UNIVERSIDADE DE SÃO PAULO
Instituto de Ciências Matemáticas e de Computação

Multi-layer analysis of convolutional neural networks for transfer learning applications

Rayner Harold Montes Condori

Tese de Doutorado do Programa de Pós-Graduação em Ciências de Computação e Matemática Computacional (PPG-CCMC)

ICMC USP
SÃO CARLOS

Rayner Harold Montes Condori

Multi-layer analysis of convolutional neural networks for transfer learning applications

Thesis submitted to the Instituto de Ciências Matemáticas e de Computação – ICMC-USP – in accordance with the requirements of the Computer and Mathematical Sciences Graduate Program, for the degree of Doctor in Science. *EXAMINATION BOARD PRESENTATION COPY*

Concentration Area: Computer Science and Computational Mathematics

Advisor: Prof. Dr. Odemir Martinez Bruno

USP – São Carlos
January 2022

Ficha catalográfica elaborada pela Biblioteca Prof. Achille Bassi
e Seção Técnica de Informática, ICMC/USP,
com os dados inseridos pelo(a) autor(a)

```
        Montes Condori, Rayner Harold
M772m      Multi-layer analysis of convolutional neural
        networks for transfer learning applications /
        Rayner Harold Montes Condori; orientador Odemir
        Martinez Bruno. -- São Carlos, 2022.
           215 p.


           Tese (Doutorado - Programa de Pós-Graduação em
        Ciências de Computação e Matemática Computacional) --
         Instituto de Ciências Matemáticas e de Computação,
        Universidade de São Paulo, 2022.


           1. Deep Learning. I. Martinez Bruno, Odemir ,
        orient. II. Título.
```

Bibliotecários responsáveis pela estrutura de catalogação da publicação de acordo com a AACR2:
Gláucia Maria Saia Cristianini - CRB - 8/4938
Juliana de Souza Moraes - CRB - 8/6176

Rayner Harold Montes Condori

Análise multicamada de redes neurais convolucionais para aplicações de transferência de conhecimento

Tese apresentada ao Instituto de Ciências Matemáticas e de Computação – ICMC-USP, como parte dos requisitos para obtenção do título de Doutor em Ciências – Ciências de Computação e Matemática Computacional. *EXEMPLAR DE DEFESA*

Área de Concentração: Ciências de Computação e Matemática Computacional

Orientador: Prof. Dr. Odemir Martinez Bruno

USP – São Carlos
Janeiro de 2022

I would like to dedicate this thesis to my mother Giovanna, my father Juvenal and girlfriend Daysi Katherine and all my family.

Also I say thank you to everybody that help me throughout these years

ACKNOWLEDGEMENTS

“As invenções são, sobretudo,
o resultado de um trabalho de teimoso.”
(Santos Dumont)

RESUMO

RAYNER H. M. CONDORI. **Análise multicamada de redes neurais convolucionais para aplicações de transferência de conhecimento**. 2022. 212 p. Tese (Doutorado em Ciências – Ciências de Computação e Matemática Computacional) – Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Carlos – SP, 2022.

O aprendizado profundo tornou-se um tema quente na inteligência artificial devido à sua capacidade de modelar conceitos complexos a partir de conceitos simples. Nesse sentido, a rede neural convolucional (CNN) é um dos tipos mais populares de redes neurais atualmente utilizadas em visão computacional e áreas afins. Em geral, os seguintes fatores contribuíram para sua popularidade. (i) Com dados suficientes, a maioria das CNNs podem ser treinadas do zero e aprender representações poderosas que resolvem a tarefa em jogo. (ii) Por outro lado, com um volume limitado de dados, é possível também aprender representações poderosas adaptando o conhecimento de um modelo CNN pré-treinado por meio de uma estratégia de aprendizagem por transferência. Como resultado, as CNNs avançaram o estado da arte em muitas tarefas de reconhecimento visual, levando a inúmeras aplicações em vários campos fora da ciência da computação, como medicina e biologia. No entanto, muitos dos melhores esforços de pesquisa estão focados em melhorar o estado da arte só em alguns conjuntos de dados, como `ImageNet` para classificação de imagens e `COCO` para detecção de objetos. Porém, o progresso da pesquisa em muitos outros domínios é reduzido a aplicar cegamente as abordagens existentes ou reinventar tudo do zero, resultando no desenvolvimento de métodos falhos em ambos os casos. Portanto, esta tese se foca em entender por meio de experimentos sistemáticos por que e quando um modelo CNN pré-treinado apresenta desempenho inferior em uma determinada tarefa, a fim de propor soluções adequadas. Na primeira parte de nosso estudo, examinamos a tarefa de reconhecimento de textura e descobrimos que todos os trabalhos anteriores tendiam a se concentrar exclusivamente em conjuntos de dados de textura baseados em categorias, levando à ideia equívoca de que apenas as camadas mais profundas tinham as informações de textura necessárias para resolver essa tarefa. . Mostramos então, propondo estratégias de aprendizagem por transferência multicamadas, que a contribuição de camadas rasas não é trivial e deve ser utilizada em determinadas aplicações. Na segunda parte do nosso estudo, focamos em tarefas desafiadoras de detecção de objetos (detecção de grãos de pólen e localização de estômatos), onde observamos uma situação semelhante à do reconhecimento de texturas. Portanto, em ambos os casos, também aplicamos a análise multicamada para propor detectores rápidos de estágio único que podem lidar com imagens muito grandes com precisão e eficiência.

Palavras-chave: Redes neurais convolucionais, Transferência de conhecimento, Visão por computador, Mapas de ativação, Classificação de Imagens, Detecção de Objetos .

ABSTRACT

RAYNER H. M. CONDORI. **Multi-layer analysis of convolutional neural networks for transfer learning applications**. 2022. 212 p. Tese (Doutorado em Ciências – Ciências de Computação e Matemática Computacional) – Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Carlos – SP, 2022.

Deep learning has become a hot topic in artificial intelligence due to its ability to model complex concepts from simple ones. In this regard, the convolutional neural network (CNN) is one of the most popular kinds of neural networks currently used in computer vision and related areas. In general, the following factors contributed to its popularity. (i) With enough data, most CNNs can be trained from scratch and learn powerful representations that solve the task at stake. (ii) On the other hand, with a limited volume of data, it is possible to also learn powerful representations by adapting the knowledge of a pre-trained CNN model via a transfer learning strategy. As a result, CNNs have advanced the state-of-the-art in many visual recognition tasks, leading to numerous applications in various fields outside of computer science, such as medicine and biology. Nevertheless, many of the best research efforts are focused on improving the state-of-the-art on a few datasets, such as ImageNet for image classification and COCO for object detection. On the other hand, research progress in many other domains is reduced to blindly applying existing approaches or re-inventing everything from scratch, resulting in the development of flawed methods in both cases. Therefore, this thesis focuses on understanding through systematic experiments why and when a pre-trained CNN model underperforms on a given task, to propose suitable solutions. In the first part of our study, we examined the task of texture recognition and discovered that all previous studies tended to focus exclusively on category-based texture datasets, leading to the misconception that only the deepest layers had the texture information needed to solve that task. We then show, by proposing multilayer transfer learning strategies, that the contribution of shallow layers is not trivial and should be used in certain applications. In the second part of our study, we focus on challenging object detection tasks (pollen grain detection and stomata localization), where we observe a situation similar to that of texture recognition. Therefore, in both cases, we also applied multilayer analysis to propose fast single-stage detectors that can handle large images accurately and efficiently.

Keywords: Convolutional neural networks, Transfer learning, Computer vision, Activation maps, Image Classification, Object Detection.

LIST OF FIGURES

LIST OF ALGORITHMS

LIST OF SOURCE CODES

LIST OF TABLES

LIST OF ABBREVIATIONS AND ACRONYMS

AdaGrad	Adaptive Gradient algorithm
ADAM	Adaptive Moment Estimation
ANN	Artificial Neural Network
ANNs	Artificial Neural Networks
ANOVA	One-way Analysis Of Variance
BN	Batch Normalization
CNNs	Convolutional Neural Networks
DCF	Deep Composite Function
DL	Deep Learning
ET	Extremely Randomized Trees
FL	Focal Loss
FPN	Feature Pyramid network
GAP	Global Average Pooling
GD	Gradient Descent
GEP	Global Entropy Pooling
GMTP	Global Mean Thresholding Pooling
GP	Global Pooling
GPU	Graphical Processing Units
GPUs	Graphics Processing Units
LSTM	Long Short-Term Memory network
MI	Mutual Information
RPN	Region Proposal Network
SE	“Squeeze-and-Excitation”
SGD	Stochastic Gradient Descent
SSA	Selective Search Algorithm
TL	Transfer Learning
TPUs	Tensor Processing Units

LIST OF SYMBOLS

N_I — Number of images or batch size (first dimension of the 4D tensor)

C — Number of channels (second dimesion of the 4D tensor)

H — Height (third dimension of the 4D tensor)

W — Width (fourth dimension of the 4D tensor)

$\mathcal{I}$ — Image Space

$\mathcal{Y}$ — Label Space

$\mathcal{D}_{\mathbf{S}}$ — Source Domain

$\mathcal{D}_{\mathbf{T}}$ — Target Domain

$\mathcal{T}_{\mathbf{S}}$ — Source Task

$\mathcal{T}_{\mathbf{T}}$ — Target Task

$D_{\mathbf{S}}$ — Source Domain Dataset

$D_{\mathbf{T}}$ — Target Domain Dataset

$n_{\mathbf{S}}$ — Number of samples in $D_{\mathbf{S}}$

$n_{\mathbf{T}}$ — Number of samples in $D_{\mathbf{T}}$

$f_{\mathbf{S}}(\cdot)$ — Source Predictive Function

$f_{\mathbf{T}}(\cdot)$ — Target Predictive Function

N_{splits} — Number of dataset splits

I — Image

$\mathbf{I}$ — Set of images

y — Target

Y — Set of targets

CONTENTS

CHAPTER 1

INTRODUCTION

1.1 Contextualization

Deep Learning (DL) is a class of machine learning techniques that allows computational models to learn features with multiple abstraction levels (LECUN; BENGIO; HINTON, 2015). Since most DL models organize their features in multiple layers, some authors also refer to DL as *layered representations learning* or *hierarchical representations learning* (CHOLLET, 2017a, Sec. 1.1.4).

DL frameworks like PyTorch (PASZKE *et al.*, 2017) and Keras (CHOLLET *et al.*, 2015) implement DL models as computational graphs to ensure adequate training and inference processes. From a high-level standpoint, every layer in a DL model becomes a node, and its connections to other layers symbolize the edges. Therefore, one way to estimate the "depth" of any DL model is to calculate the length of its computational graph's longest path (GOODFELLOW; BENGIO; COURVILLE, 2016, Ch. 1). Then, given an input sample, a DL model generates a hierarchy of feature representations by traversing its computational graph from start to finish. This means that each new visited node outputs a feature representation by applying a mathematical function to the representations previously calculated by its neighboring nodes. During training, the DL model goes from initially calculating only low-level feature representations to computing multiple low, mid, and high-level feature representations. As a consequence, every feature in a trained DL model is the result of a – typically large – composition of learned functions. In general, layers toward the end of the graph learn more abstract features than those learned in initial or intermediate layers (ZEILER; FERGUS, 2014; OLAH *et al.*, 2020).

DL models make extensive use of deep Artificial Neural Networks (ANNs) due to their great flexibility and ability to learn from vast amounts of data (ZHANG *et al.*, 2018a; Kolesnikov *et al.*, 2019; Brown *et al.*, 2020). In this regard, numerous deep ANN models have been developed over the past decade, most of which achieved impressive performance on a wide variety of challenging tasks (RAGHU; SCHMIDT, 2020). Indeed, given the ubiquity of DL, the number of research articles is staggeringly high in practically all areas of knowledge. Therefore, today it is common to find comprehensive review articles dedicated to analyzing and comparing DL models applied to particular topics, such as small object detection (TONG; WU; ZHOU, 2020), citation recommendation (ALI *et al.*, 2020) texture analysis (LIU *et al.*, 2019), plant phenotyping (SINGH *et al.*, 2018; MOCHIDA *et al.*, 2019), or medicine-related applications (MEYER *et al.*, 2018; PICCIALLI *et al.*, 2021).

Since the arrival of the ALEXNET model in 2012 (KRIZHEVSKY; SUTSKEVER; HINTON, 2012), computer vision has become one of the main fields in which DL research achieved groundbreaking advances. In this regard, although some promising *Transformer*-based DL models[1] emerged very recently (Carion *et al.*, 2020; DOSOVITSKIY *et al.*, 2021), Convolutional Neural Networks (CNNs) remain by far the most successful ANNs used in computer vision. Indeed, the scientific community continually advances the capabilities and performance of CNN models by developing innovative architectural designs (KHAN *et al.*, 2020). However, CNN models require training processes with large datasets such as ImageNet (RUSSAKOVSKY *et al.*, 2015) or JFN-300M (SUN *et al.*, 2017) to learn robust features for the task at stake. Furthermore, even with enough data, training large CNN models from scratch is impractical without the computing power of numerous Graphics Processing Units (GPUs) or Tensor Processing Units (TPUs) (JOUPPI *et al.*, 2017; TANG *et al.*, 2019). Therefore, in situations where large datasets are not available, or there is little access to high-performance computing hardware, most researchers resort to Transfer Learning (TL) strategies (OQUAB *et al.*, 2014; CIMPOI *et al.*, 2016).

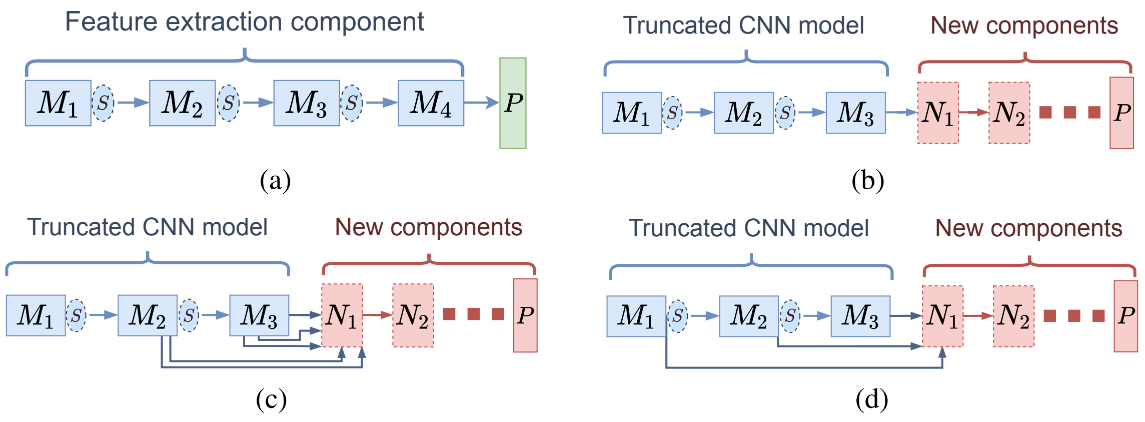
Within this thesis framework, we refer to TL as a class of strategies that take a pre-trained CNN model and adapt its semantically robust features to solve a target domain task. In this regard, the term "pre-trained" refers to any CNN model successfully trained on a proper source domain dataset. We specially focus on a TL setting known as *inductive* TL, in which both the source and target domains contain labeled data (Ribani; Marengoni, 2019). Hence, this thesis does not include any pre-trained CNN model that originated from unsupervised or self-supervised learning algorithms, e.g., SIMCLR (Chen *et al.*, 2020). Additionally, we limited our research to TL strategies applied to the following tasks: (i) image classification and (ii) object detection.

A classification task consists of assigning a new image to one of a set of predefined classes based on its visual attributes. On the other hand, a detection task involves locating and recognizing multiple objects within an image (DHILLON; VERMA, 2020). In this regard, given the sustained release of pre-trained CNN models using ImageNet as their source domain dataset (HUANG *et al.*, 2017; Hu *et al.*, 2019; TAN; LE, 2019), numerous TL strategies have been developed for classification and detection tasks over the years. As shown in Figure 1a, despite the great diversity of architectural designs, many pre-trained CNN models share a base

[1] *Transformers* are very powerful DL models mainly used in natural language processing.

architecture that has the following traits. (i) It has an initial *feature extraction component* that characterizes the input image via a set of activation maps and a final *classification component* that computes the class label probabilities. (ii) The *feature extraction component* consists of two or more consecutive blocks, each involving a simple or elaborate arrangement of primarily convolutional layers, pooling layers, normalization layers (IOFFE; SZEGEDY, 2015; WU; HE, 2018), and dropout layers (SRIVASTAVA *et al.*, 2014). (iii) There is at least one sub-sampling layer between each pair of consecutive blocks, which is responsible for halving the resolution of the set of activation maps generated from one block to the next (Springenberg *et al.*, 2015).

Figure 1 – **High-level view of exemplary modified models for TL.** (a) Base architecture of a pre-trained CNN model with four blocks (M_1 – M_4) in its *feature extraction component*. (b) Modified model that reuses the first three blocks of (a) and provides single-layer feature representations to the new components. (c – d) Modified models that compute multi-layer feature representations. Unlike (c), the choice of layers in (d) is unambiguous and valid for all truncated CNN models with the same base architecture. S: sub-sampling layer. N_i: i-th newly added component. P: component that yields the final predictions for the classification or detection task.


Feature extraction component
M_1 S M_2 S M_3 S M_4 P
(a)
Truncated CNN model
New components
M_1 S M_2 S M_3 N_1 N_2 P
(b)
Truncated CNN model
New components
M_1 S M_2 S M_3 N_1 N_2 P
(c)
Truncated CNN model
New components
M_1 S M_2 S M_3 N_1 N_2 P
(d)

Source: Elaborated by the author.

In image classification, most TL strategies construct a modified model by truncating a pre-trained CNN model at a chosen layer and replacing the removed layers with new trainable components (see Figure 1b, Figure 1c, and Figure 1d). Given a target dataset, there are two primary methodologies for training the modified model. (i) The **fine-tuning methodology**, which involves optimizing the entire model in an end-to-end manner (ANDREARCZYK; WHELAN, 2016; ZHANG; XUE; DANA, 2017; ZHENG *et al.*, 2017; LI; HOIEM, 2018; CASTRO *et al.*, 2018). (ii) The **static feature extraction methodology**, in which only the new components are optimized with feature representations from the truncated CNN model (RAZAVIAN *et al.*, 2014; YOO *et al.*, 2015; SONG *et al.*, 2016; SCABINI *et al.*, 2019). The first methodology usually leads to better performance results than the second (Kornblith; Shlens; Le, 2019), but it also requires much longer training times and more careful experimental planning to find the right hyperparameters (LI *et al.*, 2020). Additional improvements in predictive power were obtained through TL strategies that integrated the above methodologies into a unified training process (Guo *et al.*, 2019) or through the creation of modified models from several pre-trained

CNN models (Rusu *et al.*, 2016; LIN; ROYCHOWDHURY; MAJI, 2018).

Since most of the above TL strategies use only the last layer of the truncated model to generate feature representations for the new components (see Figure 1b), a critical issue that any TL strategy must address is where to truncate the pre-trained CNN model. In this regard, the layer-by-layer experiments conducted by Yosinski *et al.* (2014) (fine-tune methodology) and Cimpoi *et al.* (2016) (static feature extractor methodology) revealed that truncating the pre-trained model at its last convolutional layer often results in a successful modified model. On the other hand, Mormont, Geurts and Marée (2018) showed that earlier layers end up being better options for target domains that differ significantly from the source (e.g., digital histology vs. photographs of everyday objects). Some research works also studied modified models that incorporate multiple layers of the truncated model to increase the diversity of feature representations delivered to the new components (see Figure 1c and Figure 1d). However, while these multi-layer TL strategies achieved excellent results in some cases (SONG *et al.*, 2016; RAKHLIN *et al.*, 2018), they performed poorly in others (Mormont; Geurts; Marée, 2018). These conflicting results reveal a lack of understanding of how and when to use multi-layer feature representations.

Regarding TL strategies applied to object detection, they also construct modified models that resemble those shown in Figure 1. In this regard, most authors refer to the truncated part of these modified models as the **backbone** (CHEN *et al.*, 2019; LIU *et al.*, 2020). However, unlike image classification, TL strategies applied to object detection require backbones to compute feature representations that, in addition to being semantically strong, have enough spatial resolution to adequately detect small, medium, and large objects. Therefore, many TL strategies improved significantly as their backbone networks shifted from generating single-scale feature representations (see Figure 1b) (Girshick *et al.*, 2014; Girshick, 2015; REN *et al.*, 2017; REDMON; FARHADI, 2017) to computing multi-scale feature representations (see Figure 1d) (LIU *et al.*, 2016; LIN *et al.*, 2017; Redmon; Farhadi, 2018; CAI; VASCONCELOS, 2018; ZHAO *et al.*, 2019; Tan; Pang; Le, 2020). Indeed, the choice of layers in modern TL strategies generally follows a pyramidal pattern that is easy to implement across backbone networks that share a common base architecture (TONG; WU; ZHOU, 2020).

Given the popularity of object detection datasets like Pascal-VOC (EVERINGHAM *et al.*, 2010) and MS-COCO (Lin *et al.*, 2014), much of the research has focused on developing TL strategies that achieve high detection rates in the domain of everyday objects. In contrast, when it comes to detection tasks in other domains, significant innovations are less frequent. For example, in two challenging biological tasks: stomata detection and pollen detection, many researchers choose between blindly using famous TL strategies from the domain of everyday objects (FUENTES *et al.*, 2017; FUENTES *et al.*, 2018; BHUGRA *et al.*, 2019; SAKODA *et al.*, 2019) or implementing novel TL strategies that disregard previous advances in other domains (AONO *et al.*, 2019; FETTER *et al.*, 2019). While some of the above TL strategies have acceptable performances, we believe that greater computational efficiency and higher detection rates are achievable by developing novel TL strategies that combine the ideas from popular TL strategies with those resulting from an in-depth analysis of the detection task.

1.2 Objectives

The lack of prior work to fully understand the benefits and cons of multi-layer feature representations in different target domains motivates the present project. We hypothesize that pre-trained CNN models can lead to higher transfer learning results in classification and detection tasks by identifying the features of one or more layers that are most appropriate for the target domain at stake.

Indeed, this project relies on the work of Cui *et al.* (2018), which found that domain similarity plays a fundamental role at determining how good the transfer learning results will be. However, instead of focusing on pretraining CNN models on a source domain that is similar to the target domain, this project focuses on developing transfer learning methods that maximize the transferability power of an already pre-trained CNN model for classification and detection tasks. In more detail, this project has the following objectives.

- Analysis of the predictive power of multiple layers in classification and detection tasks.
- Investigation of the relationship between transfer learning capacity of pre-trained CNN models in different target tasks.
- Development of new methods that can adaptively select the best activation maps from a pre-trained CNN model for a target domain.

1.3 Contributions

This thesis has many contributions. The following three are the main ones.

- A systematic performance comparison of various pre-trained CNN models in classification and detection tasks.
- New methods that can perform an efficient multi-layer feature extraction from a pre-trained CNN model.
- Very fast single-stage object detectors applied to two challenging biological tasks: grain pollen detection and stomata localization.
- Almost 80000 manual annotations of bounding boxes for stomata localization tasks.

1.4 Text organization

The thesis is organized into two parts. The first part contains the theoretical background necessary to understand the second part, which, in turn, includes the description, systematic analysis, results, and discussion of proposed methods.

Consequently, the first part is divided into the following three chapters. Chapter 2 introduces the concepts of deep learning (DL) and convolutional neural networks (CNN), including the notion of base architecture, optimization algorithms, and different layer types. This chapter primarily focused on image classification and object detection tasks. In Chapter 3, we discuss more advanced concepts related to transfer learning and, also present the analysis and comparison of different CNN families (e.g., ResNet, EfficientNet, DenseNet). The description of all datasets that will be the subject of this thesis is presented in Chapter 4, along with the evaluation metrics and the general evaluation scheme that will be applied to all classification and detection tasks.

As for the second part, it is also divided into three chapters. Chapter 5 performs a systematic comparison between CNN-based TL methods and hand-engineered methods. Based on the results obtained in the previous chapter, Chapter 6 proposes viewing CNN models as collections of deep composite functions and presents TL strategies that extract feature representations from multiple layers. Chapter 7 presents TL strategies applied to two very challenging tasks: (i) pollen grain detection and (ii) stomata localization. Finally, the conclusions are shown in Chapter 8.

Part I

Theoretical Background

CHAPTER

2

DEEP LEARNING IN COMPUTER VISION

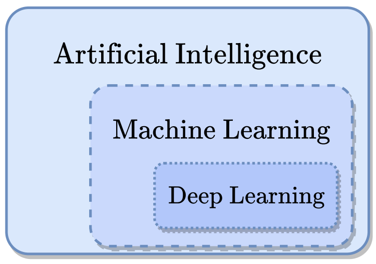
2.1 Initial considerations

Deep learning (DL) is a highly researched class of machine learning techniques (see Figure 2a) that excels at learning consecutive levels of increasingly meaningful data representations (CHOLLET, 2017a, Ch. 1). In this context, the term *representation* can be understood as a different way of looking at the data, where each piece of information in the representation is called a *feature* (GOODFELLOW; BENGIO; COURVILLE, 2016). The ability of DL systems to learn robust representations makes them suitable for many different tasks, including object detection, speech recognition, language understanding, image classification and video captioning. Therefore, most of the research in DL is focused on finding architectural designs and algorithmic improvements that allow for efficient learning and storage of data representations.

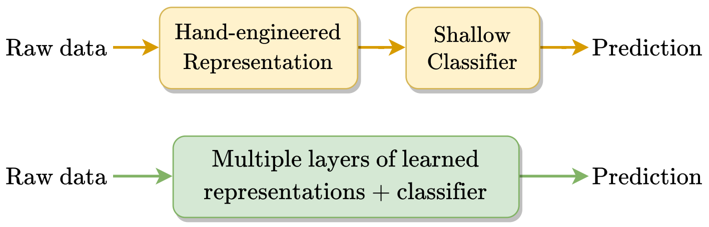
Most DL models learn representations via Deep Neural Networks (DNNs), which offer great flexibility and performance on numerous supervised learning tasks (e.g., speech recognition and object detection) (LECUN; BENGIO; HINTON, 2015). As shown in Figure 2b, the main difference between a traditional ML system and a DNN model is in the approach they use to create their representations. In the former case, human experts propose hand-engineered representations (feature engineering). In contrast, in the second case, the DNN model learns layered representations (feature learning) via an end-to-end learning process from a considerable amount of training data (O'MAHONY *et al.*, 2020).

