ResNet50	74.6$\pm$0.6	83.4$\pm$0.4
RADAM	ResNet50	75.6$\pm$1.1	85.3$\pm$0.4
GAP	ConvNeXt-T	66.0$\pm$1.0	79.0$\pm$0.4
GAP agg.	ConvNeXt-T	73.7$\pm$0.9	83.6$\pm$0.5
RADAM	ConvNeXt-T	77.0$\pm$0.7	88.7$\pm$0.4
GAP	ConvNeXt-T in IN-21K	78.4$\pm$0.7	91.4$\pm$0.3
GAP agg.	ConvNeXt-T in IN-21K	77.3$\pm$0.9	89.9$\pm$0.5
RADAM	ConvNeXt-T in IN-21K	81.4$\pm$0.7	93.0$\pm$0.3
GAP	ConvNeXt-B	69.2$\pm$1.3	83.6$\pm$0.5
GAP agg.	ConvNeXt-B	73.5$\pm$1.0	86.7$\pm$0.5
RADAM	ConvNeXt-B	76.4$\pm$0.9	90.2$\pm$0.2
GAP	ConvNeXt-B in IN-21K	79.2$\pm$0.6	91.9$\pm$0.4
GAP agg.	ConvNeXt-B in IN-21K	80.4$\pm$1.0	92.3$\pm$0.3
RADAM	ConvNeXt-B in IN-21K	82.8$\pm$0.9	94.0$\pm$0.2
GAP	ConvNeXt-L	70.9$\pm$0.9	84.6$\pm$0.4
GAP agg.	ConvNeXt-L	73.4$\pm$0.6	86.4$\pm$0.5
RADAM	ConvNeXt-L	77.4$\pm$1.1	89.3$\pm$0.3
GAP	ConvNeXt-L in IN-21K	80.4$\pm$0.9	92.2$\pm$0.4
GAP agg.	ConvNeXt-L in IN-21K	81.9$\pm$0.6	93.4$\pm$0.4
RADAM	ConvNeXt-L in IN-21K	84.0$\pm$1.0	95.2$\pm$0.4
GAP	ConvNeXt-XL in IN-21K	81.3$\pm$1.0	92.4$\pm$0.3
GAP agg.	ConvNeXt-XL in IN-21K	82.0$\pm$1.1	93.9$\pm$0.4
RADAM	ConvNeXt-XL in IN-21K	83.7$\pm$0.9	95.2$\pm$0.3

Source: By the author.

4.3.7 Conclusion

The proposed RADAM method, presented in this section, is a novel texture characterization approach for deep feature aggregation by taking advantage of pre-trained DCNNs. Transferring DCNNs pre-trained for object recognition to texture recognition became popular after deep learning popularization, and most approaches focus on feature aggregation and fine-tuning. However, RADAM does not perform fine-tuning and is con-

Table 15 – Average running time (in milliseconds) of 100 repetitions using a 224x224 RGB image, performed on a machine with a GTX 1080ti, Intel Core i7-7820X 3.60GHz processor (using 8 threads), and 64GB of RAM.

		time (ms)	
method	backbone	CPU	GPU
RADAM module alone	ResNet18	2.7±0.2	2.9±0.04
backbone's forward pass	ResNet18	15.0±1.6	4.3±0.1
backbone's backward pass	ResNet18	50.8±5.3	9.9±1.5
backbone's 1 fine-tuning epoch	ResNet18	65.8±6.0	14.2±1.6
backbone + RADAM + SVM inference	ResNet18	19.8±1.6	8.1±0.1
RADAM module alone	ResNet50	7.0±0.4	5.2±0.4
backbone's forward pass	ResNet50	34.0±2.6	11.1±0.8
backbone's backward pass	ResNet50	107.6±5.9	21.5±2.2
backbone's 1 fine-tuning epoch	ResNet50	141.7±7.6	32.6±2.3
backbone + RADAM + SVM inference	ResNet50	50.3±2.6	18.1±0.8

Source: By the author.

siderably simple to compute since it is based on RNNs. On the other hand, compared to SSN and SSR, RADAM is more costly since it still needs the forward pass of a pre-trained DCNN to obtain its deep activation maps. RADAM and the other proposed methods are analyzed in more depth in Chapter 5, where we also present a comprehensive comparison to other methods from the literature, in various benchmarks.

4.4 Fully randomized DCNNs for texture characterization

In this section, we introduce another novel proposal, inspired by RNNs and DCNNs for texture analysis. We consider fully randomized DCNNs for texture feature extraction. This is the first work to explore this approach in more depth for texture characterization. We evaluate several random DCNNs and their combination with two of our other proposed methods, the PARw (described in Section 3.2) and RADAM (described in Section 4.3). Early results of this work (evaluation of DCNNs), obtained in collaboration with researchers Rayner Condori and Lucas Correia Ribas, were published (225) in the 20th International Conference on Image Analysis and Processing (ICIAP 2019, Trento, Italy), which was presented by the author. The remaining analyses of fully random models, random feature aggregation, and improvements with PARw and RADAM resulted in another paper currently in production also in collaboration with researcher Andre Sacilotti.

The fact that random neural networks can perform well for feature extraction is not new. This is backed by the literature on RNNs (90–93) which we previously discussed. In this scenario, a hidden layer with random weights has the purpose of projecting non-linearly the input data in a higher/lower dimensional space where it is more likely that the feature vectors are linearly separable. Therefore, for these networks, the performance relies on the training of the output layer and the shape of the hidden layer rather than on its weights, as they are random. Some results (12) have also shown that convolutional networks with completely random weights can achieve impressive performance. In this case, the feature extraction part (convolutional layers) plays a similar role as the hidden layer of an RNN. The discussion presented in (12) also points out that the convolutional architecture alone is of high importance for object recognition, where they propose a fast method for architecture selection based on the performance of networks with random weights. This effect is a reflection of the inherent inductive biases that the CNN architecture possesses, such as translation equivariance and locality, which are not related to the weights themselves. However, nothing is known about how different random weight distributions, magnitudes, and structures could impact the feature extraction of random DCNNs. Our proposal is then a first step in this direction, which we describe in the following.

4.4.1 Evaluating fully randomized DCNNs

We start by exploring a variety of DCNNs for texture feature extraction when using completely random weights, applying GAP to obtain a feature vector, and training only a linear classifier (KNN, (226) LDA, and SVM) on them. The intuition is to better understand the applicability of randomized versions of common DCNNs when applied to texture characterization compared to their pre-trained counterparts. For that, we consider DCNNs using weights pre-trained on the ImageNet (54) dataset, which is the most common dataset for large-scale visual recognition pre-training. For a broad analysis of DCNN architectures, we considered 20 models with varying properties, and details are given in Table 16. These are well-known models which the deep learning community has extensively evaluated, and their source code is open access [‡]. Although these are all convolutional models, there are various differences between their architectures. However, explaining each of them is beyond the scope of our work, considering the number of models. The literature on DCNNs grows extremely fast, and there are a considerable number of published models. Here, we selected a variety of them according to their popularity, available code, and pre-trained weights, and also considering the time frame that this thesis was developed (2018 to 2023). For more details, we refer the reader to Section 2.2, to (56) which presents a review of CNNs, and to the referred work of each specific model.

Regarding the weight initialization method used for each network in this experiment, we considered the standard definition of the framework from which the model is imported, as this is the most common case when applying these networks. In the Keras library, the 2D convolutional layers employ the Glorot uniform random weight initialization (69) and the bias terms are initialized with 0. In the case of AlexNet, DenseNets, PNAS-

[‡] We use models from Keras (179) (https://keras.io/applications/) and the PyTorch Image Models library (218) (https://github.com/huggingface/pytorch-image-models/).

Table 16 – Details on each DCNN considered for analysis, where color represents the range of the values (more yellow/grayscale, higher the value). The network complexity (cost) is measured by the number of floating point operations (GFLOPs) and the number of parameters (in millions). The full network includes the classification head used for pre-training, and GAP refers to only its feature extraction part. $|\varphi|$ represents the size of the resulting 1-dimensional feature vector obtained with GAP from the last convolutional layer and used as an image representation.

	Full		GAP		
Network	GFLOPs	Param.	GFLOPs	Param.	$\|\varphi\|$
AlexNet (2012)(3)	0.72	61.1	0.7	2.47	256
VGG16 (2014)(227)	15.5	138.36	15.38	14.71	512
VGG19 (2014)(227)	19.67	143.67	19.54	20.02	512
InceptionV3 (2016)(190)	2.85	23.83	2.85	21.79	2048
ResNet18 (2016)(61)	1.82	11.69	1.82	11.18	512
ResNet50 (2016)(61)	4.12	25.56	4.12	23.51	2048
ResNet152 (2016)(61)	11.58	60.19	11.58	58.14	2048
InceptionResNetV2 (2017)(60)	6.5	55.84	6.5	54.31	1536
DenseNet121 (2017)(228)	2.85	7.98	2.85	6.95	1024
DenseNet201 (2017)(228)	4.32	20.01	4.32	18.09	1920
PNASNet5(large) (2018)(229)	10.77	86.06	10.77	81.74	4320
SENet154 (2018)(230)	20.8	115.09	20.8	113.04	2048
SEResNet50 (2018)(230)	4.12	28.09	4.12	26.04	2048
SEResNet152 (2018)(230)	11.59	66.82	11.59	64.77	2048
SEResNeXt50 (2018)(230, 231)	4.28	27.56	4.27	25.51	2048
SEResNeXt101 (2018)(230, 231)	8.04	48.96	8.04	46.91	2048
ConvNeXt-nano (2022)(67)	2.45	15.59	2.45	14.95	640
ConvNeXt-T (2022)(67)	4.46	28.59	4.46	27.82	768
ConvNeXt-B (2022)(67)	15.36	88.59	15.36	87.56	1024
ConvNeXt-L (2022)(67)	34.37	197.77	34.37	196.23	1536

Source: By the author.

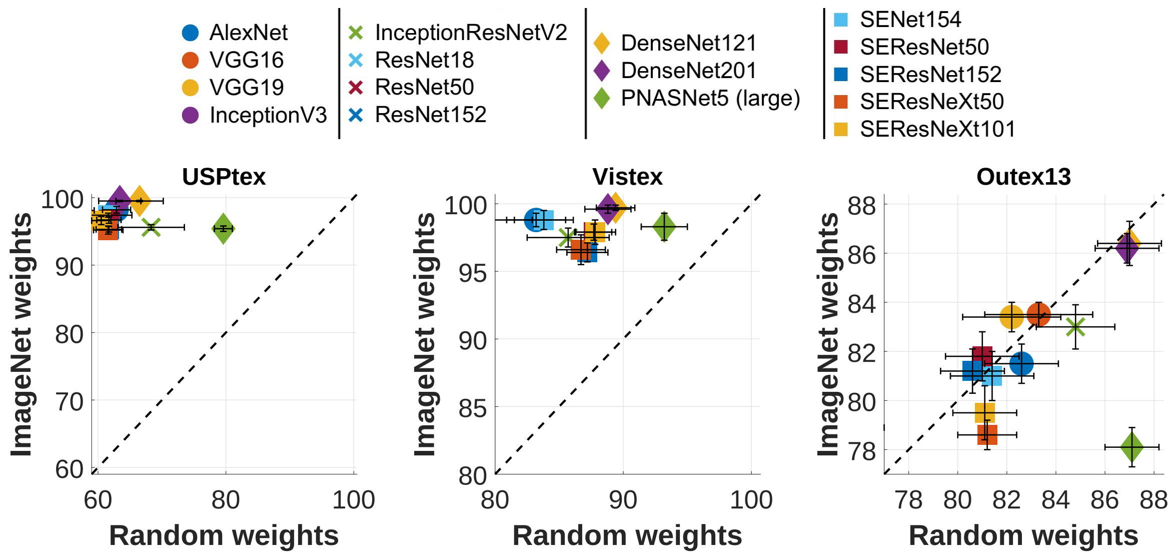
Net5, the Squeeze-and-Excitation networks, and ConvNeXts, the PyTorch default initialization policy (81) is used, that is: a uniform distribution bounded between $[-\sqrt{\frac{1}{n_l}}, \sqrt{\frac{1}{n_l}}]$ (n_l is the number of input neurons of a layer). On the other hand, in the case of the ResNets, the weights of the convolutional layers are initialized as suggested by the authors by using the Kaiming normal method, (79) while the batch normalization weights and biases are respectively initialized with 1s and 0s. In all cases, we generate 10 random networks and measure the performance by the average and standard deviation of the results obtained by them.

Figure 30 shows the results obtained with three classifiers (KNN, SVM, and LDA) using features obtained from DCNNs with either ImageNet pre-trained weights or the aforementioned random initialization, for three texture recognition datasets (USPtex, Vistex, and Outex13). We can verify the contribution of the learned weights over the random ones by the distance of the points above the dotted line, which represents $x = y$ (where random weight equals the accuracy of the ImageNet weights). For better visualization, we focus on the best networks so some are not shown (their performance is below the inferior limit of the axis). To our surprise, fully randomized DCNNs may surpass the performance of their pre-trained counterparts. Although the pre-trained weights provide better features in the majority of cases, some networks are better when random for the Outex13 task, and these results are consistent among classifiers. The best results on this dataset are achieved with the random DenseNets and PNASNet using the SVM classifier, surpassing any pre-trained network. More surprisingly, the standard deviation of the random versions indicates that different initializations have a relatively small impact on the features, and this variance can be equivalent to the variance of pre-trained models (in that case, the variance is the standard deviation of ten repetitions of 10-fold cross-validation). These results raise questions concerning the applicability of the DCNNs pre-trained on ImageNet for texture characterization.

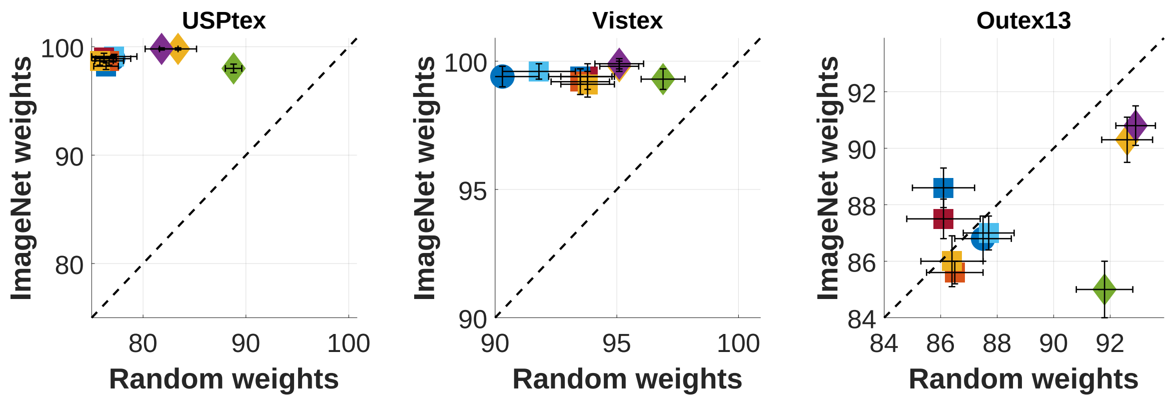
Regarding the variation among datasets, we observe similar behavior in USPtex and Vistex, where the learned weights provide a small but relevant improvement. Nonetheless, many networks with random weights yield a classification accuracy above 80% on USPtex, and above 90% on Vistex. It is also possible to notice that some networks have a higher random weight performance than the other networks in most cases, but on the other hand, are surpassed when using the ImageNet weights. This happens with the PNASNet5 and InceptionResNetV2 networks. Concerning the best networks in general, we can notice the DenseNet model, with both depths we tested (121 and 201 layers), performing above the majority of the networks o most of the tasks. In conclusion, these results point that the architecture alone may be responsible for most of the network performance in some cases, indicating that, for texture analysis, the current models need either modification of their architecture and/or new pre-training procedures other ImageNet, for the learning of better texture representations. Here we propose the use of PARw and RADAM for improving feature extraction with their fully random versions, which we evaluate in the following.

4.4.2 Improving fully randomized DCNNs using PARw and RADAM

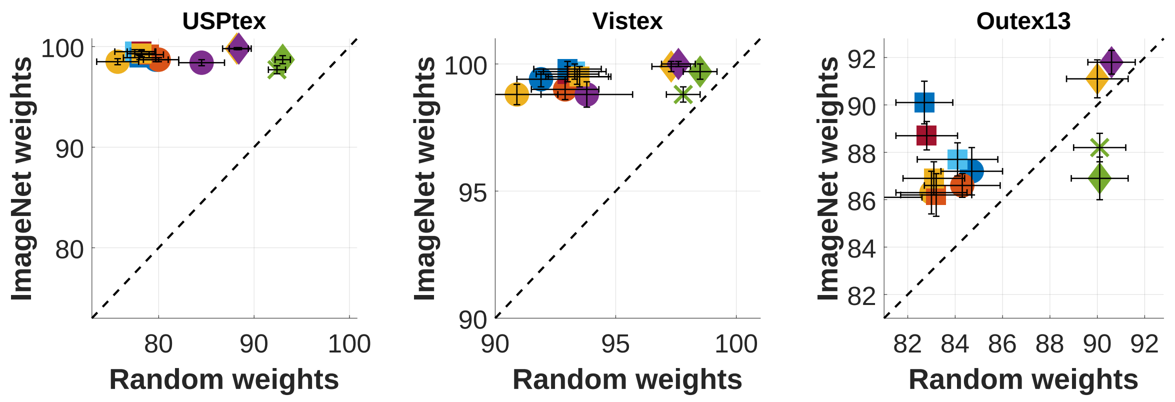
We selected the networks DenseNet201, PNASNet5, ResNet50 (gold-standard CNN), and ConvNeXt (a more recent architecture), for further evaluation with our proposed methods PARw and RADAM. Our methods are applied exactly as they were described before but in a different scenario. PARw is used for improving randomly initialized networks which are then used for feature extraction, rather than for training (as previously proposed). As for RADAM, the only difference is that the method is applied over random backbones rather than pre-trained ones, *i.e.*, it aggregates and encodes random features. In this sense, both methods can also be applied together by first rewiring the weights of



(a) KNN classification accuracy (%).



(b) SVM classification accuracy (%).



(c) LDA classification accuracy (%).

Figure 30 – Correlation between the accuracy using DCNNs with random weights or pre-trained on ImageNet. The dotted line represents the diagonal of the plan, *i.e.* $x = y$. The small black lines represent the standard deviation over 10 random networks (horizontal), or over 10-fold cross-validation (vertical).

Source: SCABINI *et al.* (225)

the backbone with PARw, and then applying RADAM for feature extraction. We name this approach as improved fully randomized DCNNs (FR-DCNN).

Table 17 shows the obtained results using LDA (the best classifier in the previous experiment) on different types of random features across three texture recognition datasets (USPtex, Vistex, and Outex13). To standardize the initialization policy, from now on we use Pytorch's default method (81) for all networks. We measure the average classification accuracy of the classifiers considering 10 random initializations of the backbones. Additionally, for the USPtex and Vistex datasets, we perform five repetitions of 10-fold cross-validation, yielding 500 training/test repetitions for each network. For Outex13, we consider the single training/test split of the dataset, yielding 10 training/test repetitions (one with each random network). The standard deviations are calculated considering the different random initializations.

Table 17 – Classification accuracy of LDA when using features obtained with a variety of DCNNs, either using ImageNet pre-trained weights, or random initialization that can also be coupled with RADAM and PARw. We highlight with bold type the best results for each architecture and dataset.

	method	USPtex	Vistex	Outex13
ResNet50	ImageNet - GAP	**99.0**±0.1	99.4±0.1	87.8
	Random - GAP	82.4±2.6	94.9±2.1	86.1±1.2
	Random - GAP + PARw	85.8±2.3	95.3±2.1	87.0±0.7
	Random - RADAM	97.9±0.8	99.8±0.4	93.9±0.7
	Random - RADAM + PARw	98.9±0.6	**99.9**±0.4	**95.9**±0.4
DenseNet201	ImageNet - GAP	**99.8**±0.0	**100**±0.0	94.1
	Random - GAP	89.3±1.8	98.1±1.5	90.4±0.7
	Random - GAP + PARw	94.7±1.5	99.3±0.9	94.1±0.4
	Random - RADAM	88.9±2.7	97.1±1.7	90.3±1.1
	Random - RADAM + PARw	97.5±0.9	99.5±0.6	**94.4**±0.7
PNASNet5	ImageNet - GAP	90.0±0.3	96.6±0.3	75.6
	Random - GAP	81.1±3.1	97.3±1.8	88.4±0.9
	Random - GAP + PARw	92.2±1.5	98.9±1.1	91.2±0.7
	Random - RADAM	95.5±1.3	98.0±1.3	90.2±0.4
	Random - RADAM + PARw	**97.3**±0.9	**99.7**±0.5	**93.4**±0.8
ConvNeXt-B	ImageNet - GAP	**99.2**±0.1	**99.9**±0.0	**90.3**
	Random - GAP	32.0±2.3	47.7±4.5	39.8±0.4
	Random - GAP + PARw	33.1±2.3	49.5±4.8	40.5±0.3
	Random - RADAM	84.1±2.0	94.8±2.5	88.8±0.6
	Random - RADAM + PARw	85.9±1.9	95.4±2.3	**90.3**±0.3

Source: By the author.

Firstly, using random weights with the default GAP is the worst approach for feature extraction in these texture recognition tasks, especially if we consider the ConvNeXt-B backbone. However, the true potential of the random models appears when applying

PARw and RADAM. PARw consistently improves the performance in all cases, either considering GAP or RADAM features. This corroborates the improved Network Science-based structure of the models achieved with PARw also when using them for feature extraction. This finding also aligns with the results observed with the networks after a single training epoch (see Section 3.2), corroborating a better initial parameter space. On the other hand, RADAM also improves performance considerably in all cases. For instance, considering the ConvNeXt-B backbone, RADAM improves an initially poor model (GAP) up to a competitive performance, even matching the ImageNet pre-trained version on the Outex13 dataset. Moreover, applying PARw and RADAM often reduces the performance variance for the random DCNNs.

In summary, the combination of RADAM and PARw (FR-DCNN) provides considerably better random features, consistently delivering the best and more stable performance in most cases, and often surpassing the performance of ImageNet pre-training. The performance may surpass the SOTA on some tasks, as we additionally show in Chapter 5. These results suggest that the synergy of RADAM and PARw may provide a viable alternative to standard pre-training approaches, especially in situations where pre-trained models are not available or transfer learning is not feasible. For instance, later on (Chapter 6), we also employ FR-DCNN for practical applications using microscopy images and show that it can be competitive with other hand-engineered and DCNN-based feature extraction approaches.

5 A SYSTEMATIC LITERATURE EVALUATION AND COMPARISON TO THE PROPOSED TEXTURE ANALYSIS METHODS

In this chapter, we show several experiments on public benchmarks performed with various texture recognition and deep learning methods from the literature and their comparison to our proposed methods. As previously discussed, nowadays we can find two parallel approaches to texture analysis in the literature: hand-engineered methods, which focus on pure-texture datasets, and deep learning-based which consider more “in-the-wild” datasets (images from the internet, objects, with backgrounds, etc). The main issue of this division is that the methods of each approach are rarely compared together, and each line uses different datasets for comparison. To address this issue, we performed a variety of experiments with various methods from the literature, selected the best candidates, and then compared them in a similar framework. However, it is important to notice that it is virtually impossible to unify all these methods and results due to several factors such as the lack of available code, omission of certain details in the evaluation protocol, etc. We tried our best to conciliate available results, run new experiments with a variety of methods, and perform a fair comparison.

5.1 Evaluating classical texture analysis methods

The results in this chapter are organized into different sections, starting with a comparison of hand-engineered features, then a comparison between deep learning networks, followed by a comprehensive comparison between the best methods, results from the literature, and the proposed methods. Throughout the following sections, most of the presented results were obtained by the author, and we indicate results with a † symbol when otherwise (when citing results presented in another work). The author is grateful to researchers from the Science for Life Laboratory (Stockholm, Sweden), from the research group led by Prof. Dr. Kevin Smith, for the discussions and suggestions received during the author’s visit to their group. Their feedback and assistance were important for performing the evaluations presented in this chapter.

Our first evaluation of literature methods considers a variety of hand-engineered texture characterization approaches. They are selected considering the different paradigms of texture characterization that emerged in the literature throughout the years. These methods are: Local Binary Patterns (LBP), (118) LBP-Variance, (118) Complete LBP (CLBP), (124) Gray Level Co-occurrence Matrix (GLCM), (117) Gray Level Difference Method (GLDM), (232) Local Oriented Statistics Booster (LOSIB), (233) Segmentation-based Fractal Texture Analysis (SFTA), (234) Fourier descriptors, (119,235) Gabor filters, (120) Fractal descriptors, (236) Binary Gabor Patterns (BGP), (237) and Binarized Statistical Image Features (BSIF). (238)

It is important to note that these methods are proposed for grayscale images, which is the predominant approach of hand-engineered texture recognition methods. In this sense, we convert the images to grayscale for this experiment. We consider three supervised classifiers (KNN, SVM, and LDA) for a more robust analysis. Three texture benchmarks (USPtex, Vistex, and Outex13) are used, and the 10-fold cross-validation procedure is performed to estimate the average classification accuracy of the classifiers (the standard deviation is given). In summary, these texture recognition datasets consist of pure-texture images, including materials and other daily-life objects, and are divided into various different categories. We refer the reader to Sections A.1 and B.1 for all details concerning the texture benchmarks, classifiers, and the evaluation protocol. The obtained results are shown in Table 5.1, where each row corresponds to a method, and each column is a different combination of a dataset and a classifier.

In summary, the CLBP method consistently provides the highest classification accuracy across all three datasets and classifiers. This indicates that the CLBP method is highly effective for texture analysis tasks in these particular cases. Other methods such as BGP and BSIF show competitive performance but do not reach the same level of accuracy as the CLBP method. As for the classifiers, we observe that although SVM occasionally achieves a competitive or slightly better performance compared to LDA for some texture analysis methods, LDA consistently demonstrates higher classification accuracies in most cases. Given these observations, we can conclude that LDA performed better compared to KNN and SVM, but SVM can also show competitive performance. It is important to notice that the focus of our research is on feature extraction rather than the classifier. This is the reason we use them with a single and simple configuration, keeping a standardized scenario for evaluation and a fair comparison to the literature. However, their hyperparameters may impact performance considerably for each specific task and feature extraction method. In this sense, tuning their configuration for each use case may yield better performance than what we report, and would be a better approach to preparing the model for a final application.

5.2 Evaluating deep convolutional networks for texture analysis

As stated before, most texture analysis methods are monochromatic, *i.e.*, they consider only one image channel, or the image luminance (grayscale). The color information is usually lost during a grayscale conversion or processed separately with non-spatial approaches such as color statistics, histograms, or with integrative methods (grayscale methods applied individually to each color channel). However, color images are present in the vast majority of imaging devices, and current CV techniques, such as deep learning-based, consider color images as the standard input type. Some studies have been carried out to assess the benefits of adding color for texture analysis (127, 128, 239) but they are

Table 18 – Analysis of hand-engineered texture characterization methods in their original grayscale version.

	USPtex			Vistex			Outex13		
method	KNN	SVM	LDA	KNN	SVM	LDA	KNN	SVM	LDA
LBP	73.7±2.5	84.2±1.7	82.2±1.4	94.2±2.7	97.5±1.6	97.2±1.8	72.5±2.5	**82.5**±1.8	80.8±2.3
LBPV	73.3±3.6	43.3±3.5	72.0±2.4	92.8±3.1	68.3±3.2	86.3±3.1	75.6±4.2	58.9±4.3	74.3±2.9
CLBP	**83.2**±2.1	86.0±1.6	**91.7**±1.5	**99.0**±1.1	**99.4**±1.1	**99.3**±1.5	**75.8**±2.9	81.0±2.8	**83.7**±2.9
GLCM	63.8±4.2	32.0±1.1	76.3±2.3	87.4±3.6	61.3±3.3	92.0±2.3	72.7±5.2	54.4±3.1	80.4±2.0
GLDM	71.9±1.9	28.7±3.1	87.0±1.3	88.9±2.8	44.3±3.0	94.8±1.7	74.0±3.7	60.2±3.1	83.3±2.5
LOSIB	43.0±2.3	15.3±1.5	65.5±2.0	70.7±4.0	32.9±2.9	81.8±3.0	53.2±4.8	29.6±3.7	72.3±2.7
SFTA	54.8±3.4	33.7±2.0	66.6±3.4	83.5±2.9	66.8±3.7	85.7±4.5	61.2±3.4	47.9±3.1	69.0±3.5
Fourier	56.9±3.2	50.7±2.1	66.1±3.1	80.4±3.6	79.9±2.7	83.6±3.0	68.8±2.8	70.0±3.1	73.4±2.3
Gabor	70.6±2.7	58.9±2.1	77.6±2.4	88.4±1.9	86.2±3.1	89.6±3.8	72.1±2.1	72.4±3.5	77.3±2.1
Fractal	29.0±3.1	10.2±1.2	68.6±2.8	52.3±5.8	29.8±4.2	85.5±5.2	48.5±3.9	23.2±3.9	66.8±4.4
BGP	75.0±1.6	**86.1**±2.1	88.2±1.7	93.9±3.4	97.8±2.4	97.8±0.9	70.5±2.5	80.8±3.0	82.6±2.5
BSIF	71.8±2.8	82.8±2.8	77.2±1.3	87.8±2.3	94.1±1.6	88.5±4.2	72.6±3.1	81.3±4.0	78.7±2.1

Source: By the author.

usually limited to the aforementioned techniques, *i.e.*, lacking a better approach to tackle the spatial patterns emerging also from the spectral variations in different color channels.

On the other hand, a typical CNN has the ability to address the relation between all the color channels since its filters connect to all the spectral information of a pixel. In this sense, we analyze a variety of DCNNs when employed for texture analysis as feature extractors for RGB images. In this sense, a DCNN backbone is only used to obtain an image descriptor (216) (a.k.a "off-the-shelf" features) which is then fed into a dedicated classifier. Results have shown (141, 142, 240) that this approach is promising for texture analysis tasks. However, these works usually focus on "in-the-wild" image datasets, where pre-trained DCNNs are known to work better since they are trained for object detection and recognition, while little is known regarding their capacity on pure-texture images. Our experiments in this section seek to close this gap and offer a more in-depth understanding of DCNNs' features for texture analysis.

We consider the same 20 DCNNs we explored in Section 4.4 (see Table 16), and they are pre-trained on the large-scale object recognition dataset ImageNet-1k (54) (see Section B.2 about the ImageNet dataset). The LDA classifier is used since it provided the highest average performance in the previous experiment, and trained over features obtained with GAP from the last convolutional layer of the DCNNs. Table 19 shows the results obtained with features from all networks, including two additional texture recognition datasets (CUReT and MBT) not present in the previous experiment. We consider these pure-texture datasets to be able to evaluate only the ability of the models to characterize texture alone, which does not involve other skills such as detecting objects, ignoring background, etc. As a baseline, we also show the CLBP method applied in an integrative fashion (features from each RGB channel are concatenated). Regarding the best networks, we can see that the DenseNet model surpasses the other networks in most cases, and ConvNeXt achieves equivalent performance. ConvNeXt provides more stable features, causing the classification standard deviations to be below 10^{-1} in the majority of the cases. On the other hand, all DCNNs are surpassed by the integrative CLBP in the MBT dataset. This highlights the fact that although DCNNs achieve outstanding performance in general, they still may be surpassed by hand-engineered methods in some texture analysis tasks.

Table 19 – Classification accuracy (%) obtained by the LDA classifier using the features obtained by ImageNet-pre-trained DCNNs on different texture recognition datasets, with the standard deviation corresponding to ten repetitions of 10-fold cross-validation. The color shading represents the differences in the results of each dataset, with lower results painted with more color, and the bold type represents the highest results.

feature extractor	USPtex	Vistex	Outex13	CUReT	MBT
CLBP(integrative)	97.2±0.3	99.4±0.1	89.5±0.3	91.7±0.2	**98.2±0.1**
Alexnet	98.7±0.2	99.4±0.3	87.2±1.0	90.5±0.7	88.2±0.9
VGG16	98.7±0.2	99.0±0.4	86.6±0.5	93.6±0.5	93.4±0.7
VGG19	98.5±0.3	98.8±0.4	86.3±0.9	93.2±0.6	92.6±0.7
InceptionV3	98.4±0.3	98.8±0.5	86.1±0.8	97.4±0.2	92.2±0.5
InceptionResNetV2	97.7±0.4	98.8±0.3	88.2±0.6	96.4±0.5	91.2±0.7
ResNet18	98.8±0.3	99.3±0.3	86.9±0.7	95.7±0.4	91.3±0.8
ResNet50	99.4±0.1	99.7±0.3	90.5±0.5	98.6±0.2	94.1±0.8
ResNet152	99.6±0.2	99.5±0.3	89.4±0.8	98.9±0.2	94.8±0.6
DenseNet121	**99.8**±0.1	99.9±0.2	91.1±0.8	98.9±0.2	97.4±0.3
DenseNet201	**99.8**±0.1	**100**±0.1	**91.8**±0.5	**99.4**±0.2	97.6±0.3
PNASNet5(large)	98.7±0.4	99.7±0.3	86.9±0.9	98.8±0.3	93.5±1.0
SENet154	99.5±0.2	99.7±0.3	87.7±0.7	98.1±0.3	93.4±0.4
SEResNet50	99.5±0.2	99.6±0.3	88.7±0.6	97.8±0.2	93.2±0.7
SEResNet152	98.9±0.2	99.8±0.3	90.1±0.9	96.9±0.3	92.1±0.4
SEResNeXt50	99.2±0.3	99.5±0.3	86.2±0.9	97.4±0.3	92.0±0.8
SEResNeXt101	99.3±0.3	99.5±0.4	86.9±0.7	96.9±0.3	91.9±0.5
ConvNeXt-nano	99.6±0.0	99.8±0.1	90.9±0.0	98.0±0.0	97.3±0.1
ConvNeXt-T	99.7±0.0	99.9±0.0	**91.8**±0.0	99.2±0.0	97.0±0.1
ConvNeXt-B	99.2±0.1	99.9±0.0	90.3±0.0	99.3±0.0	97.5±0.1
ConvNeXt-L	99.7±0.0	**100**±0.0	90.7±0.0	99.3±0.0	97.9±0.1

Source: By the author.