There are numerous DNN types, such as stacked autoencoders (Le, 2013; ZHOU; PAFFENROTH, 2017), deep belief networks (Liu *et al.*, 2014), generative adversarial networks (CRESWELL *et al.*, 2018), and deep Boltzmann machines (SRIVASTAVA; SALAKHUTDINOV, 2014). We refer the reader to (GOODFELLOW; BENGIO; COURVILLE, 2016) for details on the base architectures, main properties, and application areas of each of the above DNN types.

Figure 2 – Introduction to deep learning. (a) Deep Learning (DL) is a specific class of Machine Learning (ML) techniques. (b) Traditional ML (top diagram) frequently needs the help of human experts to create relevant representations, while DNN (bottom diagram) learns increasingly meaningful layered representations from the training data.



(a) Difference between Artificial Intelligence, Machine Learning, and Deep learning.


Raw data
Hand-engineered Representation
Shallow Classifier
Prediction
Raw data
Multiple layers of learned representations + classifier
Prediction

(b) Traditional ML (top) vs. DNN (bottom).

Source: Elaborated by the author.

Computer vision is one of the main areas where DL has been particularly successful. In this sense, despite recent advances in transformer-based DL models that achieved competitive results on some visual recognition tasks (Carion *et al.*, 2020; DOSOVITSKIY *et al.*, 2021), the Convolutional Neural Network (CNN) is still the most widely used DNN type in computer vision (VOULODIMOS *et al.*, 2018; DHILLON; VERMA, 2020; TONG; WU; ZHOU, 2020). Therefore, this chapter will mainly cover CNN related topics.

In more detail, the remainder of this chapter is structured as follows. We begin by presenting an historical perspective of DL. We then list the main visual recognition tasks and explain some fundamental CNN concepts. Next, we detail the training and inference processes that all CNN models follow to solve the given visual recognition tasks. Finally, we will present the most commonly used CNN layer types.

2.2 Historical perspective of Deep Learning and CNN

Over the years, the history of deep learning (DL) has been characterized by numerous breakthroughs and setbacks. It started in the 1940s when McCulloch and Pitts (1943) presented the first mathematical model of neurons. Later, Rosenblatt (1957) proposed *Perceptron* as the first Artificial Neural Network (ANN) with a functional learning algorithm. After the initial enthusiasm, the scientific community found intrinsic limitations in *Perceptron*, the main one being its inability to solve non-linear separable classification tasks (MINSKY; PAPERT, 1969). This limitation caused the first severe popularity decline of neural networks.

Meanwhile, Hubel and Wiesel (1959) made a relevant neuroscientific contribution by discovering particular types of neurons in the primary visual cortex, called *simple cells* and *complex cells*. In this regard, while *simple cells* can recognize edges and bars of particular orientations at specific locations, *complex cells* can collect the information provided by multiple simple cells to recognize edges and bars anywhere in the scene. The above idea of creating complex detectors out of simple ones led to the appearance of several ANN types, such as the Neocognitron (FUKUSHIMA; MIYAKE, 1982), which is the predecessor of modern CNN models.

In the early 1980s, ANN models consisting of multiple layers were known to solve non-linear separable problems. However, there was no learning algorithm that could train them efficiently. The situation substantially improved when Rumelhart, Hinton and Williams (1986) proposed a successful learning algorithm using the *back-propagation* method. This publication triggered a second wave of popularity for ANN models, leading to the emergence of famous ANN types, such as the debut of the Long Short-Term Memory network (LSTM) (HOCHREITER; SCHMIDHUBER, 1997) and the Convolutional Neural Network (CNN) (LECUN *et al.*, 1989).

However, the mid-1990s saw a further decline in the popularity of ANNs. According to Goodfellow, Bengio and Courville (2016), this decrease was mainly due to the following two factors. (i) There were irrational expectations that the ANN models of the time failed to meet. (ii) Alternative machine learning tools appeared, such as Support Vector Machines (BOSER; GUYON; VAPNIK, 1992), achieving reliable performance with much less training effort. Regardless, the fraction of the scientific community that remained was enough to publish ANN models that obtained impressive results on challenging tasks like handwritten digit recognition (LECUN *et al.*, 1998).

In the mid-2000s, the publication of a multi-layer ANN model with a low training cost (HINTON; OSINDERO; TEH, 2006) sparked the third wave of popularity. Indeed, as more research works were published, ANN models with at least one hidden layer became known as Deep Neural Networks (DNN) or Deep Learning (DL). Furthermore, two more factors were crucial for the increase in the number of research articles on DL. (i) The release of increasingly powerful Graphical Processing Units (GPU), which made the training of DNN models feasible. (ii) The publication of large datasets that allowed DNN models to learn meaningful representations. The combination of the above factors led to the work of Krizhevsky, Sutskever and Hinton (2012), who developed a CNN model with eight layers that far surpassed any other method of the time.

Since then, computer vision was one of the main fields where DL received the most attention from the scientific community. Consequently, deeper CNN models were proposed over the years, such as VGG (SIMONYAN; ZISSERMAN, 2015), INCEPTION (SZEGEDY *et al.*, 2015; SZEGEDY *et al.*, 2017), RESNET (HE *et al.*, 2016), RESNEXT (Xie *et al.*, 2017), and EFFICIENTNET (TAN; LE, 2019) leading to a breakthrough in computer vision tasks, such image classification, object detection, semantic segmentation and video understanding.

2.3 Computer vision and visual recognition tasks

Computer vision is a field of computer science that seeks to develop techniques that allow computers to understand the content of digital images or videos. According to Szeliski (2011), computer vision follows the inverse problem of computer graphics. In this sense, while computer graphics tries to model how to project visual content onto an image, computer vision attempts to describe the content of a given image by extracting relevant properties such as shape, texture, and color.

Among the many computer vision tasks, visual recognition tasks have gained much attention in the last decade due to the following factors. (i) The appearance of a significant number of robust annotated datasets (Lin *et al.*, 2014; RUSSAKOVSKY *et al.*, 2015; TRIANTAFILLOU *et al.*, 2020). (ii) The arrival of powerful Graphics Processing Units (GPUs) and Tensor Processing Units (TPUs) that enable efficient training of deep CNN models (JOUPPI *et al.*, 2017; TANG *et al.*, 2019). Below are short descriptions of well-researched visual recognition tasks.

- **Instance recognition.** Given two images (I_1 and I_2), the task is to determine whether the object appearing in I_1 also appears in I_2. Potential challenges occur when the object in question is viewed from a different perspective, with partial occlusions, or is on a cluttered background.

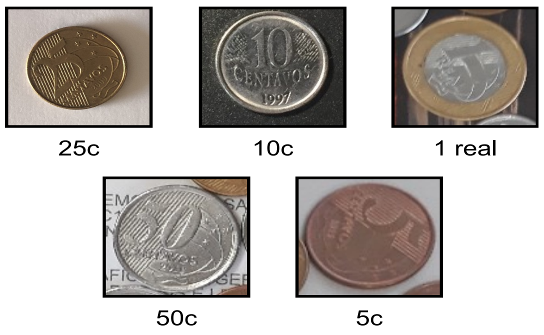
- **Image classification.** Given an image I, the task is to assign I to one of the predefined classes based on its visual features. This task is much more challenging than the previous one because it requires recognizing any valid instance of each predefined class. Figure 3a shows an example of a coin classification task, where there are five classes, namely, 5c, 10c, 25c, 50c, and 1 real.

- **Object detection.** Given an image I, the task is to **locate** and **classify** all objects in I that are valid instances of at least one predefined foreground class. As illustrated in Figure 3b, the localization sub-task involves generating bounding boxes, each denoting the presence of an object. On the other hand, the classification sub-task entails assigning a class label to each bounding box. In this sense, a detection task can also be referred to as an object localization task if there is only one foreground class. Famous object detection examples are: (i) face recognition, (ii) pedestrian recognition, and (iii) generic object detection.

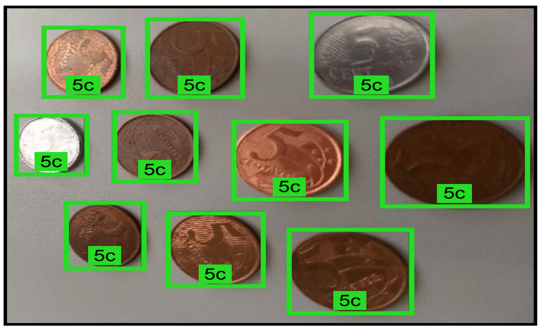
- **Image segmentation.** Given an image I, the task is to classify and delineate the boundaries of all objects in I that are valid instances of at least one predefined class. Popular examples are: (i) per-pixel segmentation, (ii) instance segmentation, and (iii) pose estimation.

- **Video understanding.** The task is to track all the objects in a video that belong to at least one predefined class. For example, monitoring human activity.

Figure 3 – **Two visual recognition tasks.** There is only one object per image in classification tasks, and there are multiple objects per image in detection tasks.



(a) Coin classification (five images)



(b) Coin detection (one image)

Source: Elaborated by the author.

2.4 Convolutional Neural Networks: Basic concepts

Convolutional neural networks (CNN) are a type of DNN initially proposed for the recognition of handwritten digits (LECUN *et al.*, 1989). They work well on several types of media, e.g., audio, image, and video (HOU; CHEN; SHAH, 2017; XU *et al.*, 2018). However, due to the characteristics and objectives of this project, we limited the scope of the following definitions to image classification and object detection tasks.

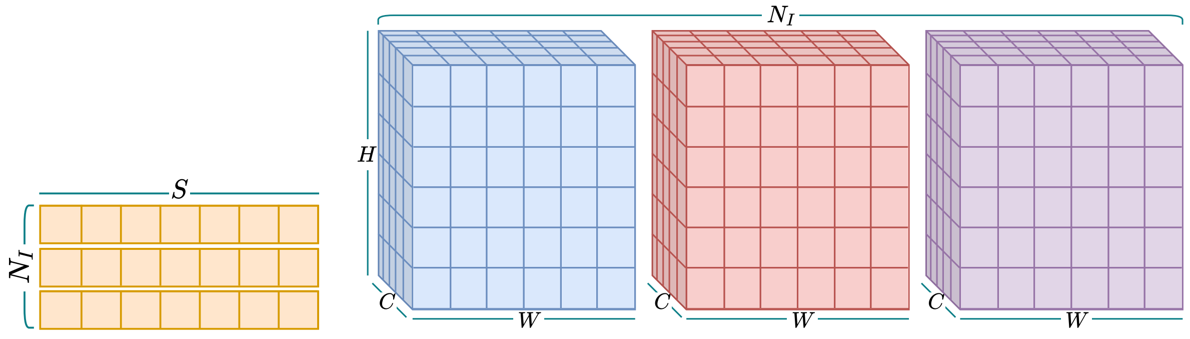
2.4.1 Tensors

Before describing the CNN architecture and other related concepts, we introduce the notion of **tensor**. In machine learning, tensors are data containers used to store and manipulate numerical data of an arbitrary number of spatial dimensions (d) (see Figure 4). Additionally, tensors obey certain transformation rules (e.g., dot product, and cross-product), making them suitable for use in deep learning. Therefore, as shown in Figure 5, the inputs, intermediate results, and predictions made by the CNN are all stored in tensors.

Tensors that only store one number are known as **scalars** or 0D tensors ($d = 0$). When the number of dimensions is at least one ($d \geq 1$), the tensor can be represented as a d-way data array. Therefore, tensors can be thought as the high-dimensional generalization of array-based data structures, like **vectors** (1D tensors) and **matrices** (2D tensors) (ZHANG *et al.*, 2017). In this regard, each numerical element in the tensor is called a **scalar component**, and it can be accessed by specifying d indices. For example, given a 3D tensor $\mathbf{T}$ of shape $(n_1 \times n_2 \times n_3)$, each scalar component is denoted $\mathbf{T}[i, j, k]$, where the indices i, j, and k are integer numbers ($1 \leq i \leq n_1$, $1 \leq j \leq n_2$, and $1 \leq k \leq n_3$). Additionally, each time we slice a tensor along one or more dimensions, we will use the colon operator ':'. For instance, $\mathbf{T}[:, j, :]$ means "to fully slice the tensor along the first and third dimensions and then to return the 2D tensor of index j".

The dimensionality of a tensor determines what data it can store. In this context, single-channel images require at least 2D tensors, while multi-channel ones require at least 3D tensors (AUDEBERT; SAUX; LEFEVRE, 2019). On the other hand, deep learning frameworks, such as Keras (CHOLLET *et al.*, 2015) and PyTorch (PASZKE *et al.*, 2017), group various multi-channel images into 4D tensors of shape $(N_I \times C \times W \times H)$, where the N_I represents the number of images, and the other three denote the number of channels, the width, and the height of the input images, respectively. Figure 4b illustrates an example of such a tensor.

Figure 4 – **Examples of 2D and 4D tensors.** The feature extraction part of most CNN models only works with 4D tensors of shape $(N_I \times C \times W \times H)$, while the classification part mainly works with 2D tensors of shape $(N_I \times S)$. S: number of features, N_I: number of images, C: number of channels, W: width, H: height.



(a) 3×7 tensor

(b) $3 \times 5 \times 6 \times 6$ tensor

Source: Elaborated by the author.

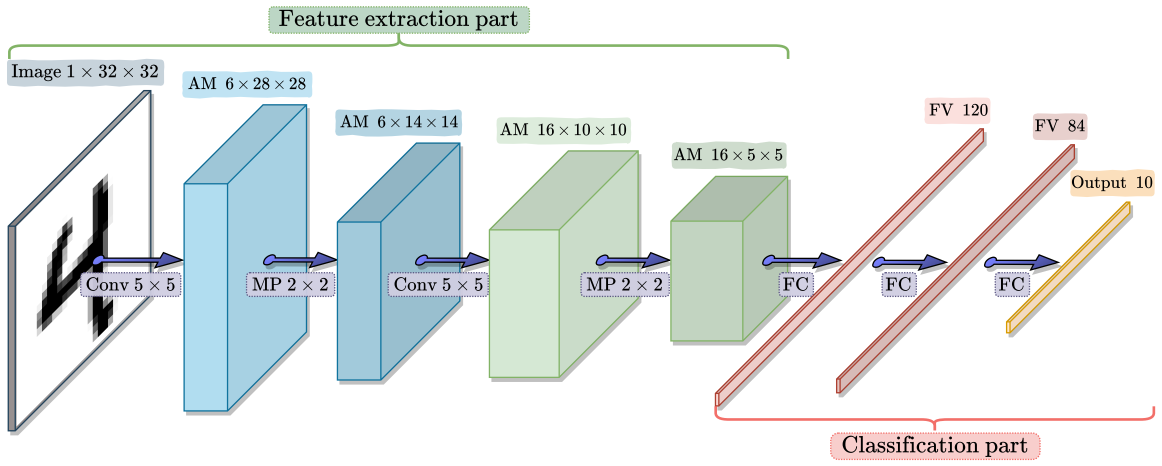
2.4.2 Parts of the CNN model

In classification tasks, the basic architecture of a CNN consists of several stages of convolutional and pooling layers followed by one or more fully connected layers (LECUN; BENGIO; HINTON, 2015). For instance, Figure 5 shows the architecture of LeNet-5, a CNN model with two convolutional layers, two pooling layers, and three fully connected layers (LECUN *et al.*, 1998). Over time, CNN models with much more complex architectures were developed (Zhang *et al.*, 2017; Tan *et al.*, 2019). However, all CNN models have two well-defined parts despite their differences: the **feature extraction part** and **the classification part**. The first part corresponds to the portion of the CNN model where all convolutional layers reside, and the second part consists of the remaining fully connected layers.

In object detection, the feature extraction part of the CNN model is called the **backbone**. Additionally, CNN models have a **regression head**, which generates bounding boxes for detected objects, and a **classification head**, which assigns class labels to the predicted bounding boxes (REN *et al.*, 2017). Of the three, the backbone is typically the longest and most important part of the CNN model from which the other parts make predictions. Therefore, backbones that perform well for image classification tasks are often reused for object detection tasks (TONG;

WU; ZHOU, 2020). Additionally, unlike the image classification case, the regression head and the classification head can include convolutional layers, fully connected layers, or both.

Figure 5 – **LENET-5 architecture**. Two convolutional layers (Conv), two pooling layers (MP), and three fully connected layers (FC). For better visualization, we display 3D tensors of shape $(C \times W \times H)$ instead of 4D tensors of shape $(N_I \times C \times W \times H)$. AM: Activation Map, FV: Feature Vector.


Feature extraction part
Image $1 \times 32 \times 32$
AM $6 \times 28 \times 28$
AM $6 \times 14 \times 14$
AM $16 \times 10 \times 10$
AM $16 \times 5 \times 5$
FV 120
FV 84
Output 10
Conv 5×5
MP 2×2
Conv 5×5
MP 2×2
FC
FC
FC
Classification part

Source: Elaborated by the author.

2.4.3 Activations maps and feature vectors

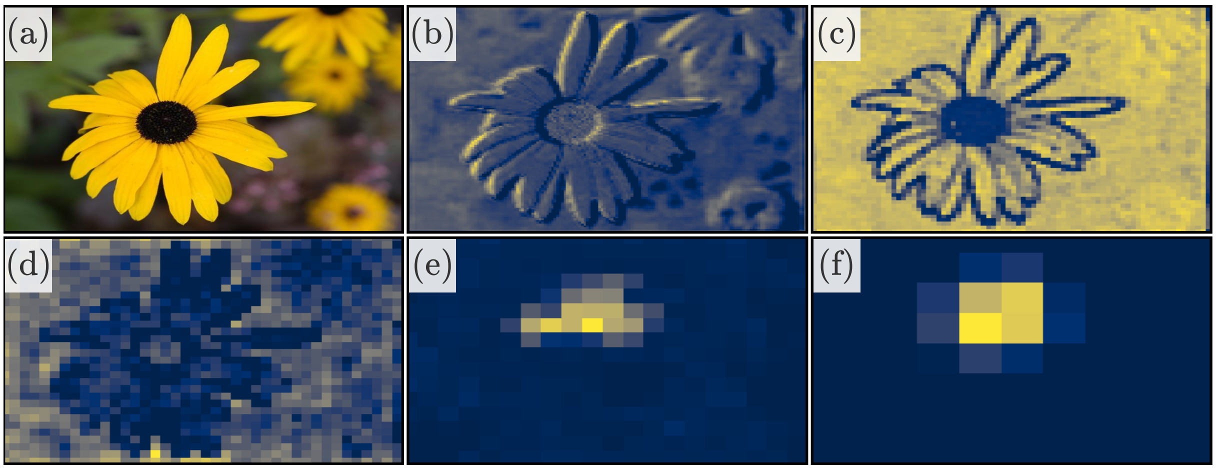
As stated in subsection 2.4.1, each batch of N_I input images given to the CNN model comes in the form of a 4D tensor. Then, for each batch, the feature extraction part of the CNN model returns a 4D tensor containing N_I sets of activation maps. Indeed, the returned tensor maintains the same format as that shown in Figure 4b, namely the first dimension represents the number of images (N_I), and the other three dimensions denote the number of channels (C), the width (W), and the height (H) of the resulting sets of activation maps (CHOLLET, 2017a).

In more detail, let us consider $\mathbf{A}$ as the tensor of size $(N_I \times C \times W \times H)$ computed by the CNN model. In this context, $\mathbf{A}[i,:,:,:]$ (or simply $\mathbf{A}[i]$) represents the set of activation maps associated with the i-th input image $(1 \leq i \leq N_I)$. Likewise, $\mathbf{A}[i,c,:,:]$ (or $\mathbf{A}[i,c]$) represents the c-th activation map of size $(W \times H)$ associated with the i-th input image $(1 \leq c \leq C)$.

Figure 6 shows the activation maps computed by different layers within a CNN model that classify flower types. While some of these activation maps only reveal generic features, others can contain high-level patterns that are meaningful for the visual recognition task at stake (ZEILER; FERGUS, 2014; Yosinski *et al.*, 2015). For example, Figure 6c shows the edges of the flower (low-level feature), whereas Figure 6f seems to indicate the existence of the flower itself (high-level feature).

In image classification tasks, the classification part of the CNN model takes 2D tensors of shape $(N_I \times S)$ as input, where N_I is the number of images and S is the number of features (see

Figure 6 – **Exemplary activation maps.** (a) Input image taken from the `Flowers` dataset (Nilsback; Zisserman, 2008), (b–f) activation maps computed by five different layers within a trained CNN model. We only display one activation map per layer. The layers are ordered by their depth, from shallowest to deepest.



Source: Elaborated by the author.

Figure 4a). In this regard, given a $(N_I \times S)$ tensor $\mathbf{X}$, we define the **feature vector** $\mathbf{X}[i,:]$ (or $\mathbf{X}[i]$) as the 1D tensor of shape (S) associated with the i-th input image ($1 \leq i \leq N_I$). Ideally, the elements in a feature vector represent information that is relevant to the visual recognition task at stake. However, unlike activation maps, there is not spatial relationship between consecutive elements in the feature vector.

2.4.4 Layers, blocks, and composite functions

As shown in subsection 2.4.2, every CNN model has distinct parts that work together to accomplish the visual recognition task at stake. Each part consists of multiple interconnected layers. In this context, a **layer** represents a function $h_\ell(\cdot)$ that takes one or more tensors as input, passes them through a mathematical transformation, and returns an output tensor. The suffix ℓ indicates the layer index. Various layer types exist; some have trainable parameters (or weights) that must be optimized in a training process, while others have only fixed parameters or none at all. We detail the most relevant layer types in section 2.6.

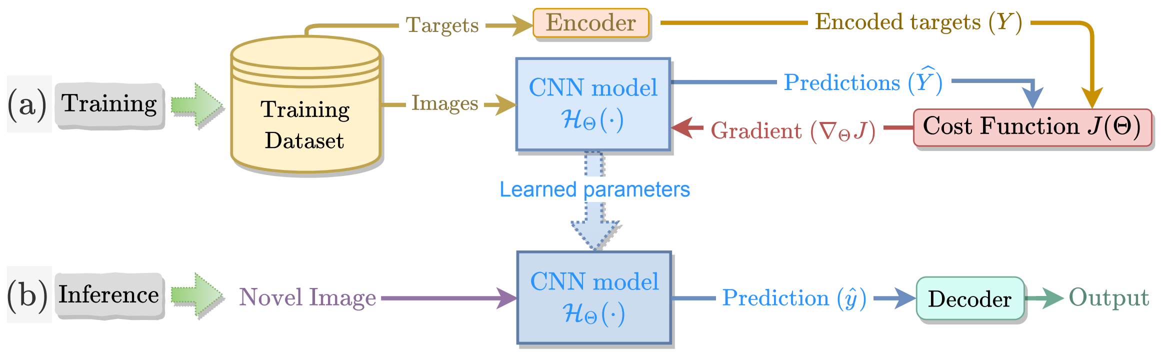
Furthermore, a **block** comprises at least two interconnected layers that work together to fulfill a given role. Hence, a **block** can be viewed as a composite function $H_b(\cdot)$, where b indicates its index. Over the years, many proposed blocks gained fame because their ingenious structures led to the emergence of increasingly powerful CNN models. Examples of such blocks are: the inception block (SZEGEDY *et al.*, 2015), the "bottleneck" building block (HE *et al.*, 2016), the dense block (HUANG *et al.*, 2017), and the squeeze-excitation block (Hu *et al.*, 2019).

In the broadest sense of the term, one could consider the parts of the CNN model and even the CNN model itself as blocks. Since the above interpretation could lead to confusing situations, we limit the scope of the term block to refer to a set of continuous layers within the CNN model's feature extraction part. Additionally, we use the block index b to differentiate the output tensors generated by distinct blocks. For example, $\mathbf{A}_b$ represents the 4D tensor that originated from the b-th block of the CNN model.

2.5 Training and inference

CNN-based solutions for visual recognition tasks involve two essential processes: training and inference. As shown in Figure 7a, training is the process of teaching the CNN model how to solve a given task using a series of examples (images and targets). In contrast, Figure 7b illustrates the inference process, which involves feeding the trained CNN model with a novel image. The model then returns a prediction based on the content of the image and the knowledge acquired during training. These two processes have many internal details, which are covered in the following subsections.

Figure 7 – Training and inference.



Source: Elaborated by the author.

2.5.1 Before training

There are at least two main aspects that must be covered before starting the training process: the design of the CNN model and its initialization strategy.

Designing the CNN model

The architecture of a CNN plays a crucial role in determining its performance in terms of predictive power, time and memory efficiency (HANIN; ROLNICK, 2018; TAN; LE, 2019). Consequently, many research articles present their own CNN models using either manual or automated approaches. Below, we briefly describe some of the manual approaches.

- The most basic manual approach is for a human expert to design the architecture through various trial and error rounds with different combinations of layers (SZEGEDY *et al.*, 2015; Zhang *et al.*, 2017).

- Since the above approach is time and resource-consuming, some works focus on testing small but effective changes to the architecture of an existing CNN model (HE *et al.*, 2019).

- An alternative approach is to combine the architectures and core ideas of two or more existing CNN models to generate a new one with greater predictive power (SZEGEDY *et al.*, 2017; Bello *et al.*, 2021).

- Another successful approach involves designing a block of layers that serves a specific purpose. Then, copies of the proposed block are used to replace others blocks within an existing CNN model or are inserted at different locations throughout the model. Examples of popular blocks are: "bottleneck" building blocks (HE *et al.*, 2016), depth-wise separable convolutions (CHOLLET, 2017b), dense blocks (HUANG *et al.*, 2017), and squeeze-excitation blocks (Hu *et al.*, 2019).

As for automated approaches, the idea is to discover increasingly solid architectures by testing different combinations of layers and blocks in a predefined search space. However, instead of a brute-force or a random search approach, the most relevant automated approaches use sophisticated search algorithms, such as those based on reinforcement learning (ZOPH *et al.*, 2018; Tan *et al.*, 2019), evolution (REAL *et al.*, 2019), and gradient-based optimization (LIU; SIMONYAN; YANG, 2019; Wu *et al.*, 2019).