5.2.1 Efficiency comparison

For a final comparison, we considered the efficiency of the DCNNs, defined here as the correlation between complexity (GFLOPs, parameters, and descriptor size) and performance (average accuracy for the five datasets). The intuition is to estimate the cost of computing an image descriptor and the dimensionality of this descriptor (which impact the cost to train a classifier). Figure 31 shows this analysis, where network size refers to the feature extraction part only. We included CLBP as a baseline for comparison, which

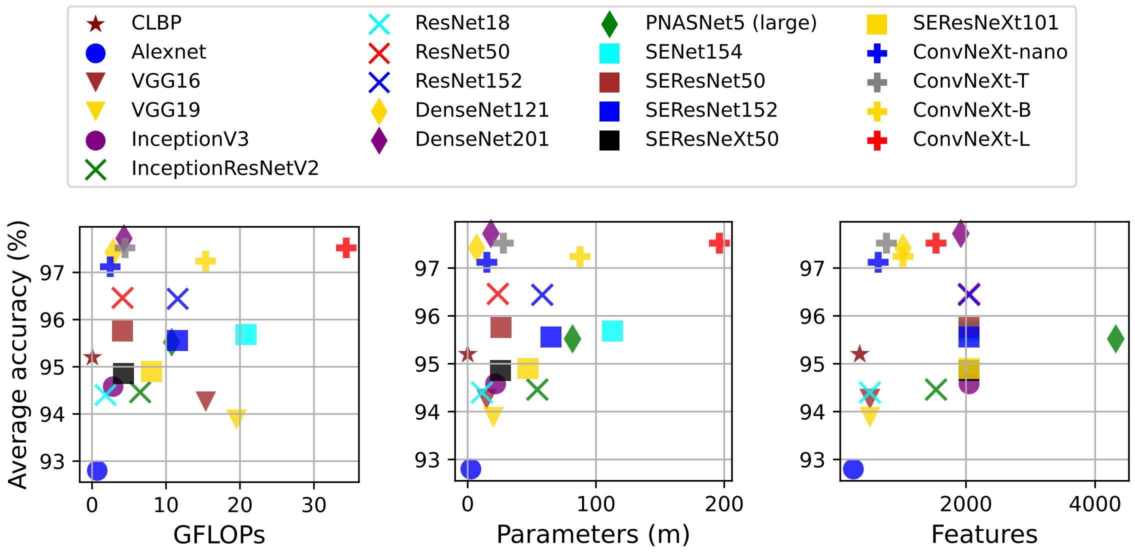


Figure 31 – The relation between the mean accuracy (LDA) attained at texture characterization and the complexity (or cost) of methods, measured by the number of floating point operations (GFLOPs), parameters (in millions), and the size of the image descriptor they output.

Source: By the author.

uses zero parameters, since it is computed on-the-fly and does not need to be stored, and outputs 354 descriptors. We avoided performing running-time experiments since it is difficult to perform a fair comparison between different types of methods. This happens because of several reasons, such as: a) the methods are implemented using a variety of programming languages and frameworks; b) hand-engineered methods do not often have an available parallel or GPU version for a fair comparison to ANN-based methods; c) the experiments performed throughout this thesis happened in a five-year window, and many things changed during this time, such as the available hardware for experimentation, library versions, and even the emergence of new models. Therefore, our discussion is based on the theoretical computational complexity of the methods, which does not depend on technical details or code optimization.

In general, there is a low positive correlation between the increase in complexity and performance among the methods. However, this is not a general rule and some methods may face a reverse effect (VGG, for instance). The best methods in terms of average performance and efficiency are the DenseNets and ConvNeXts. ResNets may also be potential candidates in some scenarios. These networks are among the only ones surpassing CLBP in terms of average accuracy, but their difference in complexity is still considerable since CLBP is a hand-engineered method with GFLOPs below 0.05, and without stored parameters, while also using fewer descriptors. In the following SOTA comparisons, we consider the DenseNets and ConvNeXts, which demonstrated the highest performance on texture characterization with pre-trained weights, and the ResNet50 which is a gold standard among CNNs and demonstrated interesting performance and efficiency.

5.3 Comparing the proposed methods within the hand-engineered texture recognition framework

We start the comparison between various texture characterization methods by considering the collection of five pure-texture datasets used in the previous section, composed of images from materials and texture categories. These datasets follow the hand-engineered texture recognition evaluation framework, employed by the majority of the works on hand-engineered methods. Later on (Section 5.4), we consider datasets with other properties, following the "in-the-wild" approach, which is more common in the deep-learning literature and where we compare to other kinds of methods. As previously mentioned, we adopt this approach to be able to compare the proposed methods with a wider range of methods from the literature and in more different scenarios.

Considering the previous results, we compare the best methods from the literature, the proposed methods, and other methods from which we cite the results (†) from the corresponding papers. For coherence in the comparisons, we considered only results that use descriptors computed from the RGB color space and without color normalization. To follow a validation procedure more similar to some previous works, here we employ leave-one-out cross-validation. The LDA classifier is used also considering previous works, and also considering that it provided the best general performance in our experiments.

Among the additional hand-engineered methods compared here (aside from CLBP), we consider the pioneering work of Complex Network-based texture analysis (CNTD) (16) (described in Section 2.3.3.1), which is applied in an integrative way for addressing the color information. The literature methods we cite results from are: Parametric spectral analysis (PSA), (241) fractal measures of color channels, (111) shortest paths on graphs, (17) fractal measure of the mutual interference of color channels, (130) Multilayer Complex Network Descriptors (MCND) (19) (an improved CN-based method for color images), and RNN-based color texture characterization. (153) We also mention the best results achieved by the reviews of Mäenpää and Pietikäinen (127) and Cernadas *et al.*. (128) It is important to note that some of these works (127, 128, 241) do not consider leave-one-out cross-validation, so the comparison should be taken cautiously.

As for the deep learning-based methods, we consider the best DCNNs according to the previous experiments. Additionally, we also include Vision Transformers (ViT), (64,65) a more recent deep-learning architecture that has been dominating many CV tasks since 2021. These models consist of employing the transformer architecture (66) which is well known for natural language processing and is on the rise due to the power of large language models. In images, the transformer is applied over a sequence of patches (small windows) obtained by dividing the pixels into smaller matrices, that are then flattened into one-dimensional vectors. Some variants of this approach have emerged recently in the literature. Here, we consider the original ViT-B/16 (64) (the base version with patch size 16), and Data-Efficient Image Transformer (DeiT3) (65) which uses the same architecture (ViT-B/16) but with an improved training pipeline. ViT-B/16 is pre-trained on ImageNet-21k (54) (the complete ImageNet). For DeiT3, we consider two variants: one that is pre-trained on ImageNet-1k only, and another one pre-trained on ImageNet-21k then fine-tuned using ImageNet-1k (IN-21K - FT1k). Both models and their corresponding pre-trained weights are available in the PyTorch Image Models library. (218) We use them as feature extractors by simply removing the classification layer and using the previous 1-dimensional representation (the *class* token), which is used to train the LDA classifier. Finally, all the obtained results are shown in Table 20.

First of all, it is important to notice the differences between hand-engineered and deep-learning-based features. Their comparison involves considering both their performance and efficiency. Deep NN features generally outperform hand-engineered features in terms of average accuracy, thanks to the hierarchical visual knowledge of the backbone pre-trained on large-scale image datasets. However, hand-engineered features can still provide competitive results in certain situations, as our results demonstrate. In terms of efficiency, hand-engineered features are less complex, requiring fewer FLOPs and having lower memory demands. This leads to faster running times, making them more suitable for real-time applications or situations with limited computational resources. On the other hand, deep NN-based methods have higher computational complexity but they can benefit from powerful GPUs and parallel processing, if available. In conclusion, the choice between hand-engineered features and deep NN-based methods depends on the trade-offs between accuracy and efficiency, considering the specific application requirements and available resources.

In contrast, the proposed methods generally outperform the two approaches from the literature, and we give special attention to RADAM with DenseNet121 which surpasses all methods from the literature on all datasets. With ConvNeXt-L, RADAM also often reaches the highest performance among all methods. For the USPtex dataset, all RADAM variations and SSN achieve very high accuracy rates, and we highlight RADAM with ConNeXt-nano for being the only method achieving a perfect classification accuracy (100%). The SSR methods achieve slightly lower accuracy rates of 99.0% to 99.3%. The DenseNet201 achieves a 99.8% accuracy rate, which is competitive with the proposed methods. For the Vistex dataset, all proposed methods (except with random ResNet50) reach a perfect accuracy rate of 100%. Among the competitors, several deep NN methods also achieve a 100% accuracy rate, specifically DenseNet201, ConvNeXt-T IN-21K, ConvNeXt-L, and ViT-B/16 IN-21K.

Table 20 – The classification accuracy rate of classifiers using features extracted with different methods on five texture datasets. The † symbol represents results from the cited paper, and empty cells indicate that the result for the corresponding paper is not available. In our experiments, we use LDA and leave-one-out cross-validation, and the mean accuracy and standard deviation are only computed for the random ResNet50 since 10 random models are used. The blue color represents the previous SOTA on the corresponding dataset, the red color represents our results matching or above that, and in **bold type** is the highest result in each column.

	Method	USPtex	Vistex	Outex13	CUReT	MBT
Hand-engineered features	(review of methods) (127) †		99.5	94.6		
	(review of methods) (128) †	96.8	97.3	90.4	95.5	
	PSA (241) †		99.5	94.4		
	Fractal-color (111) †	96.6	99.1			
	Shortest paths on graphs (17) †	96.9	99.1	91.5		
	Fractal-color interference (130) †	97.0	99.3	95.0		
	RNN-color texture (153) †	98.4	98.8	94.8		
	MCND (19) †	99.0	99.9	95.4	97.0	97.1
	CLBP	97.4	99.5	89.6	91.8	98.2
	CNTD	97.9	99.7	92.3	91.9	98.5
Deep NN features	ResNet50	99.4	99.7	90.5	98.6	94.1
	DenseNet121	99.5	99.8	89.6	96.8	94.9
	DenseNet201	99.8	**100**	94.1	99.4	98.4
	ConvNeXt-nano	99.6	99.8	90.9	98.0	97.3
	ConvNeXt-T	99.7	99.9	91.8	99.2	97.0
	ConvNeXt-T IN-21K	99.8	**100**	90.9	99.2	98.3
	ConvNeXt-L	99.7	**100**	90.7	99.3	97.9
	ViT-B/16 IN-21K	99.5	**100**	94.9	97.9	97.7
	DeiT3-B/16	99.3	99.8	89.9	99.4	97.4
	DeiT3-B/16 IN-21K-FT1K	98.6	99.6	91.2	99.0	95.5
Proposed methods	SSN_a	99.7	**100**	96.6	98.9	98.6
	SSN_b	99.8	**100**	95.2	99.1	98.3
	SSR_a	99.3	**100**	96.7	99.3	98.2
	SSR_b	99.0	**100**	**96.8**	99.6	98.0
	RADAM (ResNet50)	99.9	**100**	93.8	99.8	98.9
	RADAM (DenseNet121)	99.9	**100**	95.6	99.8	99.1
	RADAM (ConvNeXt-nano)	**100**	**100**	93.5	99.6	99.3
	RADAM (ConvNeXt-L)	99.9	**100**	95.0	**99.9**	**99.4**
	FR-DCNN (rand. ResNet50)	98.9±0.6	99.9±0.4	95.9±0.4	96.6±0.8	94.9±1.2

Source: By the author.

According to the results, we can consider that the texture recognition tasks represented by USPtex and Vistex are completely solved, so it can be hard to compare between some models. Now, we consider the remaining datasets which are more challenging. On the Outex13 dataset, SSR has the highest accuracy rate of 96.8%, with SSN closely behind at 96.6%. RADAM with pre-trained DenseNet121 or using PARw with the random ResNet50

(FR-DCNN) also surpasses the previous SOTA (MCND with 95.4%). These results represent the power of our approach of better hand-engineered and NN-based methods and their combination, in contrast to the literature methods. We simultaneously achieve SOTA with our purely hand-engineered method (SSN), with RADAM which considerably improves the performance of the DCNN-based features, while the highest performance is achieved by the combination of the approaches (SSR).

For the CUReT dataset, RADAM (ConvNeXt-L) has the highest accuracy rate of 99.9% among all methods. Other RADAM variations have accuracy rates ranging from 99.6% to 99.8%. SSR has a 99.6% accuracy rate, which also outperforms the SOTA (DenseNet201 and DeiT3-B/16 with 99.4%). Finally, for the MBT dataset, CNTD has the highest accuracy rate among the literature methods (98.5%), surpassing all deep-learning-based methods, but the proposed methods generally outperform CNTD. RADAM (ConvNeXt-L) has the highest accuracy rate again among all methods, at 99.4%, with other RADAM variations having accuracy rates ranging from 98.9% to 99.3%, and SSN reaching 98.6%. SSR has a slightly lower accuracy rate compared to CNTD.

In summary, the proposed SSN, SSR, and RADAM methods show superior performance compared to the competitors across the five texture datasets. The proposed methods achieve perfect or near-perfect accuracy rates in several cases and consistently outperform the competitors, especially on the Outex13, CUReT, and MBT datasets which represent the hardest texture recognition tasks among the considered cases. Moreover, FR-DCNN presents a surprisingly consistent performance across tasks and even surpasses the SOTA (and RADAM with pre-trained weights) on Outex13.

5.3.1 Conclusion

Our methods have considerably pushed the SOTA in pure-texture recognition (following the more "classical" benchmark datasets). These results branch out between our novel hand-engineered and deep-learning-based methods, offering powerful performance at different computational budgets and CV paradigms. When discussing the performance and efficiency of the proposed methods, it is essential to consider that each method belongs to different categories, with SSN being a hand-engineered method, RADAM being deep-learning based, and SSR combining hand-engineered texture modeling with small neural networks. FR-DCNN also falls into the deep-learning paradigm in terms of computational cost, but there is no large-scale training involved as we only use the static randomly initialized ResNet50 architecture.

In terms of performance, our methods generally outperform both hand-engineered and deep NN features from the literature on the five pure-texture texture datasets we tested, showcasing the advantages of each approach. SSN, as a hand-engineered method, leverages explainable modeling and metrics, while RADAM, a deep-learning-based method, benefits from the knowledge of DCNN backbones. SSR strikes a balance between the two paradigms, combining hand-engineered texture modeling with small neural networks to improve the characterization. On the other hand, FR-DCNN also offers a powerful and general feature extraction that can even surpass the use of large-scale pre-training.

Regarding efficiency, SSN is expected to be better compared to RADAM and SSR due to its hand-engineered nature, which typically involves fewer FLOPs and lower memory demands. RADAM, needing a DCNN backbone to operate, has higher computational complexity, but it can benefit from optimizations such as GPU acceleration and parallel processing. SSR, being a hybrid method, aims to find a compromise between the two extremes, combining the efficiency of hand-engineered image modeling with the feature learning of ANNs. This allows SSR to achieve competitive performance while maintaining a more manageable computational cost. However, these behaviors may slightly vary between tasks as we observed between the different benchmark datasets. Nevertheless, compared to other literature approaches, the proposed methods are expected to demonstrate superior performance at pure-texture characterization and a desirable efficiency.

5.4 Comparing the proposed methods within the deep-learning-based texture recognition framework

In this section, we focus on a different evaluation framework that is predominant in deep-learning-based texture recognition works. The main difference is the type of datasets employed for comparison, where the focus change to “in-the-wild” images. These datasets are DTD, FMD, and GTOS (also its mobile version). Nevertheless, KTH-TIPS2-b (pure-texture) is also often employed in these works considering that the images have considerable variability. The cross-validation protocol follows the original splits given for each dataset: 10 predefined splits for DTD, 10-fold random cross-validation for FMD, 4 pre-defined splits for KTH-TIPS2-b, 5 pre-defined splits for GTOS, and a single training/test split for GTOS-Mobile. We evaluate the average classification accuracy and standard deviations according to these splits (no standard deviation for GTOS-Mobile, except for when we use fully randomized DCNNs). We refer the reader to Chapter B for more details about the datasets and their split policy.

The following deep-learning-based methods for texture analysis are considered for comparison: DeepTEN, (143) DEPNet, (112) MAPNet, (145) DSRNet, (146) RankGP-3M-CNN++, (141) Multilayer-FV, (142) CLASSNet, (147) and DFAEN. (148) We also consider a selection of ViTs for comparison: ViT-B/16 pre-trained on Bamboo (149) and fine-tuned on DTD, ViT-L/14 using Contrastive Language Image Pre-training (CLIP) with zero-shot classification on DTD (151), multitask large-scale ViT-L/16 (μ2Net+) (150) fine-tuned on DTD, and features from the same ViTs employed before (ViT-B/16 and DeiT3) but using the SVM classifier. In general, since the training of these deep learning models is costly and complex (many hyperparameters), most of the results are referenced from the original papers (marked with †).

The majority of methods used here for comparison follow the more recent deep-learning-based paradigm for texture recognition, which consists of building personalized feature aggregation modules around a pre-trained DCNN, then fine-tuning the whole architecture. In this sense, the classification step is integrated into the model. However, our methods that use DCNNs (PARw and RADAM) are training-free with respect to the backbone, *i.e.*, we never fine-tune the DCNNs since they are used only for feature extraction, and only a separate linear classifier is trained over these features. Therefore, our methods are significantly more efficient to be adapted and ported to different applications. Here we use SVM for our methods, considering that this dedicated classifier is employed in similar works. (114, 141) We also consider the CLBP and CNTD methods as baselines of hand-engineered features (with SVM). We show all the results in Table 21, which includes a variety of DCNN backbones for the deep-learning-based methods. The table rows are divided into blocks according to the cost of the backbones.

In general, the hand-engineered methods demonstrate a much lower performance in this scenario compared to deep-learning-based methods, except on the KTH-TIPS2-b dataset. This behavior is expected, since this is the only pure-texture dataset, while the in-the-wild datasets contain a variety of noise in the images such as background, multiple objects, etc, where large models with object recognition pre-training are known to work better. Nevertheless, the proposed methods SSN and SSR surpass the compared hand-engineered methods. For KTH-TIPS2-b, SSR overcomes or matches some of the deep learning methods that use, for instance, ResNets or EfficientNet-B5. This result is remarkable considering that the method is considerably less complex and does not need pre-training,

Generally, methods employing deeper and more complex backbones tend to achieve better performance compared to methods using simpler backbones like ResNet18. RADAM, in particular, demonstrates strong performance when used with different ConvNeXt backbones, surpassing the SOTA results on all texture datasets. When used with ConvNeXt-T in IN-21K, it even surpasses various methods with more complex backbones. It also maintains strong performance even when used with lighter backbones like ConvNeXt-nano. When scaling the backbone up, for instance using ConvNeXt-L or ConvNeXt-XL, it achieves the highest classification accuracy on the benchmarks by a considerable margin.

5.4.1 Efficiency analysis

Additional analysis is performed to understand better the balance between the classification performance and computational budget of the deep-learning-based texture recognition methods. We consider the inference costs in terms of GFLOPs and the number

Table 21 – Classification accuracy of different methods on texture benchmarks. The used backbones are separated into row blocks according to their computational budget, the input size is indicated in parentheses (224x224 is used when not stated), and the two best results in each block are highlighted in bold type. Results in blue show the previous SOTA on each dataset, and red represents our results matching or above that.

method	backbone	DTD	FMD	KTH-2-b	GTOS	GTOS-M
CLBP	-	26.4±0.7	48.0±0.0	78.3±0.0	50.5±3.2	24.8
CNTD	-	23.1±0.4	41.0±0.0	74.3±0.0	48.4±4.8	20.2
SSN$_a$ (ours)	-	30.2±0.7	45.8±0.8	66.9±4.6	64.4±3.7	**43.2**
SSN$_b$ (ours)	-	**31.3**±0.8	46.0±0.5	67.2±4.9	65.9±3.7	**43.3**
SSR$_a$ (ours)	-	30.5±1.0	**51.0**±0.0	**82.2**±0.0	**68.1**±3.9	42.7
SSR$_b$ (ours)	-	**31.7**±1.0	**49.0**±0.0	**82.9**±0.0	**68.4**±3.8	40.1
DeepTEN †	ResNet18 (352)					76.1
DEPNet †	ResNet18					82.2
MAPNet †	ResNet18	69.5±0.8	80.8±1.0	80.9±1.8	80.3±2.6	83.0
DSRNet †	ResNet18	71.2±0.7	81.3±0.8	81.8±1.6	81.0±2.1	**83.7**
CLASSNet †	ResNet18	**71.5**±0.4	**82.5**±0.7	**85.4**±1.1	**84.3**±2.2	**85.3**
RADAM (ours)	ResNet18	68.1±1.0	77.7±0.5	84.7±3.6	80.6±1.7	79.5
RADAM (ours)	ConvNeXt-nano	**74.9**±0.7	**87.1**±0.4	**89.6**±3.8	**83.7**±1.5	81.8
DeepTEN †	ResNet50 (352)	69.6	80.2±0.9	82.0±3.3	84.5±2.9	
MAPNet †	ResNet50	76.1±0.6	85.2±0.7	84.5±1.3	84.7±2.2	86.6
DSRNet †	ResNet50	**77.6**±0.6	86.0±0.8	85.9±1.3	85.3±2.0	**87.0**
CLASSNet †	ResNet50	74.0±0.5	86.2±0.9	87.7±1.3	**85.6**±2.2	85.7
DFAEN †	ResNet50	73.2	86.9	86.3		**86.9**
FR-DCNN (ours)	ResNet50 (random)	26.8±1.0	41.9±4.3	62.4±11.5	58.1±4.8	29.8±1.9
RADAM (ours)	ResNet50	75.6±1.1	85.3±0.4	88.5±3.2	81.8±1.1	81.0
RADAM (ours)	ConvNeXt-T	77.0±0.7	**88.7**±0.4	**90.7**±4.0	84.2±1.7	85.3
RADAM (ours)	ConvNeXt-T in IN-21K	**81.4**±0.7	**93.0**±0.3	**91.0**±4.9	**85.4**±1.6	86.5
DFAEN †	DenseNet161	76.1	87.6	86.6		86.9
Multilayer-FV †	EfficientNet-B5 (512)	78.9	88.7	82.9		
RankGP-3M-CNN++ †	(3 backbones)		86.2±1.4	91.1±4.5		
fine-tuning †	ViT-B/16 in Bamboo 69m	81.2				
CLIP zero-shot †	ViT-L/14 (336) in WIT 400m	83.0				
μ2Net+ †	ViT-L/16 (384) in IN-21K	82.2				
SVM	ViT-B/16 in IN-21k	74.1±0.9	85.7±0.3	86.7±6.4	82.6±1.8	80.4
SVM	DeiT3-B/16	70.5±1.1	84.8±0.3	86.9±3.1	80.9±2.0	83.9
SVM	DeiT3-B/16 in IN-21K-FT1K	73.4±1.1	86.7±0.3	89.7±3.7	82.2±1.4	82.5
RADAM (ours)	ConvNeXt-B	76.4±0.9	90.2±0.2	87.7±5.6	84.1±1.6	82.2
RADAM (ours)	ConvNeXt-B in IN-21K	82.8±0.9	94.0±0.2	**91.8**±4.1	**86.6**±1.7	87.1
RADAM (ours)	ConvNeXt-L	77.4±1.1	89.3±0.3	89.3±3.4	84.0±1.8	85.8
RADAM (ours)	ConvNeXt-L in IN-21K	**84.0**±1.0	**95.2**±0.4	91.3±4.1	85.9±1.6	**87.3**
RADAM (ours)	ConvNeXt-XL in IN-21K	**83.7**±0.9	**95.2**±0.3	**94.4**±3.8	**87.2**±1.9	**90.2**

Source: By the author.

of parameters according to the backbone used by each method since this is the most resource-demanding step of every pipeline. One important aspect here is that the input size greatly impacts the FLOP count of the methods (check input sizes in parentheses in Table 21). Most works consider 224x224 inputs (the same input size employed by RADAM), and we assume this same size when not stated by the authors. For this analysis, we are not considering the preparation of the backbone either in terms of pre-training cost or chosen dataset or the fine-tuning of the methods that do so. The results are shown in Figure 32 for each dataset and its corresponding available results. Most methods use

ResNet18 and 50, and for RADAM we consider the ConvNexT variants: nano, T, B, L, and XL (from the left to the right in the figures), and their corresponding IN-21K pertaining, when available. It is possible to notice the superiority of RADAM at different budgets, especially when using ConvNeXt-T and nano. Moreover, results also scale with backbone complexity, *i.e.*, bigger backbones and better pre-training often improve performance.

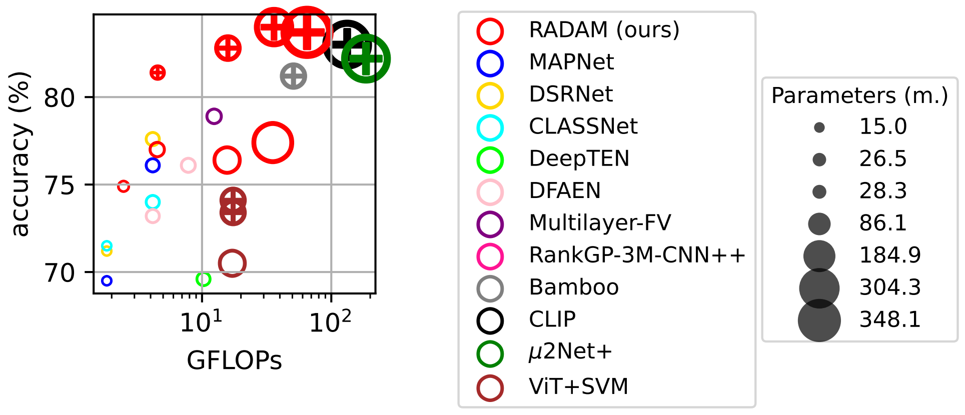
Regarding stronger backbone pre-training, *i.e.*, when datasets bigger than ImageNet-1K are used, we observe that the performance of RADAM scales accordingly. As for the literature methods, most results from methods with different pre-training (CLIP, Bamboo, and μNet+) are available only for the DTD dataset, where we observe that they achieve the current SOTA on that task compared to ImageNet-1k pre-training. On the other datasets, we can analyze the performance of ViTs with different pre-training, where a similar behavior is observed. They achieve the best results when using ImageNet-21k pre-training, except on GTOS-Mobile where ImageNet-1K pre-training performs better. Nevertheless, ImageNet-21K pre-training provides SOTA results in all datasets by a considerable margin when using ConvNeXts and RADAM. These results suggest that texture recognition can greatly benefit from better pre-training, especially when coupled with our proposed method to harness its potential.

5.5 Conclusion

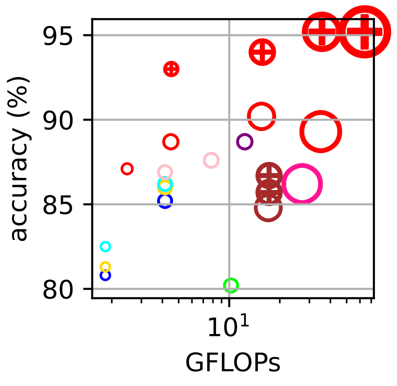
The results presented in this chapter shed new light on the performance and differences between methods and approaches to texture recognition. We have explored several hand-engineered features and deep-learning-based methods. The evaluation is performed considering ten texture benchmarks that contain images ranging between materials, terrain, and daily-life objects, collected either in controlled environments or in the wild. In this scenario, we also compare our four proposed methods against the literature approaches, and SOTA results are achieved in all datasets.

We noticed that hand-engineered features perform considerably better when dealing with pure-texture images. Deep-learning-based features work well in general, but they are surpassed by hand-engineered features in some specific pure-texture tasks. In summary, using bigger/deeper DCNN backbones for deep-learning-based methods increase their performance, but their cost also increases significantly. Nevertheless, this is currently the SOTA approach for texture recognition in uncontrolled scenarios with in-the-wild images.

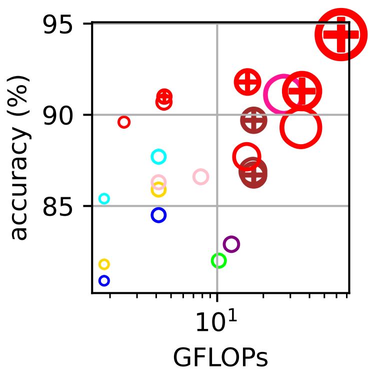
Considering the proposed methods, we developed different techniques between the two paradigms (hand-engineered or deep-learning): SSN, SSR, RADAM, and FR-DCNN. As a pure hand-engineered method, SSN achieves SOTA in pure-texture tasks and surpasses deep-learning-based methods in most cases except for in-the-wild datasets. SSR introduces the use of small neural networks and can boost the performance of SSN



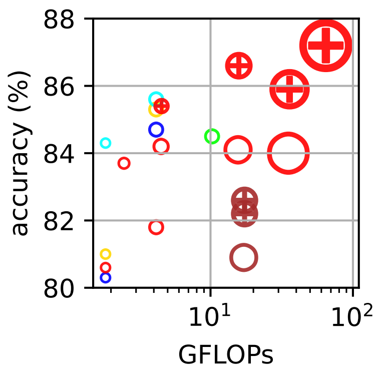
(a) DTD.



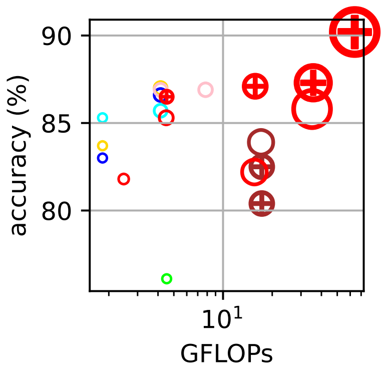
(b) FMD.



(c) KTH-TIPS2-b.



(d) GTOS.



(e) GTOS-Mobile.

Figure 32 – Computational budget (GFLOPs and the number of parameters) at inference time according to the backbone used by different methods. The symbol ⊕ represents the use of pre-training with datasets bigger than ImageNet-1k. Multiple results for the same method represent the use of different backbones. For RADAM, we consider the previously explored ConvNeXt variants.

Source: By the author.

significantly in some cases, especially for pure-texture applications. On the other hand, RADAM leverages pre-trained DCNN for obtaining robust texture descriptors without the need for fine-tuning the backbone, which saves considerable time and effort compared to other deep-learning-based methods. Using a linear dedicated classifier, it achieves SOTA results on all datasets we considered, in both the pure-texture and in-the-wild cases. The gains are considerable even compared to cutting-edge deep-learning models, such as ViTs. RADAM also demonstrates its versatility and effectiveness when combined with backbones of different sizes, even outperforming methods with more complex backbones in some cases. It can also achieve impressive pure-texture performance without even needing to pre-train the backbone when coupled with our PARw method in fully randomized DCNNs (FR-DCNN).

In conclusion, we proposed four new approaches for texture analysis that deliver SOTA results in a wide range of cases. We have shown that the Network-Science-based modeling of images is a promising approach for pure-texture tasks, as well as coupling it with RNNs. Another novel and promising finding is the significant boost in the performance of random DCNNs achieved when using PARw and RADAM (FR-DCNN), making it a viable option with SOTA results in some tasks. However, RADAM with pre-trained DCNNs proved to be the most promising method in general, achieving SOTA performance in every case we explored. The methods and results presented in Chapters 3 and 5 so far culminated in a total of three publications, (154, 204, 225) three finished articles currently submitted to international journals, (187, 212, 215) and two others currently in production. This production highlights the relevance of the proposed methods, corroborating that they are valuable alternatives to the CV community for tackling texture analysis in general.

6 APPLICATIONS

This chapter covers real-world applications where our proposed CV methods are applied. Each section corresponds to a different application, where we describe the target problem, the data, and the results of our methods also in comparison with other literature approaches.

6.1 Computer vision-assisted diagnosis based on genosensor images

The COVID-19 pandemic has reinforced the importance of low-cost, easily deployable diagnostic methodologies, where some of the main challenges are affordability, simplicity, and fast data analysis. The need for new sensing technology that can be applied across different diseases and health monitoring is particularly urgent. The ideal diagnostic tool would be one that can be used at home or in remote areas with cheap and/or common devices, reducing the need for patients to travel to healthcare facilities and thus minimizing the risk of disease transmission. Moreover, these diagnostic tools should be versatile enough to detect a wide range of diseases, not just COVID-19. This would allow for a more comprehensive approach to health monitoring, enabling the early detection and treatment of various diseases. One potential approach for this kind of diagnosis is through biosensors, compact analytical devices that detect and measures the presence of specific biological substances, such as enzymes, antibodies, or nucleic acids, by generating a measurable signal proportional to the target's concentration. A genosensor is a type of biosensor designed to detect and quantify target genetic material or sequences, such as DNA or RNA. In addition, the data generated by these diagnostic tools should be processed fastly and accurately, where AI and ML algorithms are potential candidates.

6.1.1 Prostate cancer diagnosis through computer vision

The demand for new technology for fast and automatic diagnosis extends to many medical conditions, such as prostate cancer, where early diagnosis increases the chances of cure considerably. Currently, it relies on the low-specificity prostate-specific antigen (PSA) test. Another approach for this kind of diagnosis is through biosensors, where prostate cancer gene 3 (PCA3) has emerged as a promising alternative biomarker for more accurate diagnosis using genosensors. However, despite significant progress in biosensor development, high costs and certification requirements have limited the transformation of knowledge into products. Another easy and quick diagnostic method with current technology involves capturing an image of a sensing unit exposed to a sample and processing the image. This approach differs from standard diagnostic image analysis as the picture is taken not from the biological sample but from the sensing unit. This strategy can be effective if the detection process changes the sensing unit's texture, morphology, or other visual properties. Although biosensor surfaces are known to change during measurements, utilizing these changes for diagnosis has not been extensively explored in the literature.

6.1.1.1 Proposed method

In this section, we describe a work developed in collaboration with several researchers and institutes, which introduces new genosensors for detecting a prostate cancer-specific DNA sequence (PCA3) and uses CV as an alternative tool for diagnosis. The results were published (242) in the international journal Talanta, from Elsevier. To the best of our knowledge, this was the first work to employ image analysis and ML on scanning electron microscope (SEM) images from genosensors. We focused on characterizing textural changes in these biosensors when subjected to prostate cancer PCA3. Here, we extend the published results by also analyzing the applicability of other of our texture analysis methods, and we show that the detection rate can be improved compared to the results reported in the paper. Figure 33 illustrates the general idea of this approach, composed of an image acquisition and processing pipeline, where we apply texture analysis and ML for diagnosis. The goal is to create a generic platform that can be applied to other biomarkers and diseases as well, as we also show in Section 6.1.2.

6.1.1.2 Image acquisition

For details about the genosensors, we refer the reader to the published work. (242) Here, we focus exclusively on the image acquisition and CV pipeline. SEM images were acquired from the genosensors using a Digital Scanning Microscopy Scanning Electron Microscope (DSM 960 from Zeiss West Germany), and samples were prepared with a thin platinum layer for electrical contact. 32 SEM images were acquired with scales 200 nm and 300 nm, corresponding to sensing units that were subjected to distinct concentrations of PCA3 in addition to the negative sequence (non-complementary) and a blank measurement for control. Afterwards, they are rescaled to 1024×768 pixels and cropped using different window sizes (50, 100, 200, 300, 400, and 500 pixels), with no overlap. We empirically found that the window size of 300×300 pixels was the best approach for training the models (results with other crop dimensions can be consulted in (242)), so we consider this in the following. The result is a dataset with 192 images, representing 8 classes in total: 12 images for the negative samples, 18 for the blank genosensor, 24 for the positive PCA3 concentration of 10^{-5}, 24 for 10^{-4}, 24 for 10^{-3}, 36 for 10^{-2}, 30 for 10^{-1}, and 24 for 1 μmol L^{-1}. Some examples of the final images are shown in Figure 34 for each of the eight classes. The images from different classes show high similarity and are difficult for a human observer to discriminate.

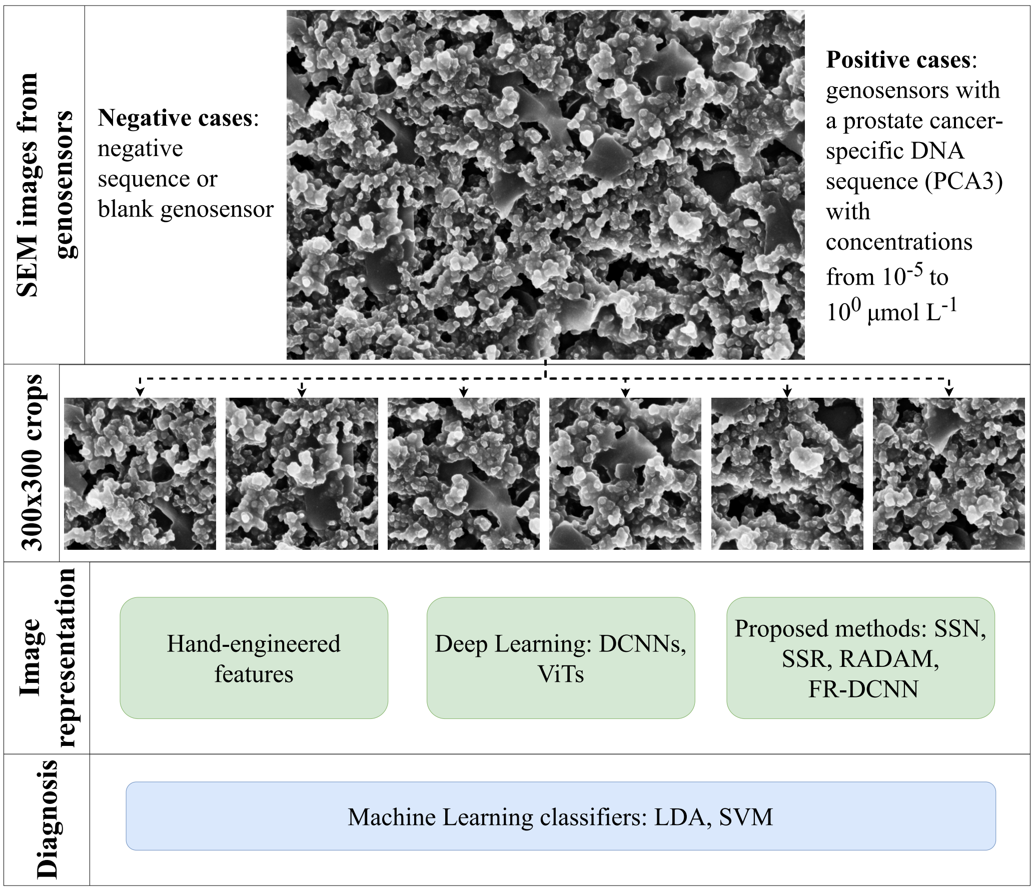

SEM images from genosensors
Negative cases: negative sequence or blank genosensor
Positive cases: genosensors with a prostate cancer-specific DNA sequence (PCA3) with concentrations from 10^{-5} to 10^{0} µmol L^{-1}
300x300 crops
Image representation
Hand-engineered features
Deep Learning: DCNNs, ViTs
Proposed methods: SSN, SSR, RADAM, FR-DCNN
Diagnosis
Machine Learning classifiers: LDA, SVM

Figure 33 – Illustration of the CV pipeline for prostate cancer diagnosis using texture analysis methods for characterizing SEM images from genosensors, which are used to train ML classifiers.

Source: By the author.

6.1.1.3 Results and discussion

We approach the dataset as either an 8-class problem, where the goal is to distinguish each of the 6 PCA3 concentrations, the negative, and the blank genosensors, or a binary case (positive or negative). Firstly, feature extraction is performed using various techniques from the literature, and the proposed SSN, SSR, RADAM, and random DCNN-based methods. We also consider an earlier version of SSR that focused exclusively on grayscale images, (154, 207, 211) thus not containing the multilayer network modeling we describe here for color images. This version contains some minor differences from the SSR methodology described in this thesis, however, we can simply consider it as using only one layer for SSN modeling ($z = 1$, thus using only the network W), followed by the RNN characterization. We consider this version since the SEM images are, in fact, grayscale. In some cases, this approach yields different results than applying the RGB version of SSN and SSR methods, as the obtained results show. We also consider hand-engineered texture features, DCNNs, and ViTs that were explored in the previous chapters of this

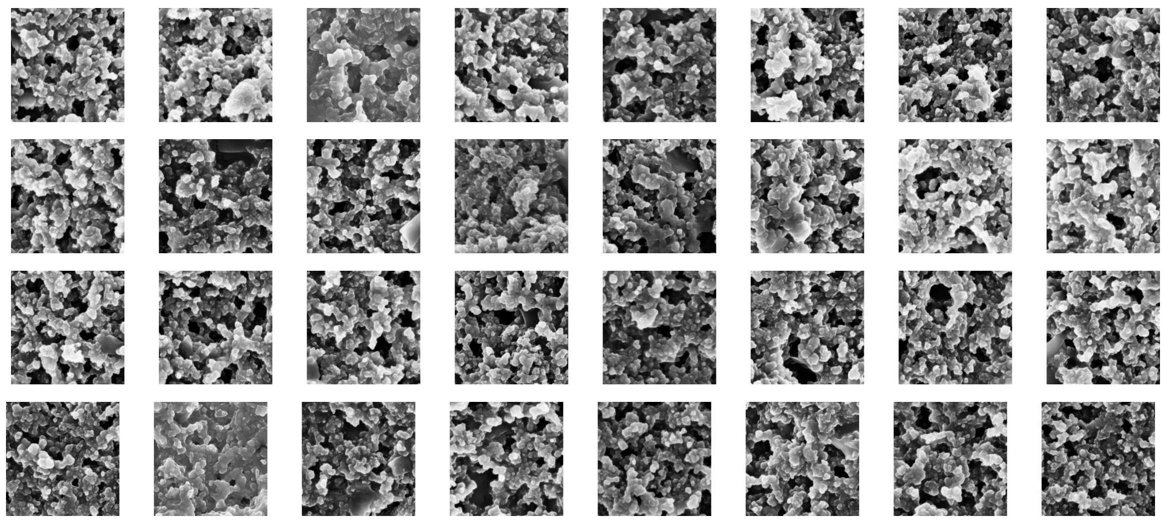


Figure 34 – Samples from the cropped SEM images, where each row is a different sample, and each column is a different class: negative sequence, blank genosensor, and PCA3 biomarker concentrations from 10^{-5} to 1 μmol L^{-1}, from the left to the right, respectively.

Source: RODRIGUES *et al.* (242)

thesis. Additionally, we consider MobileNet (243) as a lightweight DCNN alternative to explore the possibility of mobile diagnosis.

The methods are applied to obtain representations from the image crops in their original resolution, when possible, or by resizing the crops to 224×224 when not possible (for the ViTs). The extracted image representations are used to train and test supervised classifiers in a stratified 10-fold cross-validation scheme (10 repetitions). However, since the dataset is imbalanced, we employ a simple random sampling method at each iteration, selecting a number of samples per class equal to the size of the smallest class (30 samples for the binary case and 12 for multiclass). Performance evaluation was conducted using average accuracy and standard deviation from these 100 random trials (10 repetitions of 10-fold splits). All results are shown in Table 22 for the binary and multiclass tasks using two classifiers (LDA and SVM).

For the binary classification task, all methods perform above 90% with both classifiers. However, the proposed methods show the best performance, performing above 99% (except for FR-DCNN). RADAM and SSR achieve a perfect 100% classification rate in contrast to the best result of 99.8% achieved by literature methods (MobileNet and ConvNeXt-XL in IN-21K). Although FR-DCNN achieves the lowest performance among the proposed methods, it still surpasses some of the compared methods, such as DeiT3-B, ConvNeXt-T and nano, Fractal, CNTD, and Fourier.

In the multiclass classification, the performance of all methods drops compared to the binary case, which is expected due to the higher complexity of distinguishing each in-

Table 22 – Accuracy in binary and multiclass classifications for two supervised classifiers when using each of the texture analysis methods over the SEM images from genosensors. Values in blue show the best result achieved with methods from the literature, red highlight our results matching or above that, and the best result in each task is in **bold type**.

		Binary		Multiclass	
	Method	LDA	SVM	LDA	SVM
	GLDM	92.8±6.7	98.7±1.3	75.7±4.4	79.3±3.3
	Fourier	92.4±5.5	97.1±1.9	56.3±5.7	75.9±3.9
	CLBP	99.0±1.3	98.6 ±1.4	76.5±4.7	74.7±3.5
	Fractal	92.9±5.8	97.9±1.5	59.5±5.2	64.2±4.0
	CNTD	98.1±1.8	96.9±1.9	68.6±5.3	75.5±3.4
	MobileNet	99.8±0.5	99.8±0.5	70.4±4.2	70.6±3.7
	ResNet50	97.7±1.5	99.5±0.8	63.6±5.1	64.7±5.0
	DenseNet201	99.2±0.8	96.3±1.8	75.0±3.6	62.0±4.5
	InceptionResNetV2	99.0±1.1	99.0±1.5	57.9±4.5	56.8±4.0
	ConvNeXt-nano	97.8±1.3	95.3±2.7	72.3±2.7	68.0±5.2
	ConvNeXt-T (in IN-21k)	97.8±2.0	96.8±1.2	76.9±3.7	69.7±4.1
	ConvNeXt-XL (in IN-21k)	99.8±0.5	99.8±0.5	84.5±1.7	80.9±4.3
	ViT-B	99.3±1.1	99.2±1.1	71.4±3.8	66.8±5.6
	DeiT3-B	94.8±2.3	91.7±1.8	54.3±3.8	48.1±4.4
	ViT-B (in IN-21k)	99.7±0.7	99.5±0.8	70.6±4.2	68.8±3.7
	DeiT3-B (in IN-21k-FT1k)	96.5±2.3	96.0±2.1	62.2±4.4	53.7±3.3
Proposed methods	SSN$_a$	99.3±1.1	98.8±1.5	84.0±2.9	70.2±3.5
	SSN$_b$	99.2±1.1	98.0±1.8	84.0±4.4	67.3±2.0
	SSR (grayscale)	**100**±0.0	99.3±0.8	**91.4**±2.0	81.0±2.7
	SSR$_a$	99.8±0.5	**100**±0.0	82.3±3.4	77.2±2.6
	SSR$_b$	99.5±0.8	**100**±0.0	79.9±5.1	77.5±2.5
	RADAM (MobileNet)	**100**±0.0	99.8±0.5	85.5±2.4	71.1±4.7
	RADAM (ConvNeXt-nano)	99.8±0.5	99.3±0.8	87.0±1.9	74.7±4.6
	RADAM (ConvNeXt-T in IN-21k)	99.8±0.5	99.5±0.8	84.1±3.8	73.7±4.4
	RADAM (ConvNeXt-XL in IN-21k)	**100**±0.0	99.8±0.5	88.2±2.6	80.1±4.4
	FR-DCNN (random ResNet50)	98.3±1.8	91.0±8.4	75.6±3.4	53.4±2.5

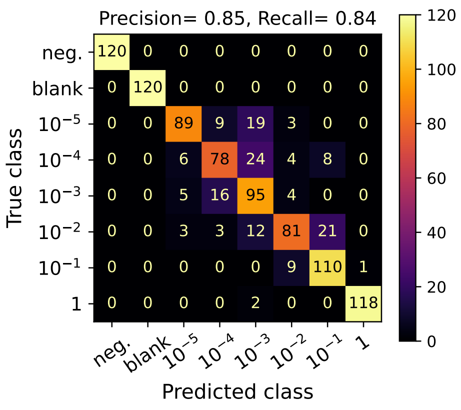
Source: By the author.

dividual PCA3 biomarker concentration. Among the methods from the literature, the best performance is achieved by ConvNeXt-XL (in IN-21k) with 84.5% and 80.9% for LDA and SVM, respectively. In contrast, the proposed methods show again the best results, with SSR grayscale achieving 91.4% with LDA, which represents a gain of 6.9% in absolute accuracy while also being significantly less complex than ConvNeXt-XL. RADAM also surpasses the literature by using ConvNeXt-XL in IN-21K, MobileNet, and ConvNeXt-nano. It is also worth noting that FR-DCNN presents the lowest results among the proposed methods, but still achieves competitive results compared to various methods from the literature. For instance, it surpasses the pre-trained version of ResNet50, showing that ImageNet pre-training is not the ideal approach for ResNet in this task, and corroborating that improved random features carry discriminative power for real tasks with increased complexity (more classes). Moreover, its performance variance is similar to the compared methods, which suggests a better robustness than one would expect from a fully randomized feature extractor.

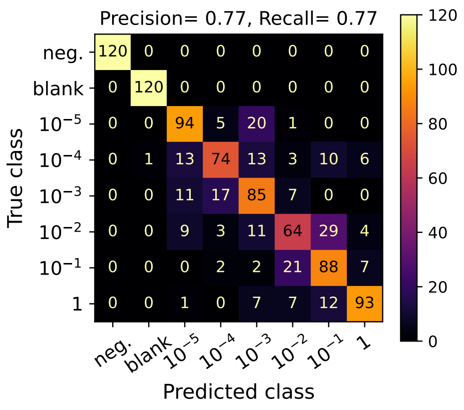
Considering methods that employ improved pre-training (IN-21k) for the DCNNs and ViTs, it is possible to observe that they often achieve the best performance compared to those employing the standard ImageNet-1k pre-training. On the other hand, the proposed methods achieve superior performance regardless of this factor. For instance, RADAM with MobileNet and ConvNeXt-nano consistently surpasses the literature approaches using IN-21k pre-training. More importantly, considering SSN and SSR that are hand-engineered features and, therefore, considerably more efficient (lower FLOPs and no stored parameters), their performance is notable in comparison to deep-learning-based methods. Our methods then not only achieve the best results but also have a lower cost considering, for instance, SSR compared to the best approach from the literature (ConvNeXt-XL), which is the largest CNN in the comparisons.

Figure 35 shows the confusion matrices for the best two methods from the literature (a-b), considering the multiclass task, and for the proposed RADAM, SSR, and FR-DCNN methods. The numbers inside the matrix sum to 120 samples per row, representing the number of trials, where 10 random sampling iterations are performed selecting 12 samples per class at each one. We also show the precision and recall (averaged over all classes), which are particularly crucial metrics for situations such as diagnosing critical diseases since they estimate the ability of the classifier to minimize the occurrence of false positives and false negatives, respectively. The results show that all methods are able to distinguish the negative and blank classes from the ones containing concentrations of prostate-cancer PCA3 sequence, except for ConvNeXt-T that incorrectly labeled a positive sample ($10^{-4}\mu$mol L^{-1}) as a blank sensor. Considering the multiclass task, RADAM and SSR achieve the highest precision and recall, showing the lowest number of misclassification (numbers outside the diagonal of the matrix). The misclassifications happen mostly between similar classes, *i.e.*, similar concentrations, which is to be expected considering the small difference between them. FR-DCNN also shows satisfactory performance, surpassing ConvNeXt-T in terms of precision and recall, and also being the only method alongside RADAM that achieves perfect classification accuracy of the images with the highest concentration of the prostate-cancer PCA3 sequence (1 μmol L^{-1}). These results are impressive considering that ConvNeXt-T uses IN-21k pre-training, while FR-DCNN uses a fully random ResNet50.

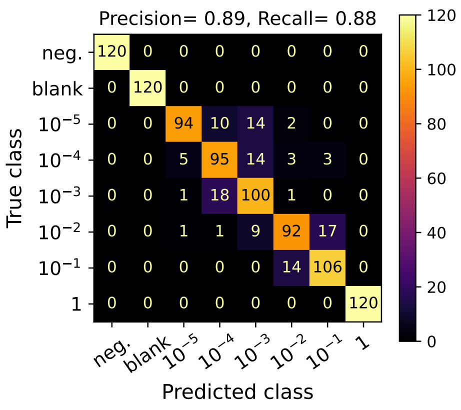
In conclusion, the proposed methods, especially SSR and RADAM, consistently outperform the literature methods for both binary and multiclass diagnosis tasks. Our novel approaches to feature extraction offer significant improvements in performance and



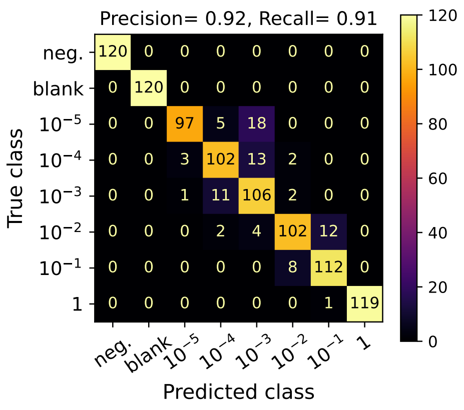
(a) ConvNeXt-XL in IN-21k.



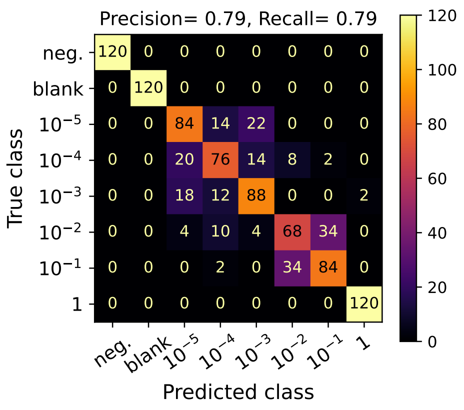
(b) ConvNeXt-T in IN-21k.



(c) RADAM (ConvNeXt-XL in IN-21k).



(d) SSR (grayscale).



(e) FR-DCNN (random ResNet50).

Figure 35 – Confusion matrices, precision, and recall for different methods (using LDA) for the eight classes of genosensor images: negative, blank, and different concentrations of prostate-cancer PCA3 sequence (in μmol L^{-1}).

Source: By the author.

efficiency over a variety of previous techniques, ranging from hand-engineered to deep-learning-based features. They offer different choices according to the computational resources available. For instance, for mobile or embedded systems we suggest either SSR or RADAM with MobileNet, and RADAM with ConvNeXt-XL if more computing resources are available. Any of these alternatives surpasses all the compared methods from the literature and offers different computational budgets. These results are promising, and the gains achieved by the proposed methods can potentially improve the automated diagnosis of prostate cancer using CV, which could have significant clinical implications such as easier early diagnosis of the condition.

6.1.2 Diagnosis of COVID-19 through computer vision

As previously mentioned, the COVID-19 pandemic highlighted the need for accessible diagnostic approaches. Early diagnosis of SARS-CoV-2 is crucial for effective pandemic management. The primary diagnostic method, real-time polymerase chain reaction (RT-PCR), necessitates sophisticated laboratory infrastructure and skilled personnel, which hampers its application in low-income countries or remote areas with limited resources. Alternative molecular detection techniques, such as Loop-mediated isothermal AMPlification (LAMP) and Clustered Regularly Interspaced Short Palindromic Repeats (CRISPR), face similar constraints. Serological tests, including enzyme-linked immunosorbent assays (ELISA) and immunosensors, detect host-generated antibodies, but their limited sensitivity during the first 1-2 weeks post-infection precludes effective identification of asymptomatic or recently infected individuals, undermining containment strategies. On the other hand, genosensors may be a long-term solution for mass testing diseases requiring sensitive genetic material detection. They have long been used in research laboratories and other settings to diagnose various diseases, including SARS. In this sense, a similar framework as the one previously described is considered here in another innovative approach for CV-assisted diagnosis of COVID-19.

The results presented in this section were achieved through a collaboration between various researchers and institutions, resulting in a publication (244) in the Materials Chemistry Frontiers journal, from the Royal Society of Chemistry. The work proposes new genosensors for the detection of ssDNA sequences from the SARS-CoV-2 genome, and detection is performed using electrical and electrochemical impedance spectroscopic, localized surface plasmon resonance, and CV. In the paper, our research group focused on the CV approach, which follows a similar framework as described in the previous section: SEM images are obtained from the genosensors, followed by image feature extraction and the supervised training of ML classifiers. In this sense, the only difference is that we do not perform cropping in this case, since sufficient images were available. We then extend the published results by including new methods from the literature and our proposed techniques SSN, SSR, RADAM, and random DCNNs, where the detection rates are improved considerably.

6.1.2.1 Image acquisition

SEM images were acquired from genosensors using a Zeiss-LEO440 electron microscope equipped with a detector 7060 (Oxford Instruments). Ten duplicate images from different sample regions were obtained to capture the visual variability of the genosensors. A magnification of 1,000X is used to assess the feasibility of employing optical microscopes for future analysis. In total, there are 200 images with a resolution of 1024×714 pixels from sensing units exposed to varying SARS-CoV-2 concentrations, negative sequences, or interferents (HPV16 and PCA3). Interferent samples served as control measurements.

The images are organized according to two classification schemes, similar to the procedure described in the previous section. In this sense, we consider a binary case, where the aim is to distinguish between SARS-CoV-2 positive or negative/interferent samples, and a multiclass case where specific concentrations have to be detected. For multiclass classification, the following eight classes are considered: control, which contains negative sequences and interferents (HPV16 and PCA3); positive SARS-CoV-2 ssDNA concentrations of 10^{-18}; 10^{-16}; 10^{-14}; 10^{-12}; 10^{-10}; 10^{-8}; and 10^{-6} mol L^{-1}. Figure 36 shows some examples from the dataset, where each column is one of the aforementioned classes (in the same order described), and each row contains different samples from that class.

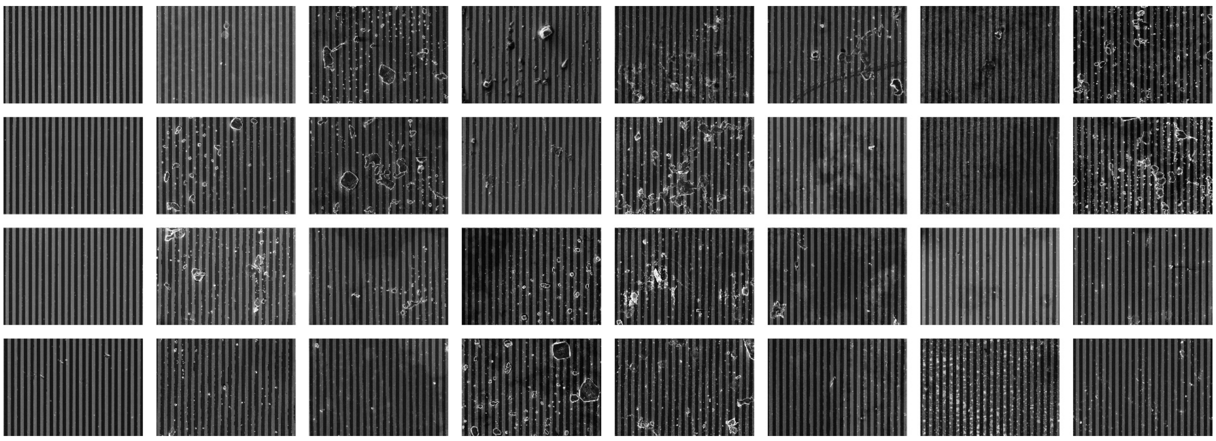


Figure 36 – Examples of the SEM images obtained from genosensors submitted to control sequences (first column, with negative and interferents), and different concentrations of SARS-CoV-2 ssDNA (remaining columns). Each row is a different sample of the same class of the column. We employ these images for either a binary classification task (positive or negative/control), or a 8-class problem to distinguish each concentration.

Source: SOARES *et al.* (244)

Compared to the previous application, in the previous section, the genosensor images here are considerably different. Firstly, the scale is much bigger, which is considered an early evaluation of the use of optical microscopes instead of SEM (which is more costly). The utmost goal in the future would be, for instance, to use common smartphones to obtain pictures, allowing for easier and faster use of the detection algorithms. However, changing the scale can pose many challenges for the feature extraction techniques. It is possible to notice that the images here contain various artifacts, which could be due to defects or variability in the surface of the genosensors, or the presence of external particles or substances. Nevertheless, the goal is that the CV algorithms learn to ignore them when classifying the genosensor.

6.1.2.2 Results and discussion

The same set of hand-engineered and deep-learning-based features evaluated in the previous section are considered here for characterizing SEM images related to SARS-CoV-2 genosensors. The validation procedure is also the same, except that we consider the original image resolution (no cropping) for the methods that allow so (all methods except the ViTs). Random downsampling is performed again to balance the classes since a minimum of 50 samples per class is available for the binary case, and 20 for multiclass. The results are shown in Table 23 for LDA and SVM, with the corresponding standard deviation for 10 repetitions of 10-fold stratified cross-validation.

The results show that the best methods from the literature are hand-engineered. Fourier descriptors achieve the best performance on the binary task, with 99.7% using SVM and CLBP, and CLBP achieves 95.8% using LDA on the multiclass task. These results are somewhat unexpected considering the high variability of the SEM images in this case, where one would expect that deep-learning-based features could perform better. However, since these images are obtained in a controlled environment, their underlying texture is well represented and the scenario is completely different from the one where the deep-learning methods are pre-trained (natural images). Therefore, although deep-learning techniques are known to automatically learn high visual abstraction, they may fail when applied to out-of-distribution tasks (when the data structure is considerably different from their training data). Nevertheless and although they are considerably more costly considering computational complexity, we can highlight the ConvNeXt variants pre-trained on IN-21k, which achieves a performance closer to Fourier and CLBP in both tasks.

Considering the proposed methods, we observe again superior performance compared to the literature considering either SSN, SSR, or RADAM. SSN surpasses Fourier on the binary task, and RADAM with ConvNeXt-T and XL surpasses CLPB on the multiclass task, achieving the highest result (97.4%) in this case. The results of RADAM

Table 23 – Accuracy for binary (positive/negative) and multiclass classification of images from genosensors with SARS-CoV-2 ssDNA, using various feature extractors ranging from hand-engineered and deep-learning approaches. Colors and bold type are used in the same way as in previous tables.

		Binary		Multiclass	
	Method	LDA	SVM	LDA	SVM
	GLDM	96.6±2.1	98.4±0.9	90.6±1.2	79.4±1.5
	Fourier	95.8±1.9	99.7±0.5	77.5±2.6	72.6±1.9
	CLBP	98.2±1.3	98.5±1.1	95.8±1.0	93.0±0.9
	Fractal	92.8±3.3	84.0±3.1	74.1±2.4	33.4±2.3
	CNTD	93.8±3.4	96.5±1.7	89.8±1.8	83.9±1.5
	MobileNet	98.5±1.1	98.7±0.9	89.1±1.6	87.2±1.1
	ResNet50	96.7±1.3	97.3±1.6	80.0±1.5	81.8±1.8
	DenseNet201	99.1±0.8	99.2±0.7	94.5±1.1	90.2±1.2
	InceptionResNetV2	98.5±1.3	98.5±0.7	86.9±1.6	79.4±1.4
	ConvNeXt-nano	98.7±1.2	98.0±1.3	84.7±1.5	81.8±1.2
	ConvNeXt-T (in IN-21k)	98.7±1.0	97.4±1.8	95.3±1.1	88.6±1.5
	ConvNeXt-XL (in IN-21k)	96.8±1.7	97.1±0.9	93.4±1.0	88.7±1.3
	ViT-B	98.4±0.9	98.2±1.2	79.0±2.3	75.8±1.7
	DeiT3-B	96.2±1.5	96.5±2.2	73.4±1.7	69.3±1.9
	ViT-B (in IN-21k)	98.0±1.5	97.1±1.9	78.9±1.9	75.8±2.5
	DeiT3-B (in IN-21k-FT1k)	97.3±1.4	97.0±1.7	80.8±2.0	78.6±1.9
Proposed methods	SSN$_a$	99.8±0.4	94.8±2.4	87.8±0.9	70.7±1.5
	SSN$_b$	99.7±0.5	93.7±1.5	91.7±0.9	73.1±2.2
	SSR (grayscale)	99.1±0.3	98.8±0.6	92.4±0.4	87.3±1.2
	SSR$_a$	**100**±0.0	97.8±0.9	95.5±0.8	86.9±0.8
	SSR$_b$	99.9±0.3	97.7±0.9	96.2±0.9	86.8±0.8
	RADAM (MobileNet)	99.1±0.7	97.1±1.4	94.1±1.5	67.8±0.8
	RADAM (ConvNeXt-nano)	99.7±0.5	98.9±0.8	95.6±0.7	77.6±1.7
	RADAM (ConvNeXt-T in IN-21k)	99.4±1.2	98.6±0.8	96.9±0.9	77.5±1.7
	RADAM (ConvNeXt-XL in IN-21k)	99.3±0.6	98.6±1.1	**97.4**±0.7	92.2±1.3
	FR-DCNN (random ResNet50)	98.8±0.4	95.2±1.2	81.4±1.4	47.8±2.8

Source: By the author.

with backbones pre-trained on IN-21k corroborate the importance of better pre-training for this task. However, the best method is SSR, which achieves perfect classification accuracy on the binary task and also surpasses CLBP on the multiclass task, *i.e.*, it surpasses every method from the literature in both cases simultaneously. For instance, while RADAM with ConvNeXt-XL performs better at discriminating each specific SARS-CoV-2 ssDNA concentration, it may fail in some cases of negative/positive classification, which is more critical from a diagnosis point of view. Nevertheless, SSR can achieve perfect positive/negative discrimination while maintaining competitive multiclass performance. Lastly, we notice that FR-DCNN presents the lowest performance among the proposed methods, but can surpass various methods from the literature, especially in the binary task.