Parameter initialization

After designing the CNN model, the next step is to initialize its set of trainable parameters Θ. This step is very challenging because inadequate parameter initialization leads to both longer training processes and poor performance (HANIN; ROLNICK, 2018).

Multiple initialization strategies have been proposed over the years (GLOROT; BORDES; BENGIO, 2011; He *et al.*, 2015; ARPIT; CAMPOS; BENGIO, 2019). Most of them involves randomly sampling initial values from a uniform distribution $\mathcal{U}(-\sqrt{k},\sqrt{k})$ or from a normal distribution $\mathcal{N}(\mu,\sigma)$. In this context, the initialization strategy proposes a method to calculate k, μ, and σ. For instance, the PyTorch framework (PASZKE *et al.*, 2017) by default initializes the parameters of fully connected layers using a uniform distribution with $k = 1/\textit{fan-in}$, where *fan-in* is the number of input units in the layer.

A very different initialization strategy is the **parameter transfer approach**. In this strategy, some of the CNN model parameters are initialized with those of an already-trained CNN model, while the remaining parameters follow the regular random initialization process (Ribani; Marengoni, 2019). The above topic becomes part of a class of methods known as Transfer Learning (TL), which will be covered in Chapter 3.

2.5.2 Training process

In supervised learning, a CNN model describes a **hypothesis** $\mathcal{H}_\Theta(\cdot)$ that maps an input image I to a predicted output $\hat{y}$. More specifically, $\mathcal{H}_\Theta(\cdot)$ represents a composite function with a set of trainable parameters $\Theta = \{\theta_1, \theta_2, \dots, \theta_{N_\Theta}\}$, such that $\hat{y} = \mathcal{H}_\Theta(I)$. In general, the performance of $\mathcal{H}_\Theta(\cdot)$ depends on Θ, which must be learned during the training process.

During training, the predictive power of $\mathcal{H}_\Theta(\cdot)$ is evaluated via the **cost function** $J(\Theta)$, which reflects the discrepancy between the model predictions for the current state of Θ and the targets. Consequently, the goal of training is to minimize $J(\Theta)$ by adjusting the parameters in Θ via an optimization algorithm (or optimizer) that learns from the training dataset. Below, more details about the optimization algorithm.

Basic gradient-based optimization algorithm

Gradient-based optimizers involve multiple forward and backward passes over the training data until $J(\Theta)$ converges to a local or global minimum. As depicted in Figure 7a, the forward pass includes the following steps:

1. Take a random batch of N_I examples from the training dataset, where the i-th example consists of one image and corresponding target, $1 \leq i \leq N_I$.
2. Use $\mathcal{H}_\Theta(\cdot)$ to process the N_I images, and obtain the set of predictions $\widehat{Y} = \{\hat{y}_1, \dots, \hat{y}_{N_I}\}$.
3. Encode the N_I targets into a format that make them compatible with $\widehat{Y}$. The output of this step is a set of N_I encoded targets $Y = \{y_1, \dots, y_{N_I}\}$.
4. Finally, use the predefined cost function $J(\Theta)$ to measure the performance of $\mathcal{H}_\Theta(\cdot)$ by comparing Y and $\widehat{Y}$.

As for the backward pass, it includes the steps below.

1. Use the back-propagation algorithm (RUMELHART; HINTON; WILLIAMS, 1986) to estimate the gradient of $J(\Theta)$ as a vector of N_Θ partial derivatives. We denote the gradient of the cost function by $\nabla_\Theta J(\Theta)$, and the partial derivative with respect to the j-th parameter by $\frac{\partial J(\Theta)}{\partial \theta_j}$; $1 \leq j \leq N_\Theta$.
2. Use an **update rule** to modify the parameters in Θ. For instance, one of the simplest learning rules is to update the parameters in the negative direction of the gradient: $\Theta \leftarrow \Theta - \eta \cdot \nabla_\Theta J(\Theta)$, where η is the learning rate.

Let us define $|D|$ as the number of examples in the training dataset. The above optimization algorithm is called the Gradient Descent (GD) algorithm if every batch of training examples covers the entire dataset ($N_I = |D|$). In general, GD is a suitable choice for training DNNs when $|D|$ is small. However, as $|D|$ grows, GD becomes much more expensive in terms of time and memory resources. Therefore, the typical choice for training large CNN models is the Stochastic Gradient Descent (SGD) algorithm (also known as the Mini-batch Gradient Descent), which is as powerful as GD but uses far fewer training examples to compute $\nabla_\Theta J(\Theta)$; $(1 \leq N_I \ll |D|)$ (GOODFELLOW; BENGIO; COURVILLE, 2016, Sec. 5.9).

Other gradient-based optimization algorithms

More complex optimizers based on GD and SGD have been proposed over the years. For example, Qian (1999) extended the SGD update rule by incorporating a momentum term to calculate the exponential moving average of previous gradients. The goal is to reduce fluctuations in the velocity and direction at which each parameter $\theta_j \in \Theta$ move within the parameter space. Therefore, the momentum term can accelerate convergence in high-curvature regions and prevent premature convergence in low-curvature ones. Additionally, Sutskever *et al.* (2013) proposed the Nesterov momentum, which is a simple variation of the standard momentum term that can obtain faster convergence times in some specific situations.

On the other hand, Duchi, Hazan and Singer (2011) proposed the Adaptive Gradient algorithm (AdaGrad). AdaGrad defines a learning rate η_j for each parameter $\theta_j \in \Theta$. Then, η_j decreases with each iteration in proportion to the accumulation of all past squared gradients calculated for θ_j. Despite its relevant theoretical properties, AdaGrad often terminates the training process prematurely as it brings most learning rates to near zero. In an effort to overcome the problems of AdaGrad, Zeiler (2012) proposed Adadelta. Although Adadelta also accumulates all past squared gradients, it does so using the exponential moving average, which causes the influence of older gradients to decrease exponentially.

The literature offers many more optimization algorithms that work well in certain situations. According to the analysis of Ruder (2016), the most successful SGD variant is the Adaptive Moment Estimation (ADAM) algorithm (KINGMA; BA, 2015). In this regard, ADAM has an update rule that includes first- and second-order moments of the gradient, which are represented in Equation 2.1 by the letters s and r, respectively.

$$s \leftarrow p_1 s + (1-p_1)\nabla_\Theta J(\Theta), \qquad r \leftarrow p_2 r + (1-p_2)\left(\nabla_\Theta J(\Theta)\right)^2, \tag{2.1}$$

Both $p_1 \in [0,1]$ and $p_2 \in [0,1]$ are hyper-parameters with default values of 0.9 and 0.999, respectively. In the first few iterations, these default values cause estimated moments to be biased towards zero. Since such outcomes are detrimental to the training process, ADAM corrects the biased estimates with Equation 2.2 below.

$$\hat{s} = \frac{s}{1-p_1^t}, \qquad \hat{r} = \frac{r}{1-p_2^t}, \tag{2.2}$$

where t is the number of iterations. Note that as t increases, the influence of p_1 and p_2 decays exponentially in Equation 2.2. Then, the update rule of ADAM is as follows: $\Theta \leftarrow \Theta - \eta\frac{\hat{s}}{\sqrt{\hat{r}}+\delta}$, where η is the global learning rate and δ is a small constant (e.g., 1e-10) for numerical stability. The experimental part of this project uses either SGD or ADAM.

2.5.3 Other topics relevant to the training process

The training process still has several topics worth describing. Although subsection 2.5.2 presented some of these topics, more details will be provided below.

Target encoding

Each training example consists of an image I and its corresponding target. In image classification tasks, a target is generally a single number that represents a class label, while in object detection tasks, it is a list containing bounding box coordinates and class labels that describe valid objects within I. During training, the dataset provides N_I training examples to the optimization algorithm. Then, as shown in Figure 7a, the targets are encoded into a format that allows the cost function $J(\Theta)$ to compare them with the predictions of the CNN model $\mathcal{H}_\Theta(\cdot)$. For instance, in image classification tasks, researchers convert labels to vectors using one-hot encoding or other more sophisticated target embedding methods (RODRÍGUEZ *et al.*, 2018). As for object detection tasks, researchers often encode the bounding box coordinates by parameterizing them relative to a set of reference boxes (REN *et al.*, 2017).

Back-propagation algorithm

The role of back-propagation is often confused with that of the optimizer. However, back-propagation is an algorithm designed only for the rapid calculation of gradients $\nabla_\Theta J(\Theta)$ (RUMELHART; HINTON; WILLIAMS, 1986). Internally, the back-propagation algorithm visits each layer of the CNN model in reverse order, from deepest to shallowest. Then, in each layer, it propagates the gradient of the previous layers using the chain rule, resulting in a set of partial derivatives of $J(\Theta)$ with respect to each parameter from the layer. In general, back-propagation works well on tensors of arbitrary dimensionality, making it portable to any DNN architecture. We refer the reader to (GOODFELLOW; BENGIO; COURVILLE, 2016, Ch. 6.5) for more information on this topic.

Loss function vs. Cost function

Although both terms are often used interchangeably, they differ in the following aspect: The **loss function** estimates how closely the predicted output matches the target in a single training example. In contrast, the cost function $J(\Theta)$, which is also known as the **objective function**, computes the average loss across a batch of training examples:

$$J(\Theta) = \frac{1}{N_I}\sum_{i=1}^{N_I} L(\mathcal{H}_\Theta(I_i), y_i) \tag{2.3}$$

where (I_i, y_i) indicates the i-th training example, $L(\cdot)$ is the loss function, and $\mathcal{H}_\Theta(\cdot)$ is the CNN model. Below is a list of the most popular loss functions used for object detection and image classification tasks.

- **Cross entropy loss (CE).** This loss function is a popular choice for classification tasks. It quantifies the difference between two probability distributions:

$$\mathrm{CE}(\hat{y}, y) = -\sum_{c=1}^{N_C} y[c] \log(\hat{y}[c]) \tag{2.4}$$

  Within the context of image classification tasks, N_C is the number of classes, and c is the class label. Furthermore, $\hat{y}$ is a vector of N_C predicted probabilities, where $\hat{y}[c]$ represents the confidence that the model has to classify the input image in the c-th class. On the other hand, y is the encoded target, represented as a binary vector, in which only a single cell contains the value 1, and the rest of the cells only include zeros. Therefore, both predicted and target vectors satisfy the following conditions: $\sum_c^{N_C} y[c] = 1$; $\sum_c^{N_C} \hat{y}[c] = 1$. Let us define $\tilde{c}$ as the class label where $y[\tilde{c}] = 1$, then Equation 2.4 can be simplified as follows:

$$\mathrm{CE}(\hat{y}, y) = -\log(\hat{y}[\tilde{c}]) \tag{2.5}$$

  As shown in Figure 8a, CE provides a higher loss value as the difference between $\hat{y}[\tilde{c}]$ and $y[\tilde{c}]$ increases. In general, CE is the default choice for image classification and object detection tasks.

- **Focal loss (FL).** It was proposed by Lin *et al.* (2017) as an improved version of the CE loss. FL modifies Equation 2.5 as follows:

$$\mathrm{FL}(\hat{y}, y) = -(1 - \hat{y}[\tilde{c}])^{\lambda} \log(\hat{y}[\tilde{c}]) \tag{2.6}$$

  where $\lambda \in [0, \infty)$ is a hyper-parameter that governs how much the relative loss of well-classified examples will be reduced. When $\lambda = 0$, the FL curve is the same as the CE curve. Figure 8a shows different FL curves for distinct values of λ.

- **Square error loss (SE).** Both CE and FL are loss functions for classification tasks. In the case of regression tasks, researchers often choose the SE loss, which has a very simple equation:

$$\mathrm{SE}(\hat{y}, y) = (\hat{y} - y)^2 \tag{2.7}$$

  In this case, $\hat{y}$ and y are scalars. Additionally, note that the quadratic formula amplifies large differences and reduces small ones (see Figure 8b).

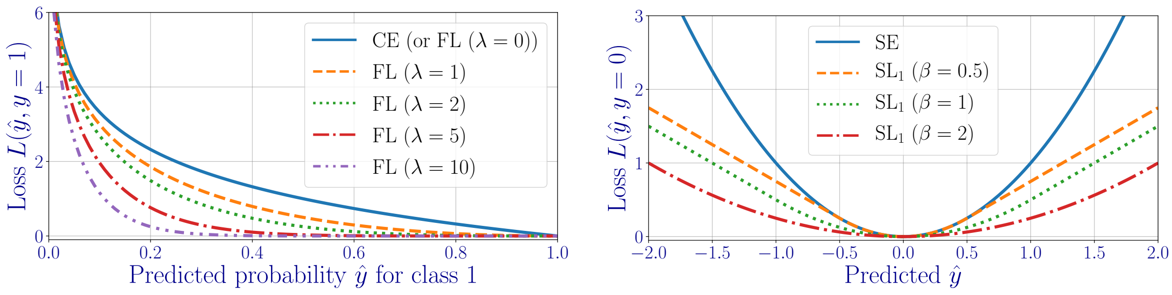
- **Smooth L_1 loss (SL_1).** It is also known as the Huber loss. This loss function is less sensitive to outliers than the SE loss, which in some cases prevents the exploding gradients problem (Girshick, 2015). Its equation is given by:

$$\mathrm{SL}_1(\hat{y}, y) = \begin{cases} \frac{(\hat{y}-y)^2}{2\beta}, & \text{if } |\hat{y} - y| < \beta, \\ |\hat{y} - y| - \frac{\beta}{2}, & \text{otherwise}, \end{cases} \tag{2.8}$$

where β is a hyper-parameter, whose default value is 1, $\hat{y}$ is the predicted scalar, and y is the target scalar. Note that Equation 2.8 has two parts, one of them is almost the same as Equation 2.7, and the other is linear. See Figure 8b for a comparison between the SE loss curve and three SL_1 loss curves (each with a different β). In general, object detectors use the SL_1 loss for training the bounding box regression subnet of the CNN model (LIN *et al.*, 2017; Tan; Pang; Le, 2020).

Additionally, $J(\Theta)$ can include two or more loss functions. For example, in object detection tasks, $J(\Theta)$ generally combines two loss functions: one for evaluating the correctness of the predicted bounding box coordinates, and another for measuring the accuracy of the predicted bounding box classes (LIN *et al.*, 2017; Tan; Pang; Le, 2020). In certain situations, researchers extends Equation 2.3 by adding a penalty norm term $\alpha\Omega(\Theta)$, where $\alpha \in [0, \infty)$ is a hyper-parameter that controls the relative contribution of the penalty term to $J(\Theta)$, and $\Omega(\Theta)$ is the penalty function that limits the growth capacity of all the CNN model parameters, so that the trained model can avoid overfitting the training dataset.

Figure 8 – **Comparison of loss curves for classification and regression tasks.**



(a) For classification tasks (b) For regression tasks

Source: Elaborated by the author.

Overfitting and regularization strategies

Overfitting is a recurring problem that occurs when $\mathcal{H}_\Theta(\cdot)$ achieves excellent results in the training data, but it is unable to predict the correct output for novel images. Multiple strategies exist to reduce overfitting. Below are brief descriptions of the most popular strategies.

- **Adding a penalty term $\alpha\Omega(\Theta)$ to $J(\Theta)$.** It forces the model parameters to take only small values, thereby making its distribution more regular. In this sense, when the additional cost of $\Omega(\Theta)$ is proportional to the absolute value of the parameters in Θ, it is called

L1 regularization. On the other hand, when the additional cost is proportional to the square of the value of the parameters in Θ, it is called **L2 regularization** or **weight decay** (CHOLLET, 2017a, Ch 4). The main drawback of this strategy is that it can cause an *underfitting* problem, which occurs when CNN model stops "learning" from the training data. Therefore, the value of $\alpha \in [0, \infty]$ plays a crucial role, since it governs how much regularization is added to $J(\Theta)$.

- **Adding one or more dropout layers.** Proposed by Srivastava *et al.* (2014). During training, the dropout layer randomly zeroes some of the previous layer's output feature elements. Then, the new set of features elements becomes the input for the next layer. These feature elements can be as small as single values from a feature vector or as big as entire activation maps. There is also a hyper-parameter called the **dropout rate**, which determines how many elements will be zeroed. Consequently, the dropout layer encourages $\mathcal{H}_{\Theta}(\cdot)$ to generalize better by learning more independent features.

- **Data Augmentation.** Its main objective is to increase the amount of training data by applying some mathematical transformations on the training images. These transformations then generate a greater variety of images with different scales, aspect ratios, perspectives, orientations, brightness, and with more complex properties (MIKOŁAJCZYK; GROCHOWSKI, 2018). Indeed, it is currently a highly researched topic because, when used correctly, it can provide significant improvements in CNN model performance, especially in domains with small to medium-sized training data available. Therefore, in most computer vision competitions, the candidate solutions generally use a large number of data augmentation techniques to increase their chances of winning (Ha; Liu; Liu, 2020). We refer the reader to (ZOPH *et al.*, 2020a) for a great analysis and categorization of data augmentations techniques applied to object detection tasks.

- **Early Stopping.** Here, the training data is typically divided into two subsets, one for training and one for validation purposes. Then, in each iteration of the optimization algorithm, two cost function results are computed, one from the training subset and the other from the validation subset. Note that the training cost is the only one that provides gradients to the optimization algorithm. In this regard, the early stopping aims to halt the optimization algorithm when the validation cost reaches a minimum value. Since there is no deterministic way to know when the validation cost is minimal, the early stopping strategy defines a hyper-parameter called **patience** as the maximum number of iterations the validation cost has to reach a new minimum (GOODFELLOW; BENGIO; COURVILLE, 2016, Sec. 7.8). However, due to recent scientific discoveries, such as the phenomenon of **double** and **triple descent**, early stopping should be used with care to avoid sub-optimal results, especially in large deep models (NAKKIRAN *et al.*, 2020; ASCOLI; SAGUN; BIROLI, 2020).

2.5.4 Inference Process

Once the training process is complete, the CNN model is ready to handle new images through a mechanism known as the **inference process**. In this sense, Figure 7b presents the basic steps that each new image must go through during the inference. A brief description of each step is below.

1. Use the trained CNN model to process the novel image I, and get a prediction $\hat{y} = \mathcal{H}_{\Theta}(I)$. This step is equivalent to the forward pass computed by the optimization algorithm during training.
2. Then follows a decoding step, consisting of transforming $\hat{y}$ into a human-readable format, such as class labels (in image classification tasks) or bounding box coordinates (in object detection tasks).

In general, the inference process takes less time than its training counterpart as no gradient calculation befalls. Furthermore, the inference process can be included in an evaluation scheme when we want to know whether $\mathcal{H}_{\Theta}(\cdot)$ achieves good levels of generalization or not. The evaluation scheme and its metrics will be explained in section 4.3.

2.6 Layer types

A layer represents a function that takes one or more tensors as input, passes them through a mathematical transformation, and delivers the resulting tensor to the next layer. Although there are several layer types, they all receive either 2D tensors of size $(N_I \times S)$ (Figure 4a) or 4D tensors of size $(N_I \times C \times W \times H)$ (Figure 4b), where N_I is the number of input images, C is the number of channel in the tensor (or depth), W is the width, H is the height, and S is the number of features. Additionally, (W, H) is often referred to as the spatial size. Below, we describe the different layer types that commonly appear on most CNN models.

2.6.1 Convolutional layer

This layer type is named after the **convolution** operation, which is the mathematical transformation the convolutional layer always applies to its inputs. In this regard, convolutional layers are very effective in learning spatially localized features from their input data (CHOLLET, 2017a, Ch. 5). Therefore, they are usually found in the feature extraction part of the CNN model (see Figure 5).

Given an input tensor $\mathbf{A}$ of shape $(N_I \times C_{in} \times W_{in} \times H_{in})$, the convolutional layer uses its internal tensor of trainable parameters called **filter bank** ($\mathbf{F}$) to compute $\widehat{\mathbf{A}}$, an output tensor of shape $(N_I \times C_{out} \times W_{out} \times H_{out})$. The subscripts *in* and *out* indicate the dimensions by which the input and output tensors may diverge.

Regarding $\mathbf{F}$, it consists of C_{out} filters of shape $(C_{in} \times W_f \times H_f)$. In this context, the filter width (W_f) and the filter height (H_f) form the **kernel size**, a hyper-parameter that require manual setting. Typical choices for the kernel size are (1×1), (3×3), (5×5), and (7×7). Over the years, new hyper-parameters were proposed that increased the complexity of this layer type. Below is a list that explains how a convolutional layer works with respect to each new hyper-parameter.

- **Basic convolution operation (without extra hyper-parameters)**. The convolution operation occurs between each input tensor $\mathbf{A}[i,:,:,:]$ of shape $(C_{in} \times W_{in} \times H_{in})$ and each filter $\mathbf{F}[j,:,:,:]$ of shape $(C_{in} \times W_f \times H_f)$, $1 \leq i \leq N_I; 1 \leq j \leq C_{out}$. Then, the output of $\mathbf{A}[i,:,:,:] \otimes \mathbf{F}[j,:,:,:]$ becomes the activation map $\widehat{\mathbf{A}}[i,j,:,:]$ of shape $(W_{out} \times H_{out})$, where $W_{out} = W_{in} - W_f + 1$ and $H_{out} = H_{in} - H_f + 1$.

- **Padding** (p). The convolution operation described above crops away some of the spatial resolution from $\mathbf{A}[i,:,:,:]$ each time the kernel size is larger than (1×1). When such a result is undesirable, the convolutional layer adds some padding to the input tensor before applying the filter. Specifically, setting $p > 0$ allows $\mathbf{A}[i,:,:,:]$ to temporarily increase its shape from $(C_{in} \times W_{in} \times H_{in})$ to $(C_{in} \times (W_{in} + 2p) \times (H_{in} + 2p))$. For example, a kernel size of (3×3) requires padding of 1 to preserve spatial resolution between the input and output tensors. Similarly, a kernel size of (5×5) requires padding of 2, and so on.

- **Step size or stride**. In the basic operation, the filter $\mathbf{F}[j,:,:,:]$ works as a sliding window function that transverse the spatial dimension of $\mathbf{A}[i,:,:,:]$ with step size of 1. Larger step sizes lead to output tensors with lower spatial resolutions. For example, as explored in (Springenberg *et al.*, 2015), convolutional layers with a step size of 2 and $p = 0$ can successfully work as pooling layers because they also halve the spatial resolution of the input tensor ($W_{out} = W_{in}/2$ and $H_{out} = H_{in}/2$).

- **Dilation rate**. This hyper-parameter defines the spacing between the filter values during the convolution operation. For example, a (3×3) kernel size with a dilation rate of 1 will match the receptive field of a (5×5) kernel size. Therefore, one of the main properties of this hyper-parameter is that it allows having larger receptive fields without incrementing the computational cost. We refer the reader to (YU; KOLTUN, 2016; LI *et al.*, 2019) for a detailed explanation of the benefits of using this hyper-parameter within the architecture of a CNN model.

- **Number of groups**. This hyper-parameter was proposed in (Xie *et al.*, 2017). In the basic convolution operation, there is only one group, which means that each input tensor $\mathbf{A}[i,:,:,:]$ will eventually be convolved with every filter $\mathbf{F}[j,:,:,:]$. When there are two groups, $\mathbf{A}[i,:,:,:]$ is partitioned into the tensors $\mathbf{A}[i, 1 : \frac{C_{in}}{2}, :, :]$ and $\mathbf{A}[i, \frac{C_{in}}{2} + 1 : C, :, :]$, each of shape $(\frac{C_{in}}{2} \times W_{in} \times H_{in})$. Then, half of the filters operate over the first tensor and the other half operate over the second one. In this context, an operation called **depthwise**

convolution occurs, when the number of groups is set to C_{in}, and $C_{out} = k^{*}C_{in}$, (k is an integer number).

2.6.2 Fully connected layer

Fully connected layers are essential for the proper functioning of CNN models. Therefore, as shown in Figure 5, the classification part of the CNN model includes at least one fully connected layer. Additionally, this layer type can serve as auxiliary classifiers that help increase the performance of the CNN model (SZEGEDY *et al.*, 2015).

Given an input tensor $\mathbf{T}$ of size $(N_I \times S_{in})$, the fully connected layer returns an output tensor $\widehat{\mathbf{T}}$ of shape $(N_I \times S_{out})$ by multiplying $\mathbf{T}$ with an internal set of trainable parameters $\mathbf{W}$, followed by the addition of multiple bias terms $\mathbf{b}$. In this regard, N_I is the number of images, S_{in} is the number of input features, and S_{out} is the number of output features. Regarding $\mathbf{W}$ and $\mathbf{b}$, the former is a 2D tensor of shape $(S_{in} \times S_{out})$, while the latter is a 1D tensor of shape (S_{out}). Therefore, unlike convolutional layers that can process input tensors of arbitrary spatial sizes, fully connected layers can only process input tensors with fixed-sized dimensions.

2.6.3 Pooling layer

Pooling layers are simple but powerful layer types that have no trainable parameters. Their purpose is to reduce the spatial size of the input tensor while preserving its spatial information. Therefore, pooling layers allow convolutional layers to focus on detecting high-level features rather than minor local distortions (KHAN *et al.*, 2020).

Given an input tensor $\mathbf{A}$ of size $(N_I \times C \times W_{in} \times H_{in})$, the pooling layer divides each activation map $\mathbf{A}[i,c,:,:]$ into multiple areas of size $(W_p \times H_p)$, where W_p and H_p are the kernel width and the kernel height of the pooling layer, respectively $(1 \leq i \leq N_I;\ 1 \leq c \leq C)$. The values of each generated area then become the input of a pooling function, which returns a single value as output. Consequently, the processing of all the activation maps in $\mathbf{A}$ yields an output tensor $\widehat{\mathbf{A}}$ of shape $(N_I \times C \times W_{out} \times H_{out})$, where $W_{out} = \frac{W_{in}}{W_p}; H_{out} = \frac{H_{in}}{H_p}$.

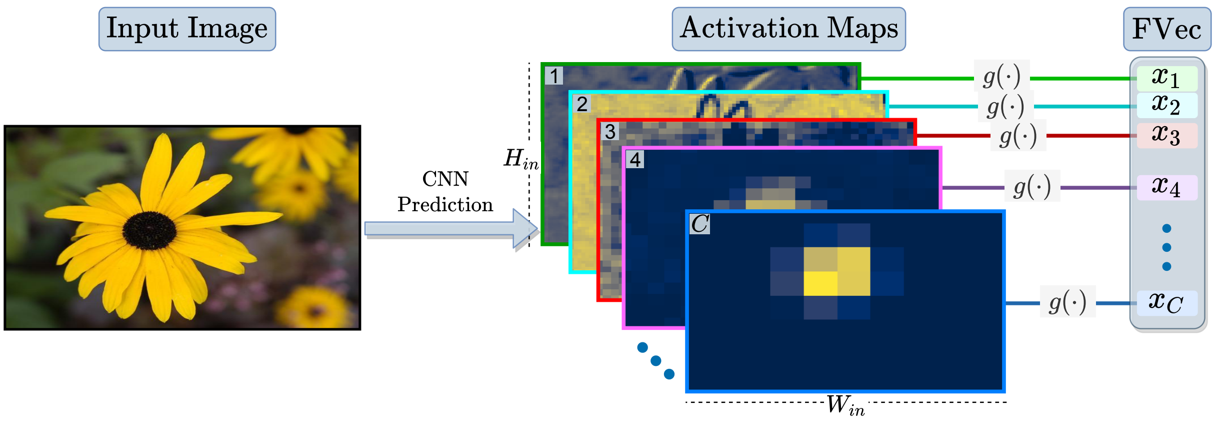
Regarding the pooling functions, the simplest ones perform **max pooling** or **average pooling** operations (LECUN *et al.*, 1998). On the other hand, more complex pooling functions are those that combine multiple levels of max-average pooling operations (YU *et al.*, 2014; LEE; GALLAGHER; TU, 2016), or those that follow second-order pooling approaches (GAO *et al.*, 2019). Additionally, He *et al.* (2015) proposed the **spatial pyramidal pooling**, which consists of using different kernel sizes to generate pooling areas of various shapes.

2.6.4 Global Average Pooling layer

One of the pooling layers that currently stands out from the rest is the Global Average Pooling (GAP) layer (LIN; CHEN; YAN, 2013). The GAP layer is equivalent to the conventional average pooling layer, whose kernel size completely covers the area of the input activation map $\mathbf{A}[i,c,:,:]$ ($W_p = W_{in}; H_p = H_{in}$). Consequently, only one pooling operation occurs for each activation map $g(\mathbf{A}[i,c,:,:])$, which returns a single scalar x_c, as shown in Figure 9.

In this context, upon processing an input tensor of size ($N_I \times C \times W_{in} \times H_{in}$), the GAP layer returns an ($N_I \times C \times 1 \times 1$) tensor, which can be written as a 2D tensor of shape ($N_I \times C$). Indeed, the simplicity and versatility of the GAP layer are what make it applicable for many CNN architectures. For example, in classification tasks, it is common for modern CNN models to include a GAP layer after their feature extraction parts (SZEGEDY *et al.*, 2016; He *et al.*, 2015; Zhang *et al.*, 2017; TAN; LE, 2019). Furthermore, the global context provided by the GAP layers has proven to be well suited for use in various locations within the feature extraction part of the CNN model (Hu *et al.*, 2019).

Figure 9 – **Global Average Pooling layer.** It returns a feature vector for each input set of activation maps. For better visualization, we show how this layer works for single image inputs ($N_I = 1$). The input image was taken from the `Flowers` dataset (Nilsback; Zisserman, 2008).



Source: Elaborated by the author.

2.6.5 Activation function

Neural networks that consist of only convolutional layers or fully connected layers cannot replicate nonlinear functions, such as the XOR function (GOODFELLOW; BENGIO; COURVILLE, 2016, ch. 6.1). This limitation occurs because convolutional layers and fully connected layers can only apply linear transformations to their input tensors.

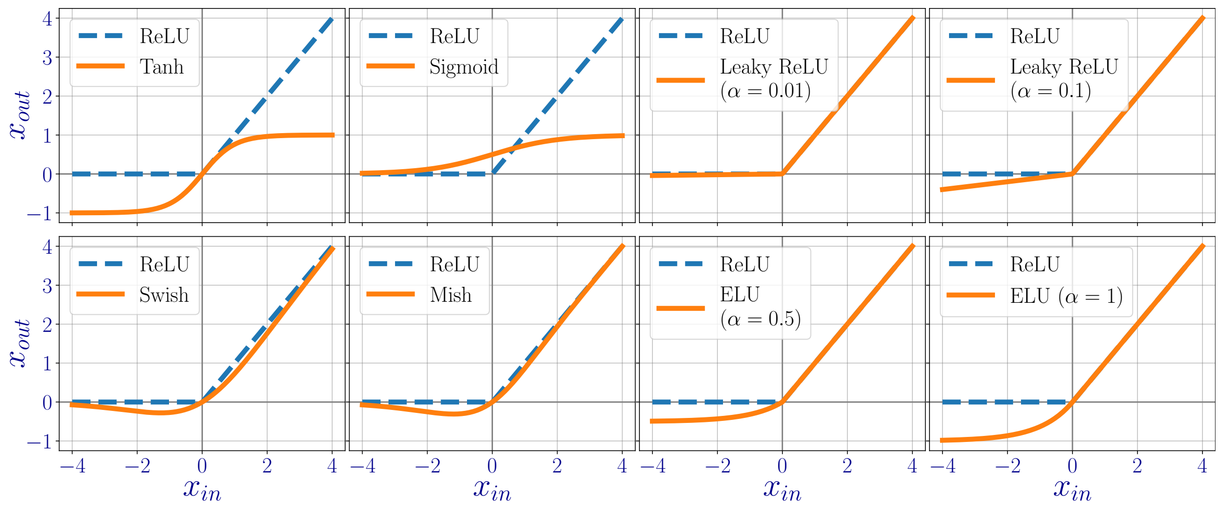
Therefore, CNN models include layers that perform nonlinear operations called **activation functions** to circumvent the above limitation. In this sense, given an input tensor $\mathbf{A}$ of an arbitrary number of dimensions, the activation function applies a nonlinear operation $z(\cdot)$ as follows: $x_{out} = z(x_{in}), \forall x_{in} \in \mathbf{A}$, where x_{in} is an element from the input tensor, and x_{out} is the transformed element. Consequently, the activation function returns an output tensor that maintains the same shape as the input.

Before deep learning became a mainstream phenomenon, the most popular activation functions were the logistic function (or **Sigmoid**): $x_{out} = 1/(1+\exp(-x_{in}))$ and the hyperbolic tangent (**Tanh**): $x_{out} = (\exp(x_{in}) - \exp(-x_{in}))/(\exp(x_{in}) + \exp(-x_{in}))$. However, although Sigmoid and Tanh are still relevant in some specific deep learning applications, they have mostly been replaced by other activation functions that offer greater advantages (APICELLA *et al.*, 2021).

Nowadays, the most popular activation function is the Rectified Linear Unit (ReLU): $x_{out} = \max(0, x_{in})$. Compared to Tanh or Sigmoid, ReLU offers faster training times and can address the vanishing gradient problem more effectively (GLOROT; BORDES; BENGIO, 2011). Additionally, Lin and Jegelka (2018) demonstrated that deep CNN models with ReLU layers can successfully approximate any nonlinear function. However, ReLU still has some drawbacks that were addressed by alternative activation functions, such as Leaky ReLU (MAAS; HANNUN; NG, 2013), Parametric ReLU (He *et al.*, 2015), Exponential Linear Unit (ELU) (CLEVERT; UNTERTHINER; HOCHREITER, 2016), Swish (RAMACHANDRAN; ZOPH; LE, 2018), and Mish (MISRA, 2020).

Figure 10 shows the curves followed by many types of activation functions, from which two conclusions emerge: (i) The curves of Tanh and Sigmoid differ greatly from the ReLU curve. (ii) Despite some of the more recent activation functions have an additional trainable parameter α that modifies the amplitude of their curves, they are all not that different from ReLU. We refer the reader to (APICELLA *et al.*, 2021) for more information on these layers.

Figure 10 – **Examples of activation functions.** All curves are compared to the ReLU curve.



Source: Elaborated by the author.

2.6.6 Normalization layer

In traditional ML, **data normalization** is one of several methods that aim to make different samples look more similar, making it easier for the model to learn and generalize to new data (CHOLLET, 2017a, ch. 7.3.1). Consequently, data normalization is a crucial pre-processing step that can significantly affect the performance of the trained ML model (LUOR, 2015). As for CNN models, despite their greater complexity compared to classic ML models, they still benefit from data normalization. Indeed, for image classification tasks, the default protocol followed by many DL libraries, such as PyTorch (PASZKE *et al.*, 2017), is to compute the per-channel mean μ_c and the per-channel standard deviation σ_c from the set of training images. Then, during the training or the inference processes, every input image I of shape $(C \times W \times H)$ is transformed as follows:

$$I[c,:,:] \leftarrow \frac{I[c,:,:]-\mu_c}{\sigma_c} \tag{2.9}$$

where c is the channel index, $1 \leq c \leq C$.

Unlike traditional ML models, CNN models typically have multiple hidden layers, where the output of one layer feeds into one or more subsequent layers. In this regard, the optimization algorithm explained in subsection 2.5.2 works under the assumption that each layer generates outputs with distributions that do not change over time. However, as the optimization algorithm updates all the CNN model parameters simultaneously, the above assumption is often unfulfilled, leading to unexpected results and instability that slows down the training process or causes it to fail (GOODFELLOW; BENGIO; COURVILLE, 2016, Ch. 8.7). This issue is called the **internal covariance shift problem**, and strategies for mitigating it have been proposed over the years.

For instance, Ioffe and Szegedy (2015) proposed Batch Normalization (BN), a layer that transforms its input tensor using mathematical operations similar to Equation 2.9. The BN layer commonly works on both 2D and 4D input tensors. In more detail, given a 4D tensor $\mathbf{A}$ of shape $(N_I \times C \times W_{in} \times H_{in})$, the batch normalization layer computes the mini-batch mean: $\mu_c = \texttt{mean}(\mathbf{A}[:,\mathbf{c},:,:])$ and the mini-batch standard deviation: $\sigma_c = \texttt{std}(\mathbf{A}[:,\mathbf{c},:,:])$ for each channel c, $1 \leq c \leq C$. Then, each activation map is normalized using the following equation:

$$\mathbf{A}[i,c,:,:] \leftarrow \gamma_c \frac{\mathbf{A}[i,c,:,:]-\mu_c}{\sigma_c} + \beta_c \quad , \tag{2.10}$$

where γ_c and β_c are trainable parameters. In general, Equation 2.10 is suitable for training the CNN model. However, it can fail during inference because an accurate estimate of μ_c and σ_c requires random mini-batches of N_{Test} images, where N_{Test} is at least equal to the training batch-size (N_I). Therefore, during training, most BN implementations replace μ_c and σ_c in Equation 2.10 with the exponential moving averages ($\widehat{\mu_c}$ and $\sqrt{\widehat{\sigma_c^2}}$) below:

$$\widehat{\mu_c} \leftarrow (m)\widehat{\mu_c} + (1-m)\mu_c, \qquad \widehat{\sigma_c^2} \leftarrow (m)\widehat{\sigma_c^2} + (1-m)\sigma_c^2 \quad , \tag{2.11}$$

where $m \in [0,1]$ is an hyper-parameter with default value of 0.1. During inference, $\widehat{\mu_c}$ and $\sqrt{\widehat{\sigma_c^2}}$ remain unchanged.

Although there is some debate about whether BN actually mitigates the internal covariance shift problem (SANTURKAR *et al.*, 2018), there is not doubt that BN allowed CNN models to achieve higher levels of generalization with less training effort. Consequently, BN has been present in the architecture of most CNN models since its introduction (KHAN *et al.*, 2020). Similar to the ReLU activation function, there are also many variants of BN, such as Layer Normalization, Instance Normalization, Group Normalization (WU; HE, 2018), and Filter Response Normalization (FRN) (SINGH; KRISHNAN, 2020). Although these variants solve many intrinsic problems of BN, only FRN seems to compete with BN in practice.

2.7 Final considerations

Convolutional Neural Networks are powerful DL tools for solving challenging visual recognition tasks. This chapter has covered the most relevant CNN definitions for efficiently using and creating CNN models. Additionally, due to the characteristics and objectives of this project, the content of this chapter was oriented towards image classification and object detection tasks. The next chapter will cover different deep learning models and transfer learning strategies.

CHAPTER

3

TRANSFER LEARNING AND DEEP CNN MODELS

3.1 Initial considerations

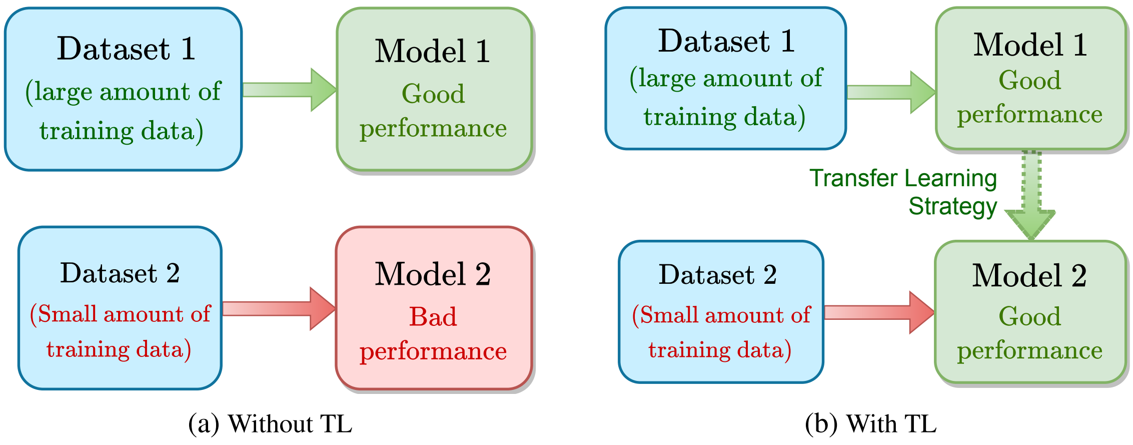
The previous chapter presented the fundamental concepts for Convolutional Neural Networks (CNN). This chapter will explore more advanced CNN concepts that are relevant to this project. In short, we will formally describe the details of Transfer Learning (TL), a powerful class of strategies that allows successful CNN models to transfer their knowledge from a source domain (e.g., everyday object images) to a target domain (e.g., biological images). We will primarily focus on inductive transfer learning strategies, where labeled data is available in the source and target domains (Ribani; Marengoni, 2019). Next, we will present some relevant CNN models commonly used for TL applications, such as ALEXNET, RESNET, POLYNET, NASNET, SENET, and EFFICIENTNET. Finally, we will show some TL strategies for image classification and object detection tasks.

3.2 Transfer Learning definitions

Applying the notation presented in (PAN; YANG, 2010) to a visual recognition context, a domain $\mathcal{D}$ is defined by an image space $\mathcal{I}$ and a marginal probability distribution $P(I)$, where $I = \{I_1, I_2, \ldots, I_n\} \in \mathcal{I}$ is a set of n training images. Additionally, a task $\mathcal{T}$ in the domain $\mathcal{D}$ is defined by two components: a label space $\mathcal{Y}$ and a predictive function $f_{\mathcal{D}}(\cdot)$. In this regard, a dataset D results from associating n training images $I_i \in \mathcal{I}$ with n class labels $y_i \in \mathcal{Y}$; $(1 \leq i \leq n)$. Then, during training, $f_{\mathcal{D}}(\cdot)$ learns from D to predict the class label for each novel input image $I_{new} \in \mathcal{D}$.

Let us now define $\mathcal{D}_\mathbf{S}$ and $\mathcal{D}_\mathbf{T}$ as the **source and target domains**, respectively. By the same logic, we designate $\mathcal{T}_\mathbf{S}$ and $\mathcal{T}_\mathbf{T}$ as the source and target tasks, respectively. Then, TL is defined as follows: given $\mathcal{D}_{\mathbf{S}}$, $\mathcal{T}_{\mathbf{S}}$, $\mathcal{D}_{\mathbf{T}}$, and $\mathcal{T}_{\mathbf{T}}$, TL attempts to improve the target predictive function $f_{\mathbf{T}}(\cdot)$ by combining the information from $\mathcal{D}_{\mathbf{S}}$ and $\mathcal{T}_{\mathbf{S}}$ with that from $\mathcal{D}_{\mathbf{T}}$ and $\mathcal{T}_{\mathbf{T}}$. As a prerequisite, the source predictive function $f_{\mathbf{S}}(\cdot)$ must have a high chance of predicting the correct output for unseen input images $\in \mathcal{I}_{\mathbf{S}}$, whereas $f_{\mathbf{T}}(\cdot)$ should still have significant room for improvement. For example, as shown in Figure 11, typical TL situations occur when the number of training samples in $\mathcal{D}_{\mathbf{S}}$ is much higher than that of $\mathcal{D}_{\mathbf{T}}$, ($n_{\mathbf{S}} \gg n_{\mathbf{T}}$).

Figure 11 – **A typical transfer learning situation**. Dataset 1 and dataset 2 become the source and target datasets, respectively. Consequently, the model learned from the first dataset becomes the source predictive function $f_{\mathbf{S}}(\cdot)$, and the model learned from the second dataset becomes the target predictive function $f_{\mathbf{T}}(\cdot)$.



Source: Elaborated by the author.

3.3 Transfer learning and CNN models

CNN models have consistently achieved outstanding results in various domains since the arrival of ALEXNET (SINGH *et al.*, 2018; TONG; WU; ZHOU, 2020; PICCIALLI *et al.*, 2021). In this regard, three factors mainly affect the performance of CNN models: (i) their architectural design, (ii) their size, and (iii) the amount of training data available. Indeed, with enough training data, deep CNN models can reach greater levels of generalization than their shallower versions (Xie *et al.*, 2017; TAN; LE, 2019). However, when the data is scarce, deep CNN models will likely have overfitting problems (OQUAB *et al.*, 2014).

The above situation has led to an increasing effort to collect large amounts of data. While there is a great success in some domains (SUN *et al.*, 2017; Sumbul *et al.*, 2019), there are still many other domains where large-scale data collection is not feasible. Fortunately, there are at least two classes of strategies that can mitigate the lack of training data. One is **data augmentation**, and the other is **transfer learning** (TL). Indeed, both classes of strategies have been actively researched throughout the years (ZOPH *et al.*, 2020a; Ha; Liu; Liu, 2020). This project focuses on a particular TL setting, **Inductive TL**, in which the training samples from the source and target domains are known in advance (Ribani; Marengoni, 2019).

3.3.1 CNN models as predictive functions

Using the definitions from section 3.2, CNN models are predictive functions that map images into class labels (in classification tasks) or labeled bounding boxes (in detection tasks). In this sense, TL requires that any CNN model serving as a source predictive function $f_{\mathbf{S}}(\cdot)$ satisfy the following criteria: (i) it must have been trained on a large-scale dataset from a domain that contains rich information, and (ii) it must have high predictive performance in the assigned task.

For the first requirement, the scientific community typically uses `ImageNet` (RUSSAKOVSKY *et al.*, 2015), a dataset with about 1.2 million training samples organized in 1000 class labels. Additionally, since `ImageNet` has a wide variety of everyday objects, it typically leads to semantically strong CNN models that satisfy the second requirement. Therefore, in terms of TL, `ImageNet` becomes the source dataset. Over the years, some datasets with many more training samples than `ImageNet` have emerged (SUN *et al.*, 2017). However, only a small number of researchers has access to these larger source datasets.

Once we have a good $f_{\mathbf{S}}(\cdot)$, we can apply TL in target domains where the training data cannot effectively cope with the given task. For instance, as shown in Figure 11a, domains with small datasets usually fall into this category. Popular target datasets include `Pascal-VOC` (EVERINGHAM *et al.*, 2010), `Flowers` (Nilsback; Zisserman, 2008), `StanfordCars` (Krause *et al.*, 2013), and `COCO` (Lin *et al.*, 2014).

3.3.2 General Transfer Learning strategy

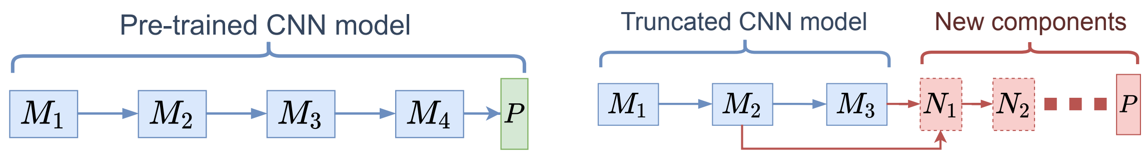
Over the years, many researchers have proposed multiple TL strategies for different target tasks (OQUAB *et al.*, 2014; Guo *et al.*, 2019; Tan; Pang; Le, 2020) and domains (SAHA *et al.*, 2016; LU *et al.*, 2018). Despite the large number of TL strategies available, most of them receive as input one source predictive function $f_{\mathbf{S}}(\cdot)$ and one target dataset (see Figure 11b). In general, $f_{\mathbf{S}}(\cdot)$ is commonly referred to as the **pre-trained CNN model** (see Figure 12a). Then, the TL strategy generates an improved target predictive function $f_{\mathbf{T}}(\cdot)$ by repurposing the knowledge stored in $f_{\mathbf{S}}(\cdot)$ with the information from the target dataset. In more detail, most TL strategies have a general template consisting of the following two steps.

1. **Modify the pre-trained CNN model.** As shown in Figure 12, this step consists of truncating the pre-trained CNN model at some layer, and then adding one or more new components. In this sense, a component may be a block, an encoder, a traditional classifier, or any other structure that receives and outputs data. Indeed, this modification can be as simple as replacing the last fully-connected layer with another (YOSINSKI *et al.*, 2014) or as complex as adding multiple components and connections to the pre-trained CNN model (CAI; VASCONCELOS, 2018; Guo *et al.*, 2019).

2. **Train the modified model.** The training protocol can be applied to only the newly added

components (RAZAVIAN *et al.*, 2014; CIMPOI *et al.*, 2016; SONG *et al.*, 2016), or to the entire modified CNN model (ZHANG; XUE; DANA, 2017; LI *et al.*, 2019), or to only a few selected layers and blocks (Rusu *et al.*, 2016; Guo *et al.*, 2019).

Figure 12 – **First step of the general TL strategy.** (a) Pre-trained CNN model with four blocks (M_1 – M_4). (b) Modified model that results from first removing the block M_4 of (a), and then adding new components ($N_1, N_2, \ldots$) and connections. P: component that computes the final predictions for the classification or detection task.



(a) $f_{\mathrm{S}}(\cdot)$ (b) Improved $f_{\mathrm{T}}(\cdot)$

Source: Elaborated by the author.

3.4 Main CNN models used as source predictive functions

The effectiveness of the TL strategy is directly associated with the quality of the pre-trained CNN model (Kornblith; Shlens; Le, 2019; ZHANG; DAVISON, 2020). Hence, the choice of a pre-trained CNN model is of particular importance in obtaining satisfactory TL results. Fortunately, numerous repositories in popular DL frameworks, including Keras (CHOLLET *et al.*, 2015) and PyTorch (PASZKE *et al.*, 2017), have made public an updated list of pre-trained CNN models that use `ImageNet` as their source dataset. However, selecting the best model is not a trivial matter, as the list of CNN models with innovative architectural designs is vast and continually growing (KHAN *et al.*, 2020).

Several factors influence the decision to prefer one pre-trained CNN model over another. Table 1 lists some of these factors. For instance, a typical approach is to select the pre-trained CNN model that achieved the highest `ImageNet` top 1 accuracy value. In support of this idea, Kornblith, Shlens and Le (2019) found a positive correlation between the ImageNet top 1 accuracy value and the one obtained by the modified model in the target task.

Another relevant factor is the size of the CNN model. For example, some of the top CNN models shown in Table 1 have above 100M trainable parameters, which becomes problematic in target tasks that require as quick inference times as possible. On the other hand, TL strategies that only train the newly-added components of the modified CNN model may find it helpful to choose the pre-trained model based on the number of activation maps it generates at its last convolutional layer. A final point to keep in mind is related to the work of Mormont, Geurts and Marée (2018), which showed that the correlation between accuracy values in the source and

Table 1 – **Comparison of the most popular CNN models.** The "Full Size" column contains the number of trainable parameters in the entire CNN model. On the other hand, the "FE Size" column counts only the parameters in the feature extraction part of the CNN model. Additionally, we provide the number of activation maps computed by the last convolutional layer of each CNN model. The `ImageNet` top 1 and top 5 accuracy values merely indicative, since they can vary from one library to another.