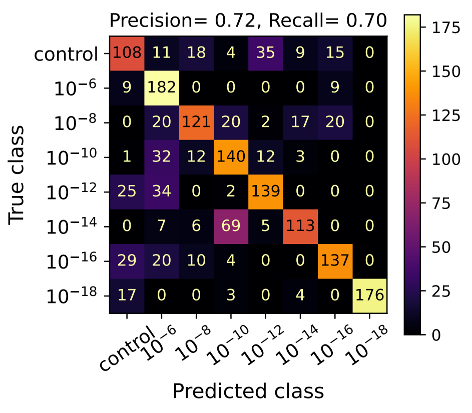
Lastly, we evaluate the best methods in terms of class-specific performance, precision, and recall, and the results are shown in Figure 37 (values correspond to the sum of the 10 iterations). It is possible to notice that although the Fourier descriptor provides the best binary classification accuracy among the literature methods, its performance is considerably degraded when increasing the complexity of the task to multiclass. In this scenario, CLBP provides the best performance among the compared methods, with 0.96 and 0.95 precision and recall, respectively. However, a considerable number of samples from the class with a SARS-CoV-2 ssDNA concentration of 10^{-12} mol L^{-1} are misclassified as other classes, particularly as the control group, which is an unstable and risky behavior in terms of diagnostics. On the other hand, SSR and RADAM perform considerably better than the literature methods, providing the highest precision and recall, better stability across classes, and even a perfect classification for some classes. The proposed FR-DCNN method is surpassed by CLBP but still provides a promising performance by surpassing the Fourier descriptor. While this approach may not be the best among the compared and proposed methods, its performance corroborates again that random features should be further explored.

These results add to our previous investigation about using the proposed methods with genosensor images. Aside from prostate cancer, this approach also achieves high performance for COVID-19 diagnosis and the detection of different viral loads. These findings corroborate that the proposed methods are powerful candidates for CV-assisted systems for the diagnostic of different diseases. The variety of proposed methods also offers options to balance the computational budget, performance, and robustness, enabling the development of high-performance mobile or embedded applications. This approach could add to the available technology for SARS-CoV-2 detection, making mass testing cheaper and faster including during different periods of infection that manifest different viral loads.

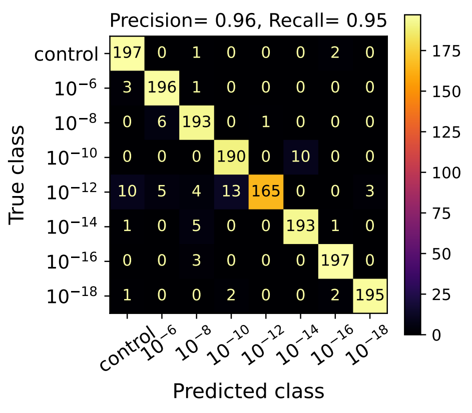
6.2 Computer vision for plant sciences

Plant species play a critical role in the ecosystem's functioning since they supply food to nearly all terrestrial organisms, maintain the atmosphere due to photosynthesis, and recycle matter. They also contribute significantly to the global economy, providing raw materials for various industries such as pharmaceuticals, textiles, and construction. Despite their importance, plant species are under constant threat from climate change, habitat loss, and illegal deforestation, among other factors. This is where the importance of plant sciences comes in, where botany, agronomy, horticulture, and plant pathology, aim to understand plant biology and apply this knowledge to improve plant health, productivity, and sustainability.

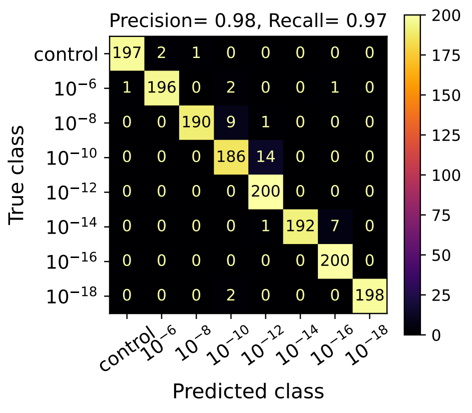
CV is increasingly being used in plant science since it enables fast and automatic



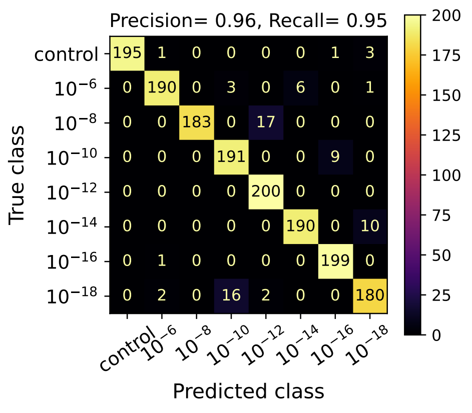
(a) Fourier.



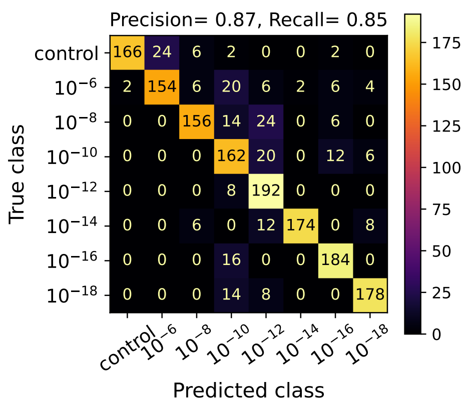
(b) CLBP.



(c) RADAM (ConvNeXt-XL in IN-21k).



(d) SSR$_a$.



(e) FR-DCNN (random ResNet50).

Figure 37 – Confusion matrices, precision, and recall for different methods (using LDA) for the eight classes of genosensor images: control (negative and interferents), and different concentrations of SARS-CoV-2 ssDNA (in mol L^{-1}).

Source: By the author.

detection and analysis of various aspects related to the vast number of plant species on the planet. It can automate the process of identifying and classifying plant species, (245) detecting diseases, (246) monitoring wildfires, (247), and many more. CV algorithms can then help to optimize agricultural practices, protect forests and endangered biomes, and enforce environmental regulations. Therefore, combining plant sciences with CV can enhance our understanding, protection, and utilization of plant resources. In this sense, this section presents two applications in plant sciences using our proposed CV methods, which we describe in the following.

6.2.1 Jacaranda Caroba leaves as environmental biosensors

As urbanization and industrialization continue to increase, air quality becomes a major concern as the levels of air pollutants change. Therefore, continuously monitoring air quality is an urgent need not only in urban centers but on the planet in general. One potential approach is the use of biosensors, which provides a cost-effective and sustainable alternative. For instance, the leaf of a tree may be a viable approach if its leaf plasticity could capture changes in its environment. This not only contributes to our understanding of the impact of pollution on our environment but also provides a valuable tool for monitoring and managing air quality and assessing its impact on plants. In this context, a significant subject of study is the Jacaranda Caroba tree, a species native to South America which is commonly used for reforestation and urban afforestation.

In this section, we describe a study to assess the potential use of the Jacaranda Caroba leaves as environmental biosensors through CV applied to microscopy images. The research was developed firstly in collaboration with the team of Prof. Dr. Rosana Kolb, which conducted the biological experiments with the Jacaranda Caroba specimens and collected the microscopy images. Afterwards, the images were analyzed in collaboration with researchers Lucas C. Ribas, Adolfo Grosso, and Marilia B. S. D. Fernandes, resulting in a paper that is in its final writing stage. The microscopic images used in this study are described in (248) where they were first employed in a simple CV pipeline using only hand-engineered methods. Here, we consider a wider range of methods including deep-learning-based, as well as our proposals.

Jacaranda is a plant of the *Bignoniaceae* family, which includes more than 800 tree species. Its distribution is vast in tropical and subtropical climates, and Brazil concentrates a large part of the species diversity. Some species of the *Bignoniaceae* family are plants with wide medicinal use, having their properties recognized for the treatment of various diseases. Jacaranda Caroba (*Jacaranda decurrens Chamisso*), popularly known as caroba or *carobinha do campo* (in Portuguese), is traditionally used in herbal preparations for different uses: its flowers are used in teas to improve digestion, while the use of macerated leaves was used for healing wounds. The cerrado biome where the Jacaranda Caroba predominates is included in the list of global hotspots, which are natural habitats that correspond to 1.4% of the planet's surface and concentrate 60% of the world's biological baggage. (249) Despite this very important feature of the biome, it continues to be one of the most deforested areas of ecosystems in the country. The recovery of savanna vegetation has been raised as a public policy for the environment, especially in the state of São Paulo, where the devastation of the biome advances along with the growth of cities and agribusiness. The biome that originally occupied 14% of the territory's vegetation now occupies less than 1%. (249)

Biome recovery through reforestation uses species of the *Bignoniaceae* family in conventional planting of areas degraded by grazing activities, agriculture, or areas degraded by fire. The plants of this family have their seeds in "wing" shapes that favor their dissemination by the wind, in addition, they have a high germination rate during the rainy season, which contributes to the natural recovery of areas damaged by deforestation. (250) They are also quite often used for urban ornamentation and afforestation in cities throughout Brazil. Therefore, employing Jacaranda Caroba as a potential biosensor would allow for assessing environmental properties not only in urban centers, where assessing the air quality is essential but also in a wide area across the country, where the detection of phenomena such as acid rains is also important. In other words, it could provide a network of biosensors spread across the country.

6.2.1.1 Biological experimentation and image acquisition

Given the potential of the Jacaranda Caroba tree as a biosensor, a detailed study was conducted to understand its response to different concentrations of pollutants. A set of Jacaranda Caroba specimens is divided into six groups, seeded and cultivated in a controlled environment, where five groups are exposed to potassium fluoride spray for 60 days, and one is a control group. This compound is chosen because fluoride salts and hydrofluoric acid are the main fluorides of industrial value and the primary source of the fluoride ion for applications in manufacturing and in chemistry. Therefore, it is an important chemical signature of pollutants generated by human activity. Each group of specimens received a different concentration of the substance (in mg/L): 5, 15, 30, 60, and 1162. The control group was not exposed to the substance.

Microscopy images are collected from the Jacaranda Caroba specimens using a Zeiss LSM 780 (CLSM), which uses the technique of fluorescence microscopy. The microscope captures wavelengths between $400nm$ and $700nm$ with a high-definition spatial resolution of 2048×2048 pixels. After the exposure period, images were collected using between four and five leaves of six specimens, from each group. In total, there are 148 images, 25 for the control group and those exposed to potassium fluoride 5 mg/L, 26 for the 15 mg/L group, and 24 for the groups of 30, 60, and 1162 mg/L. Figure 38 shows examples from the dataset, where each column is a class (or group from the experiment), and each row is a different sample.

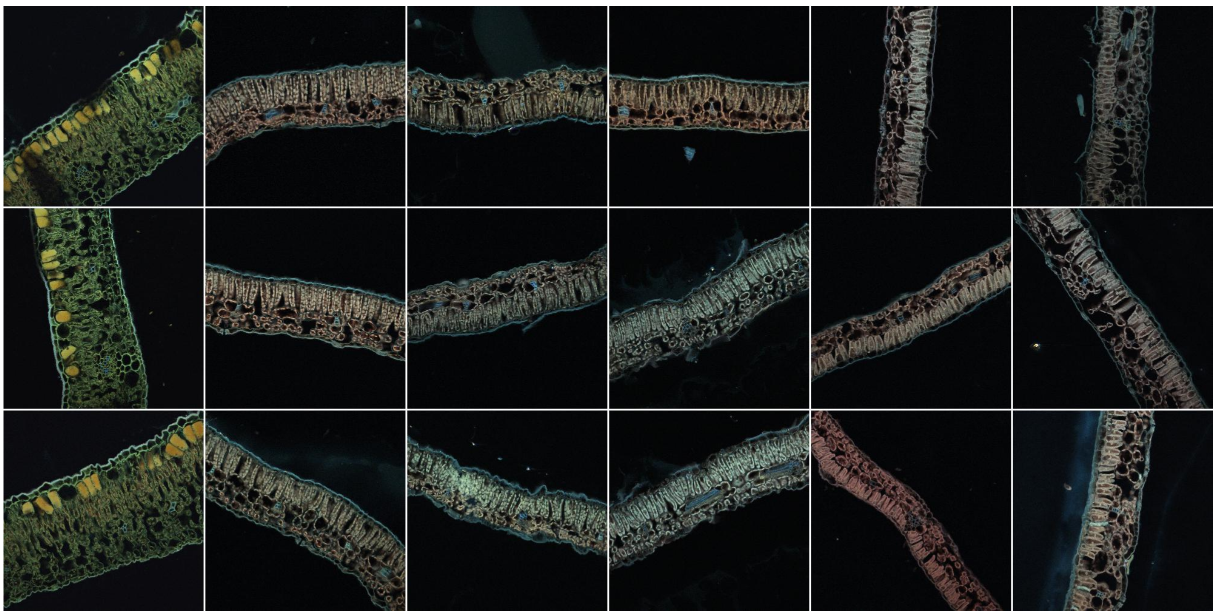


Figure 38 – Microscopy images from leaf samples of Jacaranda Caroba specimens subjected to daily doses of potassium fluoride with different concentrations. From the left to the right, each column represents a different concentration (in mg/L): 0 (control group), 5, 15, 30, 60, and 1162.

Source: FALVO. (248)

6.2.1.2 Results and discussion

We apply various CV models for distinguishing between the different pollutant groups of the dataset. Firstly, three DCNN models are considered to assess the possibility of training their entire architecture in the dataset. We consider the InceptionV3 (190) and RestNet (61) since they are well-known CNN models, and two ResNet sizes (50 and 101 layers) are used to evaluate the impact of depth. They are then trained from scratch, *i.e.*, all weights are randomly initialized and trained until accuracy stops to increase in a 10-fold cross-validation scheme. Classification is performed using a linear layer with a softmax classifier. In this setup, we tuned the hyperparameters for better training the models, and we found that the best results are achieved using the ADAM optimizer, (71) a batch size of six, a learning rate $\alpha = 0.005$ and stopping criteria if no validation loss improvement happens within a window of 40 training epochs, where we observed an average between 100 and 200 total training epochs. Another approach we take on the compared CNNs is fine-tuning after pre-training the model on ImageNet-1k, *i.e.*, pre-trained weights are ported by keeping them in the model except for the classification layer, which is replaced. Afterward, we retrain the model using a smaller learning rate of $\alpha = 0.00005$ with the same settings as before for the other hyperparameters (average between 80 and 150 total training epochs). For both training from scratch and fine-tuning, we use the recommended input dimensions for these models (299 for Inception and 224 for ResNet). It is important to notice that we tried to train these models using full-resolution images, but the memory requirements are huge and exhausted the memory of our 24GB GPU card (Quadro P6000).

The following hand-engineered techniques are also considered for comparison: CLBP (124) which is a gold standard among this kind of method, and CNTD, (16) an earlier Network-Science-based method for texture analysis. Moreover, we use the aforementioned DCNNs as feature extractors by considering ImageNet pre-training and using the GAP of the last convolutional layer. In this case, we are also able to use full-resolution images. For this approach, we also considered the ConvNeXt (67) variants nano with ImageNet-1k pre-training and T and XL with ImageNet-21k pre-training. Additionally, feature extraction is performed with ViT-B/16 (64) with ImageNet-21k pre-training, and DeiT3 (65) (with either ImageNet-1k and 21k). Notice that these ViT methods are limited to a fixed input size (224×224), regardless if we only use them for feature extraction. For all the feature extraction methods using either hand-engineered features, DCNNs, Vits, and our proposed methods, we use the LDA classifier with the 10-fold cross-validation technique for evaluating their performance. Finally, the results for all methods are given in Table 24.

Our results show that training DCNNs from scratch is the worst approach, as they often achieve 100% of train accuracy but much poorer testing accuracy. This behavior corroborates the difficulty of training such complex models on small datasets, which leads to critical overfitting levels. They are better employed through transfer learning, where weights are pre-trained in large-scale vision tasks. Using their pre-trained version, fine-tuning is an interesting alternative when using their original input sizes. This approach achieves similar results to the hand-engineered methods from the literature that use full-resolution images. On the other hand, by using the DCNNs only for feature extraction with full-resolution images, the results are considerably superior. In this sense, we consider the ConvNeXt variants only for feature extraction at the full-resolution input size, and also the ViTs only for feature extraction (with their fixed input size). Among the literature methods, ConvNeXt-nano achieves the best result, with 98.8% accuracy. As for the ViTs, it is possible to observe that they perform at a similar level to hand-engineered features.

Concerning the proposed methods, we notice that although SSN overcomes the compared hand-engineered methods and fine-tuned DCNNs, it is surpassed by the other more complex methods. The highest classification accuracy is achieved with the proposed RADAM over ConvNeXt-nano, with 99.3%. RADAM also surpasses the literature when using ResNet50 with input size 2048×2048. SSR and FR-DCNN also achieve impressive

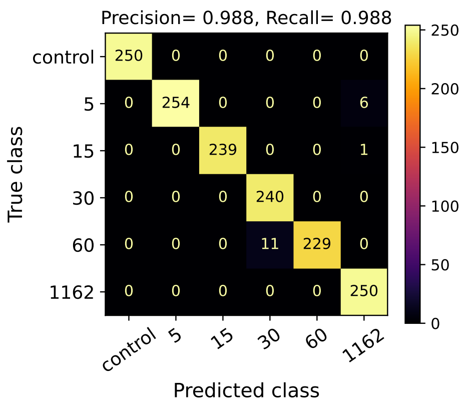
Table 24 – Classification accuracy (10-fold cross-validation) for different methods for the characterization of pollutant levels on the Jacaranda Caroba leaves. Different input sizes are evaluated for DCNN-based methods (according to their architecture). Methods for feature extraction, such as the proposed methods, are coupled with the LDA classifier. The result in blue highlight the best method from the literature, and red are our results matching or above that.

	Method	Input size	LDA acc.
Trained	InceptionV3	299 × 299	55.1±3.5
	ResNet50	224 × 224	59.6±5.3
	ResNet101	224 × 224	60.5±5.3
Fine-t.	InceptionV3	299 × 299	81.8±3.6
	ResNet50	224 × 224	83.1±2.7
	ResNet101	224 × 224	82.7±1.5
Feature extraction	CLBP (integrative)	2048 × 2048	79.2±1.9
	CNTD (integrative)	2048 × 2048	68.1±3.0
	InceptionV3	299 × 299	67.1±2.4
	InceptionV3	2048 × 2048	86.9±1.5
	ResNet50	224 × 224	72.2±2.2
	ResNet50	2048 × 2048	89.4±1.2
	ResNet101	224 × 224	74.6±2.2
	ResNet101	2048 × 2048	91.3±1.3
	ConvNeXt-nano	2048 × 2048	98.8±0.5
	ConvNeXt-T (in IN-21k)	2048 × 2048	97.4±0.9
	ConvNeXt-XL (in IN-21k)	2048 × 2048	97.2±0.8
	ViT-B/16 (in IN-21K)	224 × 224	84.2±1.7
	DeiT3-B/16	224 × 224	79.9±1.6
	DeiT3-B/16 (in IN-21K-FT1K)	224 × 224	80.8±2.0
Proposed methods	SSN$_a$	2048 × 2048	83.0±2.5
	SSR$_a$	2048 × 2048	94.1±2.3
	RADAM (ResNet50)	224 × 224	88.4±1.9
	RADAM (ResNet50)	2048 × 2048	98.8±0.5
	RADAM (ConvNeXt-nano)	2048 × 2048	**99.3**±0.5
	RADAM (ConvNeXt-T in IN-21k)	2048 × 2048	97.8±0.8
	RADAM (ConvNeXt-XL in IN-21k)	2048 × 2048	97.8±0.5
	FR-DCNN (random ResNet50)	2048 × 2048	91.9±0.7

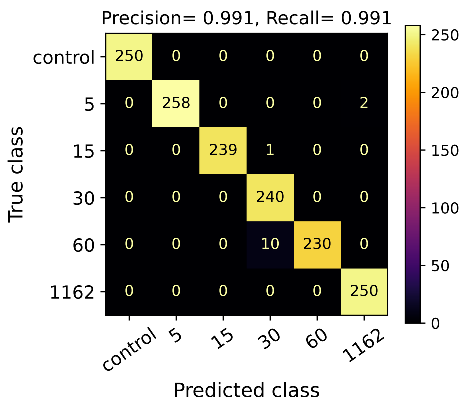
Source: By the author.

results, surpassing all the compared methods except ConvNeXt. Moreover, SSR requires considerably less computation in comparison to deep-learning-based methods. The results with FR-DCNN show that ImageNet pre-training is not a good approach for a common ResNet in this task. However, when using RADAM the results are increased significantly (both on pre-trained or random backbones through FR-DCNN).

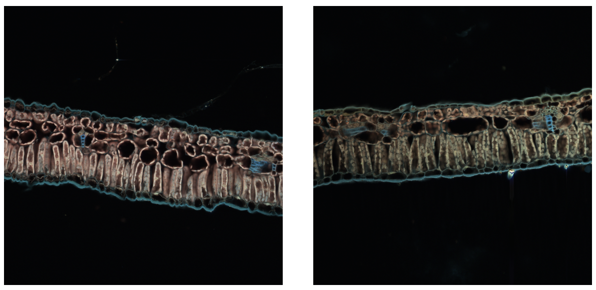
To better understand the behavior of the best methods (ConvNeXt-nano with or without RADAM), Figure 39 shows their confusion matrices, where the values correspond to the sum of the 10 cross-validation iterations. In terms of precision and recall, both methods show high performance. RADAM provides metrics above 0.99, which surpasses the original ConvNeXt-nano, with 0.988. Upon careful inspection of the misclassifications, we notice that both models achieve perfect classification for the groups of control, 30 and 1162 mg/L. On the other hand, the highest error rates appear in the group of 60 mg/L, where misclassifications happened between the group of 30 mg/L. Nevertheless, these two groups are close concerning the concentration of pollutants they were submitted to, and similar images are expected. However, the most concerning performance is related to the few misclassifications of samples from groups with lower pollutant concentrations classified as the highest one (1162 mg/L). For instance, for ConvNeXt-nano this happens with 6 samples from the group of 5 and 1 sample from the group of 15 mg/L. RADAM diminishes this problem, where only 2 samples from the group of 5 are misclassified as 1162 mg/L.



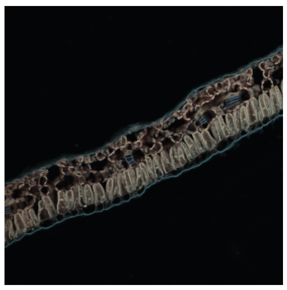
(a) ConvNeXt-nano.



(b) RADAM (ConvNeXt-nano).



(c) 30 mg/L samples (misclassified as 60 mg/L).



(d) 60 mg/L sample.

Figure 39 – Confusion matrices, precision, and recall for ConvNeXt-nano with or without RADAM for distinguishing the Jacaranda Caroba leaves between the control group and the ones submitted to five potassium fluoride concentrations (in mg/L).

Source: By the author.

In conclusion, the results suggest that the leaf plasticity of Jacaranda Caroba is affected by the potassium fluoride pollutant. Using the microscopy images from the leaves, both hand-engineered and deep-learning methods are able to detect this effect, but the latter is considerably superior in detecting the specific concentration of the pollutants. However, our proposed methods offer different alternatives that may escape this behavior. A detection mechanism using SSN is the best approach among other hand-engineered features and could be highly efficient in comparison to DCNNs. SSR operates at a slightly increased computational budget but delivers deep-learning-level performance with better efficiency than DCNNs. SSN and SSR are then viable alternatives for mobile or embedded devices to be used in field research. FR-DCNN can serve as a baseline and simpler approach among DCNNs, or for evaluating the benefits of using pre-trained weights in unusual applications. On the other hand, RADAM with pre-trained DCNNs is the only method performing above 99% and is the ultimate choice if high accuracy is a priority for detecting the exact concentrations of pollutant levels. Detailed examination of the class-specific performance of the best method from the literature and RADAM shows that our method improves the overall precision and recall, and reduces errors between distant pollutant levels

6.2.2 Brazilian plant species identification through leaf midrib images

Another important task in plant sciences is the classification of plant species. Considering the vast amount of species, this can be, even for specialists, a hard task. (251) Traditionally, taxonomists use identification keys to discover the unknown species that is being analyzed; this approach is time-consuming and can lead to problems (especially with species that are similar). To surpass problems brought via manual identification processes, studies using ML techniques have been conducted, many of them using deep learning. (252) Using automated methodologies, the process of identification can be more accurate, less time-consuming, and more reproducible. (253) Nowadays, the automated identification field may lie on molecular methods, such as DNA sequencing, or can make use of ML techniques to classify images from different plant structures. (253) In this context, this section describes an application of our proposed CV methods for the detection of a variety of Brazilian plant species. This work was developed in collaboration with researchers Rayner M. Condori and Isabella C. L. Munhoz, and part of the results was published (254) in the 18th International Conference on Computer Analysis of Images and Patterns (CAIP 2019) in Salerno, Italy, which was presented by the author.

Different studies have been conducted to develop better strategies in automated plant classification using CV, and, a majority of them utilize leaves for plant discrimination. (252) Although the use of these structures may be an interesting option since leaves are available for sampling throughout the year, environmental conditions can cause these structures to vary drastically. (255) In order to avoid this environmental effect, some internal structures that potentially store information as discrimination patterns can be used in plant identification. One of them, called leaf midrib, is composed of sets of specialized tissues, phloem and xylem, and other cells. (256) As shown in previous studies, (245) these structures may become potential plant identifiers because they are less plastic than other regions of the leaf, they are stable when submitted to the image acquisition process and their characteristics vary among different species.

6.2.2.1 Considered species and image acquisition

We focus on a selection of 50 different plant species found in the Cerrado biome in central Brazil, at the IBGE (Brazilian Institute of Geography and Statistics) Ecological Reserve. For identifying them, leaf samples were harvested from the third and fourth nodes from the tips of fully mature branches. The central part of these leaves, including the midrib, was preserved in FAA 70 (a mixture of Formalin, Acetic acid, and 70% Alcohol) for a period of 48 hours. Following this, the samples were dehydrated via a graded series of ethanol solutions and subsequently embedded within paraffin. Cross-sections of these embedded samples were cut to a thickness of 8 μm. These thin sections were then stained using 1% astra blue and 1% basic fuchsin. Midrib images were captured at a 10x magnification using a trinocular Axio Lab A1 microscope, outfitted with an Axiocam ICc 1 digital camera. The captured images were then pre-processed; the background was removed through manual segmentation, leaving only the midrib region for analysis. The process resulted in a dataset with a total of 606 high-quality RGB images, with varying resolutions and aspect ratios (since segmentation and cropping were performed on each image individually). Figure 40 shows samples from 20 out of the 50 species, and in (245) the reader may check the name and additional details of all the species in the dataset.

6.2.2.2 Results and discussion

The midrib images show expressive texture patterns that vary between species, which suggests that texture descriptors could be useful for identification. However, results with hand-engineered descriptors (245) show the difficulty of effectively distinguishing between all the considered species. This happens due to several image variations such as rotation, scale, and also morphological and structural differences between specimens of the same species. On the other hand, the dataset contains an insufficient number of samples to train DCNNs. Here, we expand the analysis presented in (245) by including more methods that also tackle RGB images, different CNN and ViT architectures, better pre-training, and also our proposed methods. The results are dramatically improved, especially when using the proposed methods. We follow a very similar evaluation framework to the previous sections in this chapter, with the inclusion of results from the hand-engineered combined fractal features. (245) Since image resolution varies greatly, we use the default input sizes for the DCNNs and ViTs, while the original resolutions are used for the hand-engineered methods. The results are shown in Table 25.

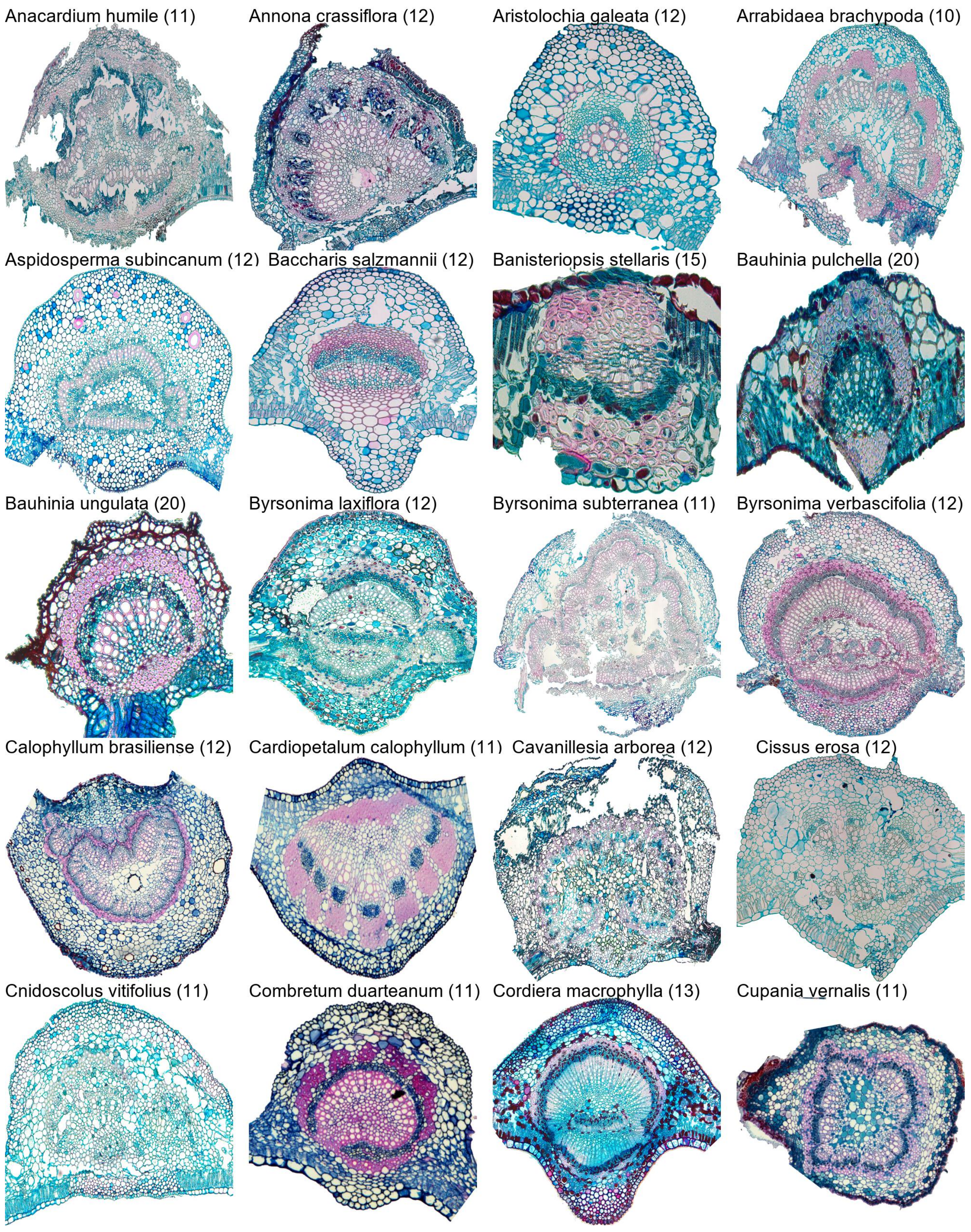

Anacardium humile (11)
Annona crassiflora (12)
Aristolochia galeata (12)
Arrabidaea brachypoda (10)
Aspidosperma subincanum (12)
Baccharis salzmannii (12)
Banisteriopsis stellaris (15)
Bauhinia pulchella (20)
Bauhinia ungulata (20)
Byrsonima laxiflora (12)
Byrsonima subterranea (11)
Byrsonima verbascifolia (12)
Calophyllum brasiliense (12)
Cardiopetalum calophyllum (11)
Cavanillesia arborea (12)
Cissus erosa (12)
Cnidoscolus vitifolius (11)
Combretum duarteanum (11)
Cordiera macrophylla (13)
Cupania vernalis (11)

Figure 40 – Samples from the first 20 species (in alphabetical order) from the leaf midrib microscopic images dataset, which contains 50 species in total. The name of the species is given above each image, and its number of samples is in parenthesis.