CNN model	Full Size	FE Size	# Act. maps	Top 1 Acc.	Top 5 Acc.
ALEXNET (KRIZHEVSKY; SUTSKEVER; HINTON, 2012)	61M	2M	256	56.5%	79.1%
VGG-16 (SIMONYAN; ZISSERMAN, 2015)	138M	15M	512	71.6%	90.4%
VGG-19 (SIMONYAN; ZISSERMAN, 2015)	144M	20M	512	72.4%	90.9%
RESNET-18 (HE *et al.*, 2016)	12M	11M	512	69.8%	89.1%
RESNET-50 (HE *et al.*, 2016)	26M	24M	2048	76.1%	92.9%
RESNET-152 (HE *et al.*, 2016)	60M	58M	2048	78.3%	94.1%
DENSENET-121 (HUANG *et al.*, 2017)	8M	7M	1024	74.4%	92.0%
DENSENET-201 (HUANG *et al.*, 2017)	20M	18M	1920	77.3%	93.5%
INCEPTION-V3 (SZEGEDY *et al.*, 2016)	24M	22M	2048	77.3%	93.5%
XCEPTION (CHOLLET, 2017b)	23M	21M	2048	79.1%	94.4%
INCEPTION-V4 (SZEGEDY *et al.*, 2017)	43M	41M	1536	80.1%	95.0%
INCEPTIONRESNET-V2 (SZEGEDY *et al.*, 2017)	56M	54M	1536	80.5%	95.3%
POLYNET (Zhang *et al.*, 2017)	95M	93M	2048	81.0%	95.6%
RESNEXT-50-32X4D (Xie *et al.*, 2017)	25M	23M	2048	77.6%	93.7%
RESNEXT-101-32X4D (Xie *et al.*, 2017)	44M	42M	2048	78.8%	94.4%
SERESNET-50 (Hu *et al.*, 2019)	28M	26M	2048	77.6%	93.8%
SERESNET-152 (Hu *et al.*, 2019)	67M	65M	2048	78.7%	94.4%
SERESNEXT-50-32X4D (Hu *et al.*, 2019)	28M	26M	2048	79.1%	94.4%
SERESNEXT-101-32X4D (Hu *et al.*, 2019)	49M	47M	2048	80.2%	95.0%
SENET-154 (Hu *et al.*, 2019)	115M	113M	2048	81.3%	95.5%
NASNET-A (ZOPH *et al.*, 2018)	89M	85M	4032	82.7%	96.1%
EFFICIENTNET-B0 (TAN; LE, 2019)	5M	4M	1280	77.7%	93.5%
EFFICIENTNET-B1 (TAN; LE, 2019)	8M	7M	1280	78.6%	94.2%
EFFICIENTNET-B2 (TAN; LE, 2019)	9M	8M	1408	80.6%	95.3%
EFFICIENTNET-B3 (TAN; LE, 2019)	12M	11M	1536	82.0%	96.1%
EFFICIENTNET-B4 (TAN; LE, 2019)	19M	18M	1792	83.4%	96.6%
EFFICIENTNET-B5 (TAN; LE, 2019)	30M	28M	2048	83.4%	96.6%
EFFICIENTNET-B6 (TAN; LE, 2019)	43M	41M	2304	84.0%	96.9%
EFFICIENTNET-B7 (TAN; LE, 2019)	66M	64M	2560	84.1%	96.9%
RESNET-RS-50 (Bello *et al.*, 2021)	36M	34M	2048	79.9%	95.0%
RESNET-RS-101 (Bello *et al.*, 2021)	64M	62M	2048	82.3%	96.0%
RESNET-RS-152 (Bello *et al.*, 2021)	87M	85M	2048	83.7%	96.6%
RESNET-RS-270 (Bello *et al.*, 2021)	130M	128M	2048	84.4%	97.0%
RESNET-RS-350 (Bello *et al.*, 2021)	164M	162M	2048	84.7%	97.0%
RESNET-RS-420 (Bello *et al.*, 2021)	192M	190M	2048	85.0%	97.1%

Source: Research data.

target tasks is more diffuse in target domains that substantially differ from the source domain The following subsections will detail the architectures of the most relevant CNN models used for TL.

3.4.1 LeNet, AlexNet, *and* ZFNet

LENET is a pioneering family of small CNN models that were used for recognizing low-resolution images, such as handwritten digits. For instance, Figure 5 shows LENET-5 (LECUN *et al.*, 1998), a CNN model with five trainable layers: two convolutional layers in its feature extraction part and three fully-connected layers in its classification part. Furthermore, there is a flattening layer between the feature extraction and classification parts that reshapes the 4D tensors into 2D ones Other important traits of LENET-5 are as follows: (i) it has a sub-sampling layer after each convolutional layer, (ii) its activation function is a hyperbolic tangent, and (ii) its last fully-connected layer has Gaussian connections. Although LENET-5 achieved high predictive power, it was only applied to low resolution images at the time due to limited computing resources.

Many years later, Krizhevsky, Sutskever and Hinton (2012) proposed ALEXNET, a CNN model that dramatically outperformed all traditional methods in the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) (RUSSAKOVSKY *et al.*, 2015). In general, ALEXNET is a larger version of LENET-5 and its training would not have been possible without the arrival of Graphics Processing Units. In more detail, ALEXNET is deeper than LENET-5 as it has two more convolutional layers, and it is wider since it includes more filters per convolutional layer. Additionally, its feature extraction part reduces the size of each input image by approximately 32 times. Last, ALEXNET uses two regularization techniques to minimize overfitting: (i) it places a dropout layer between two consecutive fully-connected layers (SRIVASTAVA *et al.*, 2014) and (ii) data augmentation.

Zeiler and Fergus (2014) proposed ZFNET, a CNN model derived from smartly tweaking the ALEXNET hyper-parameters. For example, ZFNET uses 7×7 filters instead of the 11×11 filters from ALEXNET. The main contribution of ZFNET was to demonstrate that it is not necessary to drastically change the architecture of a CNN model to achieve significant improvements in recognition accuracy.

3.4.2 VGG *models*

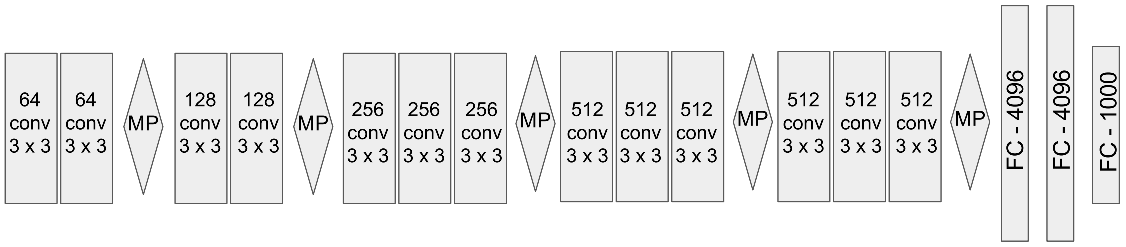
The Visual Geometry Group [1] developed VGG (SIMONYAN; ZISSERMAN, 2015), a family of CNN models that also follow the LENET design pattern. In more detail, as shown in Figure 13, the feature extraction part of all VGG models comprises five blocks of contiguous convolutional layers. Each block ends with a max-pooling layer that reduces the spatial size of the input activation maps by half. As for the classification part, it always involves three fully-connected layers.

Indeed, the researchers created different VGG models to explore the limits of how many convolutional layers can be included in a CNN model without adversely affecting performance. In particular, the researchers found that VGG-16 (see Figure 13a) and VGG-19 (see Figure 13b), with 13 and 16 convolutional layers, respectively, offered the best trade between training cost and predictive power. However, beyond 16 convolutional layers, the marginal increase in predictive power does not compensate for the additional cost associated with training the model.

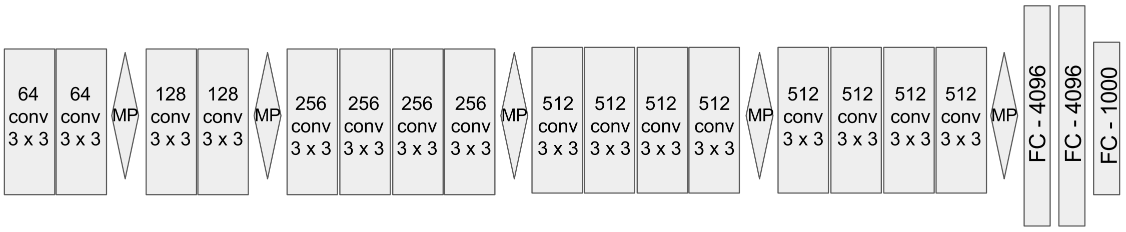
[1] The Visual Geometry Group belong to the Department of Engineering Science, University of Oxford. Website: <http://www.robots.ox.ac.uk/~vgg/>.

VGG models also have the following properties: (i) They only include 3×3 filters with a stride of one. (ii) They used mini-batch GD with momentum to optimize the trainable parameters of the CNN model. (iii) Lastly, they combined two regularization techniques to avoid overfitting during training (dropout layers and L2 penalty)

Figure 13 – **Architecture of VGG-16 and VGG-19.** Both models have five modules in their feature extraction part, and three fully-connected layers in their classification part. There is a flattening layer between the last max-pooling layer and first fully-connected layer (not shown in the image). FC: Fully-connected layer, MP: max-pooling layer, conv: convolutional layer.



(a) VGG-16



(b) VGG-19

Source: Elaborated by the author.

3.4.3 Network in Network

A major drawback of the LeNet, AlexNet, and VGG models is that they can only process fixed-size input images. This limitation stems from the flattening layer that all these models use to convert the feature extraction part output into a format suitable for the classification part. In more detail, the flattening layer, as the name suggests, reshapes the (N_I, C, W, H) input tensor into an $(N_I, C*W*H)$ output tensor by treating each spatial position as a scalar value.

In this context, Lin, Chen and Yan (2013) developed NiN, a CNN model whose feature extraction part consists of four blocks, with three convolutional layers per block. The architecture of NiN presents the following particularities.

- Two of the three convolutional layers in each NiN block have kernel sizes of 1×1. Internally, each 1×1 filter works as a fully-connected layer that acts independently on every pixel location of the tensor along the channel dimension.

- Additionally, NIN uses a global average pooling layer (GAP) instead of a flattening layer. Since the GAP layer does not take into account the spatial information of the input tensor (see subsection 2.6.4), the NIN model can process input images of arbitrary sizes.
- Lastly, NIN does not have an explicit classification part, as the output of the GAP layer becomes the prediction.

Despite not being as popular as other CNN models, NIN greatly influenced the architecture of many modern CNN models. Indeed, the rest of this section will present only CNN models that have at least have one GAP layer and one convolutional layer with kernel sizes of 1×1.

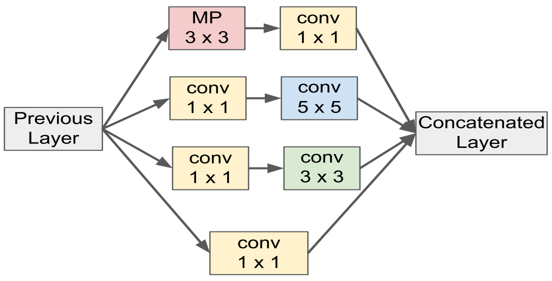
3.4.4 Inception *and* Xception

INCEPTION is a family of CNN models that combines the ideas from NIN (GAP layer, 1×1 filters) with the paradigm of repeated blocks. In general, there are four pure INCEPTION models, where each of them is based on an older version, if any. In this sense, the first version is INCEPTION-V1, and the last is INCEPTION-V4. Each of these versions are described below.

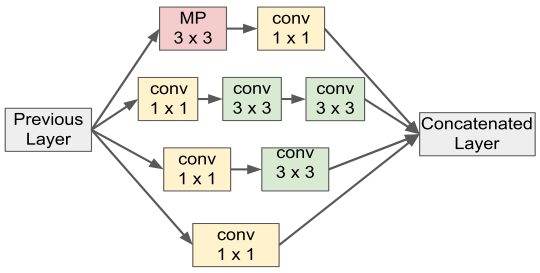
- INCEPTION-V1 or GOOGLENET (SZEGEDY *et al.*, 2015). It consists of two initial convolutional layers followed by nine **inception blocks**, a GAP layer, and a fully-connected layer. To accelerate the training process, INCEPTION-V1 also has auxiliary classifiers that are connected to earlier inception blocks. This version rapidly popularized for three primary reasons: (i) it won the ILSVRC challenge in 2014, (ii) it dramatically reduced the number of trainable parameters related to previous ILSVRC winners by having a classification part with only one fully-connected layer instead of three, and (iii) it introduced the inception block. As shown in Figure 14a, an inception block contains parallel subnetworks that apply different sets of operations to the same input. For example, one subnetwork may apply a max pooling operation followed by a convolution with a filter size of 5×5, while another may only apply a convolution with a 1×1 filter size. Then, at the end of each inception block, there is a concatenation layer responsible for merging the outputs of all the subnetworks.
- INCEPTION-V2 and INCEPTION-V3 (SZEGEDY *et al.*, 2016). It increased the number of inception blocks to eleven. Also, it introduced the idea of factoring convolutions, which within the context of *convolutional layers* it is the replacement of big filter sizes with a series of smaller ones. For instance, Figure 14b shows how a convolutional layer with a filter size of 5×5 can be replaced by two contiguous convolutional layers of 3×3. INCEPTION-V2 introduced the idea of asymmetric convolutions, in which it replaced a 7×7 filter with one 7×1 filter followed by another 1×7 filter. As for INCEPTIONV3, it maintains almost the same architecture as INCEPTIONV2, but it adds batch normalization (IOFFE; SZEGEDY, 2015) to the **auxiliary classifiers**.

- INCEPTION-V4 (SZEGEDY *et al.*, 2017). It applied all the concepts introduced in the previous versions to create a more uniform architecture. In more detail, this version has three primary inception blocks, which were repeated several times. Consequently, the total number of inception blocks in INCEPTION-V4 is higher than the one in INCEPTION-V3. Additionally, this version explicitly defines two **reduction blocks**, in charge of reducing the spatial size of the input sets of activation maps.

Figure 14 – **Two different versions of the same inception block.** The INCEPTION-V2 version replaces the 5×5 filters with two consecutive 3×3 filters.



(a) Inception block (INCEPTION-V1)



(b) Inception block (INCEPTION-V2)

Source: Elaborated by the author.

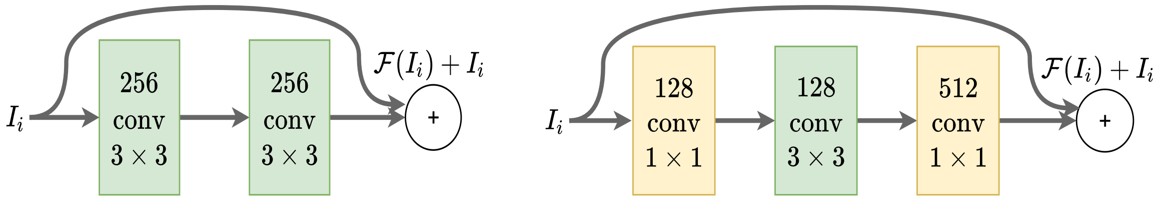
On the other hand, Chollet (2017b) proposed XCEPTION, a CNN model that reinterprets all INCEPTION-V3 blocks as **depthwise separable convolutions**. In the paper, the depthwise separable convolution is defined as a **pointwise convolution** followed by a **depthwise convolution**. In this regard, the pointwise convolution is equivalent to the standard convolution with a 1×1 filter size. In contrast, the depthwise convolution is the **grouped convolution** when the number of groups is equal to the number of channels in the input tensor. See subsection 2.6.1 for more information concerning the grouped convolutions. XCEPTION achieved slightly better results than INCEPTION-V3 on the `ImageNet` and `JFT` datasets using the same number of trainable parameters.

3.4.5 ResNet, DenseNet, ResNeXt, *and* SENet *models*

RESNET (HE *et al.*, 2016) is a simple yet powerful family of CNN models whose feature extraction part can easily contains more than 100 convolutional layers. Indeed, its architecture is almost as simple as the VGG model. However, RESNET can reach much deeper architectures (up to 152 layers) while still providing a reasonable balance between training cost and predictive power. The success of RESNET stems from the concept of building blocks. As shown in Figure 15, each building block receives an input tensor I_i and applies a transformation $\mathcal{F}(I_i,\{W_i\})$ over it. Then, instead of returning just $\mathcal{F}(I_i,\{W_i\})$, the building block returns the sum $I_{i+1} = \mathcal{F}(I_i,\{W_i\})+I_i$. Note that W_i contains the trainable parameters of the building block, and $\mathcal{F}$ becomes the residual mapping to learn.

There are two types of building blocks in RESNET models: (i) normal (see Figure 15a) and (ii) "bottleneck" (see Figure 15b). According to the original authors, both types are suitable for small and large RESNET models. However, the "bottleneck" building blocks are significantly less expensive than the normal ones. Therefore, they are recommended for RESNET models with 50 layers or more. Regarding their training process, the RESNET models usually adopt the parameter initialization from (He *et al.*, 2015), use BN layers (IOFFE; SZEGEDY, 2015), and avoid the dropout layers.

Figure 15 – **Types of building blocks in RESNET.** The normal building block works very well on relatively small RESNET models (up to 49 layers), and the "bottleneck" building block works best on larger CNN models (up to 152 layers).

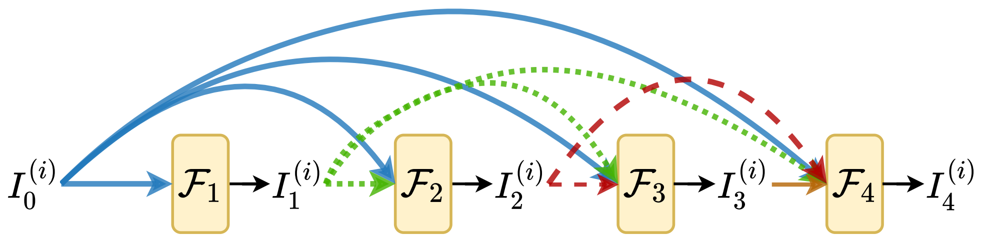


(a) Normal building block (b) Bottleneck building block

Source: Elaborated by the author.

In response to the success of RESNET, Huang *et al.* (2017) developed DENSENET, a family of CNN models that support architectures with up to 264 layers. In this regard, the feature extraction part of every DENSENET model is organized into four **dense blocks**, each of which is subdivided into several **dense layers**. As shown in Figure 16, within the i-th dense block ($1 \leq i \leq 4$), the j-th dense layer must process the concatenation of activation maps generated by all preceding dense layers $[I_0^{(i)}, I_1^{(i)}, \ldots, I_{j-1}^{(i)}]$. Then, the transformation $\mathcal{F}_j$ is applied as follows: $I_j^{(i)} = \mathcal{F}_j([I_0^{(i)}, I_1^{(i)}, \ldots, I_{j-1}^{(i)}])$. In general, $\mathcal{F}_j$ is defined as a composite function of three consecutive operations: BN + ReLU + convolutional layer. Additionally, between each pair of contiguous dense blocks, a **transition layer** reduces the spatial size of the tensor given from one dense block to the next. Finally, there is a GAP layer that communicates the output from the last dense block to the fully-connected layer that generates the predictions.

Figure 16 – **Dense Block operations.** At the j-th dense layer, the composite function $\mathcal{F}$ process the output tensors from all previous dense layers $[I_0^{(i)}, I_1^{(i)}, \ldots, I_{j-1}^{(i)}]$.



Source: Elaborated by the author.

ResNeXt (Xie *et al.*, 2017) and SENet (Hu *et al.*, 2019) are other highly influential CNN models built on ResNet. As for the ResNeXt models, they are multi-branch architectures that were proposed as an extension of the ResNet models. More specifically, the building blocks of these models include **grouped convolutions** (see subsection 2.6.1), in which independent set of filters see different parts of the input activation maps. Regarding the SENet models, they include several “Squeeze-and-Excitation” (SE) blocks, which can be inserted after any other block or layer. For example, it is frequently plugged after the building blocks of the ResNet or ResNeXt models. A SE block explicitly models the channel inter-dependencies of its input activation maps, which has proven to be useful for improving the performance of other CNN models in several datasets.

3.4.6 NASNet *and* EfficientNet

Zoph *et al.* (2018) used Google’s vast computing resources to develop NASNet, a family of CNN models whose architectures were learned directly from the dataset. In more detail, the goal was to find the optimal blocks and cells (groups of blocks) within a search space that included 13 different operations, such as 1×1 convolutions, 3×3 depthwise separable convolutions, and 3×3 max-pooling. The authors used a reinforcement learning (RL)-based approach to perform the search process, and used the `CIFAR-10` dataset as their learning target. In this regard, the RL-based approach rewarded architectures with high accuracy values at each search step. Following the RL-based approach’s execution, the architecture obtained was extended by stacking multiple copies of its cells. The resulting architecture became a NASNet model, which then followed the training from scratch procedure on `ImageNet` (see subsection 2.5.2).

NASNet has the merit of paving the way for large-scale neural architecture search approaches. Consequently, CNN models with automatically derived architectures have been proposed over the years, outperforming their hand-engineered counterparts on various reference datasets (LIU; SIMONYAN; YANG, 2019; REAL *et al.*, 2019; Wu *et al.*, 2019; TAN; LE, 2021). One of the most prominent examples is EfficientNet (TAN; LE, 2019), a family of CNN models typically included in the winning solutions to a range of visual recognition challenges (Ha; Liu; Liu, 2020).

With the aid of an RL-based algorithm, EfficientNet is the result of a Pareto solution that optimizes both time efficiency (measured in FLOPS) and accuracy. Unlike NASNet, EfficientNet performed the architecture search on the `ImageNet` dataset. We refer the reader to (Tan *et al.*, 2019) for details about how the RL-based search algorithm works. The output of the searching process was a baseline CNN model that received the name of EfficientNet-B0. Regarding its architecture, EfficienNet-B0 consists of seven blocks, where the operations, connections, and number of identical layers in each block were all determined by the RL-based search algorithm. In the case of EfficientNet-B1, it is a scaled-up version of the baseline CNN model. As a result, the scaling increased (i) the input image resolution at training time from 224×224 to 240×240, (ii) the depth by self-repeating the sub-blocks of layers within each block, and (iii) the width by generating more activation maps at each block. These three dimensions are further scaled up in EFFICIENTNET-B2, EFFICIENTNET-B3, and so on.

3.4.7 Other important CNN models

Many other CNN models can fulfill the role of source predictive functions. For example, Szegedy *et al.* (2017) proposed INCEPTION-RESNET, a family of CNN models that successfully combined the ideas of inception blocks and residual units. In more detail, the INCEPTION-RESNET models have two types of blocks: (i) **pure inception blocks** and (ii) **residual inception blocks**. Essentially, the first type of block reduces the resolution of the input set of activation maps by half, while the second type follows the residual unit formula $I_{i+1} = F(I_i) + I_i$. However, in this case, F is an inception block that differs from those described in (SZEGEDY *et al.*, 2015) in two ways: (i) it ends with a 1×1 convolutional layer that forces the output channel dimension of $F(I_i)$ to match that of I_i, and (ii) it does not include pooling layers.

POLYNET (Zhang *et al.*, 2017) is the polynomial version of the INCEPTION-RESNET model. In short, as opposed to the RESNET and INCEPTION-RESNET models whose blocks only follow the structure $F(I_i) + I_i$, POLYNET proposed the following types of blocks: (i) $I_i + F(I_i) + F(I_i)^2$, (ii) $I_i + F(I_i) + GF(I_i)$, and (iii) $I_i + F(I_i) + G(I_i)$, where F and G are inception blocks.

Lastly, Bello *et al.* (2021) presented RESNET-RS, another highly effective class of CNN models with the potential to compete with much more complex CNN models in both accuracy and inference speed. According to RESNET-RS, scaling rules are more important than optimizing architectural changes. Consequently, RESNET-RS did not substantially modify the architecture of the classic RESNET models. Instead, it employed an exhaustive search algorithm to derive the following scaling rules. First, use depth-scaling when overfitting is a concern, otherwise use width-scaling. Second, increase the input image resolution more slowly than EFFICIENTNET. For instance, instead of starting with 224×224 input images, RESNET-RS began with 128×128.

3.5 Transfer Learning for image classification tasks

In essence, (i) `ImageNet` is the source domain dataset (D_S), (ii) the source task ($\mathcal{T}_\text{S}$) is generic object recognition, and (iii) the source predictive function ($f_\text{S}(\cdot)$) is the chosen pre-trained CNN model. Now, this section will cover TL strategies when given a target domain T, a target domain dataset D_T, and a image classification task $\mathcal{T}_\text{T}$.

Due to the boom in deep learning research, there is also an enormous variety of TL strategies. Nonetheless, as described in subsection 3.3.2, the goal of all TL strategies is to repurpose the knowledge from the pre-trained CNN model to solve the target classification task at stake. Consequently, every TL strategy involves two basic steps: (i) modify the pre-trained

CNN model and (ii) train the modified model. Based on these basic steps, there are two primary types of TL strategies.

- TL strategies that employ the pre-trained part of the modified model to yield feature vectors that later will serve to train the newly-added components (RAZAVIAN *et al.*, 2014; SONG *et al.*, 2016; Wang *et al.*, 2019)
- TL strategies that follow a **fine-tuning** process. During fine-tuning, old and new components are trained together (ANDREARCZYK; WHELAN, 2016; ZHANG; XUE; DANA, 2017; Guo *et al.*, 2019).

The next subsections contain brief descriptions of the main TL strategies for each type.

3.5.1 Type 1: Pre-trained part of the modified CNN model used as a feature extractor

In this type of TL strategies, all the parameters belonging to the pre-trained part of the modified CNN model are frozen. Then, these TL strategies propose algorithms for transforming the pre-trained part output tensors into semantically-strong feature vectors. Next, a dedicated classifier is trained and evaluated using feature vectors computed from the labeled and unlabeled samples. Brief descriptions of some TL strategies of this type are provided below.

- FC-CNN (RAZAVIAN *et al.*, 2014). It truncates the pre-trained CNN model up to one of its hidden fully-connected layers and attaches a dedicated classifier (e.g., Support Vector Machine classifier) at the end of truncated model. Consequently, unlike other TL strategies, FC-CNN does not propose a new algorithm for computing feature vectors. It instead trains and evaluates the dedicated classifier using the feature vectors computed by the fully-connected layer. FC-CNN has a fast computation time and consistently produces good results with both ALEXNET and VGG models. Nevertheless, it is not recommended for pre-trained CNN models with a single fully-connected layer because it could result in feature vectors that are too biased towards the source domain.
- GAP-CNN. It has been integrated into many DL frameworks despite not being formally proposed by any research group. For example, in Keras (CHOLLET *et al.*, 2015), the definition of every pre-trained CNN model includes two parameters (`"include_top"` and `"pooling"`) that facilitate the creation and configuration of GAP-CNN. In this regard, the following two situations arise when assembling the modified CNN model. (i) The pre-trained CNN model has a GAP layer[2] connecting its feature extraction part with its classification part, or (ii) it does not have one. In the former case, GAP-CNN truncates

[2] Global Pooling Average layers are covered in subsection 2.6.4.

the model up to its GAP layer, while in the latter case, GAP-CNN truncates the model up to its last convolutional layer and then attaches a new GAP layer. The modified model ends with a dedicated classifier that processes the features vectors computed by the GAP layer. Mormont, Geurts and Marée (2018) and Scabini *et al.* (2019) used GAP-CNN to compare different pre-trained CNN models for digital pathology classification and texture recognition tasks, respectively. Moreover, Mormont, Geurts and Marée (2018) conducted experiments by truncating the pre-trained CNN model at intermediate convolutional layers. In this project, we will extend the above idea by including feature selection approaches into GAP-CNN.