Source: SILVA *et al.* (245)

The result shows that the use of color is important for improving the performance of the methods (except for Gabor and InceptionV3). The DCNN performance between

Table 25 – Classification results of LDA for 50 Brazilian plant species characterized with feature extraction methods applied to microscopic images from the leaf midrib. Both the hand-engineered and DCNN-based descriptors are evaluated using 10 repetitions of 10-fold cross-validation and varying the image color space (grayscale or RGB). The result in blue highlight the best method from the literature, and red are our results matching or above that.

	Method	**Grayscale**	**RGB**
	Gabor	42.0±0.7	40.7±9.8
	BGP	55.8±5.9	74.1±4.9
	CLBP	43.1±6.2	58.2±11.9
	Combined fractal (245) †	83.7±0.7	-
	InceptionV3	98.5±1.4	97.5±1.5
	ResNet50	98.5±1.0	98.9±1.0
	ConvNeXt-nano	93.2±0.4	96.2±0.3
	ConvNeXt-T (in IN-21k)	98.4±0.2	99.2±0.1
	ConvNeXt-XL (in IN-21k)	99.1±0.1	99.6±0.0
	ViT-B/16 (in IN-21K)	93.3±0.4	96.5±0.2
	DeiT3-B/16	96.9±0.2	97.8±0.2
	DeiT3-B/16 (in IN-21K-FT1K)	92.3±0.4	94.7±0.2
Proposed methods	SSN_b	85.2±0.3	93.4±0.2
	SSR_b	96.8±0.3	98.6±0.3
	RADAM (ResNet50)	99.0±0.2	99.6±0.1
	RADAM (ConvNeXt-nano)	98.6±0.1	98.9±0.1
	RADAM (ConvNeXt-T in IN-21k)	99.1±0.1	99.6±0.1
	RADAM (ConvNeXt-XL in IN-21k)	99.5±0.1	**100**±0.0
	FR-DCNN (random ResNet50)	74.5±0.7	88.6±1.1

Source: By the author.

grayscale or RGB images varies very little despite the highest ones being achieved using colored information. On the other hand, the results of most traditional descriptors are significantly improved when using color. This happens due to the increase in the number of descriptors, as they are applied in an integrative way (combining descriptors from each channel) for the RGB images. However, even with this increase, these methods perform below the combined fractal technique, which uses only the image grayscale. It shows that beyond the importance of color, there are complex patterns on the leaf midrib that could be related to the shape and structural composition. Nevertheless, DCNNs and ViTs show the best results among the literature methods, surpassing the combined fractal descriptors by a considerable margin, where ConvNeXt-XL achieves the best results (99.6%). The superior performance of the deep-learning-based methods can be explained by their ability to automatically identify regions of interest in the image. Therefore, they have more flexibility to deal with different image variations, such as varying background and image sizes, aspect ratios, etc.

With the proposed methods, we achieve considerably superior performance in general. SSN and SSR consistently surpass the hand-engineered techniques, with SSR also overcoming the ViTs and some of the DCNNs. These results are expressive considering their simplified computational budget compared to DCNNs. FR-DCNN also surpasses the hand-engineered methods but performs below the pre-trained deep-learning models. This behavior suggests that ImageNet pre-training is highly beneficial for this task, corroborating its ability to deal with high variability in image structure. The highest results are achieved by RADAM using ResNet50 and ConvNeXt (T and XL), with the latter obtaining perfect classification accuracy and also the highest robustness (lowest standard deviation). These results highlight the effectiveness of the proposed methods in providing high discrimination power at different budgets by taking advantage of improved hand-engineered techniques (SSN and SSR) and expanding the power of pre-trained DCNNs (using RADAM).

Our findings show that it is possible to obtain perfect discrimination of the 50 plant species by analyzing only its leaf midrib using the proposed RADAM method. Our results then reinforce the quality of leaf midrib as a taxonomic factor and corroborate the power of the proposed methods in real-world applications. The improvement they represent compared to the previous result of 83.7% (245) proposed for this task is considerable. Therefore, we believe our methods are strong candidates for being applied in practice for plant species recognition, and that our results extend also to other species from different biomes, countries, etc. Moreover, they are flexible in terms of computational budget and offer different options according to specific needs and available resources.

7 CONCLUSION

AI is one of the hottest topics nowadays, either in the scientific community or in the industry. New and better AI tools are emerging faster than ever before, and bringing impressive advances to many fields. Therefore, understanding and improving the concepts behind these algorithms is of utmost importance. In this context, networks are essential mechanisms in current AI models, such as ANNs. In this thesis, we employ Network Science to explore patterns and randomness in AI networks used for CV tasks. Throughout the chapters, we have delved into the properties of graphs and neural networks following three main objectives: understanding and improving neural networks, building new CV techniques, and applying them to real-world problems.

Our results show how patterns emerge from randomness and how randomness can be explored to create more robust, adaptable, and faster neural networks. We also explored complex networks for modeling and characterizing images, and how they can be combined with neural networks for creating hybrid models with improved CV performance. In summary, the study has opened up new avenues for research and development in Network Science and neural networks, resulting in novel CV systems that achieve SOTA performance in a variety of tasks.

7.1 Overview of the main results

The first research objective resulted in exciting insights into how the ANN structure is related to its functioning. Fully-connected neural networks are studied as weighted graphs, where neurons are vertices and synapses are edges. For this analysis, we built an ANN dataset by considering different weight initialization, varying performance achieved throughout training, and different train domains in image classification tasks. We compute a set of neuronal centrality measurements to each network after training, describing the neuronal functioning on hidden layers. Significant patterns correlate these features with the neural network performance (image classification accuracy). These patterns show complex organizations and similar structures between neural networks of similar performance trained in different domains. We then propose an unsupervised clustering approach to group neurons with similar centrality properties, name BoN, and show how the neuron type distribution is directly related to the network performance. In other words, our results show that Network Science allows us to discriminate between ANNs with different performances by looking only at their final (after training) graph structure. These findings, presented in Section 3.1 of Chapter 3, help in understanding the importance of topology on performance and for developing new techniques for ANN construction, training, etc.

Building upon our findings about the structure and performance of ANNs, we develop new techniques for optimizing random weight initialization through Network Science. The neuronal strength is further analyzed for a variety of randomly initialized ANNs, and we uncover important connectivity patterns related to networks that train better and also provide a better final solution. One of the main factors responsible for this effect is the neuronal strength variance, where we show that minimizing it can improve the models significantly. A new method is then proposed to reorganize neuronal connections in order to minimize the strength variance, which we name PARw. The method is based on a well-known Network Science model (the scale-free model) and can be applied to any ANN structure, weight initialization, and task. The results are promising, where we observe statistically significant gains in training and testing performance for a variety of ANNs such as DCNNs and ViTs. The reader can find the description and results of this method in Section 3.2 of Chapter 3.

We also explore combinations of Network Science and ANNs in new CV techniques specifically designed for texture analysis. A novel hand-engineered method is developed for image modeling using complex networks, where each image pixel is a node and similar pixels are connected in a directed and weighted graph. We show how this network reflects the image patterns, and how Network-Science-based centrality can distinguish between different images. This method, named SSN, achieves SOTA results in a variety of texture analysis tasks by using a set of statistics from network centrality as image descriptors. To improve it, we also propose a randomized neural network as a tool for characterizing the centrality instead of using statistics, which resulted in another method named SSR. This new proposal merges the benefits of hand-engineered image representation using SSN with the learning capacity of neural networks and also achieves SOTA results on a variety of texture recognition tasks. The two methods are presented in Sections 4.1 and 4.2 of Chapter 4.

Also within texture analysis, we developed two new approaches using deep neural networks. In contrast to SSN and SSR, which falls more in the hand-engineered paradigm, using deep models demands considerably more computing resources but can achieve superior performance on a wider variety of tasks by taking advantage of pre-training. The first method we propose following this approach, named RADAM, performs deep feature aggregation and encoding in pre-trained DCNNs. It uses randomized autoencoders for pixel-wise randomized encoding from deep activation maps that are aggregated across multiple depths of the DCNN. Afterwards, only a linear classifier is trained over the aggregated and encoded features, and SOTA results are achieved on various tasks even comparing to methods that fine-tune the DCNNs. Another method is developed by combining PARw and RADAM applied over fully randomized DCNNs, where we show that it can provide useful features for texture analysis and even surpass the use of pre-trained DCNNs in some cases, achieving SOTA results on some tasks. These results are surprising and show that DCNNs with random weights can be a viable alternative in some cases and that our approach is promising for evaluating DCNN architectures. These methods are described in Sections 4.3 and 4.4 of Chapter 4;

In Chapter 5 we present an evaluation of several methods on ten texture recognition benchmark datasets. We first compare hand-engineered methods using different linear classifiers and then compare image features obtained with DCNNs and their efficiency (relation between cost and performance). Afterwards, we compare them with the proposed methods in two different paradigms: pure-texture datasets, with more controlled images, and in-the-wild datasets with images from the internet or obtained without any control. All our methods achieve SOTA performance in the first scenario, surpassing a variety of methods from the literature. In the second scenario, which is considerably more complex, our hand-engineered methods (SSN and SSR) and fully randomized DCNNs struggle in comparison to deep learning models with strong pre-training and fine-tuning. On the other hand, RADAM achieves SOTA results in this case as well, offering various cost alternatives according to different backbones while not needing fine-tuning, which makes it considerably more efficient.

Lastly, Chapter 6 presents four innovative real-world problems that we solve through CV, where we compare the performance of our proposed methods with the literature. The first part of the chapter introduces CV-assisted diagnosis of diseases by using SEM images from genosensors. Two diagnosis tasks are explored, the detection of prostate cancer and COVID-19. In both cases, the proposed methods surpass all the compared methods, offering a high level of precision at different computational budgets that can be explored for mobile applications, embedded systems, etc. In the second part of the chapter we discuss the use of CV in plant sciences, and two tasks are explored: the detection of pollutant levels on the leaves of the Jacaranda Caroba tree, and the classification of plant species through microscopic images of the leaf midrib. Again, we show that the proposed methods outperform the techniques from the literature on both tasks, proving to be powerful tools in this field as well.

7.2 Scientific output generated during the doctorate

Considering the aforementioned results and throughout the period of this doctorate research, considerable literature production was achieved. In this section, we list all works derived from this doctoral research, directly or indirectly, or results from collaborations in related fields and applications of the studied methods and techniques. The list below concerns all the already published works (a total of 16) in this period:

- Machicao, J., Ribas, L. C., Scabini, L. F., & Bruno, O. M. (2018). Cellular automata rule characterization and classification using texture descriptors. Physica A: Statistical Mechanics and its Applications.

- Cantero, S. V. A. B., Goncalves, D. N., dos Santos Scabini, L. F., & Goncalves, W. N. (2018). Importance of vertices in complex networks applied to texture analysis. IEEE Transactions on Cybernetics.

- Farfán, A. J., Scabini, L. F., & Bruno, O. M. (2019). A Web-Based System to Assess Texture Analysis Methods and Datasets. In Computer Analysis of Images and Patterns: 18th International Conference, CAIP, Salerno, Italy. Springer International Publishing.

- Scabini, L. F., Condori, R. H., Ribas, L. C., & Bruno, O. M. (2019). Evaluating deep convolutional neural networks as texture feature extractors. In Image Analysis and Processing–ICIAP 2019: 20th International Conference, Trento, Italy. Springer International Publishing.

- Scabini, L. F., Condori, R. H., Gonçalves, W. N., & Bruno, O. M. (2019). Multilayer complex network descriptors for color–texture characterization. Information Sciences.

- Scabini, L. F., Condori, R. M., Munhoz, I. C., & Bruno, O. M. (2019, August). Deep Convolutional Neural Networks for Plant Species Characterization Based on Leaf Midrib. In Computer Analysis of Images and Patterns: 18th International Conference, CAIP 2019, Salerno, Italy. Springer International Publishing.

- Scabini, L. F., Ribas, L. C., & Bruno, O. M. (2020). Spatio-spectral networks for color-texture analysis. Information Sciences.

- Ribas, L. C., Junior, J. J. D. M. S., Scabini, L. F., & Bruno, O. M. (2020). Fusion of complex networks and randomized neural networks for texture analysis. Pattern Recognition.

- Soares, J. C., Soares, A. C., Rodrigues, V. C., Oiticica, P. R. A., Raymundo-Pereira, P. A., Bott-Neto, J. L., Buscaglia, L. A., de Castro, L. D. C., Ribas, L. C., Scabini, L., Brazaca, L., Correa, D. S., Mattoso, L. H. C., Oliveira, M. C. F., Carvalho, A. C. P. L. F., Carrilho, E., Bruno, O. M., & Oliveira, O. N. (2021). Detection of a SARS-CoV-2 sequence with genosensors using data analysis based on information visualization and machine learning techniques. Materials Chemistry Frontiers.

- Rodrigues, V. C., Soares, J. C., Soares, A. C., Braz, D. C., Melendez, M. E., Ribas, L. C., Scabini, L.; Bruno, O. M.; Carvalho, A. L.; Reis, R. M., Sanfelice, R. C., & Oliveira Jr, O. N. (2021). Electrochemical and optical detection and machine learning applied to images of genosensors for diagnosis of prostate cancer with the biomarker PCA3. Talanta.

- Scabini, L. F., Ribas, L. C., Neiva, M. B., Junior, A. G., Farfan, A. J., & Bruno, O. M. (2021). Social interaction layers in complex networks for the dynamical epidemic modeling of COVID-19 in Brazil. Physica A: Statistical Mechanics and its Applications.
- Scabini, L., Ribas, L., Ribeiro, E., & Bruno, O. (2022, February). Deep Topological Embedding with Convolutional Neural Networks for Complex Network Classification. In Network Science: 7th International Winter Conference, NetSci-X 2022, Porto, Portugal. Springer International Publishing.
- Zielinski, K. M., Ribas, L. C., Scabini, L. F., & Bruno, O. M. (2022, April). Complex Texture Features Learned by Applying Randomized Neural Network on Graphs. In 2022 Eleventh International Conference on Image Processing Theory, Tools and Applications, IPTA, Salzburg, Austria. IEEE.
- Ribas, L. C., Scabini, L., & Bruno, O. M. (2022, April). A complex network approach for fish species recognition based on otolith shape. In 2022 Eleventh International Conference on Image Processing Theory, Tools and Applications, IPTA, Salzburg, Austria. IEEE.
- de Castro, L. D., Scabini, L., Ribas, L. C., Bruno, O. M., & Oliveira Jr, O. N. (2023). Machine learning and image processing to monitor strain and tensile forces with mechanochromic sensors. Expert Systems with Applications.
- Scabini, L. F., & Bruno, O. M. (2023). Structure and performance of fully connected neural networks: Emerging complex network properties. Physica A: Statistical Mechanics and its Applications.

Aside from the already published works, we also list papers that are under review or in the final writing phase (a total of six):

- Ribas, L. C., Scabini, L. F., Junior, J. J. D. M. S., & Bruno, O. M. (2020). Learning Local Complex Features using Randomized Neural Networks for Texture Analysis. arXiv preprint arXiv:2007.05643.
- Ribas, L. C., Scabini, L. F., Condori, R. H., & Bruno, O. Spatio-Spectral Representation Learning for Color-Texture Classification. Available at SSRN 4266882.
- Scabini, L., De Baets, B., & Bruno, O. M. (2022). Improving Deep Neural Network Random Initialization Through Neuronal Rewiring. arXiv preprint arXiv:2207.08148.
- Scabini, L., Zielinski, K. M., Ribas, L. C., Gonçalves, W. N., De Baets, B., & Bruno, O. M. (2023). RADAM: Texture Recognition through Randomized Aggregated Encoding of Deep Activation Maps. arXiv preprint arXiv:2303.04554.

- Scabini, L., Ribas, L. C., Grosso, A., Fernandes, M. B. S. D., Bruno, O. M.. Network Science for Hyperspectral Image Analysis: Characterizing the Jacaranda Caroba Leaf Plasticity for Environmental Biosensing.
- Borzooei, S., Scabini, L., Daneshgar, S., Deblieck, L., Cornelissen, R., Broeck, E., De Langhe, P., Bruno, O., De Baets, B., Nopens, I., Torfs, E.. Prediction of activated sludge settling characteristics in wastewater treatment plants using deep convolutional networks applied over microscopy images.

7.3 Future works

This thesis presented a variety of methods and results in the fields of ANNs and CV. The obtained results surpassed SOTA methods on several tasks and open new directions for future research. Firstly, considering the study of the complex network properties of ANNs, one approach is to consider them dynamically, *i.e.*, to understand how they evolve during training. Another interesting direction would be analyzing the network stability, such as the relation between adversarial attacks with their structure. Other future directions are to extend the analysis to more architectures, tasks (such as regression), and the optimization of more elaborated Network Science measures for efficient computation on deeper ANNs (using GPUs). These findings can then be applied to a variety of practical tools, such as network pruning, neural architecture search, improved training mechanisms, etc.

Considering our new ANN initialization technique (PARw), we believe that the organization of the weights is an important aspect of random initialization that should be further investigated. One interesting direction for future work is to explore other Network Science measures and tools, once more efficient implementations of them are available. The proposed PA rewiring method can also be explored in different ways, for instance by using different levels of rewiring (instead of rewiring the whole layer), and/or applying it heterogeneously in different layers. Another interesting idea is to analyze the potential use of PARw as a regularization technique during training, such as applying it after every couple of epochs to overcome local minima and reduce overfitting. In summary, deep ANN architectures are complex and dynamic systems, and their graph structure should be further investigated.

On texture analysis, our proposed works also offer a variety of potential future research directions. Firstly, the most obvious choice is the use of PARw in the randomized neural networks employed by SSR and RADAM. We performed some initial tests, and some gains were observed, but the networks we employed are very small since we gave priority to efficiency. Nevertheless, this should be further investigated according to the network size and properties, and we believe that significant gains can be achieved. Another potential candidate for future research is the use of random DCNNs and their combination with PARw and RADAM. This was our last research development during the doctorate since it employs two of our previous methods, and we believe more attention should be given to it. For instance, further research should focus on understanding the underlying reasons for the success of the random deep networks and their combination with PARw and RADAM, analyzing different random initializers, and weight magnitudes, and expanding the idea to different architectures (*e.g.*, ViTs).

In conclusion, this thesis has demonstrated the power and potential of integrating Network Science with ANNs, particularly in the field of CV. The research conducted has not only deepened our understanding of the intricate relationship between network structure and performance, but it has also led to the development of novel methods and techniques that have achieved SOTA results on various tasks. The exploration of randomness and patterns in AI networks has opened up new possibilities for creating more robust and efficient methods, creating several new directions for future research. Furthermore, the application of these findings in real-world problems, such as disease diagnosis and plant sciences, has shown the practical and impactful nature of this research. As we move forward, we believe that the fusion of Network Science and AI holds great promise for the future. The insights and methods presented in this thesis provide a solid foundation for further research in this exciting and rapidly evolving field. We hope this work contributes to pushing the boundaries of what is possible in CV and Network Science, and that future research on the subject could advance technology and its benefits for society.

REFERENCES

1 WATTS, D. J.; STROGATZ, S. H. Collective dynamics of ‘small-world’ networks. **Nature**, Nature Publishing Group, v. 393, n. 6684, p. 440–442, 1998.

2 BARABÁSI, A.-L.; ALBERT, R. Emergence of scaling in random networks. **Science**, American Association for the Advancement of Science, v. 286, n. 5439, p. 509–512, 1999.

3 KRIZHEVSKY, A.; SUTSKEVER, I.; HINTON, G. E. Imagenet classification with deep convolutional neural networks. **Communications of the ACM**, AcM New York, NY, USA, v. 60, n. 6, p. 84–90, 2017.

4 GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A. **Deep learning**. Massachusetts: MIT Press, 2016.

5 NAJAFABADI, M. M. *et al.* Deep learning applications and challenges in big data analytics. **Journal of Big Data**, Nature Publishing Group, v. 2, n. 1, p. 1, 2015.

6 BALDI, P.; SADOWSKI, P.; WHITESON, D. Searching for exotic particles in high-energy physics with deep learning. **Nature Communications**, Nature Publishing Group, v. 5, p. 4308, 2014.

7 GÜÇLÜ, U.; GERVEN, M. A. van. Deep neural networks reveal a gradient in the complexity of neural representations across the ventral stream. **Journal of Neuroscience**, Soc Neuroscience, v. 35, n. 27, p. 10005–10014, 2015.

8 CASTELVECCHI, D. Can we open the black box of ai? **Nature News**, v. 538, n. 7623, p. 20, 2016.

9 FRANKLE, J.; CARBIN, M. **The lottery ticket hypothesis**: finding sparse, trainable neural networks. 2018. Available at: https://arxiv.org/pdf/1803.03635.pdf. Access at: 20 Jan. 2019.

10 ZHOU, H. *et al.* **Deconstructing lottery tickets:** zeros, signs, and the supermask. 2019. Available at: https://proceedings.neurips.cc/paper_files/paper/2019/file/1113d7a76ffceca1bb350bfe145467c6-Paper.pdf. Access at: 20 Feb. 2021.

11 RAMANUJAN, V. *et al.* What’s hidden in a randomly weighted neural network? *In*: CONFERENCE ON COMPUTER VISION AND PATTERN RECOGNITION, 2020, Seattle. **Proceedings [...]**. Seattle: IEEE, 2020. p. 11893–11902.

12 SAXE, A. M. *et al.* On random weights and unsupervised feature learning. *In*: INTERNATIONAL CONFERENCE ON MACHINE LEARNING, 2011, Bellevue. **Proceedings [...]**. Bellevue: ACM, 2011. p. 1089–1096.

13 LI, Y.; LIANG, Y. Learning overparameterized neural networks via stochastic gradient descent on structured data. *In*: ADVANCES IN NEURAL INFORMATION PROCESSING SYSTEMS, 2018, Montreal. **Proceedings [...]**. Montreal: NIPS, 2018.

14 JESUS, R. J. *et al.* Effect of initial configuration of weights on training and function of artificial neural networks. **Mathematics**, Multidisciplinary Digital Publishing Institute, v. 9, n. 18, p. 2246, 2021.

15 CHALUMEAU, T. *et al.* Optimized texture classification by using hierarchical complex network measurements. **Proceedings of the SPIE**, v. 6070, p. 60700Q, 2006. DOI: 10.1117/12.655592.

16 BACKES, A. R.; CASANOVA, D.; BRUNO, O. M. Texture analysis and classification: a complex network-based approach. **Information Sciences**, Elsevier, v. 219, p. 168–180, 2013. DOI: 10.1016/j.ins.2012.07.003.

17 SÁ JUNIOR, J. J. M.; CORTEZ, P. C.; BACKES, A. R. Color texture classification using shortest paths in graphs. **IEEE Transactions on Image Processing**, IEEE, v. 23, n. 9, p. 3751–3761, 2014.

18 CANTERO, S. V. A. B. *et al.* Importance of vertices in complex networks applied to texture analysis. **IEEE Transactions on Cybernetics**, IEEE, v. 50, n. 2, p. 777–786, 2018.

19 SCABINI, L. F. *et al.* Multilayer complex network descriptors for color–texture characterization. **Information Sciences**, Elsevier, v. 491, p. 30–47, 2019. DOI: 10.1016/j.ins.2019.02.060.

20 BARABÁSI, A.-L. *et al.* **Network science**. Cambridge: Cambridge University Press, 2016.

21 BIANCONI, G.; PIN, P.; MARSILI, M. Assessing the relevance of node features for network structure. **Proceedings of the National Academy of Sciences**, National Academy of Sciences, v. 106, n. 28, p. 11433–11438, 2009.

22 ERDOS, P.; RÉNYI, A. On random graphs i. **Publicationes Mathematicae**, v. 6, p. 290–297, 1959.

23 ERDOS, P.; RÉNYI, A. On the evolution of random graphs. **Publication of the Mathematical Institute of the Hungarian Academy of Sciences**, Citeseer, v. 5, p. 17–61, 1960.

24 COSTA, L. *et al.* Analyzing and modeling real-world phenomena with complex networks: a survey of applications. **Advances in Physics**, Taylor & Francis, v. 60, n. 3, p. 329–412, 2011.

25 BARABASI, A.-L.; OLTVAI, Z. N. Network biology: understanding the cell's functional organization. **Nature Reviews Genetics**, Nature Publishing Group UK London, v. 5, n. 2, p. 101–113, 2004.

26 GUIMERA, R.; AMARAL, L. A. N. Functional cartography of complex metabolic networks. **Nature**, Nature Publishing Group, v. 433, n. 7028, p. 895–900, 2005.

27 RUBINOV, M.; SPORNS, O. Complex network measures of brain connectivity: uses and interpretations. **Neuroimage**, Elsevier, v. 52, n. 3, p. 1059–1069, 2010.

28 GIRVAN, M.; NEWMAN, M. E. Community structure in social and biological networks. **Proceedings of the National Academy of Sciences**, National Academy of Sciences, v. 99, n. 12, p. 7821–7826, 2002.

29 NEWMAN, M. E.; PARK, J. Why social networks are different from other types of networks. **Physical Review E**, APS, v. 68, n. 3, p. 036122, 2003.

30 SCABINI, L. F. *et al.* Social interaction layers in complex networks for the dynamical epidemic modeling of covid-19 in brazil. **Physica A**, Elsevier, v. 564, p. 125498, 2021. DOI: 10.1016/j.physa.2020.125498.

31 COSTA, L. d. F. *et al.* Characterization of complex networks: a survey of measurements. **Advances in Physics**, Taylor & Francis, v. 56, n. 1, p. 167–242, 2007.

32 BARRAT, A. *et al.* The architecture of complex weighted networks. **Proceedings of the National Academy of Sciences**, National Academy of Sciences, v. 101, n. 11, p. 3747–3752, 2004.

33 KERMARREC, A.-M. *et al.* Second order centrality: distributed assessment of nodes criticity in complex networks. **Computer Communications**, Elsevier, v. 34, n. 5, p. 619–628, 2011.

34 ESTRADA, E.; RODRIGUEZ-VELAZQUEZ, J. A. Subgraph centrality in complex networks. **Physical Review E**, APS, v. 71, n. 5, p. 056103, 2005.

35 BRON, C.; KERBOSCH, J. Algorithm 457: finding all cliques of an undirected graph. **Communications of the ACM**, ACM New York, NY, USA, v. 16, n. 9, p. 575–577, 1973.

36 CAZALS, F.; KARANDE, C. A note on the problem of reporting maximal cliques. **Theoretical Computer Science**, Elsevier, v. 407, n. 1-3, p. 564–568, 2008.

37 LATAPY, M.; MAGNIEN, C.; VECCHIO, N. D. Basic notions for the analysis of large two-mode networks. **Social Networks**, Elsevier, v. 30, n. 1, p. 31–48, 2008.

38 BOLDI, P.; VIGNA, S. Axioms for centrality. **Internet Mathematics**, Taylor & Francis, v. 10, n. 3-4, p. 222–262, 2014.

39 DIJKSTRA, E. W. *et al.* A note on two problems in connexion with graphs. **Numerische Mathematik**, v. 1, n. 1, p. 269–271, 1959.

40 BRANDES, U.; FLEISCHER, D. Centrality measures based on current flow. *In*: SYMPOSIUM ON THEORETICAL ASPECTS OF COMPUTER SCIENCE, 2005, Stuttgart. **Proceedings [...]**. Stuttgart: Springer, 2005. p. 533–544.

41 CAJAL, S. Ramón y. **Textura del sistema nervioso del hombre y de los vertebrados**. Madrid: Moya, 1899.

42 PANDYA, S. *et al.* The beautiful brain: the drawings of santiago ramon y cajal. **Neurology India**, Medknow, v. 65, n. 4, p. 934, 2017.

43 PLACE, U. T. Is consciousness a brain process? **British Journal of Psychology**, Wiley Online Library, v. 47, n. 1, p. 44–50, 1956.

44 MCCULLOCH, W. S.; PITTS, W. A logical calculus of the ideas immanent in nervous activity. **The Bulletin of Mathematical Biophysics**, Springer, v. 5, n. 4, p. 115–133, 1943.

45 ROSENBLATT, F. The perceptron: a probabilistic model for information storage and organization in the brain. **Psychological Review**, American Psychological Association, v. 65, n. 6, p. 386, 1958.

46 MINSKY, M.; PAPERT, S. A. **Perceptrons**: an introduction to computational geometry. Massachusetts: MIT Press, 2017.

47 BISHOP, C. M. *et al.* **Neural networks for pattern recognition**. Oxford: Oxford University Press, 1995.

48 RUMELHART, D. E.; HINTON, G. E.; WILLIAMS, R. J. Learning representations by back-propagating errors. **Nature**, Nature Publishing Group UK London, v. 323, n. 6088, p. 533–536, 1986.

49 HINTON, G. E.; OSINDERO, S.; TEH, Y.-W. A fast learning algorithm for deep belief nets. **Neural Computation**, MIT Press, v. 18, n. 7, p. 1527–1554, 2006.

50 RAINA, R.; MADHAVAN, A.; NG, A. Y. Large-scale deep unsupervised learning using graphics processors. *In*: INTERNATIONAL CONFERENCE ON MACHINE LEARNING, 2009. **Proceedings [...]**. Montreal: ACM, 2009. p. 873–880.

51 LECUN, Y. *et al.* Backpropagation applied to handwritten zip code recognition. **Neural Computation**, MIT Press, v. 1, n. 4, p. 541–551, 1989.

52 HUBEL, D. H.; WIESEL, T. N. Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. **The Journal of Physiology**, Wiley Online Library, v. 160, n. 1, p. 106–154, 1962.

53 FELLEMAN, D. J.; VAN, D. E. Distributed hierarchical processing in the primate cerebral cortex. **Cerebral Cortex**, v. 1, n. 1, p. 1–47, 1991.

54 DENG, J. *et al.* Imagenet: A large-scale hierarchical image database. *In*: CONFERENCE ON COMPUTER VISION AND PATTERN RECOGNITION. 2009. **Proceedings [...]**. Miami: IEEE, 2009. p. 248–255.

55 RUSSAKOVSKY, O. *et al.* Imagenet large scale visual recognition challenge. **International Journal of Computer Vision**, Springer, v. 115, n. 3, p. 211–252, 2015.

56 ALZUBAIDI, L. *et al.* Review of deep learning: concepts, cnn architectures, challenges, applications, future directions. **Journal of Big Data**, Springer, v. 8, p. 1–74, 2021.

57 KHAN, A. *et al.* A survey of the recent architectures of deep convolutional neural networks. **Artificial Intelligence Review**, Springer, v. 53, n. 8, p. 5455–5516, 2020.

58 HAN, K. *et al.* A survey on vision transformer. **IEEE Transactions on Pattern Analysis and Machine Intelligence**, IEEE, v. 45, p. 87–110, 2022. DOI: 10.1109/TPAMI.2022.3152247.

59 SZEGEDY, C. *et al.* Going deeper with convolutions. *In*: CONFERENCE ON COMPUTER VISION AND PATTERN RECOGNITION, 2015, Boston. **Proceedings [...]**. Boston: IEEE, 2015.

60 SZEGEDY, C. *et al.* Inception-v4, inception-resnet and the impact of residual connections on learning. *In*: CONFERENCE ON ARTIFICIAL INTELLIGENCE, 2017, San Francisco. **Proceedings [...]**. San Francisco: AAAI Press, 2017.

61 HE, K. *et al.* Deep residual learning for image recognition. *In*: CONFERENCE ON COMPUTER VISION AND PATTERN RECOGNITION, 2016, Las Vegas. **Proceedings [...]**. Las Vegas: IEEE, 2016. p. 770–778.

62 TOLSTIKHIN, I. O. *et al.* Mlp-mixer: An all-mlp architecture for vision. **Advances in Neural Information Processing Systems**, MIT Press, v. 34, p. 24261–24272, 2021.

63 LIU, H. *et al.* Pay attention to mlps. **Advances in Neural Information Processing Systems**, MIT Press, v. 34, p. 9204–9215, 2021.

64 DOSOVITSKIY, A. *et al.* **An image is worth 16x16 words**: transformers for image recognition at scale. 2020. Available at: https://arxiv.org/abs/2010.11929. Access at: 20 June 2021.

65 TOUVRON, H.; CORD, M.; JÉGOU, H. Deit iii: revenge of the vit. *In*: EUROPEAN CONFERENCE ON COMPUTER VISION, 2022, Tel Aviv. **Proceedings [...]**. Tel Aviv: Springer, 2022. p. 516–533.

66 VASWANI, A. *et al.* Attention is all you need. *In*: CONFERENCE ON NEURAL INFORMATION PROCESSING SYSTEMS, 2017, Long Beach. **Proceedings [...]**. Long Beach: MIT Press, 2017.

67 LIU, Z. *et al.* A convnet for the 2020s. *In*: CONFERENCE ON COMPUTER VISION AND PATTERN RECOGNITION, 2022, New Orleans. **Proceedings [...]**. New Orleans: IEEE / CVF, 2022. p. 11976–11986.

68 GLOROT, X.; BORDES, A.; BENGIO, Y. Deep sparse rectifier neural networks. *In*: INTERNATIONAL CONFERENCE ON ARTIFICIAL INTELLIGENCE AND STATISTICS, 2011, Lauderdale. **Proceedings [...]**. Lauderdale: JMLR, 2011. p. 315–323.

69 GLOROT, X.; BENGIO, Y. Understanding the difficulty of training deep feedforward neural networks. *In*: INTERNATIONAL CONFERENCE ON ARTIFICIAL INTELLIGENCE AND STATISTICS, 2010, Sardinia. **Proceedings [...]**. Sardinia: JMLR, 2010. p. 249–256.

70 SUTSKEVER, I. *et al.* On the importance of initialization and momentum in deep learning. *In*: INTERNATIONAL CONFERENCE ON MACHINE LEARNING, 2013, Atlanta. **Proceedings [...]**. Atlanta: ACM, 2013. p. 1139–1147.

71 LOSHCHILOV, I.; HUTTER, F. **Decoupled weight decay regularization**. 2017. Available at: https://arxiv.org/pdf/1711.05101.pdf. Access at: 20 Jan. 2019.

72 ZEILER, M. D. **Adadelta**: an adaptive learning rate method. 2012. Available at: https://arxiv.org/pdf/1212.5701.pdf. Access at: 25 Jan. 2019.

73 ZOPH, B. *et al.* Learning transferable architectures for scalable image recognition. *In*: CONFERENCE ON COMPUTER VISION AND PATTERN RECOGNITION, 2018, Salt Lake. **Proceedings [...]**. Salt Lake: IEEE / CVF, 2018. p. 8697–8710.

74 MIIKKULAINEN, R. *et al.* Evolving deep neural networks. *In*: KOZMA, R. *et al.* (ed.). **Artificial Intelligence in the Age of Neural Networks and Brain Computing**. Cambridge: Academic Press, 2019. p. 293–312.

75 GOODFELLOW, I. J.; SHLENS, J.; SZEGEDY, C. **Explaining and harnessing adversarial examples**. 2014. Available at: https://arxiv.org/pdf/1412.6572.pdf. Access at: 22 Jan. 2019.

76 NARKHEDE, M. V.; BARTAKKE, P. P.; SUTAONE, M. S. A review on weight initialization strategies for neural networks. **Artificial Intelligence Review**, Springer, p. 1–32, 2021. DOI: 10.1007/s10462-021-10033-z.

77 HINTON, G. E.; SALAKHUTDINOV, R. R. Reducing the dimensionality of data with neural networks. **Science**, American Association for the Advancement of Science, v. 313, n. 5786, p. 504–507, 2006.

78 BENGIO, Y. *et al.* Greedy layer-wise training of deep networks. *In*: NEURAL INFORMATION PROCESSING SYSTEMS, 2007, Vancouver. **Proceedings [...]**. Vancouver: MIT Press, 2007. p. 153–160.

79 HE, K. *et al.* Delving deep into rectifiers: surpassing human-level performance on imagenet classification. *In*: INTERNATIONAL CONFERENCE ON COMPUTER VISION, 2015, Santiago. **Proceedings [...]**. Santiago: IEEE, 2015. p. 1026–1034.

80 KLAMBAUER, G. *et al.* Self-normalizing neural networks. *In*: CONFERENCE ON NEURAL INFORMATION PROCESSING SYSTEMS, 2017, Long Beach. **Proceedings [...]**. Long Beach: MIT Press, 2017. p. 972–981.

81 PASZKE, A. *et al.* Pytorch: an imperative style, high-performance deep learning library. *In*: CONFERENCE ON NEURAL INFORMATION PROCESSING SYSTEMS, 2019, Vancouver. **Proceedings [...]**. Vancouver: MIT Press, 2019.

82 SAXE, A. M.; MCCLELLAND, J. L.; GANGULI, S. **Exact solutions to the nonlinear dynamics of learning in deep linear neural networks**. 2013. Available at: https://arxiv.org/pdf/1312.6120.pdf. Access at: 20 Jan. 2019.

83 HU, W.; XIAO, L.; PENNINGTON, J. **Provable benefit of orthogonal initialization in optimizing deep linear networks**. 2020. Available at: https://arxiv.org/pdf/2001.05992.pdf. Access at: 15 Jan. 2021.

84 SUSSILLO, D.; ABBOTT, L. **Random walk initialization for training very deep feedforward networks**. 2014. Available at: https://arxiv.org/pdf/1412.6558.pdf. Access at: 20 Jan. 2019.

85 HENDRYCKS, D.; GIMPEL, K. **Adjusting for dropout variance in batch normalization and weight initialization**. 2016. Available at: https://arxiv.org/pdf/1607.02488.pdf. Access at: 20 Jan. 2019.

86 SKORSKI, M.; TEMPERONI, A.; THEOBALD, M. Revisiting weight initialization of deep neural networks. **Proceedings of Machine Learning Research**, v. 157, p. 1192–1207, 2021.

87 MISHKIN, D.; MATAS, J. **All you need is a good init**. 2015. Available at: https://arxiv.org/pdf/1511.06422.pdf. Access at: 20 Jan. 2020.

88 SALIMANS, T.; KINGMA, D. P. Weight normalization: a simple reparameterization to accelerate training of deep neural networks. *In*: CONFERENCE ON NEURAL INFORMATION PROCESSING SYSTEMS, 2019, Vancouver. **Proceedings [...]**. Vancouver: MIT Press, 2016. p. 901–909.

89 FRANKLE, J.; SCHWAB, D. J.; MORCOS, A. S. **The early phase of neural network training**. 2020. Available at: https://arxiv.org/pdf/2002.10365.pdf. Access at: 20 Jan. 2021.

90 SCHMIDT, W. F.; KRAAIJVELD, M. A.; DUIN, R. P. W. Feedforward neural networks with random weights. *In*: INTERNATIONAL CONFERENCE ON PATTERN RECOGNITION, 1992, The Hague. **Proceedings [...]**. The Hague: IAPR, 1992. p. 1–4.

91 PAO, Y.-H.; TAKEFUJI, Y. Functional-link net computing: theory, system architecture, and functionalities. **Computer**, IEEE, v. 25, n. 5, p. 76–79, 1992.

92 PAO, Y.-H.; PARK, G.-H.; SOBAJIC, D. J. Learning and generalization characteristics of the random vector functional-link net. **Neurocomputing**, Elsevier, v. 6, n. 2, p. 163–180, 1994.

93 HUANG, G.-B.; ZHU, Q.-Y.; SIEW, C.-K. Extreme learning machine: theory and applications. **Neurocomputing**, Elsevier, v. 70, n. 1, p. 489–501, 2006.

94 JARRETT, K. *et al.* What is the best multi-stage architecture for object recognition? *In*: INTERNATIONAL CONFERENCE ON COMPUTER VISION, 2009, Kyoto. **Proceedings [...]**. Kyoto: IEEE, 2009. p. 2146–2153.

95 COX, D.; PINTO, N. Beyond simple features: a large-scale feature search approach to unconstrained face recognition. *In*: INTERNATIONAL CONFERENCE ON AUTOMATIC FACE & GESTURE RECOGNITION, 2011, Santa Barbara. **Proceedings [...]**. Santa Barbara: IEEE, 2011. p. 8–15.

96 PARK, S. K.; MILLER, K. W. Random number generators: good ones are hard to find. **Communications of the ACM**, ACM, v. 31, n. 10, p. 1192–1201, 1988.

97 MOORE, E. H. On the reciprocal of the general algebraic matrix. **Bulletin of the American Mathematical Society**, v. 26, p. 394–395, 1920.

98 PENROSE, R. A generalized inverse for matrices. **Mathematical Proceedings of the Cambridge Philosophical Society**, Cambridge University Press, v. 51, n. 3, p. 406–413, 1955.

99 CAMBRIA, E. *et al.* Extreme learning machines [trends & controversies]. **IEEE Intelligent Systems**, IEEE, v. 28, n. 6, p. 30–59, 2013.

100 GRILL-SPECTOR, K.; MALACH, R. The human visual cortex. **Annual Review of Neuroscience**, Annual Reviews, v. 27, p. 649–677, 2004. DOI: 10.1146/annurev.neuro.27.070203.144220.

101 LIU, L. *et al.* From bow to cnn: two decades of texture representation for texture classification. **International Journal of Computer Vision**, Springer, v. 127, n. 1, p. 74–109, 2019.

102 HUMEAU-HEURTIER, A. Texture feature extraction methods: a survey. **IEEE Access**, IEEE, v. 7, p. 8975–9000, 2019. DOI: 10.1109/ACCESS.2018.2890743.

103 PIETIKÄINEN, M.; OJALA, T. Texture analysis in industrial applications. **Image Technology**, Springer, p. 337–359, 1996. DOI: 10.1007/978-3-642-58288-2_13.

104 KASSNER, A.; THORNHILL, R. Texture analysis: a review of neurologic mr imaging applications. **American Journal of Neuroradiology**, Am Soc Neuroradiology, v. 31, n. 5, p. 809–816, 2010.

105 DANA, K. J. *et al.* Reflectance and texture of real-world surfaces. **ACM Transactions on Graphics**, v. 18, n. 1, p. 1–34, 1999.

106 OJALA, T. *et al.* Outex-new framework for empirical evaluation of texture analysis algorithms. *In*: INTERNATIONAL CONFERENCE ON PATTERN RECOGNITION, 2002, Quebec. **Proceedings [...]**. Quebec: IEEE, 2002. p. 701–706.

107 CAPUTO, B.; HAYMAN, E.; MALLIKARJUNA, P. Class-specific material categorisation. *In*: NTERNATIONAL CONFERENCE ON COMPUTER VISION, 2005, Beijing. **Proceedings [...]**. Beijing: IEEE, 2005. p. 1597–1604.

108 MALLIKARJUNA, P. *et al.* **The kth-tips2 database**. 2006. Available at: https://www.csc.kth.se/cvap/databases/kth-tips/kth-tips2.pdf. Access at: 20 Jan. 2019.

109 SHARAN, L.; ROSENHOLTZ, R.; ADELSON, E. Material perception: what can you see in a brief glance? **Journal of Vision**, The Association for Research in Vision and Ophthalmology, v. 9, n. 8, p. 784–784, 2009.

110 PICKARD, R. *et al.* **VisTex vision texture database**. 1995. Available at: http://vismod.media.mit.edu/vismod/imagery/VisionTexture/vistex.html. Access at: 2 nov. 2019.

111 BACKES, A. R.; CASANOVA, D.; BRUNO, O. M. Color texture analysis based on fractal descriptors. **Pattern Recognition**, Elsevier, v. 45, n. 5, p. 1984–1992, 2012.

112 XUE, J.; ZHANG, H.; DANA, K. Deep texture manifold for ground terrain recognition. *In*: CONFERENCE ON COMPUTER VISION AND PATTERN RECOGNITION, 2018, Salt Lake. **Proceedings [...]**. Salt Lake: IEEE / CVF, 2018.

113 XUE, J. *et al.* Differential viewpoints for ground terrain material recognition. **IEEE Transactions on Pattern Analysis and Machine Intelligence**, IEEE, v. 44, n. 3, p. 1205–1218, 2020. DOI: 10.1109/TPAMI.2020.3025121.

114 CIMPOI, M. *et al.* Describing textures in the wild. *In*: CONFERENCE ON COMPUTER VISION AND PATTERN RECOGNITION, 2014, Columbus. **Proceedings [...]**. Columbus: IEEE, 2014. p. 3606–3613.

115 ABDELMOUNAIME, S.; DONG-CHEN, H. New brodatz-based image databases for grayscale color and multiband texture analysis. **ISRN Machine Vision**, Hindawi, 2013. DOI: 10.1155/2013/876386.

116 JULESZ, B. Visual pattern discrimination. **IRE Transactions on Information Theory**, IEEE, v. 8, n. 2, p. 84–92, 1962.

117 HARALICK, R. M. Statistical and structural approaches to texture. **Proceedings of the IEEE**, IEEE, v. 67, n. 5, p. 786–804, 1979.

118 OJALA, T.; PIETIKAINEN, M.; MAENPAA, T. Multiresolution gray-scale and rotation invariant texture classification with local binary patterns. **IEEE Transactions on Pattern Analysis and Machine Intelligence**, IEEE, v. 24, n. 7, p. 971–987, 2002.

119 AZENCOTT, R.; WANG, J.-P.; YOUNES, L. Texture classification using windowed fourier filters. **IEEE Transactions on Pattern Analysis and Machine Intelligence**, IEEE, v. 19, n. 2, p. 148–153, 1997.

120 HOANG, M. A.; GEUSEBROEK, J.-M.; SMEULDERS, A. W. Color texture measurement and segmentation. **Signal Processing**, Elsevier, v. 85, n. 2, p. 265–275, 2005.

121 ZHANG, J.; TAN, T. Brief review of invariant texture analysis methods. **Pattern Recognition**, Elsevier, v. 35, n. 3, p. 735–747, 2002.

122 HARALICK, R. M.; SHANMUGAM, K.; DINSTEIN, I. H. Textural features for image classification. **IEEE Transactions on Systems, Man, and Cybernetics**, IEEE, v. SMC-3, n. 6, p. 610–621, 1973.

123 KHADIRI, I. E. *et al.* Repulsive-and-attractive local binary gradient contours: new and efficient feature descriptors for texture classification. **Information Sciences**, Elsevier, v. 467, p. 634–653, 2018. DOI: 10.1016/j.ins.2018.02.009.

124 GUO, Z.; ZHANG, L.; ZHANG, D. A completed modeling of local binary pattern operator for texture classification. **IEEE Transactions on Image Processing**, v. 19, n. 6, p. 1657–1663, 2010.

125 BACKES, A. R.; BRUNO, O. M. A new approach to estimate fractal dimension of texture images. *In*: INTERNATIONAL CONFERENCE ON IMAGE AND SIGNAL PROCESSING, 2008, Cherbourg-Octeville. **Proceedings [...]**. Cherbourg-Octeville: Springer, 2008. p. 136–143.

126 HAFNER, J. *et al.* Efficient color histogram indexing for quadratic form distance functions. **IEEE Transactions on Pattern Analysis and Machine Intelligence**, IEEE, v. 17, n. 7, p. 729–736, 1995.

127 MÄENPÄÄ, T.; PIETIKÄINEN, M. Classification with color and texture: jointly or separately? **Pattern Recognition**, Elsevier, v. 37, n. 8, p. 1629–1640, 2004.

128 CERNADAS, E. *et al.* Influence of normalization and color space to color texture classification. **Pattern Recognition**, Elsevier, v. 61, p. 120–138, 2017. DOI: 10.1016/j.patcog.2016.07.002.

129 JAIN, A.; HEALEY, G. A multiscale representation including opponent color features for texture recognition. **IEEE Transactions on Image Processing**, v. 7, n. 1, p. 124–128, 1998.

130 CASANOVA, D. *et al.* Texture analysis using fractal descriptors estimated by the mutual interference of color channels. **Information Sciences**, Elsevier, v. 346, p. 58–72, 2016. DOI: 10.1016/j.ins.2016.01.077.

131 GONÇALVES, W. N.; MACHADO, B. B.; BRUNO, O. M. A complex network approach for dynamic texture recognition. **Neurocomputing**, Elsevier, v. 153, p. 211–220, 2015. DOI: 10.1016/j.neucom.2014.11.034.

132 GONÇALVES, W. N. *et al.* Texture descriptor based on partially self-avoiding deterministic walker on networks. **Expert Systems with Applications**, Elsevier, v. 39, n. 15, p. 11818–11829, 2012.

133 SCABINI, L. F.; GONÇALVES, W. N.; JR, A. A. C. Texture analysis by bag-of-visual-words of complex networks. *In*: IBEROAMERICAN CONGRESS ON PATTERN RECOGNITION, 2015, Montevideo. **Proceedings [...]**. Montevideo: Springer, 2015. p. 485–492.

134 LIMA, G. V. de *et al.* Classification of texture based on bag-of-visual-words through complex networks. **Expert Systems with Applications**, Elsevier, v. 133, p. 215–224, 2019. DOI: 10.1016/j.eswa.2019.05.021.

135 LEUNG, T.; MALIK, J. Representing and recognizing the visual appearance of materials using three-dimensional textons. **International Journal of Computer Vision**, Springer, v. 43, n. 1, p. 29–44, 2001.

136 CSURKA, G. *et al.* Visual categorization with bags of keypoints. *In*: EUROPEAN CONFERENCE ON COMPUTER VISION, 2004, Prague. **Proceedings [...]**. Prague: Springer, 2004. p. 1–22.

137 LOWE, D. G. Distinctive image features from scale-invariant keypoints. **International Journal of Computer Vision**, Springer, v. 60, n. 2, p. 91–110, 2004.

138 SINGH, C.; SINGH, J. Geometrically invariant color, shape and texture features for object recognition using multiple kernel learning classification approach. **Information Sciences**, Elsevier, v. 484, p. 135–152, 2019. DOI: 10.1016/j.ins.2019.01.058.

139 LECUN, Y. *et al.* Gradient-based learning applied to document recognition. **Proceedings of the IEEE**, IEEE, v. 86, n. 11, p. 2278–2324, 1998.

140 CIMPOI, M.; MAJI, S.; VEDALDI, A. Deep filter banks for texture recognition and segmentation. *In*: CONFERENCE ON COMPUTER VISION AND PATTERN RECOGNITION, 2015, Boston. **Proceedings [...]**. Boston: IEEE, 2015. p. 3828–3836.

141 M. Condori, R. H.; BRUNO, O. M. Analysis of activation maps through global pooling measurements for texture classification. **Information Sciences**, v. 555, p. 260–279, 2021. ISSN 0020-0255.

142 LYRA, L. O.; FABRIS, A. E.; FLORINDO, J. B. **Multilayer deep feature extraction for visual texture recognition**. 2022. Available at: https://arxiv.org/pdf/2208.10044.pdf. Access at: 5 Jan. 2023.

143 ZHANG, H.; XUE, J.; DANA, K. Deep ten: texture encoding network. *In*: CONFERENCE ON COMPUTER VISION AND PATTERN RECOGNITION, 2017, Honolulu. **Proceedings [...]**. Honolulu: IEEE / CVF, 2017. p. 2896–2905.

144 LIN, T.-Y.; ROYCHOWDHURY, A.; MAJI, S. Bilinear cnn models for fine-grained visual recognition. *In*: INTERNATIONAL CONFERENCE ON COMPUTER VISION, 2015, Santiago. **Proceedings [...]**. Santiago: IEEE, 2015. p. 1449–1457.

145 ZHAI, W. *et al.* Deep multiple-attribute-perceived network for real-world texture recognition. *In*: INTERNATIONAL CONFERENCE ON COMPUTER VISION, 2019, Seoul. **Proceedings [...]**. Seoul: IEEE / CVF, 2019. p. 3612–3621.

146 ZHAI, W. *et al.* Deep structure-revealed network for texture recognition. *In*: CONFERENCE ON COMPUTER VISION AND PATTERN RECOGNITION, 2020, Seattle. **Proceedings [...]**. Seattle: IEEE / CVF, 2020. p. 11007–11016.

147 CHEN, Z. *et al.* Deep texture recognition via exploiting cross-layer statistical self-similarity. *In*: CONFERENCE ON COMPUTER VISION AND PATTERN RECOGNITION, 2021, Nashville. **Proceedings [...]**. Nashville: IEEE / CVF, 2021. p. 5231–5240.

148 YANG, Z. *et al.* Dfaen: double-order knowledge fusion and attentional encoding network for texture recognition. **Expert Systems with Applications**, v. 209, p. 118223, 2022. ISSN 0957-4174.

149 ZHANG, Y. *et al.* **Bamboo**: building mega-scale vision dataset continually with human-machine synergy. 2022. Available at: https://arxiv.org/pdf/2203.07845.pdf. Access at: 20 Jan. 2023.

150 GESMUNDO, A. **A continual development methodology for large-scale multitask dynamic ml systems**. 2022. Available at: https://arxiv.org/pdf/2209.07326.pdf. Access at: 20 Jan. 2023.

151 RADFORD, A. *et al.* **Learning transferable visual models from natural language supervision**. 2021. 8748–8763 p. Available at: http://proceedings.mlr.press/v139/radford21a/radford21a.pdf. Access at: 20 Jan. 2022.

152 SÁ JUNIOR, J. J. M.; BACKES, A. R. ELM based signature for texture classification. **Pattern Recognition**, v. 51, p. 395–401, 2016. DOI: 10.1016/j.patcog.2015.09.014.

153 SÁ JUNIOR, J. J. M.; BACKES, A. R.; BRUNO, O. M. Randomized neural network based signature for color texture classification. **Multidimensional Systems and Signal Processing**, Springer, v. 30, n. 3, p. 1171–1186, 2019.

154 RIBAS, L. C. *et al.* Fusion of complex networks and randomized neural networks for texture analysis. **Pattern Recognition**, Elsevier, v. 103, p. 107189, 2020.

155 FLORINDO, J. B. *et al.* Visgraphnet: a complex network interpretation of convolutional neural features. **Information Sciences**, v. 543, p. 296–308, 2021. ISSN 0020-0255.

156 BASU, S. *et al.* A theoretical analysis of deep neural networks for texture classification. *In*: INTERNATIONAL JOINT CONFERENCE ON NEURAL NETWORKS, 2016, Vancouver. **Proceedings [...]**. Vancouver: IEEE, 2016. p. 992–999.

157 NGUYEN, A.; YOSINSKI, J.; CLUNE, J. Deep neural networks are easily fooled: high confidence predictions for unrecognizable images. *In*: CONFERENCE ON COMPUTER VISION AND PATTERN RECOGNITION, 2015, Boston. **Proceedings [...]**. Boston: IEEE / CVF, 2015. p. 427–436.

158 PICARD, D. **Torch. manual_seed(3407) is all you need**: on the influence of random seeds in deep learning architectures for computer vision. 2021. Available at: https://arxiv.org/pdf/2109.08203.pdf. Access at: 20 Jan. 2022.

159 WIGHTMAN, R.; TOUVRON, H.; JÉGOU, H. **Resnet strikes back**: an improved training procedure in timm. 2021. Available at: https://arxiv.org/pdf/2110.00476.pdf. Access at: 20 Jan. 2022.

160 HUANG, Z.; LAM, H.; ZHANG, H. **Quantifying epistemic uncertainty in deep learning**. 2021. Available at: https://arxiv.org/pdf/2110.12122.pdf. Access at: 20 Jan. 2022.

161 BARABÁSI, A.-L.; ALBERT, R. Emergence of scaling in random networks. **Science**, American Association for the Advancement of Science, v. 286, n. 5439, p. 509–512, 1999.

162 STAM, C. J. Functional connectivity patterns of human magnetoencephalographic recordings: a 'small-world'network? **Neuroscience Letters**, Elsevier, v. 355, n. 1-2, p. 25–28, 2004.

163 SPORNS, O.; ZWI, J. D. The small world of the cerebral cortex. **Neuroinformatics**, Springer, v. 2, n. 2, p. 145–162, 2004.

164 HUMPHRIES, M. D.; GURNEY, K.; PRESCOTT, T. J. The brainstem reticular formation is a small-world, not scale-free, network. **Proceedings of the Royal Society B**, The Royal Society London, v. 273, n. 1585, p. 503–511, 2005.

165 SPORNS, O. The human connectome: a complex network. **Annals of the New York Academy of Sciences**, Wiley Online Library, v. 1224, n. 1, p. 109–125, 2011.

166 STAUFFER, D. *et al.* Efficient hopfield pattern recognition on a scale-free neural network. **The European Physical Journal B**, Springer, v. 32, n. 3, p. 395–399, 2003.

167 TORRES, J. J. *et al.* Influence of topology on the performance of a neural network. **Neurocomputing**, Elsevier, v. 58, p. 229–234, 2004. DOI: 10.1016/j.neucom.2004.01.048.

168 SIMARD, D.; NADEAU, L.; KRÖGER, H. Fastest learning in small-world neural networks. **Physics Letters A**, Elsevier, v. 336, n. 1, p. 8–15, 2005.

169 ERKAYMAZ, O.; OZER, M. Impact of small-world network topology on the conventional artificial neural network for the diagnosis of diabetes. **Chaos, Solitons & Fractals**, Elsevier, v. 83, p. 178–185, 2016. DOI: 10.1016/j.chaos.2015.11.029.

170 GRAY, S.; RADFORD, A.; KINGMA, D. P. **Gpu kernels for block-sparse weights**. 2017. Available at: https://cdn.openai.com/blocksparse/blocksparsepaper.pdf. Access at: 20 Jan. 2019.

171 XIE, S. *et al.* Exploring randomly wired neural networks for image recognition. *In*: INTERNATIONAL CONFERENCE ON COMPUTER VISION, 2019, Seoul. **Proceedings [...]**. Seoul: IEEE / CVF, 2019. p. 1284–1293.

172 YOU, J. *et al.* Graph structure of neural networks. *In*: INTERNATIONAL CONFERENCE ON MACHINE LEARNING, 2020, Lille. **Proceedings [...]**. Lille: PMLR, 2020. p. 10881–10891.

173 ERKAYMAZ, O. Resilient back-propagation approach in small-world feed-forward neural network topology based on newman–watts algorithm. **Neural Computing and Applications**, Springer, v. 32, n. 20, p. 16279–16289, 2020.

174 TESTOLIN, A.; PICCOLINI, M.; SUWEIS, S. Deep learning systems as complex networks. **Journal of Complex Networks**, Oxford University Press, v. 8, n. 1, p. cnz018, 2020.

175 ZAMBRA, M.; MARITAN, A.; TESTOLIN, A. Emergence of network motifs in deep neural networks. **Entropy**, Multidisciplinary Digital Publishing Institute, v. 22, n. 2, p. 204, 2020.

176 MALFA, E. L. *et al.* Characterizing learning dynamics of deep neural networks via complex networks. *In*: INTERNATIONAL CONFERENCE ON TOOLS WITH ARTIFICIAL INTELLIGENCE, 2021, Washington. **Proceedings [...]**. Washington: IEEE, 2021. p. 344–351.

177 SCABINI, L. F.; BRUNO, O. M. Structure and performance of fully connected neural networks: emerging complex network properties. **Physica A**, Elsevier, v. 615, p. 128585, 2023. DOI: 10.1016/j.physa.2023.128585.

178 YING, C. *et al.* NAS-bench-101: towards reproducible neural architecture search. *In*: INTERNATIONAL CONFERENCE ON MACHINE LEARNING, 2019, Long Beach. **Proceedings [...]**. Long Beach: PMLR, 2019. p. 7105–7114.

179 CHOLLET, F. *et al.* **Keras**. 2015. Available at: https://keras.io. Access at: 20 Jan. 2019.

180 RECHT, B. *et al.* Do imagenet classifiers generalize to imagenet? *In*: INTERNATIONAL CONFERENCE ON MACHINE LEARNING, 2019, Long Beach. **Proceedings [...]**. Long Beach: PMLR, 2019. p. 5389–5400.

181 XIAO, H.; RASUL, K.; VOLLGRAF, R. **Fashion-mnist**: a novel image dataset for benchmarking machine learning algorithms. 2017. Available at: https://arxiv.org/pdf/1708.07747.pdf. Access at: 20 Jan. 2019.

182 KRIZHEVSKY, A.; HINTON, G. *et al.* **Learning multiple layers of features from tiny images**. 2009. Available at: http://www.cs.utoronto.ca/~kriz/learning-features-2009-TR.pdf. Access at: 20 Jan. 2019.

183 HAGBERG, A.; SWART, P.; CHULT, D. S. **Exploring network structure, dynamics, and function using networkx**. 2008. Available at: https://www.osti.gov/biblio/960616. Access at: 20 Jan. 2019.

184 ARTHUR, D.; VASSILVITSKII, S. K-means++: the advantages of careful seeding. *In*: SYMPOSIUM ON DISCRETE ALGORITHMS, 2006, New Orleans. **Proceedings [...]**. New Orleans: ACM, 2007. p. 1027–1035.

185 MEGHANATHAN, N. Correlation coefficient analysis of centrality metrics for complex network graphs. *In*: SILHAVY, R. *et al.* (ed.). **Intelligent systems in cybernetics and automation theory**. Cham: Springer, 2015. p. 11–20. (Advances in intelligent systems and computing, v. 348).

186 KULLBACK, S.; LEIBLER, R. A. On information and sufficiency. **The Annals of Mathematical Statistics**, JSTOR, v. 22, n. 1, p. 79–86, 1951.

187 SCABINI, L.; BAETS, B. D.; BRUNO, O. M. **Improving deep neural network random initialization through neuronal rewiring**. 2022. Available at: https://arxiv.org/pdf/2207.08148.pdf. Access at: 20 Jan. 2023.

188 LOSHCHILOV, I.; HUTTER, F. **SGDR**: stochastic gradient descent with warm restarts. 2016. Available at: https://arxiv.org/pdf/1608.03983.pdf. Access at: 20 Jan. 2019.

189 LEE, S. H.; LEE, S.; SONG, B. C. **Vision transformer for small-size datasets**. 2021. Available at: https://arxiv.org/pdf/2112.13492.pdf. Access at: 20 Jan. 2022.

190 SZEGEDY, C. *et al.* Rethinking the inception architecture for computer vision. *In*: CONFERENCE ON COMPUTER VISION AND PATTERN RECOGNITION, 2016, Las Vegas. **Proceedings [...]**. Las Vegas: IEEE / CVF, 2016.

191 HUANG, G. *et al.* Deep networks with stochastic depth. *In*: EUROPEAN CONFERENCE ON COMPUTER VISION, 2016, Amsterdam. **Proceedings [...]**. Amsterdam: Springer, 2016. p. 646–661.

192 YUN, S. *et al.* Cutmix: regularization strategy to train strong classifiers with localizable features. *In*: INTERNATIONAL CONFERENCE ON COMPUTER VISION, 2019, Seoul. **Proceedings [...]**. Seoul: IEEE / CVF, 2019. p. 6023–6032.

193 ZHANG, H. *et al.* **mixup**: beyond empirical risk minimization. 2017. Available at: https://arxiv.org/pdf/1710.09412.pdf. Access at: 20 Jan. 2019.

194 TROCKMAN, A.; KOLTER, J. Z. **Patches are all you need?** 2022. Available at: https://arxiv.org/pdf/2201.09792.pdf. Access at: 20 Jan. 2023.

195 CUBUK, E. D. *et al.* Randaugment: practical automated data augmentation with a reduced search space. *In*: CONFERENCE ON COMPUTER VISION AND PATTERN RECOGNITION, 2020, Seattle. **Proceedings [...]**. Seattle: IEEE / CVF, 2020. p. 702–703.

196 ZHONG, Z. *et al.* Random erasing data augmentation. *In*: CONFERENCE ON ARTIFICIAL INTELLIGENCE, 2020, New York. **Proceedings [...]**. New York: AAAI, 2020. p. 13001–13008.

197 WANG, L. *et al.* Sample-efficient neural architecture search by learning actions for monte carlo tree search. **IEEE Transactions on Pattern Analysis and Machine Intelligence**, IEEE, v. 44, n. 9, p. 5503–5515, 2021.

198 IOFFE, S.; SZEGEDY, C. Batch normalization: accelerating deep network training by reducing internal covariate shift. *In*: INTERNATIONAL CONFERENCE ON MACHINE LEARNING, 2015, Lille. **Proceedings [...]**. Lille: PMLR, 2015. p. 448–456.

199 GONÇALVES, W. N.; BRUNO, O. M. Dynamic texture analysis and segmentation using deterministic partially self-avoiding walks. **Expert Systems with Applications**, Elsevier, v. 40, n. 11, p. 4283–4300, 2013.

200 RIBAS, L. C.; MANZANERA, A.; BRUNO, O. M. A fractal-based approach to network characterization applied to texture analysis. *In*: INTERNATIONAL CONFERENCE ON COMPUTER ANALYSIS OF IMAGES AND PATTERNS, 2019, Salerno. **Proceedings [...]**. Salerno: Springer, 2019. p. 129–140.

201 RIBAS, L. C.; BRUNO, O. M. Dynamic texture classification using deterministic partially self-avoiding walks on networks. *In*: INTERNATIONAL CONFERENCE ON IMAGE ANALYSIS AND PROCESSING, 2019, Trento. **Proceedings [...]**. Trento: Springer, 2019. p. 82–93.

202 RIBAS, L. C.; GONCALVES, W. N.; BRUNO, O. M. Dynamic texture analysis with diffusion in networks. **Digital Signal Processing**, Elsevier, v. 92, p. 109–126, 2019. DOI: 10.1016/j.dsp.2019.03.017.