- FV-CNN (CIMPOI *et al.*, 2016). The Fisher Vector encoder (FV) is one of the most powerful versions of the bag of visual words (BoW) approach (PERRONNIN; SáNCHEZ; MENSINK, 2010). Here, FV is used to transform sets of activation maps into high-dimensional feature vectors. Then, FV-CNN reduces the size of the resulting feature vectors with the Principal Component Analysis (PCA) method. As with the previous TL methods, the modified model also ends with a dedicated classifier (e.g., a Support Vector Machine), where the training and evaluation processes occur. Experimental results presented in the original study indicated that truncating the pre-trained CNN model at its last convolutional layer led to higher accuracy rates for **category-based texture datasets**[3] than when truncating the CNN model at any other layer. Furthermore, FV-CNN consistently outperforms FC-CNN across a variety of datasets. However, FV-CNN is also more computationally expensive than FC-CNN because it needs to generate a **codebook** before computing the feature vectors. This codebook can be thought of as an intermediate representation that summarizes the information provided by the truncated model.

- Neural-based Feature Transformation (NFT) (SONG *et al.*, 2016). It is a computationally expensive TL strategy based on FV-CNN and FC-CNN. In more detail, NFT starts by concatenating the feature vectors generated by FV-CNN and FC-CNN for each input image. This set of feature vectors then feeds a set of **feed-forward neural networks** (FFN). By taking the intermediate feature representations from the hidden layer of FNN, NFT creates another set of feature vectors. Lastly, NFT uses the new set of feature vectors to train and evaluate the dedicated classifier. Based on the research paper, the NFT results are superior to the FV-CNN results, but training will take more time.

- EASYTL (Wang *et al.*, 2019). This TL strategy focuses on improving the dedicated classifier. Therefore, the first step of EASYTL is to perform an intra-domain alignment between two different sets of feature vectors. Specifically, the first set should be drawn from the training data in D_{T}, whereas the second set should be unlabeled data. This alignment is intended to identify common elements between the two sets. After completing the alignment process, it yields a set of feature vectors that act as fixed parameters for a

[3] Definition and examples of category-based texture datasets are in subsection 4.2.2.

predefined cost function J. Then, following the minimization of J, unlabeled samples will receive corresponding class labels. According to the authors, EASYTL is most appropriate for datasets where there is a high degree of intra-class variability between the labeled and unlabeled data. Indeed, datasets that meet the above criteria are not highly uncommon. Examples of such datasets are: `KTH-TIPS2b`, `Outex10`, and `Outex12`.

3.5.2 Type 2: Fine-tuning the modified CNN model

This type of TL also involves modifying the pre-trained CNN model. Consequently, as shown in Figure 12, the modified model must include a pre-trained component with connections to one or more newly-added components. Then, the fine-tuning process consists of partially or entirely retraining the modified model with samples from the target dataset D_T.

In general, TL strategies of this type can achieve outstanding results. However, they have two primary drawbacks. (i) the fine-tuning process may take various minutes or several days, depending on the size of the dataset. (ii) There is a need to set hyper-parameters carefully, as they can dramatically affect the fine-tuning process positively or negatively. Both drawbacks are intertwined, as it is necessary to fine-tune the modified CNN model numerous times to find an optimal hyper-parameter set (LI *et al.*, 2020). The following paragraphs describe some TL methods of this type.

- Standard fine-tuning. Here, the pre-trained CNN model is truncated at some layer. Afterward, the modified model is built by adding a fully-connected layer of c neurons at the end of the truncated model, where c is the number of classes in the target dataset. Additionally, Yosinski *et al.* (2014) demonstrated that simply replacing the original fully-connected layer with one containing c neurons is sufficient to observe the benefits of using this TL strategy.

- T-CNN (ANDREARCZYK; WHELAN, 2016). In addition to adding a fully-connected layer of c neurons to the truncated CNN model, T-CNN proposed the inclusion of an **energy layer** between the last convolutional layer and the first fully-connected layer of the modified CNN model. This energy layer is responsible for computing the energy statistic for each given activation map. As an additional note, T-CNN discussed the benefits of reusing earlier convolutional layers of the AlexNet model.

- Deep Texture Encoding Network (DEEPTEN) (ZHANG; XUE; DANA, 2017). This TL strategy results from combining the main ideas of FV-CNN with the standard fine-tuning method. In this regard, DEEPTEN introduced a new encoding layer, which unlike the one in FV-CNN, is fully differentiable, allowing optimization algorithms such as SGD to be applied. Then, DEEPTEN builds a modified CNN model by placing the new encoding layer between the truncated CNN model and the fully-connected layer of c neurons. As an additional note, the encoding layer can learn robust residual encoders, which makes it appropriate for texture classification tasks.

- SpotTune (Guo *et al.*, 2019). Only ResNet models are compatible with this TL strategy. In more detail, SpotTune performs adaptive fine-tuning by running two copies of the pre-trained ResNet model, referred to as (i) **the fine-tune route** and (ii) **the frozen route**. As their names suggest, the fine-tune route optimizes its internal parameters during training, while the frozen route does not. Moreover, SpotTune introduced a policy network, which determines which route each training sample from the target dataset must take. For example, the first routing decision entails selecting the initial building block from one of the two ResNet copies. Likewise, there are two possible paths to take for each successive routing decision. Hence, each training sample follows a dynamic route, which terminates in a fully-connected layer of c neurons. SpotTune optimizes all non-frozen layers, including those from the policy network, during training. Performance-wise, this TL strategy achieves promising results across many target domains (e.g., sketch images and texture images).

3.6 Transfer Learning for object detection tasks

Once again, (i) the source domain (D_S) is ImageNet, (ii) the source task ($\mathcal{T}_S$) is generic object recognition, and the source predictive function ($f_s(\cdot)$) is the chosen pre-trained CNN model. This section will briefly explain some object detection models ($f_T(\cdot)$) that deal with an object detection task ($\mathcal{T}_T$), when given a dataset (D_T) in a target domain T.

3.6.1 From image classification to object detection

With the tremendous advances made in image classification tasks over the last decade, object detection tasks have also received a great deal of attention. Indeed, it is becoming more common for researchers to publish a novel CNN model that faces image classification tasks alongside a modified version designed for object detection tasks (ZOPH *et al.*, 2018; Tan *et al.*, 2019; WANG *et al.*, 2020).

In general, when a CNN model performs well in image classification tasks it modified version also does well in object detection tasks (LIU *et al.*, 2020). Several factors can contribute to this situation. For example, as described in section section 2.3, all object detection tasks have an internal classification sub-task. Indeed, even when there is a single foreground class, a classification step must be performed to distinguish foreground from background.

In practice, the modified versions used for object detection result from using an intrinsic TL strategy, in which the feature extraction part of the original CNN model receives new connections to components tasked with locating and classifying the objects in an image. Consequently, as discussed in subsection 2.4.2, the feature extraction part of an object detector is referred to as the **backbone**, and it is easily interchangeable with any other backbone that uses the same base architecture (Ghiasi; Lin; Le, 2019; TONG; WU; ZHOU, 2020).

Two backbones share the same base architecture, when each can be divided into N_b blocks such that blocks at the same level compute activation maps with the same spatial resolution. For instance, all RESNET models share a five-block architecture, in which there is a subsampling layer at the end of each block that reduces the spatial resolution of the input activation maps by half. As the above situation occurs with other CNN families, such as DENSENET, RESNEXT, and EFFICIENTNET, it is relatively easy to replace the backbones within an object detector.

3.6.2 Types of object detectors

As described in section 2.3, object detection tasks are inherently more challenging than image classification tasks. Therefore, the two types of TL strategies discussed in section section 3.5 are irrelevant in this case as the majority of TL strategies for object detection rely on fine-tuning approaches. Nevertheless, these TL strategies can still be categorized based on the number of stages the modified model will take to generate its final predictions (LIU *et al.*, 2020). The following are brief descriptions of each type and its principal representatives.

Two-stage and multi-stage detectors

The modified model of a two-stage detector generally includes the backbone of the pre-trained CNN model, a region-proposal component, and a final prediction component. More specifically, two-stage detectors became popular after the introduction of R-CNN (Girshick *et al.*, 2014). The first stage of R-CNN generates numerous region proposals using the Selective Search Algorithm (SSA), which is directly applied to the input images. Then, the second stage encodes, refines, and classifies those region proposals based on the pre-trained CNN model, a bounding box regression module and a class-specific linear SVM classifier. Note that only the second stage of R-CNN used the pre-trained CNN model.

The author of R-CNN also proposed FAST R-CNN (Girshick, 2015), a TL strategy that achieved faster processing times than RCNN by doing the following modifications. (i) It only reuses the feature extraction part of the pre-trained CNN model (the backbone), while R-CNN reused layers up to the classification part. (ii) Although FAST R-CNN also uses SSA to generate the region proposals, it applies SSA to the set of activation maps generated by the backbone instead of applying it to the input image. (iii) Finally, in the prediction stage, FAST R-CNN uses fully-connected layers to classify and refine the region proposals.

FASTER R-CNN (REN *et al.*, 2017) is an object detector that significantly improved its region-proposal stage by introducing the Region Proposal Network (RPN). In this sense, RPN is a fast and independent component that receives activation maps from the backbone and generates region proposals. Additionally, RPN can be trained simultaneously with the rest of the modified model. On the other hand, dai *et al.* (2016) developed R-FCN, a fully-convolutional object detector, allowing it to achieve better processing speed than FASTER R-CNN. In more detail, R-FCN modifies the second stage of FASTER R-CNN by replacing the fully-connected layers with convolutional layers. These convolutional layers are responsible for generating one activation map per class. Then, each set of feature maps computes a class score for every region proposal provided by RPN. Lin *et al.* (2017) presented the Feature Pyramid network (FPN), an upgrade to RPN that exploits the inherent multi-scale nature of the backbone architecture. Thus, unlike RPN, FPN can generate region proposals for both small and large objects within a single image.

Regarding multi-stage object detectors, they often include one or more prediction stages, responsible for refining and correcting the results of the previous stages. Consequently, this type of object detector typically achieves higher predictive power but at the cost of taking longer processing times. In this regard, CASCADE R-CNN (CAI; VASCONCELOS, 2018) is one of the most reliable multi-stage object detectors due to its simple implementation and consistent detection gains across several applications. Indeed, CASCADE R-CNN inspired other multi-stage object detectors, such as HTC (Chen *et al.*, 2019) and CASCADE R-CNN-RS (Du *et al.*, 2021).

One-stage detectors

By merging the region-proposal and prediction stages into one, one-stage (or unified) object detectors can significantly reduce the computational burden of two- and multi-stage object detectors. They became popular after the publication of OVERFEAT, which locates and classifies objects using a sliding-window approach over the set of activation maps computed by the pre-trained backbone of ALEXNET. Moreover, OVERFEAT creates a pyramid of images of different resolutions for each image, allowing it to detect small and large objects. In this sense, OVERFEAT may be considerably slower when the task involves detecting both small and large objects due to the need for higher image resolutions. In general, OverFeat inspired other very popular one-stage detectors, such as SSD (LIU *et al.*, 2016) and YOLO (REDMON; FARHADI, 2017; Redmon; Farhadi, 2018).

However, the higher speed of one-stage detectors came at the expense of lower accuracy results. In this regard, Lin *et al.* (2017) concludes that this lower accuracy results primarily from an imbalance between the foreground and background classes, i.e., RPN or FPN tend to create many more region proposals for the background class than for the foreground class. Therefore, Lin *et al.* (2017) proposed an improved one-stage detector called RETINANET that could equal the accuracy of state-of-the-art two-stage detectors. In more detail, RETINANET adds a factor to the cross-entropy loss function to boost the reward of well-classified region proposals. This slightly modified loss function received the name of Focal Loss (FL).

The success of RETINANET inspired more evolved one-stage detectors such as RE-

FINEDET (ZHANG *et al.*, 2018b), M2DET(ZHAO *et al.*, 2019), EFFICIENTDET (Tan; Pang; Le, 2020), and RETINANET-RS (Du *et al.*, 2021).

3.7 Final considerations

When training data is limited, transfer learning (TL) is one of the best techniques in DL for producing CNN models with enough predictive power to handle challenging target tasks. This chapter covered different the basic definitions in TL, the general steps that each TL strategy follows, the most used pre-trained CNN models, and the main TL strategies for image classification and object detection tasks. Finally, we also discussed that, despite the large and growing number of TL strategies, it is crucial to choose a pre-trained CNN model that can maximize the benefits of the chosen TL strategy.

CHAPTER 4

DATASETS AND EVALUATION METRICS

4.1 Initial considerations

This chapter describes the datasets and evaluation metrics used in the experimental part of this thesis. In a supervised machine learning environment, the above two elements are essential in determining whether a proposed predictive model outperforms others on a specific task.

Datasets are helpful to identify the strengths and weaknesses of different predictive models. Therefore, the scientific community continually publishes robust datasets as benchmarking tools that help researchers identify the domains where their predictive models outperform others (Lin *et al.*, 2014; RUSSAKOVSKY *et al.*, 2015; Bansal *et al.*, 2017; Sumbul *et al.*, 2019). This situation has spawned a growing effort to collect diverse and challenging publicly available datasets to understand better the pros and cons of predictive models across multiple domains (Hurt *et al.*, 2018; TRIANTAFILLOU *et al.*, 2020).

As for evaluation metrics, they allow scientists to quantify the performance of individual predictive models on a specific task. This quantification typically involves splitting a given dataset B into two disjoint sets (B_{train}, B_{test}). The model then goes through a training phase at B_{train}, followed by a test phase at B_{test}. The chosen evaluation metric assigns a score to the predictive model by comparing its predicted outputs with the real ones. Although multiple evaluation metrics are available for each task (HOSSIN; SULAIMAN, 2015; MOCCIA *et al.*, 2018; LIU *et al.*, 2020), some may give misleading results in certain situations. For example, in classification tasks with imbalanced datasets, a badly-chosen evaluation metric can erroneously assigns good scores to models that only predict a single class (Fatourechi *et al.*, 2008).

4.2 Datasets

This project considers twenty-two annotated datasets: fourteen for image classification tasks and eight for object detection tasks. Regarding the classification tasks, Table 2 presents the description of 14 texture datasets. Regarding the detection tasks, Table 3 presents the description of seven datasets for single-class stomata detection, and Table 4 describe one dataset for single-class and multi-class pollen detection.

4.2.1 Splits and splitting strategies

Before detailing the datasets for classification and detection tasks, here we define two fundamental terms: (i) *split* and (ii) splitting strategy. Given the annotated dataset B, the term *split* refers to one of the many possible partitions of B into two non-overlapping sets, namely the training set (B_{train}) and the test set (B_{test}), where $B = B_{\text{train}} \cup B_{\text{test}}$. In this regard, every *split* is essential as it allows predictive models to have proper training and evaluation procedures.

Some datasets provide predefined *splits* that allow researchers to successfully benchmark their models against one another. On the other hand, when a dataset B has no predefined *splits*, it is up to the researcher to generate them using a *splitting strategy*. Indeed, there are various *splitting strategies* in the literature, each with its own advantages and disadvantages (Raschka, 2018). We briefly describe some of them in the following paragraphs.

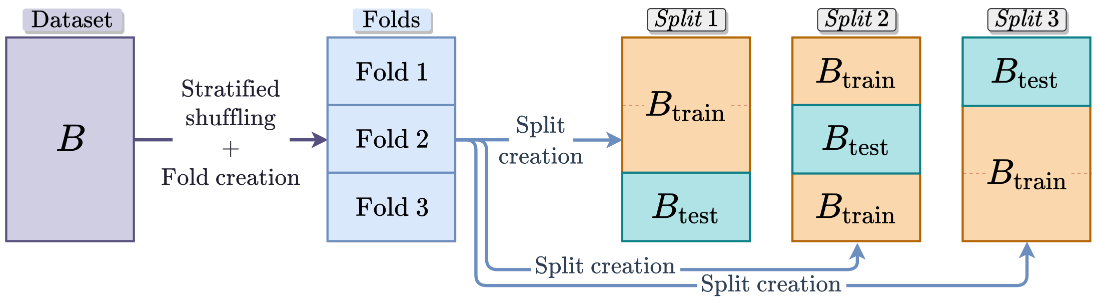
- **Stratified k-fold cross-validation.** It first generates k non-overlapping folds by randomly shuffling B in a stratified manner. Then, it creates k *splits*, where the training set (B_{train}) of each *split* contains a different combination of $k-1$ folds, and B_{test} contains the remaining fold. See Figure 17a for an example of the stratified three-fold cross-validation.

- **Leave-one-out validation.** It is equal to the k-fold cross-validation approach with $k = n_B$, where n_B is the total number of samples in B. In more detail, this strategy generates n_B *splits*, where each one assigns a single sample to B_{test}, and the remaining $n_B - 1$ samples to B_{train}.

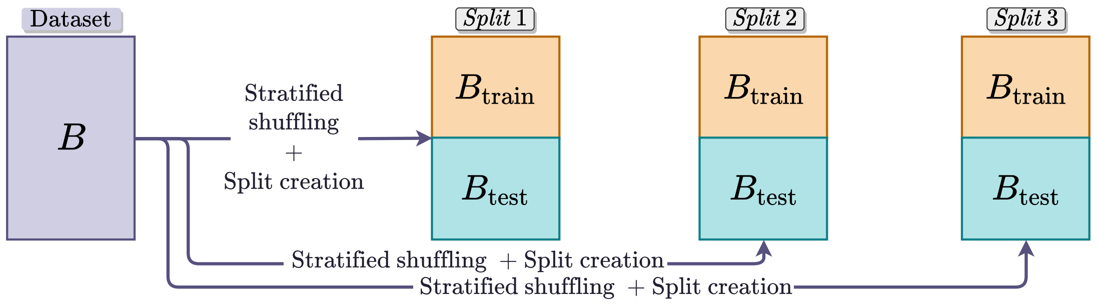
- **Stratified m-repeated holdout validation.** It involves creating m stratified random *splits*, each with p% of the samples from B going to B_{train}, and the other (100-p)% going to B_{test}. See Figure 17b for an example of this strategy with $m = 3$ and $p = 50\%$.

For benchmarking purposes, this project generally uses the predefined *splits* of each dataset, if available. Otherwise, we generate *splits* with the k-fold cross-validation approach or the m-repeated holdout validation approach. The following subsections describe the twenty-two datasets in more detail.

Figure 17 – **High-level view of two splitting strategies.** In the first case, the folds ensure that each image in the dataset will appear exactly once in the test set, while in case two, there is no such guarantee.



(a) Stratified k-fold cross validation (k=3)



Dataset
B
Stratified shuffling + Split creation
Stratified shuffling + Split creation
Stratified shuffling + Split creation
Split 1
Split 2
Split 3
B_{train}
B_{test}

(b) Stratified m-repeated holdout validation (m = 3, p = 50%)

Source: Elaborated by the author.

4.2.2 Datasets for texture classification

In image analysis, *texture* is an intrinsic visual attribute that contains rich semantic information. Although there is no consensus about the exact definition of *texture*, we perceive it as the organization (or even the disorganization) of simple or complex pixel patterns in an image. Its analysis is of great importance in many disciplines (ZARITSKY *et al.*, 2011; VIDYA *et al.*, 2015). Therefore, the scientific community continuously searches for the methods that best characterize the texture information in an image (HUMEAU-HEURTIER, 2019).

In this sense, Table 2 provides details on fourteen datasets typically used in following texture recognition task: given an image with some unknown texture, the question is to assign the image to one of a set of predefined classes based on the information that is embodied in its pixel values. Except for `CUReT_Q4`, all the datasets in Table 2 have been extensively studied over the years (HOSSAIN; SERIKAWA, 2013; CAVALIN; OLIVEIRA, 2017; LIU *et al.*, 2019). Indeed, as shown in Figure 18, these datasets have different challenges and goals resulting from two main factors: (i) the type of texture problem they address and (ii) the splitting strategy used.

– First factor: instance-based vs category-based datasets

Texture datasets can be grouped into (i) instance-based datasets and (ii) category-based datasets. We refer the reader to the last column of Table 2 to learn which type of texture problem

Table 2 – Benchmark datasets used for texture recognition. The symbols N_{ipc} and N_{splits} indicate the number of images per class and the total number of dataset *splits*, respectively. Additionally, $N_{ipc}^{(\text{train})}$ and $N_{ipc}^{(\text{test})}$ are, respectively, the average number of training and test images per class. Datasets with two splitting strategies can also have two different values for $N_{ipc}^{(\text{train})}$, $N_{ipc}^{(\text{test})}$ and N_{splits}. (C): category-based datasets, (I): instance-based datasets.

Texture Dataset	# Classes	N_{ipc}	Splitting strategy	N_{splits}	$N_{ipc}^{(\text{train})}$	$N_{ipc}^{(\text{test})}$	(C)-(I)?
`Brodatz` (BRODATZ, 1966)	111	10	Holdout/CV	10	5/9	5/1	I
`UIUC` (LAZEBNIK; SCHMID; PONCE, 2005)	25	40	Holdout/CV	10	20/36	20/4	I
`USPtex` (BACKES; CASANOVA; BRUNO, 2012)	191	12	Holdout/CV	10	6/10.8	6/1.2	I
`STex` (KWITT, 2010)	476	16	Holdout	10	8	8	I
`Vistex` (PICARD *et al.*, 1995)	54	16	Holdout	10	8	8	I
`CUReT` (DANA *et al.*, 1999)	61	92	Holdout	10	46	46	I
`CUReT_Q4` (modified from Drbohlav and Leonardis (2010))	61	368	Predefined	6	184	184	I
`Outex10 (O10)` (OJALA; PIETIKAINEN; MAENPAA, 2002)	24	180	Predefined	1	20	160	I
`Outex12 (O12)` (OJALA; PIETIKAINEN; MAENPAA, 2002)	24	380	Predefined	1	20	360	I
`Outex13 (O13)` (OJALA *et al.*, 2002)	68	20	Predefined/CV	1/10	10/18	10/2	I
`MBT` (ABDELMOUNAIME; DONG-CHEN, 2013)	154	16	Holdout/CV	10	8/14.4	8/1.6	I
`FMD` (SHARAN; ROSENHOLTZ; ADELSON, 2009)	10	100	Holdout	10	50	50	C
`Kth-Tips2b` (CAPUTO; HAYMAN; MALLIKARJUNA, 2005)	11	432	Predefined	4	324	108	C
`DTD` (CIMPOI *et al.*, 2016)	47	120	Predefined	10	80	40	C

Source: Research data.

each dataset addresses.

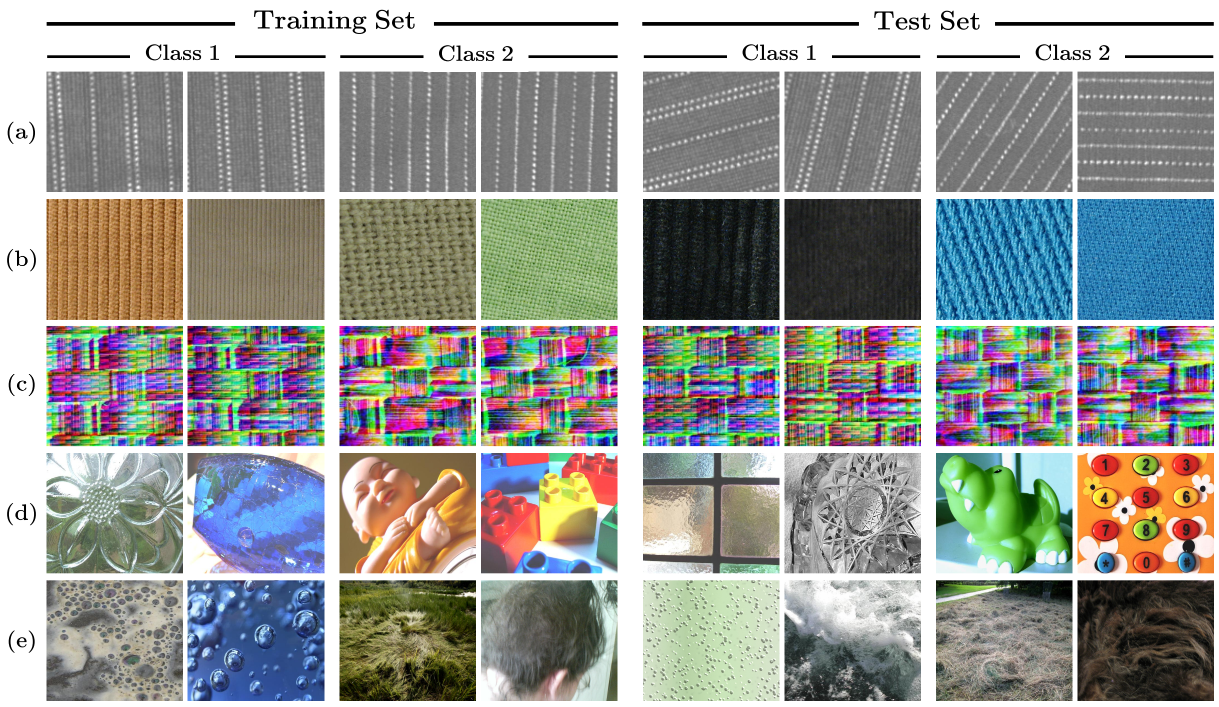
Instance-based datasets contain classes with images that belong to the same physical sample. In this sense, more challenging datasets are those that consider rotation, illumination, and/or viewpoint changes like the `CUReT` and `Outex` suites. Figure 18a and Figure 18c show exemplary images from instance-based datasets. On the other hand, category-based datasets consider more physical samples for the same class. Therefore, the task is to discriminate between the physical samples that belong to different classes. See Figure 18b, Figure 18d, and Figure 18e for exemplary images from category-based datasets.

In general, there is more intra-class variability in category-based datasets than in instance-based datasets. However, while instance-based datasets seem easier than their category-based equivalents, some texture datasets such as `Outex10` and `Outex12` proved particularly challenging for some highly regarded predictive models, like FV-CNN (LIU *et al.*, 2019).

– *Second factor: predefined splits vs. randomly-generated splits*

We can also group texture datasets according to the splitting strategy they follow in the experiments. In this sense, the fourth column of Table 2 shows the strategies we use for each dataset. As stated in subsection 4.2.1, except for the `Outex13` dataset, we do not generate new *splits* on datasets that already include predefined ones. This decision is motivated by the fact that datasets that provide predefined *splits*, generally evaluate specific aspects of the proposed models. For example, `Outex10` and `Outex12` evaluate whether the proposed model is robust against rotation and illumination changes (see Figure 18a). Similarly, each predefined *split* in

Figure 18 – Exemplary images from five texture datasets. Only images of two classes are displayed for each dataset. The training and test sets correspond to one *split* of the dataset (either randomly-generated or predefined). (a) `Outex10` *(predefined)*, (b) `Kth-Tips2b` *(predefined)*, (c) `MBT` *(random)*, (d) `FMD` *(random)*, and (e) `DTD` *(predefined)*.