203 RIBAS, L. C.; BRUNO, O. M. Dynamic texture analysis using networks generated by deterministic partially self-avoiding walks. **Physica A**, Elsevier, v. 541, p. 122105, 2020. DOI: 10.1016/j.physa.2019.122105.

204 SCABINI, L. F.; RIBAS, L. C.; BRUNO, O. M. Spatio-spectral networks for color-texture analysis. **Information Sciences**, Elsevier, v. 515, p. 64–79, 2020. DOI: 10.1016/j.ins.2019.11.042.

205 RIBAS, L. C. *et al.* A complex network based approach for knee osteoarthritis detection: data from the osteoarthritis initiative. **Biomedical Signal Processing and Control**, Elsevier, v. 71, Part A, p. 103133, 2022.

206 RIBAS, L. C. *et al.* Learning graph representation with randomized neural network for dynamic texture classification. **Applied Soft Computing**, Elsevier, v. 114, p. 108035, 2022. DOI: 10.1016/j.asoc.2021.108035.

207 ZIELINSKI, K. M. *et al.* Complex texture features learned by applying randomized neural network on graphs. *In*: INTERNATIONAL CONFERENCE ON IMAGE PROCESSING THEORY, TOOLS AND APPLICATIONS, 2022, Salzburg. **Proceedings [...]**. Salzburg: IEEE, 2022. p. 1–6.

208 FOSTER, M. **A text-book of physiology**. Philadelphia: Lea Brothers & Company, 1895.

209 FISHER, R. A. The use of multiple measurements in taxonomic problems. **Annals of Eugenics**, Wiley Online Library, v. 7, n. 2, p. 179–188, 1936.

210 RIPLEY, B. D. **Pattern recognition and neural networks**. Cambridge: Cambridge University Press, 2007.

211 RIBAS, L. C. *et al.* **Learning local complex features using randomized neural networks for texture analysis**. 2020. Available at: https://arxiv.org/pdf/2007.05643.pdf. Access at: 20 Jan. 20121.

212 RIBAS, L. C. *et al.* **Spatio-spectral representation learning for color-texture classification**. 2022. Available at: https://papers.ssrn.com/sol3/papers.cfm?abstract_id=4266882. Access at: 20 Jan. 2023.

213 TIKHONOV, A. N. On the solution of ill-posed problems and the method of regularization. **Doklady Akademii Nauk**, Russian Academy of Sciences, v. 151, n. 3, p. 501–504, 1963.

214 CALVETTI, D. *et al.* Tikhonov regularization and the l-curve for large discrete ill-posed problems. **Journal of Computational and Applied Mathematics**, Elsevier, v. 123, n. 1-2, p. 423–446, 2000.

215 SCABINI, L. *et al.* **RADAM**: texture recognition through randomized aggregated encoding of deep activation maps. 2023. Available at: https://arxiv.org/pdf/2303.04554.pdf. Access at: 10 June. 2023.

216 DONAHUE, J. *et al.* Decaf: a deep convolutional activation feature for generic visual recognition. *In*: INTERNATIONAL CONFERENCE ON MACHINE LEARNING, 2014, Beijing. **Proceedings [...]**. Beijing: PMLR, 2014. p. 647–655.

217 LIN, M.; CHEN, Q.; YAN, S. **Network in network**. 2013. Available at: https://arxiv.org/pdf/1312.4400.pdf. Access at: 20 Jan. 2019.

218 WIGHTMAN, R. **PyTorch image models**. 2019. Available at: https://github.com/rwightman/pytorch-image-models. Access at: 20 Jan. 2020.

219 WANG, Z.; LIU, J.-C. Translating math formula images to latex sequences using deep neural networks with sequence-level training. **International Journal on Document Analysis and Recognition**, Springer, v. 24, n. 1, p. 63–75, 2021.

220 RIBAS, L. C. *et al.* Fusion of complex networks and randomized neural networks for texture analysis. **Pattern Recognition**, v. 103, p. 107189, 2020. ISSN 0031-3203. DOI: 10.1016/j.patcog.2019.107189.

221 WORTSMAN, M. *et al.* Model soups: averaging weights of multiple fine-tuned models improves accuracy without increasing inference time. *In*: INTERNATIONAL CONFERENCE ON MACHINE LEARNING, 2022, Baltimore. **Proceedings [...]**. Baltimore: PMLR, 2022. p. 23965–23998.

222 PEDREGOSA, F. *et al.* Scikit-learn: machine learning in python. **Journal of Machine Learning Research**, v. 12, n. 85, p. 2825–2830, 2011.

223 CORTES, C.; VAPNIK, V. Support-vector networks. **Machine Learning**, Springer, v. 20, p. 273–297, 1995. DOI: 10.1007/BF00994018.

224 ZAFAR, A. *et al.* A comparison of pooling methods for convolutional neural networks. **Applied Sciences**, MDPI, v. 12, n. 17, p. 8643, 2022.

225 SCABINI, L. F. *et al.* Evaluating deep convolutional neural networks as texture feature extractors. *In*: INTERNATIONAL CONFERENCE ON IMAGE ANALYSIS AND PROCESSING, 2019, Trento. **Proceedings [...]**. Trento: Springer, 2019. p. 192–202.

226 FIX, E.; HODGES, J. L. Discriminatory analysis, nonparametric estimation: consistency properties. **Report 4, Project nº 21-49**, v. 4, 1951.

227 SIMONYAN, K.; ZISSERMAN, A. **Very deep convolutional networks for large-scale image recognition**. 2014. Available at: https://arxiv.org/pdf/1409.1556.pdf. Access at: 20 Jan. 2019.

228 HUANG, G. *et al.* Densely connected convolutional networks. *In*: CONFERENCE ON COMPUTER VISION AND PATTERN RECOGNITION, 2017, Honolulu. **Proceedings [...]**. Honolulu: IEEE / CVF, 2017. p. 4700–4708.

229 LIU, C. *et al.* Progressive neural architecture search. *In*: EUROPEAN CONFERENCE ON COMPUTER VISION, 2018, Munich. **Proceedings [...]**. Munich: Springer, 2018. p. 19–34.

230 HU, J.; SHEN, L.; SUN, G. Squeeze-and-excitation networks. *In*: CONFERENCE ON COMPUTER VISION AND PATTERN RECOGNITION, 2018, Salt Lake. **Proceedings [...]**. Salt Lake: IEEE / CVF, 2018. p. 7132–7141.

231 XIE, S. *et al.* Aggregated residual transformations for deep neural networks. *In*: CONFERENCE ON COMPUTER VISION AND PATTERN RECOGNITION, 2017, Honolulu. **Proceedings [...]**. Honolulu: IEEE / CVF, 2017. p. 1492–1500.

232 WESZKA, J. S.; DYER, C. R.; ROSENFELD, A. A comparative study of texture measures for terrain classification. **IEEE Transactions on Systems, Man, and Cybernetics**, v. 4, n. SMC-6, p. 269–285, 1976.

233 GARCÍA-OLALLA, O. *et al.* Local oriented statistics information booster (losib) for texture classification. *In*: INTERNATIONAL CONFERENCE ON PATTERN RECOGNITION, 2014, Stockholm. **Proceedings [...]**. Stockholm: IEEE, 2014. p. 1114–1119.

234 COSTA, A. F.; HUMPIRE-MAMANI, G.; TRAINA, A. J. M. An efficient algorithm for fractal analysis of textures. *In*: CONFERENCE ON GRAPHICS, PATTERNS AND IMAGES, 2012, Ouro Preto. **Proceedings [...]**. Ouro Preto: IEEE, 2012. p. 39–46.

235 JOURNAUX, L. *et al.* Texture classification with generalized fourier descriptors in dimensionality reduction context: an overview exploration. *In*: PREVOST, L.; MARINAI, S.; SCHWENKER, F. (ed.). **Artificial neural networks in pattern recognition**. Paris: Springer, 2008. p. 280–291.

236 VARMA, M.; GARG, R. Locally invariant fractal features for statistical texture classification. *In*: INTERNATIONAL CONFERENCE ON COMPUTER VISION, 2007, Rio de Janeiro. **Proceedings [...]**. Rio de Janeiro: IEEE, 2007. p. 1–8.

237 ZHANG, L.; ZHOU, Z.; LI, H. Binary gabor pattern: an efficient and robust descriptor for texture classification. *In*: INTERNATIONAL CONFERENCE ON IMAGE PROCESSING, 2012, Orlando. **Proceedings [...]**. Orlando: IEEE, 2012. p. 81–84.

238 KANNALA, J.; RAHTU, E. Bsif: binarized statistical image features. *In*: INTERNATIONAL CONFERENCE ON PATTERN RECOGNITION, 2012, Tsukuba. **Proceedings [...]**. Tsukuba: IEEE, 2012. p. 1363–1366.

239 DRIMBAREAN, A.; WHELAN, P. F. Experiments in colour texture analysis. **Pattern Recognition Letters**, Elsevier, v. 22, n. 10, p. 1161–1167, 2001.

240 CIMPOI, M. *et al.* Deep filter banks for texture recognition, description, and segmentation. **International Journal of Computer Vision**, Springer, v. 118, n. 1, p. 65–94, 2016.

241 ALATA, O. *et al.* Choice of a pertinent color space for color texture characterization using parametric spectral analysis. **Pattern Recognition**, Elsevier, v. 44, n. 1, p. 16–31, 2011.

242 RODRIGUES, V. C. *et al.* Electrochemical and optical detection and machine learning applied to images of genosensors for diagnosis of prostate cancer with the biomarker pca3. **Talanta**, Elsevier, v. 222, p. 121444, 2020. DOI:10.1016/j.talanta.2020.121444.

243 SANDLER, M. *et al.* Mobilenetv2: inverted residuals and linear bottlenecks. *In*: CONFERENCE ON COMPUTER VISION AND PATTERN RECOGNITION, 2018, Salt Lake. **Proceedings [...]**. Salt Lake: IEEE / CVF, 2018. p. 4510–4520.

244 SOARES, J. C. *et al.* Detection of a sars-cov-2 sequence with genosensors using data analysis based on information visualization and machine learning techniques. **Materials Chemistry Frontiers**, Royal Society of Chemistry, v. 5, n. 15, p. 5658–5670, 2021.

245 SILVA, N. R. da *et al.* Plant identification based on leaf midrib cross-section images using fractal descriptors. **PloS One**, Public Library of Science, v. 10, n. 6, p. e0130014, 2015.

246 SALEEM, M. H.; POTGIETER, J.; ARIF, K. M. Plant disease detection and classification by deep learning. **Plants**, MDPI, v. 8, n. 11, p. 468, 2019.

247 GONÇALVES, D. N. *et al.* Transformers for mapping burned areas in brazilian pantanal and amazon with planetscope imagery. **International Journal of Applied Earth Observation and Geoinformation**, Elsevier, v. 116, p. 103151, 2023. DOI: 10.1016/j.jag.2022.103151.

248 FALVO, M. **Método de mapeamento espaço-espectral em imagens multi-espectrais e sua aplicação em tecidos vegetais**. 2016. 155 p. Tese (Doutorado) — Instituto de Física de São carlos, Universidade de São Paulo, São Carlos, 2016.

249 DURIGAN, G. **Manual para recuperação da vegetação de cerrado**. São Paulo: Instituto Florestal, 2003.

250 RIBEIRO, J. F.; FONSECA, C. E. L. da; SOUSA-SILVA, J. C. **Cerrado**: caracterização e recuperação de matas de galeria. Planaltina: Embrapa Cerrados, 2001.

251 SEELAND, M. *et al.* Image-based classification of plant genus and family for trained and untrained plant species. **BMC Bioinformatics**, BioMed Central, v. 20, n. 1, p. 4, 2019.

252 WÄLDCHEN, J.; MÄDER, P. Plant species identification using computer vision techniques: a systematic literature review. **Archives of Computational Methods in Engineering**, Springer, v. 25, n. 2, p. 507–543, 2018.

253 GASTON, K. J.; O'NEILL, M. A. Automated species identification: why not? **Philosophical Transactions of the Royal Society of London B**, The Royal Society, v. 359, n. 1444, p. 655–667, 2004.

254 SCABINI, L. F. *et al.* Deep convolutional neural networks for plant species characterization based on leaf midrib. *In*: INTERNATIONAL CONFERENCE ON COMPUTER ANALYSIS OF IMAGES AND PATTERNS, 2019, Salerno. **Proceedings [...]**. Salerno: Springer, 2019. p. 389–401.

255 SÁ JUNIOR, J. J. M. *et al.* A computer vision approach to quantify leaf anatomical plasticity: a case study on gochnatia polymorpha (less.) cabrera. **Ecological Informatics**, Elsevier, v. 15, p. 34–43, 2013. DOI: 10.1016/j.ecoinf.2013.02.007.

256 KEATING, R. C. Leaf histology and its contribution to relationships in the myrtales. **Annals of the Missouri Botanical Garden**, JSTOR, v. 71, n. 3, p. 801–823, 1984.

257 LLOYD, S. Least squares quantization in pcm. **IEEE Transactions on Information Theory**, IEEE, v. 28, n. 2, p. 129–137, 1982.

258 WU, X. *et al.* Top 10 algorithms in data mining. **Knowledge and Information Systems**, Springer, v. 14, n. 1, p. 1–37, 2008.

259 VARMA, M.; ZISSERMAN, A. A statistical approach to texture classification from single images. **International Journal of Computer Vision**, v. 62, n. 1, p. 61–81, 2005. ISSN 1573-1405.

260 RIDNIK, T. *et al.* **Imagenet-21k pretraining for the masses**. 2021. Available at: https://arxiv.org/pdf/2104.10972.pdf. Access at: 20 Jan. 2022.

APPENDIX

APPENDIX A – MACHINE LEARNING MODELS AND EVALUATION

This chapter covers aspects of the ML techniques employed throughout the thesis. It is divided into two main sections: the first describes the ML methods, and the second includes their validation and evaluation schemes.

A.1 Machine learning methods

In this section, we present an overview of ML algorithms, starting with supervised classifiers. Classification is one of the most common tasks in ML. This is the procedure of automatically identifying the category, label, or class of a given data sample. In the supervised setting, this procedure considers previous knowledge obtained from a labeled training set. For that, there are several approaches in the literature, and in the following, we cover the ones that are used in this thesis.

A.1.1 k-nearest neighbors (KNN)

There are several supervised classifiers in the literature and numerous improvements and adjustments that were proposed for them throughout the years. One of the most simple and fundamental approaches is the k-nearest neighbors (KNN) algorithm. It is a type of instance-based learning method, which is non-parametric and is categorized as a supervised learning technique. It was first proposed by Fix and Hodges (226) and has been widely used in ML ever since. The primary principle of KNN is classifying an object based on the majority votes of its k nearest neighbors. Consider a labeled training set $X = \{xi\}$, where xi are z-dimensional vectors with labels (classes) $Y = \{yi\}$, where each training sample xi is associated with a class yi. The KNN algorithm assigns an unlabeled testing sample $\overline{x}$ to the most common class y among its k nearest neighbors in the training set. In practice, for a given test sample $\overline{x}$, its nearest neighbors are defined in terms of their distance in a z-dimensional space. The most common approach is to use the Euclidean distance, although other metrics are also often employed. Therefore, for a training sample x, the distance is defined as:

$$d(\overline{x}, x) = \sqrt{(\overline{x_1} - x_1)^2 + (\overline{x_2} - x_2)^2 + \ldots + (\overline{x_z} - x_z)^2}, \tag{A.1}$$

where $d()$ denotes the Euclidean distance.

If $k = 1$, a.k.a. simply the nearest neighbor algorithm (the most common use case), the class assigned to $\overline{x}$ corresponds to the class yj of $xj = \{argmin\ d(\overline{x}, xi),\ \forall xi \in X\}$. When $k > 1$, the most frequent class among the k nearest training samples is assigned to the test sample $\overline{x}$. Although simple, this algorithm works well in a variety of cases, especially if z is small, and can be applied for various ML tasks. On the other hand, if the number of training samples in X is large, the process may become inefficient since the algorithm (in its most common form) must compute the distance between every new test sample to every training sample. In conclusion, the k is the only numerical hyperparameter of the algorithm, and other design choices may include selecting the distance metric or heuristics to compute the nearest neighbors more efficiently. For simplicity, we use KNN with a fixed $k = 1$ in the experiments presented in this thesis, since the intuition is to highlight the quality of the features themselves, rather than the classifiers.

A.1.2 Linear Discriminant Analysis (LDA)

Linear Discriminant Analysis (LDA) (210) is a method used in statistics, pattern recognition, and ML to find a linear combination of features that characterizes or separates two or more classes of objects or events. It is a generalization of the earlier Fisher linear discriminant. (209) This approach is often used for dimensionality reduction and is related to the principal component analysis (PCA), and can also be extended into a linear classifier. For that, the aim is to project a dataset into a lower-dimensional space where class separability is maintained (or improved), and computational costs are reduced.

Mathematically, the LDA approach can be described through the concept of maximizing the ratio of between-class variance to the within-class variance. It works by fitting class conditional densities to the data and using Bayes' rule, *i.e.*, the model fits a Gaussian density to each class, assuming that all classes share the same covariance matrix. Consider a set of c classes $K = \{i_1, ..., i_c\}$, the mean of class i as μ_i, and a common covariance matrix Σ, computed for the training set. The LDA classifier can be defined as:

$$\delta_i(x) = -\frac{1}{2}\ln(|\Sigma|) - \frac{1}{2}(x - \mu_i)^T \Sigma^{-1}(x - \mu_i), \tag{A.2}$$

where $\delta_i(x)$ represents the discriminant function for class i, which is the probability of vector x to belong to class i. The classification is then performed by selecting $y = \{argmax\ \delta_i(x),\ \forall i \in K\}$, *i.e.*, selecting the class with the highest probability for x. It is important to notice that the LDA can be tuned in several ways such as varying the covariance estimator, and the number of components for dimensionality reduction, among others (according to different versions of the algorithm). In this thesis, we use LDA with a least-squares solution with automatic shrinkage using the Ledoit–Wolf lemma.

A.1.3 Support Vector Machines (SVM)

The Support Vector Machine (SVM) (223) is another supervised model widely used for classification tasks. The main idea behind SVMs is to build a hyperplane as a decision surface by maximizing the separation between classes. In the case of linearly separable classes, the decision function of SVM is a hyperplane defined by:

$$f(x) = \mathbf{w}^T \phi(x) + b, \tag{A.3}$$

where x is the input sample, $\mathbf{w}$ is the weight vector that defines the orientation of the hyperplane, $\phi(x)$ is a transformation of the input vector x (identity transformation for linear SVM), and b is the bias term that defines the offset of the hyperplane from the origin.

The support vectors are the training samples that lie closest to this decision hyperplane, hence, they are the data points that, if moved, would alter the position of the dividing hyperplane. The training of the SVM is then an optimization problem and consists of finding the hyperplane that minimizes the distance to the support vectors. In the case of linearly separable classes, the problem can be stated as:

$$\min_{\mathbf{w},b} \frac{1}{2}|\mathbf{w}|^2, \tag{A.4}$$

subject to:

$$f(x) \geq 1,\ i = 1, \dots, N, \tag{A.5}$$

where N is the number of training samples.

For non-linearly separable classes, a so-called “soft-margin” SVM is often used. In this case, slack variables $\xi_i \geq 0$ are introduced to allow some samples to be on the wrong side of the margin, and the optimization problem becomes:

$$\min_{\mathbf{w},b,\boldsymbol{\xi}} \frac{1}{2}|\mathbf{w}|^2 + C\sum_{i=1}^{N} \xi_i, \tag{A.6}$$

subject to:

$$y_i(\mathbf{w}^T\phi(x_i) + b) \geq 1 - \xi_i,\ \xi_i \geq 0,\ i = 1, \dots, N, \tag{A.7}$$

where $C > 0$ is a hyperparameter that controls the trade-off between maximizing the margin and minimizing the classification error.

To solve these optimization problems, the concept of Lagrange duality is used. The problem is transformed into its dual form, which can be more easily solved. In general, SVMs are effective in high-dimensional spaces and versatile in the sense that different kernel functions can be used for the decision function. For cases where the classes are not linearly separable, SVM uses a “kernel trick” to map the input vectors into a high-dimensional feature space. Nevertheless, in this thesis SVMs are used in their simplest linear form, and with $C = 1$.

A.1.4 k-means

The k-means clustering algorithm is a vector quantization method originating from signal processing, which was later adapted for cluster analysis in data mining. Lloyd (257) provided one of the most widely used adaptations, which is usually employed due to its simplicity and performance, and it is commonly cited among the best data mining methods. (258) It differs from the previously described ML techniques in two ways. First, it is an unsupervised learning method, which means that the labels for the training data are not known by the algorithm. And second, it is used for data clustering, which is slightly different from classification. The goal of this algorithm is to find groups in the data, where k is a given parameter that represents the number of desired or estimated groups. In other words, although the data labels are not needed, the user needs to provide k beforehand (in the simplest form of this algorithm), *i.e.*, to know the number of existing groups in the data.

The k-means algorithm works iteratively to assign each data point to one of the k groups based on their z-dimensional feature representation. Consider a training set $X = \{x_p\}$, where x_p are z-dimensional vectors representing each data point p. The algorithm aims to partition X into k clusters, where each observation belongs to the cluster with the nearest mean. For that, it calculates the centroid of each group as the mean vector of the features for the samples within that group and then assigns each data sample to the group with the closest centroid. This is achieved by an iterative refinement technique. Given an initial set of randomly defined z-dimensional k means $\{\mu_1^{(1)}, ..., \mu_k^{(1)}\}$, the algorithm proceeds by alternating between two steps:

- Assignment step: Assign each observation to the cluster with the nearest mean:

$$S_i^{(t)} = \{x : |x - \mu_i^{(t)}|^2 \leq |x - \mu_j^{(t)}|^2 \ \forall j, 1 \leq j \leq k\}, \tag{A.8}$$

  where each $S_i^{(t)}$ is the set of all points closer to $\mu_i^{(t)}$ than to any other mean.

- Update step: Calculate the new means (centroids) of the observations in the new clusters as:

$$\mu_i^{(t+1)} = \frac{1}{|S^{(t)}i|} \sum x \in S_i^{(t)} x. \tag{A.9}$$

The algorithm converges when the assignments no longer change or according to a tolerance factor. Therefore, the resulting clusters correspond to the minimization of within-cluster variances:

$$\arg\min_S \sum_{i=1}^{k} \sum_{x \in S_i} |x - \mu_i|^2. \tag{A.10}$$

There are many modifications and improvements that were proposed for the original k-means algorithm. For instance, the $k++$ method (184) is often used for better initialization of the centroids, rather than a simple random sampling, which spreads the initial clusters with probability proportional to its squared distance from the point's closest existing cluster center. Another commonly used technique is to measure the quality of the partitions and select the best one among a set of random repetitions. This can also be used for estimating the optimal number of clusters k, rather than requiring it from the user. In the following, we discuss these techniques in more detail.

A.2 Validation and evaluation of machine learning techniques

There are several ways to measure the performance and compare ML methods. Here, we focus on the techniques adopted throughout this thesis. The following subsections present a general description and formulation of validation procedures and performance metrics for supervised and unsupervised cases.

A.2.1 Validation procedure

In the context of supervised classification, the validation procedure involves splitting the original labeled data into a training set and a validation set. The training set is used to build the model and the validation set is used to evaluate the model's performance. This is performed to avoid overfitting, *i.e.*, when the model learns the training data too well and performs poorly on unseen data. In some cases, a common approach is to have a third split, the test set. In this sense, the model is built/trained on the training set, the validation set is used to estimate its performance and tune its hyperparameters, and the best model selected from this procedure is evaluated on the test set. Therefore, even if the model does not have access to the labels of the validation set, it serves as a separate set for tuning. This procedure allows a better tuning of the model without causing overfitting and represents a better performance estimation in real cases.

Various benchmark datasets provide the data splits beforehand, which facilitate the evaluation and comparison of different ML methods. However, this is a common approach in research, where the same benchmarks are considered for comparing between methods. In real problems or when new data is collected, the data splits are not available beforehand. For these cases, there are several splitting techniques for evaluating an ML model without overestimation or underestimation of its performance. One of the most popular validation techniques is the k-fold cross-validation. It consists of partitioning the data into k equal-sized sets, or folds, and the ML model is trained on $k-1$ folds and tested on the remaining fold, while a validation split can be obtained as an additional partition of the training set. The k-fold cross-validation consists of repeating this procedure k times, *i.e.*, each fold is used as a test set once. The results are then averaged to produce a more reliable estimate of model performance.

In practice, there are various approaches to perform k-fold cross-validation. The most common is to split the data randomly. In these cases, it is recommended to perform a set of repetitions for the whole procedure, since the folds may change drastically. Another common approach is to keep the number of samples per class balanced across each fold, which is usually referred to as stratified k-fold cross-validation. One particular case of k-fold cross-validation that does not involve random/stochastic procedures is when $k = n$, where n is the number of data points in the dataset. This is known as leave-one-out cross-validation, and consists of using each data point as a test set once, resulting in a total of n training/test iterations. However, the computational load can be considerable for large datasets. One common approach in these cases is the hold-out scheme, where the data is split using a fraction of the dataset (a common case is 80% for training and 20% for testing). Nevertheless, since this procedure is also random, repetitions are needed for better performance estimation.

A.2.2 Metrics used for supervised classification

During cross-validation or other validation schemes, the ML models are evaluated in terms of their performance. In this sense, quantitative measures are needed to understand what is happening during training, or the ability of the model to predict unseen data. In this thesis, we employ three fundamental metrics for supervised classification tasks: accuracy, precision, and recall. These metrics allow us to quantify the performance of our models in different aspects. Accuracy is the most intuitive performance metric. It is simply the ratio of correct predictions among the total number of predictions made. Firstly, consider a simple binary classification task, *i.e.*, there are only two classes (positive or negative). After training an ML model following an appropriate data split, and applying it to predict the instances from the test set, the accuracy can be computed by:

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}, \tag{A.11}$$

where TP are True Positives (instances correctly predicted as positive), TN are True Negatives (instances correctly predicted as negative), FP are False Positives, or "type I error" (instances incorrectly predicted as positive), and FN are False Negatives, or type II error (instances incorrectly predicted as negative). Intuitively, the multiclass case can be approached by including the metrics for each class, respectively.

Precision refers to the ability of the classifier to accurately identify negative samples, avoiding any misclassification as positive. This essentially means that a high-precision classifier minimizes the occurrence of false positives, which can be calculated as:

$$\text{Precision} = \frac{TP}{TP + FP}. \tag{A.12}$$

Recall, also known as sensitivity, hit rate, or true positive rate, measures the capability of the classifier to correctly detect and identify all instances that are positive within the data set. A model with a high recall rate successfully locates all positive samples, minimizing the occurrence of false negatives, which can be computed by:

$$\text{Recall} = \frac{TP}{TP + FN}. \tag{A.13}$$

Precision and recall can be extended to the multiclass case as well by simply accounting for the predictions of all classes, or computing it to each class against all the others and then averaging the results. Each of these metrics provides a unique perspective on the performance of an ML model applied for supervised classification, and together they offer a comprehensive view of the model's effectiveness. However, it is important to understand the trade-off between precision and recall while making predictions. High precision means that an algorithm returned substantially more relevant results than irrelevant ones, while high recall means that an algorithm found most of the relevant results.

A.2.3 Unsupervised validation and metrics

For unsupervised clustering methods like k-means, validation is slightly different than in supervised learning. Since there are no "correct" labels, it is not straightforward to compare the clusters that were found to the ground truth. Instead, we must use other metrics to evaluate the quality of the clusters. One common approach is the Silhouette score, which measures how similar each sample is to the other samples in its cluster compared to samples in other clusters. Or in other words, it simply measures the degree of separation between clusters. The score ranges from -1 to 1, with a high score indicating that the data point fits well within its cluster and poorly in neighboring clusters, and a negative score signifying that the data point might have been assigned to the wrong cluster. The silhouette score for each sample x is computed as:

$$\text{Silhouette } (x) = \frac{b(x) - a(x)}{\max(a(x), b(x))}, \tag{A.14}$$

where $a(x)$ is the average distance between sample x and all other points in the same cluster (mean intra-cluster distance), and $b(x)$ is the distance between x and the nearest cluster that the sample is not a part of (mean nearest-cluster distance). To estimate the clustering quality for all samples, the most common approach is to simply consider their average silhouette.

Another common usage for the silhouette metric is to improve the results of k-means by reducing the effect of random initialization. In practice, one can consider several random initializations, optimize the centroids, and compute the silhouette for the results. The clustering with the highest silhouette can be used as the final result.

The Elbow method is a technique that helps to select the optimal number of clusters for k-means by fitting the model with a range of numbers of clusters and selecting the one that adds the least marginal benefit. This method involves plotting the variance explained as a function of the number of clusters, and then selecting the number of clusters where the increase in variance explained begins to level off—this point resembles an "elbow". Mathematically, the within-cluster sum of squares (or distortion) for k clusters is defined as:

$$\text{distortion} = \sum_{i=1}^{k} \sum_{x \in S_i} |x - \mu_i|^2, \tag{A.15}$$

where x are the observations, S_i are the clusters, and μ_i are the cluster means. Finally, by plotting the distortion as a function of k, it is possible to detect the "elbow" in the curve, *i.e.*, the point where the rate of decrease of the distortion dramatically slows down. This point is then considered as a good estimate for the number of clusters.

APPENDIX B – PUBLIC COMPUTER VISION DATASETS

There are many public benchmarks used by the CV community to evaluate and compare techniques, according to different tasks. Here, we focus on supervised image classification, so we explore datasets where the data points are pairs of (image, label). However, even within these delimitations, there is a vast amount of possibilities. We selected a set of benchmarks with different properties and according to the method we want to evaluate. In the following, we describe these datasets, show various examples from them, and compare their characteristics. We start by covering texture analysis datasets, as this is the main application within our proposals and results.

B.1 Texture datasets

The public texture datasets we employ can be roughly divided into two categories: pure texture (homogeneous) or "in-the-wild". In the following, we describe each of them in different sections.

B.1.1 Pure texture bechmarks

We refer to a dataset as "pure texture" when its images are composed of a homogeneous texture surface of a single sample, filling the whole image. Or, in other words, when there is no background or additional information in the image, such as other objects or other textures. Historically speaking, this kind of dataset is the most common since the early days of texture recognition, measuring the capacity of a model to characterize a single texture in question. This scenario is common in several applications when image acquisition is controlled, or when studying textures in a laboratory. Among some examples, we can mention microscopy imaging, industrial inspection, and medical imaging. In the following, the reader can observe different image samples from this kind of dataset, as we describe them.

We refer to each dataset in chronological order considering when they were published. In this sense, the first one we employ is the Vision Texture dataset, a.k.a. Vistex. (110) It was published in 1995 as one of the first comprehensive datasets of high-quality color textures that are representative of real-world conditions. Vistex contains images that do not adhere to standardized frontal plane perspectives and controlled studio lighting requirements. The content of the images varies from textures of tree bark, fur, stones, vegetables, water, soil, and others. In total, there are 54 texture classes in the dataset, which can be seen in Figure 41. For each texture, there is a set of 16 patches (each one is an image) with dimensions (width and height) 128 $\times$ 128 pixels, totaling 864 images. When employing this dataset for evaluation, the most common approach is to randomly split it iteratively into independent training and test sets, such as with stratified k-fold or leave-one-out cross-validation.

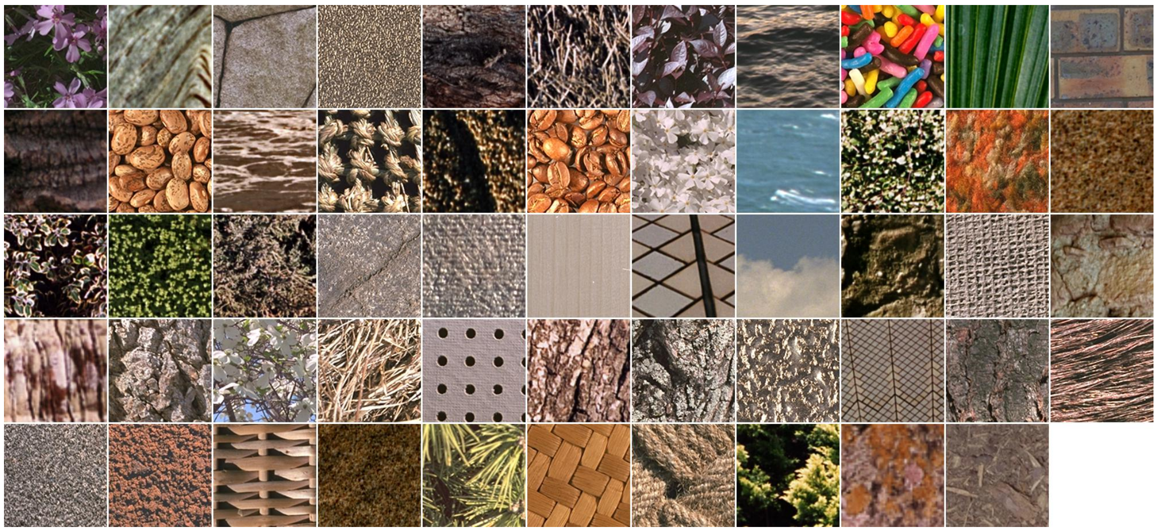


Figure 41 – All the 54 texture classes from the texture recognition dataset Vistex.

Source: (110).

In 1999, the Columbia-Utrecht Reflectance and Texture dataset (CUReT) (105) was proposed, containing colored images of various materials. The most important characteristic of this dataset is the wide variety of geometric and photometric properties of the textures, such as over 200 different combinations of viewing and illumination directions. We use the dataset as in (259) by obtaining central crops of 200 $\times$ 200 pixels. This produces a total of 92 image samples for each of 61 textures, resulting in 5612 images in total. To illustrate the intra-class variations in this dataset, in Figure 42 we show three samples (a, b, and c) of each of a set of 20 randomly selected texture classes.

The Outex framework (106) from 2002 is proposed for the empirical evaluation of texture analysis methods and consists of several different datasets. Here we approach two of them, Outex_TC_00010 and 13, which we refer to as Outex10 and Outex13, for simplicity. Outex10 is composed of 4320 grayscale images of 128 $\times$ 128 pixels, evenly divided into 24 different texture classes. It focuses on rotation invariance, so the images are rotated at 9 different angles (0°, 5°, 10°, 15°, 30°, 45°, 60°, 75°, 90°). The dataset has a fixed training and testing split which we illustrate in Figure 43 (a-b) by showing one randomly selected sample for each of the 24 classes, in each split (training or test). Similarly, Outex13 also has a fixed training and testing split, but this dataset focuses on color texture analysis. The dataset contains 1360 images of 128 $\times$ 128 pixels divided into 68 color texture classes, that is, 20 samples per class. Differences in illumination and color patterns are present for each class, between the training and test sets, as we show in Figure 43 (c-d).

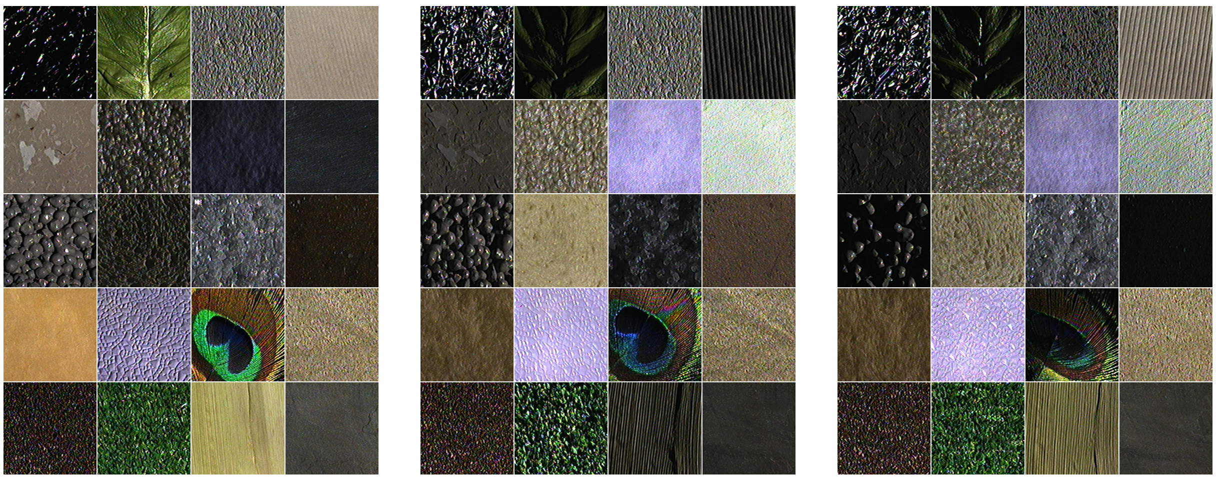
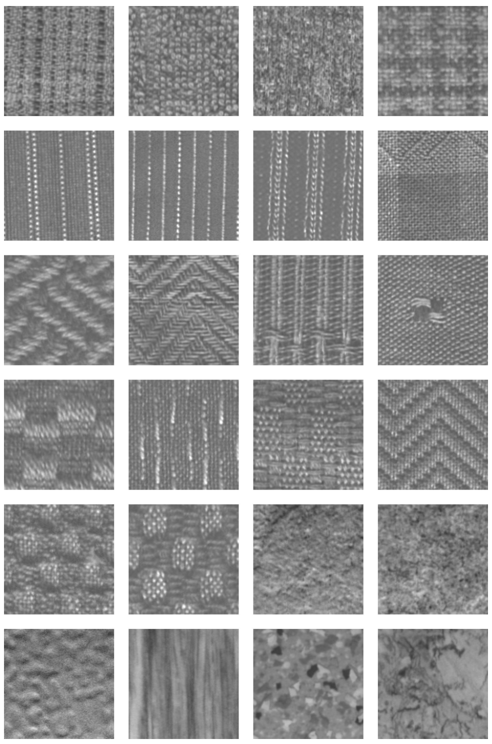

(a) (b) (c)

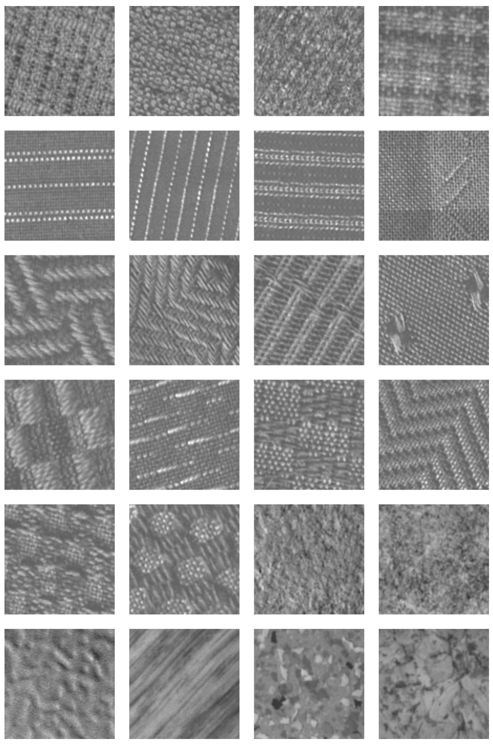
Figure 42 – A set of 20 randomly selected textures from the CUReT dataset, and three different samples for each one of them (a-c).

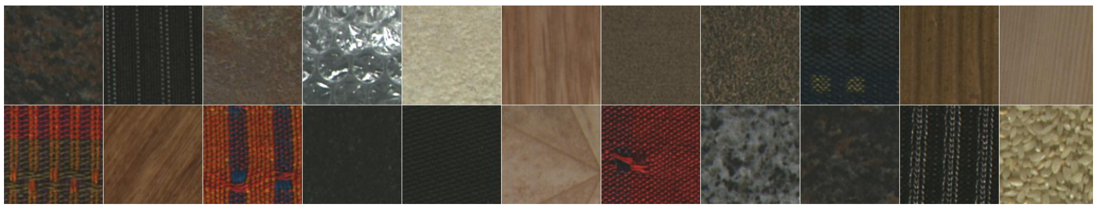
Source: (105).

A more challenging texture recognition benchmark is KTH-TIPS2-b. (107, 108) It contains 4752 images with 200 × 200 pixels from 11 different materials, which are split in a fixed set for 4-fold cross-validation. The images are captured under 9 different scales equally spaced logarithmically per sample, 3 camera poses (frontal, 22.5° left and 22.5° right), and 4 illumination conditions (front, from the side at roughly 45°, from the top at roughly 45°, and using ambient lighting). Figure 44 (a-d) shows images from each of the 11 texture classes, for the four independent splits of the dataset, where each split is composed of images from a different sample (a different object/material was used for image acquisition). Additionally, we built a smaller version (MNIST-like) of KTH-TIPS2-b by cropping the original 200 × 200 samples into 28 × 28 non-overlapping grayscale patches, from 10 selected texture classes (aluminum foil, cork, wool, corduroy, linen, cotton, brown bread, white bread, wood, and cracker). We divide the new crops into training and testing folds by ensuring that no crops of the same image appear together in the same fold. It yields a total of 165 thousand samples for training and 35 thousand for the test. We then randomly select 60 thousand images for training and 10 thousand for the test.

In 2012, the USPtex dataset (111) was proposed with the intuition to provide texture images of objects and materials found daily. It was built by the University of São Paulo and contains 191 classes of natural-colored textures. Originally, one image for each texture class was obtained with a resolution of 512 × 384. They are then divided into 12 crops of 128 × 128 pixels without overlap, totaling 2292 images in total. A selection of 70 texture classes from USPtex is shown in Figure 45. Since this dataset does not provide fixed validation splits, the evaluation is usually done with stratified k-fold or leave-one-out


(a) Outex10 training set.


(b) Outex10 test set.


(c) Outex13 training set.

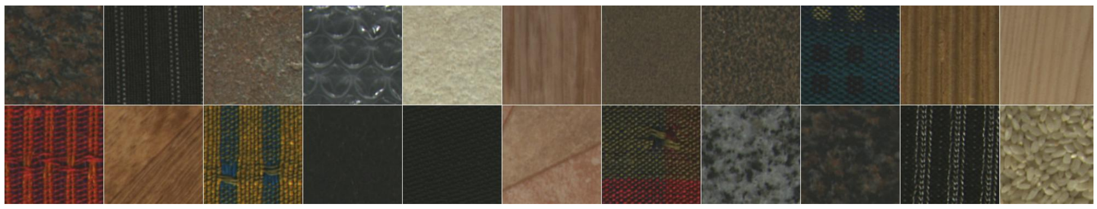

(d) Outex13 test set.

Figure 43 – Samples randomly selected from the Outex10 and Outex13 datasets. For Outex10, we show one sample for each of its 24 texture classes, for both its training (a) and test (b) split. For Outex13, we randomly select 22 classes out of the 68 and show training (c) and testing (d) samples from them.

Source: (106).

cross-validation.

The last pure-texture dataset we use is the Multi-Band Texture (MBT). (115) Compared to the previous datasets, MBT textures are far more unusual in daily life. The images are produced by combinations of intraband and interband spatial varia-

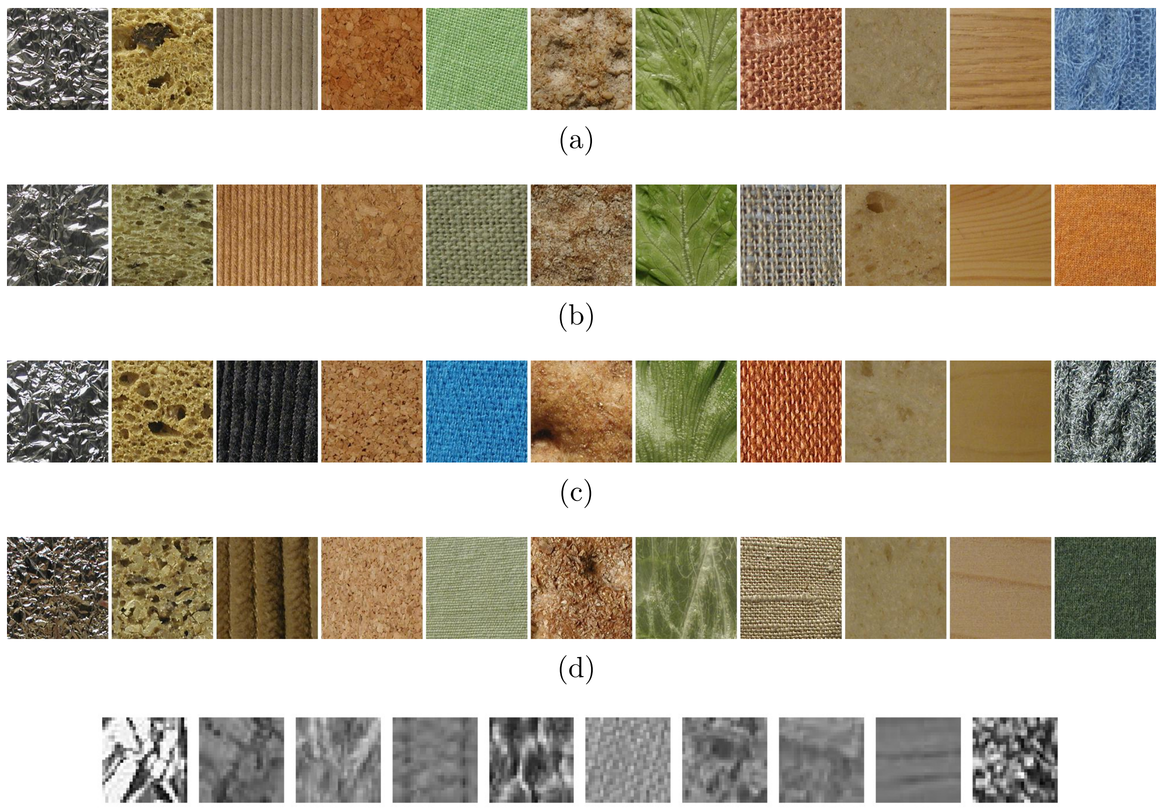


Figure 44 – Samples from the KTH-TIPS2-b dataset. In this work, we used the original dataset (a-d) and a reduced version (e) where non-overlapping grayscale crops of size 28 × 28 are obtained from 10 classes of the original dataset.

Source: (107).

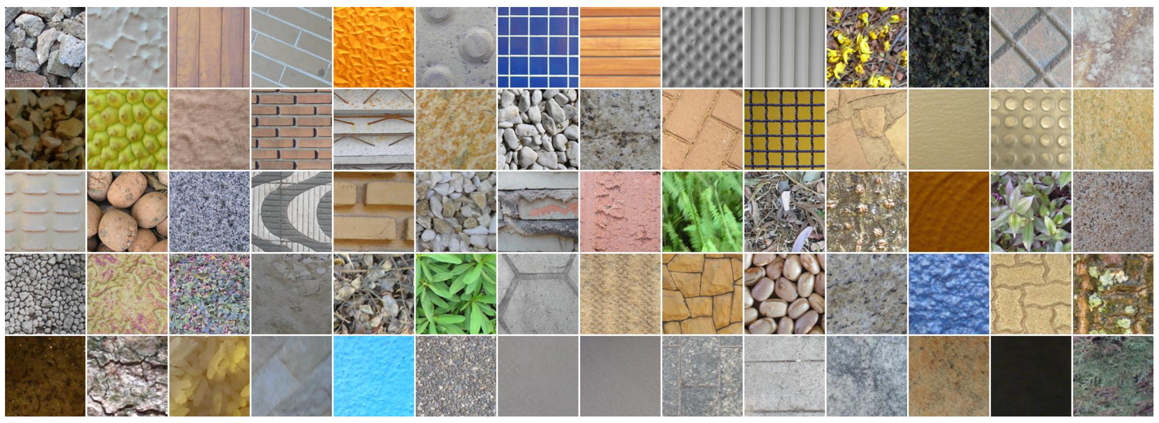


Figure 45 – A set of 70 randomly selected textures from the USPtex dataset.

Source: (111).

tions. This type of pattern appears in images with high spatial resolution and multi-spectral/hyperspectral information, which are common in areas such as astronomy and remote sensing. The original dataset contains 154 textures with a resolution of 640 ×

640. Here, we divided each of them into 16 non-overlapping crops of 160 × 160 pixels, composing a set of 2464 images (as in (19)). They are split using common cross-validation techniques since no pre-defined splits are given. Figure 46 shows a selection of 70 textures from MBT, where it is possible to see their particular visual characteristics compared to the previous datasets. These patterns usually emerge when image acquisition captures spectral bands outside the visible spectrum, and visualizing the spatial patterns in these bands in a trichromatic color space (such as RGB) results in uncommon visual characteristics.

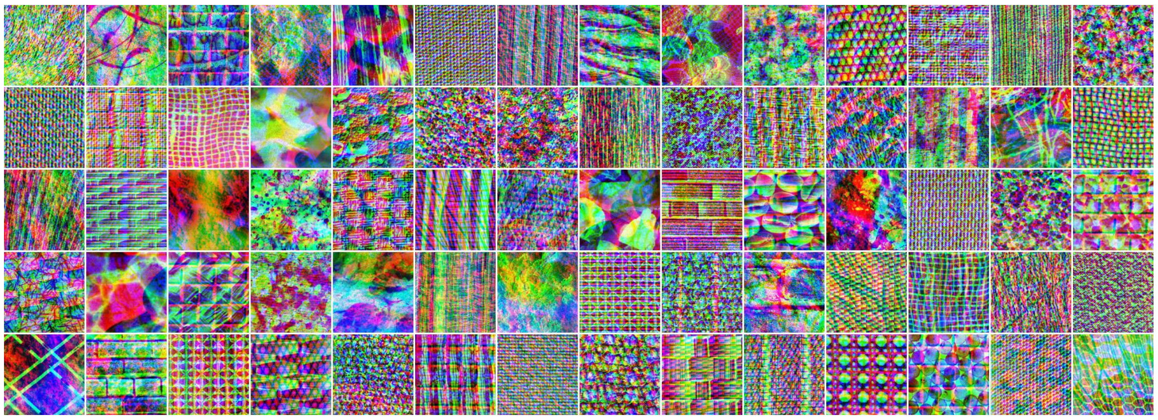


Figure 46 – 70 randomly selected texture classes from the cropped version of the MBT dataset.

Source: (115).

B.1.2 Textures in-the-wild

Another class of texture benchmarks, which we refer to here as *textures in-the-wild*, follows different principles than pure texture. The images are usually obtained from the web or generated outside a strictly controlled setting. This type of dataset became particularly popular in texture analysis after the popularization of deep learning. The first dataset that we employ and which fits into this category is the Flickr Material Dataset (FMD). (109) It was proposed to evaluate how quickly humans can recognize natural materials under various conditions, including brief presentations and image degradation through manipulations like removing color, blurring, or inverting contrast. The dataset contains 1000 images obtained from websites like *flickr.com*, with a resolution 512 × 384 and considering 10 different materials: fabric, foliage, glass, leather, metal, paper, plastic, stone, water, and wood. The images are chosen for their diverse appearances, and cropped to eliminate object shapes as a cue. Methods are evaluated using common cross-validation techniques (usually 10-fold). Figure 47 shows 40 image samples from FMD, where each column is a class, and each row is a different sample from the same class. It is possible to notice the complex structure of the samples, where there is no control over the background, multiple objects appear in the same image, and even multiple textures. For instance, the image in the 2nd row and 9th column contain both water and foliage textures.

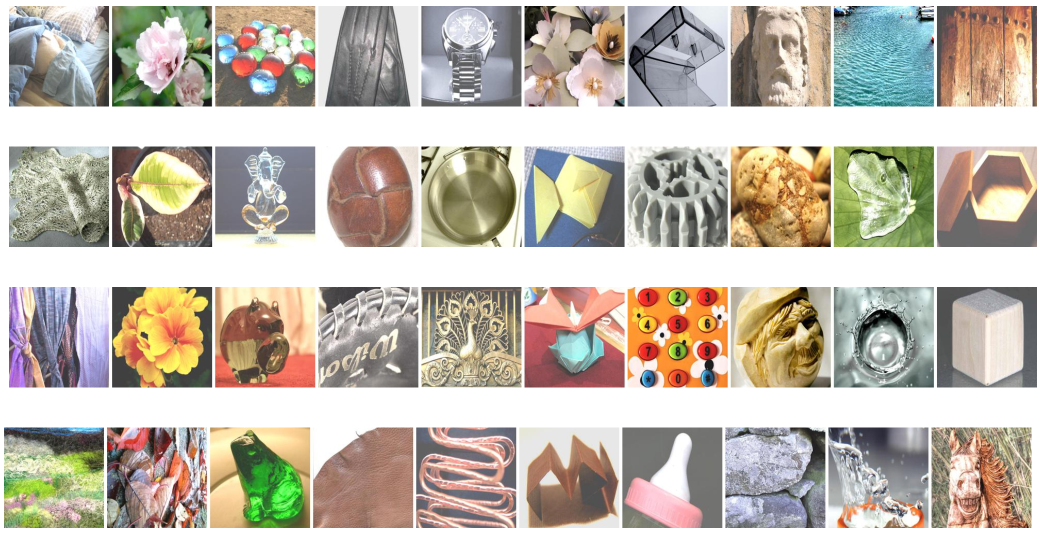
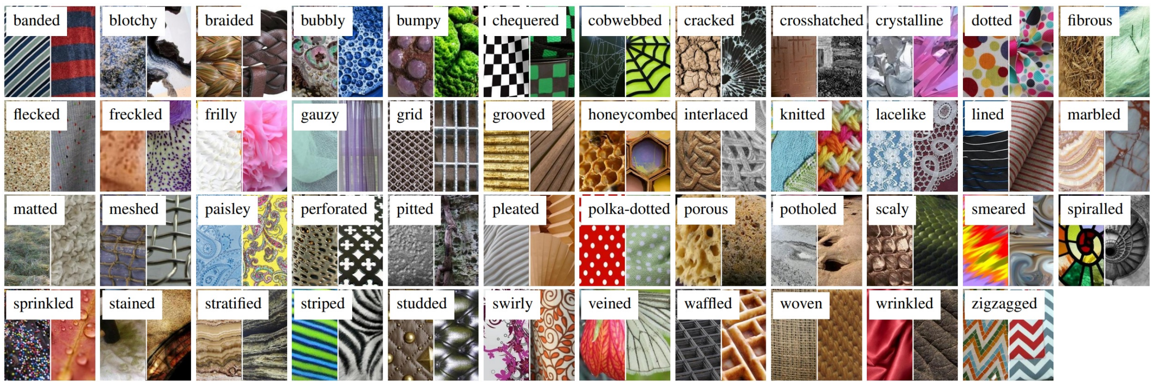


Figure 47 – Examples from the 10 classes of the FMD dataset, where each column is a class, and each row is a different sample.

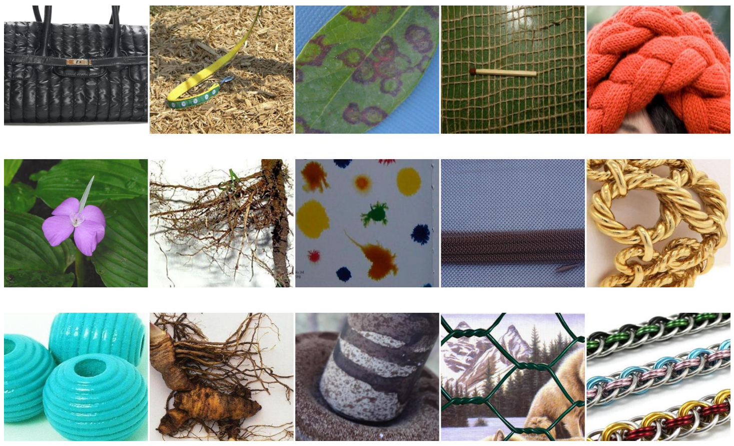
Source: (109).

The term *textures in-the-wild* is employed by Cimpoi *et al.* (114) to describe their proposed Describable Texture Dataset (DTD). According to the authors, it targets the problem of texture description, intended as the recognition of describable texture attributes. In contrast to material recognition, which is the target of other datasets such as CUReT, KTH, FMD, or USPtex, texture attributes do not imply materials (*e.g.*, veined may equally apply to leaves or marble), and materials do not imply attributes (not all marbles are veined). (114) Therefore, the DTD dataset emphasizes the importance of human-centric texture descriptors, which are essential for understanding and modeling human perception of textures. The *in-the-wild* term comes from the fact that the images are collected from the internet, and using a semi-automatic process that involves querying internet image search engines using human-generated textual queries. This resulted in a dataset of 5640 images of 47 different texture classes, or attributes (Figure 48 (a)), with varying resolutions. Examples of DTD images may contain background and multiple objects, as can be seen in Figure 48 (b). The evaluation of this dataset is performed using a set of 10 provided splits for training, validation, and testing.

The Ground Terrain in Outdoor Scenes (GTOS) (112, 113) dataset is designed to facilitate ground terrain texture recognition in applications such as autonomous driving and robot navigation. Compared to the previous datasets, this is a medium/large-sized dataset with more than 30,000 images, divided into 40 distinct classes of outdoor ground terrain. The images capture a wide range of weather and lighting conditions, with 18 viewing directions, 4 illumination conditions, and 3 exposure settings per sample. They



(a) DTD's 47 texture attributes/classes.

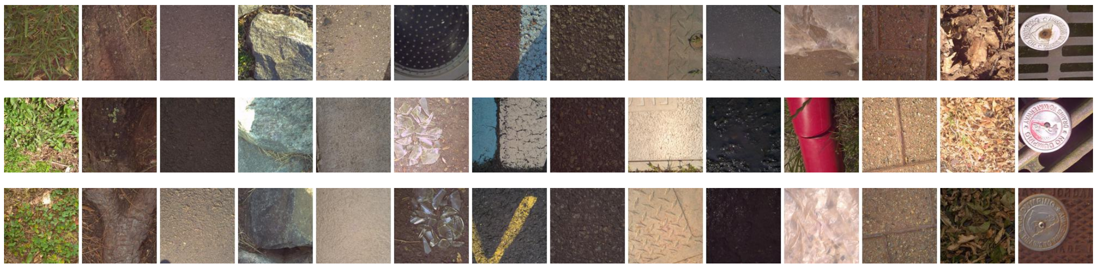


(b) A selection of samples from the classes "blotchy", "braided", "fibrous", "grooved", and "meshed".

Figure 48 – The texture classes from the DTD dataset (a), and a selection of three samples from some of them (b).

Source: (114).

also provide a pre-defined set of 5 train/test splits. Figure 49 (a) shows 42 samples from GTOS, where each column is a class and each row is a different sample from that class. Additionally, the GTOS-Mobile dataset (Figure 49 (b)) is a bigger version with over 100 thousand images obtained from frames of videos captured using a hand-held mobile phone. This version is composed of 31 classes that are similar, but not the same as in the original GTOS dataset. In the original work, the authors trained a model on GTOS and tested it on GTOS-Mobile, but it is often the case to also evaluate a model on each dataset separately. This is possible considering that they also provide a predefined training/test split for GTOS-Mobile. At first glance, the GTOS datasets may appear as pure texture, but we cannot assume that for all samples, as Figure 49 shows.


(a) GTOS.

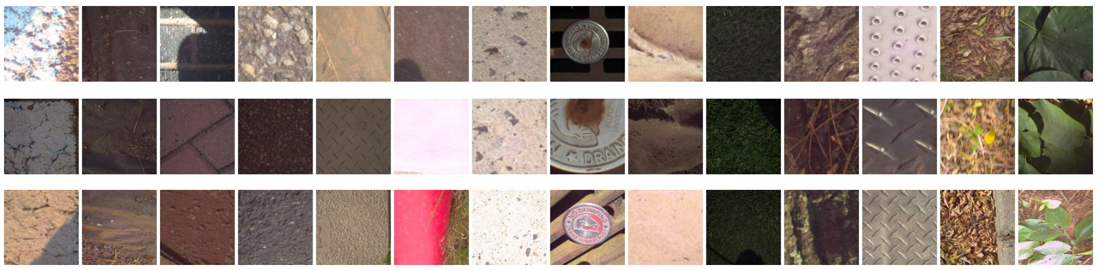

(b) GTOS-Mobile.

Figure 49 – 42 samples from both the GTOS and GTOS-Mobile datasets, where each column is a class and each row is a different sample.

Source: (112, 113).

B.2 Other image classification datasets

During the development of this thesis, other datasets were employed aside from texture recognition. We start by describing ImageNet (54, 55) which is one of the most important large-scale datasets in the CV community. It has contributed significantly to the growth and progress of deep learning models for object recognition and classification. It also has become an important resource for the pre-training of deep models, and also a benchmark for comparisons. ImageNet contains millions of images that are hierarchically organized (see Figure 50). In this work, we employ ImageNet-21k and ImageNet-1k, which are two different versions. ImageNet-21k is simply the full version of the dataset, which consists of 14,197,122 images, divided into 21,841 classes, but which lacks an official train-validation split. In this sense, this dataset is used as in (260) which combines several techniques for using this dataset for pre-training. ImageNet-1k, as the name suggests, is a smaller subset specifically designed for the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) (55) in 2012. This subset consists of approximately 1.2 million high-resolution images that are distributed across 1,000 object categories. Each category in ImageNet-1k contains roughly 1,000 images, ensuring a balanced distribution across classes. The dataset is split into training, validation, and testing sets, with 1,000 images per class allocated for training, 50 images per class for validation, and an undisclosed number of images per class for testing. Works usually compare the performance on the validation set.

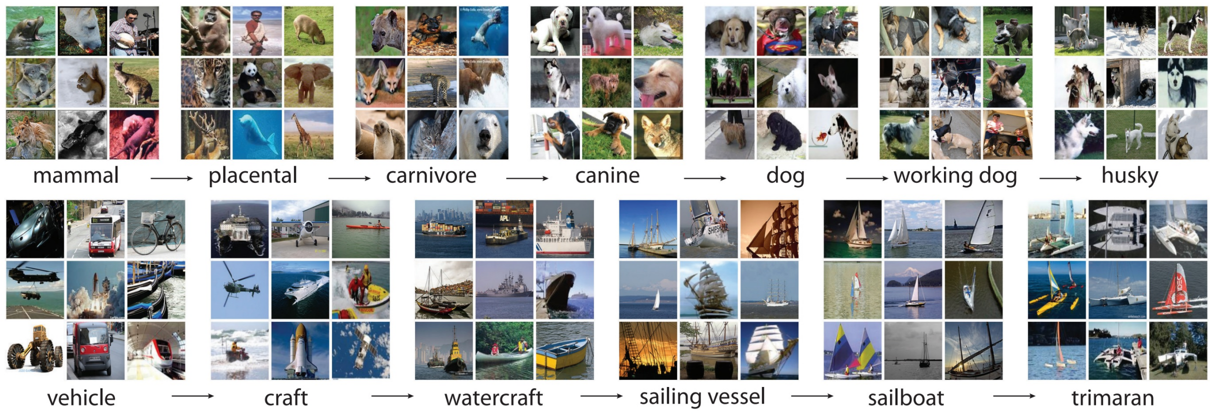
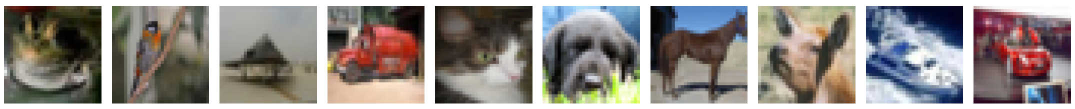


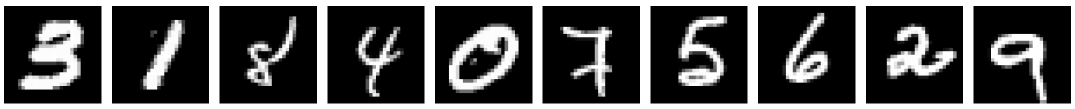
Figure 50 – An example of the ImageNet hierarchical organization, where images of objects are organized according to root-to-leaf branches of related concepts.

Source: (54).

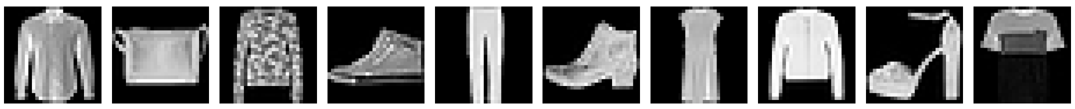
The Canadian Institute For Advanced Research (CIFAR-10) dataset (182) is another object recognition benchmark but on a much lower scale compared to ImageNet. It consists of 60,000 color images, divided into 10 classes, with each image having 32 × 32 pixels. The dataset is split into 50,000 training images and 10,000 test images. The 10 classes represent common objects such as airplanes, automobiles, birds, cats, deer, dogs, frogs, horses, ships, and trucks. One example for each class is shown in Figure 51 (a). Another dataset with a similar dimension is the Modified National Institute of Standards and Technology (MNIST) (139) dataset, a widely-used benchmark for handwritten digit recognition tasks. It contains 70,000 grayscale images of handwritten digits (from 0 to 9), with each image having a size of 28x28 pixels. The dataset is split into 60,000 training images and 10,000 test images, with an equal distribution of samples across the 10 classes. Figure 51 (b) shows some examples. Fashion MNIST (181) is a more challenging alternative to the MNIST dataset, designed to address its limitations in terms of simplicity and overuse. It contains 70,000 grayscale images of 10 fashion items: t-shirts, trousers, pullovers, dresses, coats, sandals, shirts, sneakers, bags, and ankle boots (examples shown in Figure 51 (c)). Each image has 28 × 28 pixels, and it is also split into 60,000 samples for training and 10,000 for testing.



(a) CIFAR10.



(b) MNIST.



(c) Fashion MNIST.

Figure 51 – One example for each class of each of the corresponding datasets.

Source: (139, 181, 182).