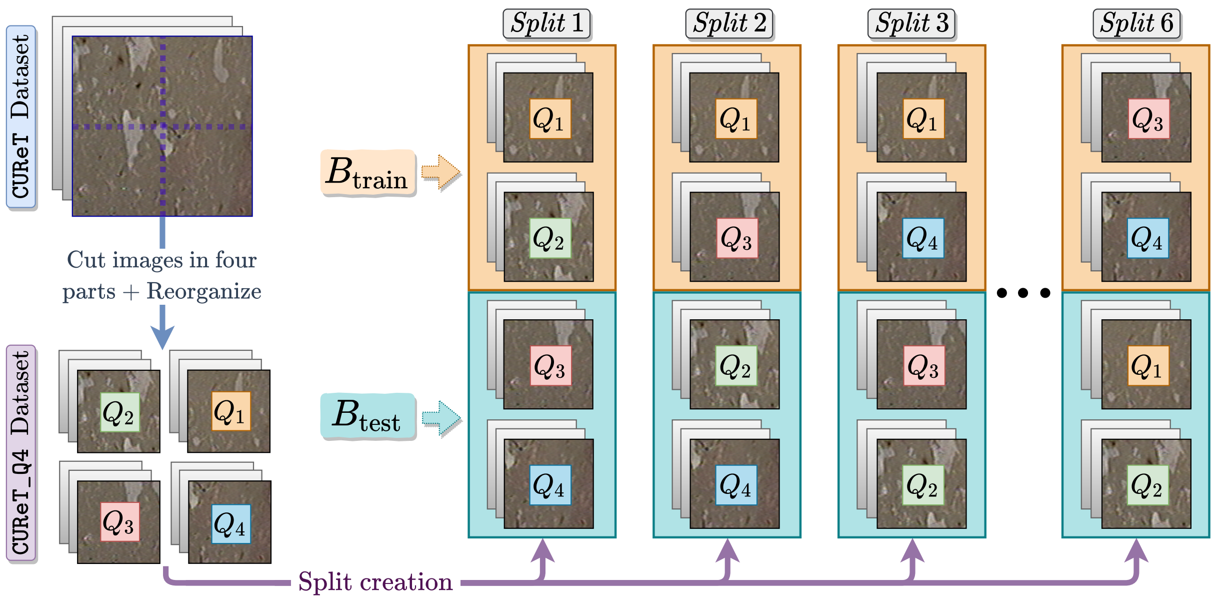


Source: Research data.

`Kth-Tips2b` ensures that images belonging to the same physical sample can only go either to B_{train} or to B_{test} (see Figure 18b). Additional details for datasets with predefined *splits* are as follows:

- **`DTD`.** Since each predefined *split* includes a training, validation, and test set, the first two sets are merged into B_{train}, and the remaining one becomes B_{test}.
- **`Kth-Tips2b`.** In each *split*, B_{train} contains the images from three of the four physical samples, and B_{test} includes the images from the remaining physical sample.
- **`Outex` suites.** We use their predefined *splits* without further processing.
- **`CUReT_Q4`.** It was inspired by the work of Drbohlav and Leonardis (2010), who discovered inherent problems with the `CUReT` dataset that cause predictive models to yield misleading results. In this sense, Drbohlav and Leonardis (2010) achieved more reliable outcomes by using the following splitting strategy. (i) First, he doubled the number of samples in `CUReT` by horizontally slicing every `CUReT` image into two sub-images of the same size. (ii) Then, the sub-images in the upper half went to B_{train}, and those in the lower half went to B_{test}. On the other hand, `CUReT_Q4` is the result of cutting every image from `CUReT` horizontally and vertically. This slicing procedure generates four sub-images of equal size

Figure 19 – CUReT_Q4 and its predefined *splits*. CUReT_Q4 is the result of transforming every image from CUReT into four sub-images. These sub-images are reorganized according to their regions of origin (Q_1, Q_2, Q_3, Q_4). Each predefined *split* uses a different combination of two regions to create B_{train} and B_{test}.



Source: Elaborated by the author.

per image. Consequently, CUReT_Q4 has four times the number of samples as CUReT, and their sub-images are organized according to their region of origin: Q_1, Q_2, Q_3, Q_4 (see Figure 19). Next, we define six *splits* with the following splitting strategy: the training set of each *split* includes all the sub-images that belong to a different combination of two regions (e.g., Q_1 and Q_3), and the test set consists of all the sub-images that belong to the other two regions (e.g., Q_2 and Q_4).

Most of the datasets listed in Table 2 do not provide predefined *splits*. In such cases, we use the following splitting strategies to generate stratified random *splits*.

- Stratified m-repeated holdout validation (with $m = 10, p = 50\%$), which is also used by many deep learning related publications (CIMPOI *et al.*, 2016; LIU *et al.*, 2016; ZHANG; XUE; DANA, 2017; LIU *et al.*, 2019)
- The stratified k-fold cross validation (with $k = 10$), which is highly recommended by many researchers (Raschka, 2018).

4.2.3 Datasets for single-class stomata detection

This thesis also proposes predictive models that deal with the task of single-class stomata detection. We refer to as stomata detection, the automatic generation of bounding boxes that correctly enclose all stomata in a given microscopic image of the leaf epidermis. In this regard, we use the term “single-class” because there is only one foreground class: the stoma.

– A little context in plant stomata research

Stomata are vital microstructures found primarily in the epidermis of plant leaves. Anatomically, a stoma is a microscopic pore whose aperture is regulated by a pair of surrounding *guard cells* (SANTELIA; LAWSON, 2016). Additionally, many plant species have special epidermal cells, known as *subsidiary cells*, that surround each stoma. Indeed, many authors use the number, shape and arrangement of subsidiary cells as relevant features that define different stomata types (CARPENTER, 2005; RUDALL; CHEN; CULLEN, 2017).

When opened, the stoma allows gas exchange between the plant and the environment[1]. This exchange is crucial for the photosynthesis and transpiration of plants. Therefore, many research works have studied the impact of stomatal features (i.e., size, density, and distribution pattern) on the productivity and transpiration efficiency of some plant species (LAWSON; BLATT, 2014; MORALES-NAVARRO *et al.*, 2018; CAINE *et al.*, 2019).

In general, each new study requires biologists to perform manual stomata phenotyping on microscopic images of epidermal leaf tissues. The quality of these images depends on the sample preparation technique and the imaging method used (YUAN *et al.*, 2020). For example, some sample preparation techniques require placing the epidermal tissues in staining solutions to improve stomata visualization (EISELE *et al.*, 2016). In this context, predictive models that perform automatic stomata detection can substantially speed up new studies.

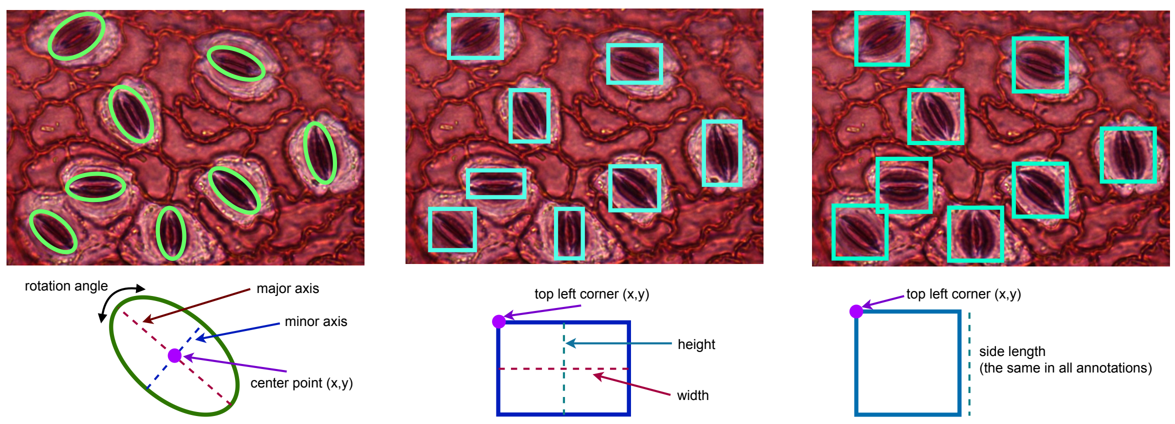
– Manual microscopic image annotation

Within the context of a generic object detection task, a valid training sample consists of an image (I) and a set of rectangular bounding box annotations (G) (Lin *et al.*, 2014). Each annotation ($g \in G$) can be described by five parameters $\langle g_l, g_x, g_y, g_w, g_h \rangle$, where g_l is its class label, (g_x, g_y) indicates the pixel coordinates of its top left corner, and (g_w, g_h) defines its size.

For stomata detection, a valid training sample also consists of an image and a set of annotations. However, the number and type of parameters used in the annotations may differ from those commonly used in generic object detection. For example, since there is only one foreground class, g_l becomes an implicit parameter that always represents the class “stoma”. Additionally, as all existing stomata datasets provide incomplete training samples (images without annotations), we annotated them manually. This situation allowed us to choose the annotation type that best suits the task. In more detail, we considered three annotation types: (i) rotated ellipses, (ii) rectangular bounding boxes, and (iii) square bounding boxes. Of the three, the first and third types take advantage of the stomatal morphological characteristics (see

[1] Typically, the entry of CO_2 and the release of water vapor.

Figure 20 – Comparison of different annotation types and their number of parameters. This thesis performed manual annotation of microscopic images with square bounding boxes. At the image level, annotations with square bounding boxes can depend on two parameters instead of three if the same side length is used in all of them.



(a) Rotated ellipses (b) Rectangular bounding boxes (c) Square bounding boxes

Source: Elaborated by the author.

Figure 20). At the end, given that our goal was to annotate as many images as possible, we opted for the third approach. Therefore, Table 3 reports the number of images and the number of stomata we annotated using square bounding boxes. The details on how we arrived at the above decision are as follows.

Although most object detection literature relies on bounding boxes (LIU *et al.*, 2020; DHILLON; VERMA, 2020), some evidence suggests that using rotated ellipses can yield more reliable results in certain domains (WANG *et al.*, 2019). As shown in Figure 20a, this evidence may also be valid for stomata detection, where the shape of a stoma is close to that of a rotated ellipse. However, manual annotation with rotated ellipses requires adjusting five parameters per stoma (Figure 20a), while rectangular and square bounding box annotations require four (Figure 20b) and three parameters (Figure 20c), respectively.

In practice, the time it takes to create an annotation is directly proportional to the number of parameters we need to adjust. This time difference becomes evident when the annotation occurs under unfavorable imaging conditions (e.g., the stoma is partially outside the image limits or in a blurred region) and when the number of stomata to annotate is large. Since our study includes up to 350 stomata per image, we used square bounding boxes. Consequently, we manually annotated 77202 stomata within 1703 microscopic images (see Table 3).

As shown in Figure 20c, each square bounding box annotation depends on three parameters: the pixel coordinates of its top left corner and its side length. However, as the stomata within an image are similar in size, we managed to speed up the annotation process by using only bounding boxes of equal side length. Therefore, for each image, the annotation process focused on adjusting two parameters instead of three. Finally, we corrected the size of the bounding

Table 3 – **Datasets used to train and evaluate** StomaDet. H: holdout, P: predefined, N_{img}: number of images, N_{sp}: number of species, N_{stoma}: average number of stomata per split.

Dataset	Dataset Splits	Training Set			Test Set			Magnification	Sample Preparation Technique	Imaging Method
		N_{img}	N_{sp}	N_{stoma}	N_{img}	N_{sp}	N_{stoma}			
`SimoneDB`	H(5)	58	4	7863	58	4	7791			
`Ctenanthe` (FILHO; DE, 2018)	H(5)	44	1	7093	43	1	7147	50, 100	Direct Isolation	Reflected light
`WoodyL` (SILVA *et al.*, 2016)	P(1)/H(5)	95	21	7887	116	32	9410	100, 200	Clear & stain	Transmitted light
`Poplar` (FETTER *et al.*, 2019)	P (1)	24	1	825	60	1	2074	400	Nail Polish	DIC
`Ginkgo` (BARCLAY; WING, 2016)	P (1)	120	1	1431	64	1	1099	200	Lamina Peel	SEM
`CuticleDB` (BARCLAY *et al.*, 2007)	P (1)	140	137	3596	252	231	6149	400	Clear & stain	Brightfield
`USNM/USBG` (FETTER *et al.*, 2019)	P (1)	239	103	5138	390	116	9699	100, 200, 400	Nail Polish	DIC, SEM
`StomaDB` (all datasets)	P(1)/H(5)	720	268	33833	983	386	43369	–	–	–

Source: Research data.

boxes that needed it.

The amount of manually annotated images per dataset can be calculated by summing the N_{img} values from the “Training set” and “Test set” columns in Table 3. In general, the rate of annotated images is not very high in many datasets. For example, we only annotated $N_{\text{img}} = 140$ of the 678 training images in `CuticleDB`. Two main reasons explain the above situation. (i) The dataset had many low-quality images that we removed (e.g., stomata are in large blurred regions). (ii) Because manual annotation is still a laborious task, we preferred diversity to quantity. Therefore, datasets that include fewer plant species or fewer magnification factors (e.g., 50x, 100x, 200x) are less likely to end up with a high proportion of annotated images.

– *Details of each stomata dataset*

This thesis performs experiments on the seven datasets listed in Table 3. These datasets contain microscopic images of leaf surfaces and involve different plant species, sample preparation techniques, and imaging methods. Figure 21 presents one or two exemplary images per dataset. For better visualization of the stomata, these exemplary images display relatively small cropped regions rather than full-size microscopic images.

In general, microscopic images within a dataset share similar global characteristics, and images between datasets do not. For example, as shown in Figure 21e and Figure 21f, `WoodyL` images are predominantly red, while those from `SimoneDB` are mainly blue (see Figure 21a and Figure 21b). However, these exemplary images also reveal that, despite sharing the same coloration, epidermal surfaces are often quite different between plant species. Below is a brief description of each stomata dataset.

- **`SimoneDB`**. It contains 15654 stomata distributed in 116 microscopic images from four plant species. Since this dataset does not provide any predefined *split*, we created five *splits* using the repeated holdout validation method with $p = 50\%$. We stratified every *split* by plant species to ensure that all training and test sets have approximately the same number of stomata.

- `Ctenanthe` (FILHO; DE, 2018). It consists of 87 microscopic images containing a total of 14240 stomata. We generated five *splits* using the stratified holdout validation method with $p = 50\%$. In this case, the term "stratified" means that both the training and test sets ended up with approximately the same amount of 50x and 100x microscopic images. All images belong to the abaxial side of *Ctenanthe Oppenheimiana* leaf surfaces. Figure 21c and Figure 21d show cropped images at 100x and 50x magnification, respectively. These leaves come from plants, which at an early stage of development, were placed in controlled growth chambers and subjected for 60 days to the following conditions: relative humidity of 55%, photon flux density of $150\mu mol/m^2$, and hydration regimen of 50ml of H_2O per day. Additionally, half of the plants were subjected to 4-hour photoperiods and the other half to 24-hour photoperiods. The sample preparation involved mounting $0.5cm^2$ sections of live *Ctenanthe* leaves on glass slides. These sections were then hydrated with distilled water and covered with glass coverslips. Finally, the abaxial surface of each leave were digitalized using the Axio-Lab light microscope from Zeiss. We refer the reader to (FILHO; MACHICAO; BRUNO, 2017) for more details concerning the sample preparation and imaging methods used in this dataset.

- `WoodyL` (SILVA *et al.*, 2016). It contains 17297 stomata spread across 211 microscopic images of abaxial leaf surfaces from 32 woody species. We refer the reader to (SILVA *et al.*, 2017) for details of the protocol followed to create this dataset. Two exemplary images are shown in Figure 21e and Figure 21f. We rearranged this dataset by first partitioning it into 32 subsets, where each subset includes all images of a single woody species. Then, subsets with at least six elements formed the first group, and the remaining subsets formed the second group. The first group ended up with about 66% of the woody species and concentrated more than 85% of the images. The two groups followed different splitting strategies. In the first group, we created five *splits* using the stratified holdout validation method with $p = 50\%$, where each *split* received about the same number of images per woody species. By contrast, the images from the second group formed a predefined *split* of only one test set. Then, we generated five new *splits* by appending the images from the predefined *split* to every randomly generated *split*. Therefore, Table 3 shows that `WoodyL` has, on average, more samples in its test set than in its training set.

- `Poplar`. This dataset contains microscopic images of the abaxial and adaxial sides of *Populus Balsamifera* leaves. We refer the reader to (FETTER *et al.*, 2019) for details on the sample preparation technique and the imaging method used in this dataset. The original version of this dataset includes a predefined *split* with 3123 training images and 175 test images. Since it is not possible to manually annotate all the images, we selected some of them. Indeed, as this dataset only involves one plant species and one magnification (x400), all images are visually similar to the cropped image shown in Figure 21g. Therefore, we annotated 24 images from the training set and 60 images from the test set.

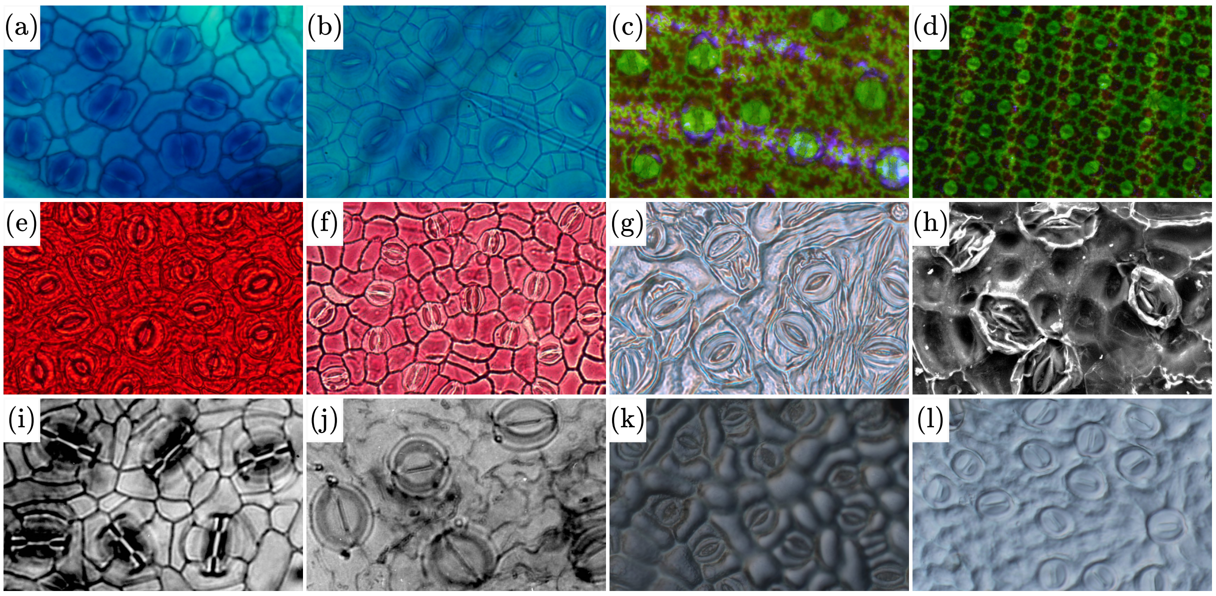
- **`Ginkgo`**. This dataset contains microscopic images of *Ginkgo Biloba* leaf surfaces at x200 magnification. The unannotated version of `Ginkgo` provides a predefined *split* with 408 training images and 200 test images. The sample preparation technique and the imaging method used in this dataset are described in (BARCLAY; WING, 2016). Additionally, as shown in Figure 21h, `Ginkgo` images possess unique characteristics that make them stand out from images in other datasets. In total, we manually annotated 120 images from the training set and 64 images from the test set.

- **`CutibleDB`** (BARCLAY *et al.*, 2007). The original dataset (without annotations) includes a predefined *split* of 678 training images and 696 test images. Each image shows optical microscopy of adaxial and abaxial plant cuticles placed side by side. Besides, unlike previous datasets, `CutibleDB` involves many different plant species, with almost one microscopic image per plant species. The annotation process consisted of the following steps. (i) We discarded poor-quality microscopic images as in previous datasets. (ii) We removed the adaxial side of each image because it generally exhibits poorly visualized stomata. (iii) We randomly selected 140 images from the training set and 252 images from the test set for manual annotation. Given the high number of plant species in this dataset, we tried to annotate as many images as possible, especially in the test set. Therefore, the proportion of manually annotated images in the training set is around 20%, while in the test set, it is 36%. Figure 21i and Figure 21j show two exemplary images from this dataset.

- **`USNM/USBG`** (FETTER *et al.*, 2019). This dataset also includes a large number of plant species, although not as many as `CuticleDB`. The unannotated version of `USNM/USBG` has 409 training images and 696 test images. As in the above datasets, we first discarded all images where stomata are poorly visualized. We randomly selected 239 training images and 390 test images from the original dataset for manual annotation. Therefore, the proportion of annotated images is 58% in the training set and 56% in the test set. Figure 21k and Figure 21l show two exemplary images from this dataset.

- **`StomaDB`**. It has 77202 annotated stomata distributed over 1703 microscopic images of leaf surfaces. `StomaDB` resulted from merging the seven datasets in Table 3. Since the merging process's main task was to obtain the *splits* for `StomaDB`, we arranged the input datasets into two groups according to their number of *splits*. More specifically, datasets with one *split* went to the first group, and those with five *splits* went to the second group. Next, we obtained two temporary datasets ($\texttt{StomaDB}_A$ and $\texttt{StomaDB}_B$) by merging each group's datasets. In more detail, $\texttt{StomaDB}_A$ acquired one *split* by concatenating the individual *splits* provided by `Poplar`, `Ginkgo`, `CuticleDB`, and `USNM/USBG`. Additionally, $\texttt{StomaDB}_B$ obtained five splits, where its *i*-th *split* resulted from concatenating the *i*-th

Figure 21 – **Exemplary microscopic images of the leaf epidermis of 11 species.** (a) *Hymenaea Courbaril* (`SimoneDB` dataset), (b) *Schizolobium parahyba* (`SimoneDB` dataset), (c-d) *Ctenanthe Oppenheimiana* (`Ctenanthe` dataset), (e) *Ilex affinis* (`WoodyL` dataset), (f) *Symplocos nitens* (`WoodyL` dataset), (g) *Populus balsamifera* (`Poplar` dataset), (h) *Ginkgo biloba* (`Ginkgo` dataset), (i) *Diatenopteryx sorbifolia* (`CuticleDB` dataset), (j) *Diospyros nicaraguensis* (`CuticleDB` dataset), (k) *Cinnamomum camphora* (`USNM/USBG` dataset), (l) *Carya glabra* (`USNM/USBG` dataset).



Source: Research data.

splits of `SimoneDB`, `Ctenanthe`, and `WoodyL` ($1 \leq i \leq 5$). Finally, `StomaDB` obtained five *splits* by concatenating each *split* of $\texttt{StomaDB}_B$ with a copy of the *split* of $\texttt{StomaDB}_A$. Given the different imaging methods, sample preparation techniques, and plant species involved, `StomaDB` has enough data to train and evaluate solid predictive models.

4.2.4 A dataset for multi-class pollen detection

This thesis proposes predictive models that address the task of multi-class pollen detection. In more detail, given a microscopic image, the task is to locate and classify the pollen grains within the image. Therefore, unlike stomatal detection, predictive models for pollen detection must not only generate precise bounding boxes but also assign the correct pollen type to each of them. This thesis considers six different pollen types. Table 4 shows the name and family of each pollen type.

– A little context in pollen research

Palynology is an interdisciplinary science that focuses on the study of pollen and other bioparticles. In this context, pollen is a non-living type of bioaerosol that plays a crucial role in the development and evolution of ecosystems (FRÖHLICH-NOWOISKY *et al.*, 2016). Therefore,

Table 4 – **Pollen dataset summary**. Corylus and Betula have significantly more samples than the other pollen types.

Pollen Genus	Family	# Images	# Pollen grains
Betula	Betulaceae	20	138
Carpinus	Betulaceae	26	29
Corylus	Betulaceae	25	194
Fagus	Fagaceae	30	34
Quercus	Fagaceae	36	47
Salix	Salicaceae	27	59

Source: Research data.

the study of different pollen types has led to multiple applications, such as allergy diagnosis (D'AMATO *et al.*, 2007; BARBER *et al.*, 2008; PABLOS *et al.*, 2016), climate change analysis (ZISKA *et al.*, 2011; D'AMATO *et al.*, 2015), palaeoclimatic reconstruction (AMOROSI *et al.*, 2004; BERTINI, 2010; MAGRI *et al.*, 2017) and pesticide monitoring during agronomic seasons (MCART *et al.*, 2017; BÖHME *et al.*, 2018). In general, these applications rely on the correct identification of pollen grains in microscopic images. However, since manual identification of pollen types is time-consuming, automated approaches have been proposed throughout the years. We refer the reader to (HOLT; BENNETT, 2014) for a review of proposed automated methods from 1997 to 2013.

– Manual annotation of pollen grains

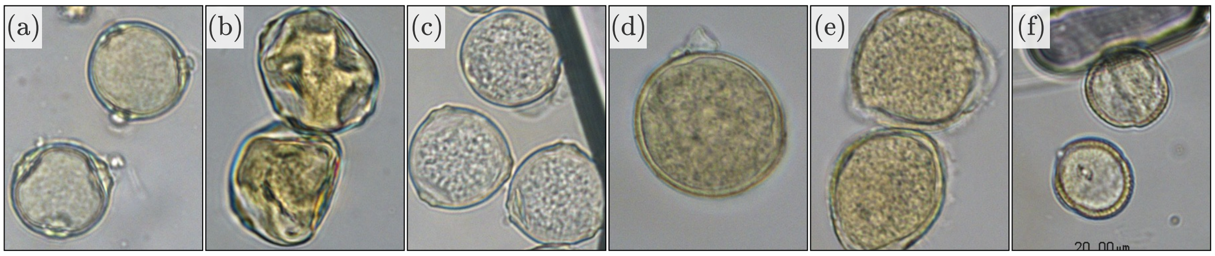
As explained in subsection 4.2.3, a valid training sample consists of an image (I) and a set of annotations (G). Therefore, since we received a dataset consisting of 164 microscopic images without annotations, we manually annotated them using square bounding boxes. We opted for the above annotation type for the following reasons. (i) Given that pollen grains are nearly spherical (see Figure 22), they can fit well in bounding boxes with aspect ratios close to 1:1. (ii) Besides, manual annotation processes are much faster with square bounding boxes than with their rectangular equivalents because the former requires one less parameter to adjust than the latter (see Figure 20).

Therefore, within the framework of a pollen detection task, each annotation ($g \in G$) is described by four parameters $\langle g_l, g_x, g_y, g_s \rangle$, where g_l is its pollen type, (g_x, g_y) indicates the coordinates of its top left corner, and g_s is its side length.

– Details of the pollen dataset

This dataset consists of 501 pollen grains distributed over 164 microscopic images. These images are (1280×960) in size, and there is at least one pollen grain per image. Besides, this dataset considers the six tree pollen types listed in Table 4. These pollen types are often found in

Figure 22 – **Exemplary microscopic images of pollen grains of six tree genus.** (a) Betula, (b) Carpinus, (c) Corylus, (d) Fagus, (e) Quercus, (f) Salix.



Source: Research data.

air samples and are directly related to a significant number of type I hypersensitivity reactions (D'AMATO *et al.*, 2007; PABLOS *et al.*, 2016; ŠAULIENĖ *et al.*, 2016).

A relevant aspect of this dataset is that each microscopic image contains pollen grains that belong to the same pollen type. Therefore, Table 4 also shows the number of images and pollen grains associated with each pollen type. Finally, since this dataset does not provide predefined *splits*, we generated ten random *splits* using the stratified holdout validation method with $p = 50\%$.

4.3 Evaluation Metrics

Evaluation metrics play a fundamental role in discriminating between solid and weak predictive models (Raschka, 2018). Within the framework of this thesis, each evaluation metric is a function that receives (i) model predictions and (ii) target values; and returns a performance score. In general, this performance score is a real number, where the higher the score, the better the model. In this section, we first introduce the general evaluation scheme and then describe the evaluation metrics we used in the experiments.

4.3.1 General Evaluation Scheme

Algorithm 1 presents the general evaluation scheme for estimating the average performance of a predictive model (M) when given a dataset (B) and an evaluation metric ($\mathcal{E}$). In more detail, as shown in lines 5–8 of Algorithm 1, M follows training, prediction, and evaluation processes with each *split* of B. Eventually, the evaluation scheme returns the average of the partial scores generated by $\mathcal{E}$ (line 11).

In addition to $\mathcal{E}$, Algorithm 1 shows two other relevant functions: TRAINMODEL and PREDICT. Since these functions vary from one proposed predictive method to another, they will be detailed in later chapters.

Algorithm 1 – General Evaluation Scheme

```
 1: procedure GenEvaluation($M, B, \mathcal{E}$)                ▷ Model $M$, dataset $B$, and eval metric $\mathcal{E}$
 2:     $b \leftarrow 0$
 3:     $c \leftarrow 0$
 4:     for all $(B_{\text{train}}, B_{\text{test}}) \in B$ do                    ▷ For all splits in $B$
 5:         $\widehat{M} \leftarrow$ TrainModel($M$, $B_{\text{train}}$)              ▷ $\widehat{M}$ is the trained model
 6:         $(\mathbf{I}_{\text{test}}, Y_{\text{test}}) \leftarrow B_{\text{test}}$        ▷ Separate $B_{\text{test}}$ into images and target values
 7:         $\widehat{Y}_{\text{test}} \leftarrow$ Predict($\widehat{M}, \mathbf{I}_{\text{test}}$)           ▷ $\widehat{Y}_{\text{test}}$ has the model predictions
 8:         $b \leftarrow b + \mathcal{E}(\widehat{Y}_{\text{test}}, Y_{\text{test}})$                ▷ Performance estimation
 9:         $c \leftarrow c + 1$
10:     end for
11:     return $b/c$                                              ▷ Average performance
12: end procedure
```

4.3.2 Evaluation metrics for classification tasks

In classification tasks, most evaluation metrics compute a confusion matrix by comparing predicted class labels (model predictions) against target class labels (HOSSIN; SULAIMAN, 2015). For instance, Table 5 shows the confusion matrix for a binary classification task ($n_c = 2$), where n_c represents the number of classes, $\mathtt{tp}$ and $\mathtt{tn}$ denote the number of correctly classified samples, and $\mathtt{fn}$ and $\mathtt{fp}$ denote the number of misclassified samples.

Table 5 – **Confusion matrix for binary classification tasks.** There are two classes: the positive and negative class.

	Target positive class	Target negative class
Predicted positive class	# true positives ($\mathtt{tp}$)	# false positives ($\mathtt{fp}$)
Predicted negative class	# false negatives ($\mathtt{fn}$)	# true negatives ($\mathtt{tn}$)

Source: Elaborated by the author.

Several metrics can be computed using the confusion matrix above. Some of them are briefly described below.

- **Accuracy rate ($\mathtt{Acc}$).** It measures the ratio of correct predictions over the total number of predictions. It can be computed as follows: $\mathtt{Acc} = \frac{\mathtt{tp}+\mathtt{tn}}{\mathtt{tp}+\mathtt{tn}+\mathtt{fp}+\mathtt{fn}}$.

- **Precision ($\mathtt{Pr}$).** It measures the ratio of correct predictions in the positive class over the total number of positive predictions. It has the following equation: $\mathtt{Pr} = \frac{\mathtt{tp}}{\mathtt{tp}+\mathtt{fp}}$.

- **Recall ($\mathtt{Re}$).** It measures the ratio of correct predictions in the positive class over the total number of positive samples. It can be computed as follows: $\mathtt{Re} = \frac{\mathtt{tp}}{\mathtt{tp}+\mathtt{fn}}$.

- **False positive rate ($\mathtt{FPR}$).** It measures the ratio of false positives over the total number of negative samples. It can be computed as follows: $\mathtt{FPR} = \frac{\mathtt{fp}}{\mathtt{tn}+\mathtt{fp}}$.

- **F1-Score ($F1$).** It is the harmonic mean between the precision and recall values. It has the following equation: $F1 = \frac{2*Pr*Re}{Pr+Re}$.

- **Area under the ROC curve ($AUC\text{-}ROC$).** Unlike the previous metrics, $AUC\text{-}ROC$ works with estimated probabilities rather than predicted labels. Each probability p represents the chance of a sample belonging to the positive class. Therefore, given a threshold T, each sample is classified as positive if $p > T$, and negative otherwise. Since the number of positive predictions tends to decrease as T increases, $AUC\text{-}ROC$ measures how much the recall and FPR results vary with respect to different values of T. In general, $AUC\text{-}ROC$ can take values between 0 and 1, where 0.5 indicates random predictions, zero symbolize predictions that are always wrong, and one indicates predictions that are always correct.

The above evaluation metrics can be extended to classification tasks with $n_c > 2$ classes by constructing $n_c \times n_c$ confusion matrices. In this setting, every class has its own tp, fp, and fn. For example, a *true positive* for the i-th class occurs if both the predicted and target labels belong to that class. A *false positive* for the i-th class occurs if only the predicted label belongs to the i-th class, while a *false negative* for the i-th class occurs if only the target label belongs to the i-th class.

After calculating the value of tp, fp, and fn for each class, the precision, recall, and F1-score calculation follows the same formulas as in the binary classification case. The results by class are then averaged to obtain a single value per evaluation metric. Finally, the accuracy metric is equal to the number of correct predictions (tp of all classes) over the sum of all the elements of the confusion matrix.

4.3.3 Evaluation metrics for detection tasks

Evaluating the performance of predictive models for object detection involves comparing sets of ground-truth bounding boxes against predicted ones. This comparison returns three integer numbers for each foreground class: number of true positives (tp), number of false positives (fp), and number of false negatives (fn).

A *true positive* occurs whenever a ground-truth bounding box and a predicted one meet the following criteria: (i) they belong to the same class, and (ii) their *Intersection-over-Union* (IoU) is equal to or greater than the threshold t_{IoU}. In any other case, the predicted and ground-truth bounding boxes are, respectively, counted as fp and fn.

In general, we can correctly assess the size and location of every predicted bounding box by setting $t_{\text{IoU}} \geq 0.5$. Indeed, as proposed in (Lin *et al.*, 2014), studying predictive models at various t_{IoU} is helpful for identifying those that predict the most accurate bounding boxes. Once tp, fp, and fn are computed for each foreground class, they become essential pieces of the following evaluation metrics.

- **Precision ($\mathtt{Pr}$), Recall ($\mathtt{Re}$), and F1-score ($\mathtt{F1}$).** In detection tasks with a single foreground class ($n_c = 1$), these metrics have the same formulas as their equivalents defined for classification tasks (subsection 4.3.2). In the case of $n_c \geq 2$, these metrics are computed per class and then averaged.

- **Mean Average Precision ($\mathtt{mAP}$).** It can take values from zero to one (or 0% to 100%), where one indicates that the detector can perfectly locate and classify the objects within an image. In more detail, the steps to compute $\mathtt{mAP}$ are as follows. (i) Generate the precision vs. recall curve for every foreground class. (ii) At each curve, compute the 101-point interpolated Average Precision ($\mathtt{AP}$). (iii) Finally, take the mean of the computed $\mathtt{AP}$s. Indeed, $\mathtt{mAP}$ has been used in popular papers, such as (He *et al.*, 2017; Redmon; Farhadi, 2018). Additionally, the value of $t_{\mathtt{IoU}}$ is used as a suffix of $\mathtt{mAP}$ to distinguish between $\mathtt{mAP}$ results computed with different $t_{\mathtt{IoU}}$ values. For example, for $t_{\mathtt{IoU}} = 0.5$ and $t_{\mathtt{IoU}} = 0.75$, the corresponding abbreviations are $\mathtt{mAP}_{0.5}$ and $\mathtt{mAP}_{0.75}$, respectively.

4.4 Final considerations

This chapter presented the datasets and the evaluation metrics we will use throughout the experimental part of this thesis. In total, we consider twenty-two datasets, fourteen of which are for image classification. Specifically, the fourteen datasets address the task of texture recognition. Meanwhile, the remaining eight datasets pertain to object detection tasks. Indeed, seven of eight datasets will deal with stomata localization tasks, while the remaining one will deal with grain pollen detection.

We also introduced the concept of split as the division of a dataset into training and test sets. Next, we discussed the most relevant splitting strategies, including the stratified k-fold cross-validation and the stratified m-repeated hold-out validation. Finally, in addition to describing the functioning of some evaluation metrics, we presented the general evaluation scheme as a subroutine applicable to both image classification and object detection tasks.

Part II

Experiments

CHAPTER

5

CNN-BASED FEATURE EXTRACTORS VS. HAND-ENGINEERED METHODS FOR TEXTURE ANALYSIS

5.1 Initial considerations

The previous chapters of this thesis covered the theoretical background of CNNs, from fundamental definitions, such as tensors and activation maps, to advanced topics, such as different types of TL strategies. Additionally, as explained in section 2.1, CNN-based methods mainly differ from traditional hand-engineered methods in the approach they take to compute their features. In more detail, hand-engineered methods require a human expert to design features in advance, while CNN-based methods derive they automatically from multiple data observations.

According to (LIU *et al.*, 2019), CNN-based methods can successfully outperform hand-engineered feature extractors in many datasets. Hence, given the target task of gray-level texture recognition, the purpose of this chapter is to present situations where CNN models are not as good as traditional hand-engineered methods. Specifically, this chapter systematically examines several factors using seventeen feature extractors and five gray-level texture datasets. In this context, a feature extractor is represented by either (i) a hand-engineered method or (ii) a CNN-based method derived from the TL strategy that uses pre-trained CNN models as feature extractors (see subsection 3.5.1).

In more detail, we first compare the individual performances of the seventeen feature extractors using a dedicated classifier. Then, we demonstrate that several hand-engineered methods can still be useful when comparing the accuracy results of (i) all pairings of two CNN-based methods, and (ii) all pairings of a CNN-based method with a hand-engineered method. Finally, this chapter analyzes the effect of changing the dedicated classifier and the effects of the so-called "curse of dimensionality" in the performance of high-dimensional feature vectors.

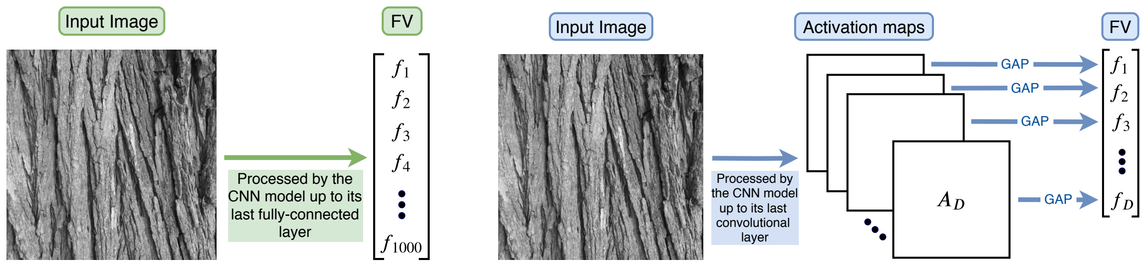
5.2 Target task, datasets and pre-trained models

The prevailing idea of all methods studied here, whether hand-engineered or CNN-based, is to transform an image into a feature vector. Then, each feature vector is fed to a dedicated classifier such as Support Vector Machine (CORTES; VAPNIK, 1995) to solve the following texture task. Given a gray-level image with some unknown texture, the classification task involves assigning the image to one of several predefined classes based on the information contained in its feature vector. Hence, after training and evaluating the dedicated classifier with non-overlapping sets of feature vectors, the result is an accuracy rate that indicates to what extent the task has been accomplished.

We conducted systematic experiments on five texture datasets `Outex013`, `USPtex`, `MBT`, `CUReT_Q4`, and `UIUC` (described in subsection 4.2.2) with the 17 feature extractors listed below.

- Ten CNN-based methods. They result from two TL strategies: GAP-CNN and FC-CNN (introduced in subsection 3.5.1), each with five pre-trained CNN models: VGG-19 (SIMONYAN; ZISSERMAN, 2015), INCEPTION-V3 (SZEGEDY *et al.*, 2015), RESNET-50 (HE *et al.*, 2016), DENSENET-121 (HUANG *et al.*, 2017) and EFFICIENTNET-B1 (TAN; LE, 2019). Figure 23 a high-level view of how GAP-CNN and FC-CNN compute a feature vector from an input image.
- Seven hand-engineered methods. Gray Level Co-occurrence Matrix (GLCM) (HARALICK; SHANMUGAM; DINSTEIN, 1973), Gabor Wavelet (GW) (MANJUNATH; MA, 1996), Local Binary Patterns (LBP) (OJALA; PIETIKAINEN; MAENPAA, 2002), Deterministic Tourist Walks (DTW) (A. R. Backes and W. N. Gonçalves and A. S. Martinez and O. M. Bruno, 2010), the Fractal Dimension (FD) (FLORINDO; BRUNO, 2012), MR8 Texton histograms (TH-MR8) (VARMA; ZISSERMAN, 2005) and Fourier Magnitude Sampling (FMS) (GONZALEZ; WOODS; EDDINS, 2009). We refer the reader to Appendix A for the description of each traditional method.

Figure 23 – **Feature extraction with two TL strategies.** Both FC-CNN and GAP-CNN compute feature vectors that are used to train and evaluate a dedicated classifier.


Input Image
FV
f_1
f_2
f_3
f_4
f_{1000}
Processed by the CNN model up to its last fully-connected layer
Input Image
Activation maps
FV
GAP
GAP
GAP
GAP
A_D
f_1
f_2
f_3
f_D
Processed by the CNN model up to its last convolutional layer

(a) FC-CNN (b) GAP-CNN

Source: Elaborated by the author.

5.3 General experimental settings

In all experiments, we used the general evaluation scheme described in Algorithm 1. In this regard, the evaluation metric $\mathcal{E}$ was the accuracy rate, and the predictive model M comprised both a feature extractor M_{feat} and a dedicated classifier M_{clf}. Additionally, the training and inference processes of Algorithm 1 are detailed in Algorithm 2 and Algorithm 3, respectively.

In more detail, as shown in Algorithm 2, M_{feat} transforms the set of training images I_{train} into a set of feature vectors X_{train} during training. Next, X_{train} follow a feature standardization process which first computes the mean and standard deviation values for each feature, and then transforms X_{train} according to those statistics. Lastly, the dedicated classifier is trained using the transformed set of feature vectors and corresponding labels. During inference, as shown in Algorithm 3, M_{feat} again transforms all test images into a set of feature vectors X_{test}. After that, X_{test} is subjected to a standardization process using the mean and standard deviation values computed during training. Finally, the trained classifier computes the predicted labels.

Algorithm 2 – Feature extractors – Training process

```
 1: procedure TRAINMODEL($M, B_{\text{train}}$)                     ▷ Model $M$ and training *split* $B_{\text{train}}$
 2:     $\langle M_{\text{feat}}, M_{\text{clf}}\rangle \leftarrow M$           ▷ feature extractor $M_{\text{feat}}$ and dedicated classifier $M_{\text{clf}}$
 3:     $\langle \mathbf{I}_{\text{train}}, Y_{\text{train}}\rangle \leftarrow B_{\text{train}}$          ▷ Training images $\mathbf{I}_{\text{train}}$ and labels $Y_{\text{train}}$
 4:     $X_{B_{\text{train}}} \leftarrow M_{\text{feat}}(\mathbf{I}_{\text{train}})$                 ▷ $X_{B_{\text{train}}}$ is the set of feature vectors
 5:     $\langle \widehat{X}_{B_{\text{train}}}, \mathbf{U}, \Sigma\rangle \leftarrow$ STANDARDIZEFVS($X_{B_{\text{train}}}$)   ▷ $\mathbf{U}$ is the set of mean values
 6:                                                       ▷ $\Sigma$ is the set of standard deviation values
 7:     $\widehat{M}_{\text{clf}} \leftarrow$ TRAINCLASSIFIER($M_{\text{clf}}, \widehat{X}_{B_{\text{train}}}, Y_{\text{train}}$)   ▷ $\widehat{M}_{\text{clf}}$ is the trained classifier
 8:     $\widehat{M} \leftarrow \langle M_{\text{feat}}, \widehat{M}_{\text{clf}}, \mathbf{U}, \Sigma\rangle$
 9:     return $\widehat{M}$
10: end procedure
```

Algorithm 3 – Feature extractors – Prediction process

```
1: procedure PREDICT($\widehat{M}, \mathbf{I}_{\text{test}}$)              ▷ Trained model $\widehat{M}$ and test images $\mathbf{I}_{\text{test}}$
2:     $\langle M_{\text{feat}}, \widehat{M}_{\text{clf}}, \mathbf{U}, \Sigma\rangle \leftarrow \widehat{M}$                ▷ See line 8 of Algorithm 2 .
3:     $X_{B_{\text{test}}} \leftarrow M_{\text{feat}}(\mathbf{I}_{\text{test}})$                  ▷ Set of test feature vectors $X_{B_{\text{test}}}$
4:     $\widehat{X}_{B_{\text{test}}} \leftarrow$ STANDARDIZEFVS($X_{B_{\text{test}}}, \mathbf{U}, \Sigma$)        ▷ See line 5 of Algorithm 2
5:     $\widehat{Y}_{\text{test}} \leftarrow \widehat{M}_{\text{clf}}(\widehat{X}_{B_{\text{test}}})$                     ▷ Predicted labels $\widehat{Y}_{\text{test}}$
6:     return $\widehat{Y}_{\text{test}}$
7: end procedure
```

Regarding the splitting strategy, `Outex013`, `USPtex`, `MBT` and `UIUC` followed the stratified ten-fold cross validation methodology (subsection 4.2.1), while `CUReT_Q4` used the six predefined *splits* shown in Figure 19. The specific settings for the hand-engineered and CNN-based methods are given in the following subsections.

5.3.1 Hand-engineered feature extraction details

The parameters values for FMS, LBP, GW, and GLCM were chosen through an exhaustive grid search with images from the `Brodatz` dataset (BRODATZ, 1966). On the other hand, the parameters for DTW and TH-MR8 were respectively taken from (GONçALVES, 2010) and (VARMA; ZISSERMAN, 2005). As for FD, no parameters are needed.

- GLCM. Two distances and four orientations were considered, namely $d = \{1,2\}$ and $\theta = \{0°, 45°, 90°, 135°\}$, leading to 8 GLCMs. From each of them we computed the contrast, energy, correlation and homogeneity, leading to 32 feature values. Additionally, we used GLCM of dimension 128×128, which is referred to as *GLCM 128r*.
- FMS. 140 radial samples and 140 circular samples were used at all experiments. Also, we consider two variants of this method, one that normalize the Fourier spectrum magnitude to the range [0, 1] (*FMS-N*), and the other one who do not use any normalization method (*FMS-U*).
- GW. we set $u_l = 0.01$ and $u_h = 0.4$, and considered the setting of 8 scales and 6 orientations (*GW 8s 6o*), giving rise to feature vectors of 96 elements.
- LBP. histograms with 64 and 256 bins were considered, leading to feature vectors with 64 (*LBP 64f*) and 256 (*LBP 256f*) descriptors respectively.
- FD. the number of features of this method varied automatically according to the size of the input image. For the `CUReT_Q4` dataset, this number was 64, while 68 features were calculated for the `USPtex` and `Outex013` datasets. Finally, 79 features were extracted from the `UIUC` dataset.
- DTW. we chose $R = 4$, $t_0 = 13$, $t_{inc} = 59$, $N_T = 4$, $\mu = 1$, while the number of bins of the resulting histogram was 7. Altogether, 56 descriptors were computed.
- TH-MR8. about 30% of the training set was used for the creation of the *visual dictionary*. In this sense, each class contributes with 10 *textons*. Therefore, the size of the resulting feature vectors is $10 \times N_c$, where N_c corresponds to the total number of classes.

Furthermore, a C++ implementation was written for FMS, GLCM, LBP, GW, and DTW, while FD and TH-MR8 were implemented in MATLAB instead. Since MATLAB is typically slower than C++, more careful optimizations were done for the former, such as using vectorized methods instead of loops.

5.3.2 Implementation details for the CNN-based methods

We performed experiments using two TL strategies: FC-CNN and GAP-CNN. Both strategies are shown in Figure 23. In total, there are ten CNN-based feature extractors as each TL strategy is applied to five pre-trained CNN models. They are named as follows: FC-VGG-19, FC-INCEPTION-V3, FC-RESNET-50, FC-DENSENET-121, FC-EFFICIENTNET-B1, GAP-VGG-19, GAP-INCEPTION-V3, GAP-RESNET-50, GAP-DENSENET-121, and GAP-EFFICIENTNET-B1.

Pre-processing the texture image was required to comply with the input requirements of each pre-trained CNN model. These preprocessing steps vary across the CNN models and the deep learning libraries. In the experiments, we used the pre-trained CNN models from the PyTorch library (PASZKE *et al.*, 2017). Therefore, each gray-level texture image I undergoes to the following preprocessing steps:

1. *Range scaling.* The pixel values of I, originally in the range [0, 255], are linearly re-scaled to the interval of [0, 1].

2. *Fixed spatial size.* Then, the image I is resized to a predefined spatial resolution ($S \times S$) using bilinear interpolation. The term S is the same as that used by the CNN model when it was trained in the source domain: $S = 299$ for INCEPTION-V3, $S = 240$ for EFFICIENTNET-B1, and $S = 224$ for the remaining CNN models (VGG-19, RESNET-50 and DENSENET-121).

3. *Three-channel conversion.* By definition, I is a one-channel gray-level image. Therefore, I is converted into a "color" image by copying its content into three channels.

4. *Image standardization.* Finally, we applied a per-channel standardization process of the form $I^{(i)} = \frac{I^{(i)} - \mu^{(i)}}{\sigma^{(i)}}$, where i is the channel index, $\mu^{(i)}$ is the i-th value from $\mathbf{U} = \{0.485, 0.456, 0.406\}$, and $\sigma^{(i)}$ is the i-th value from $\mathbf{\Sigma} = \{0.229, 0.224, 0.225\}$. Both $\mathbf{\Sigma}$ and $\mathbf{U}$ were collected from the PyTorch documentation (PASZKE *et al.*, 2017).

We used the PyTorch 1.1 library (PASZKE *et al.*, 2017) for the overall deep learning support and its TorchVision 0.3 component to import four pre-trained CNN models: VGG-19, INCEPTION-V3, RESNET-50 and DENSENET-121. EFFICIENTNET-B1, not yet included in the official PyTorch repository yet, so we used an alternative PyTorch based library for this pre-trained CNN model[1].

5.4 Experiments

The experiments were organized in five parts. For each part, the feature extractors were systematically compared for what concerns a different aspect. In the first part CNN-based methods and hand-engineered methods were compared in terms of their accuracy rates and

[1] Github repository for the PyTorch implementation of EFFICIENTNET-B1: <https://github.com/rwightman/pytorch-image-models>

feature extraction times, where GAP-DENSENET-121, GAP-EFFICIENTNET-B1 and FC-EFFICIENTNET-B1 achieved the highest accuracy rates at the five analyzed datasets.

The second part suggests the existence of a strong correlation between the features computed by different CNN-based methods as no significant accuracy improvements were typically obtained by their combinations. On the other hand, the third part reveals that some hand-engineered methods such as *FMS-N*, *FMS-U* and *LBP 256f* are good options to be used in combination with CNN-based methods.

The fourth part focuses on GAP-DENSENET-121, where different classifiers were used in the experiments, showing that in some specific situations involving the SVM-L classifier, GAP-EFFICIENTNET-B1 can successfully compete with its hand-engineered counterparts at improving the accuracy rates of GAP-DENSENET-121. Finally, the fifth part shows that slightly better accuracy rates were achieved at the analyzed datasets by including a dimensionality reduction phase in the evaluation scheme.

5.4.1 CNN-based methods vs. Hand-Engineered (HE) methods

In this experiment, we used the general experimental settings and evaluation scheme described in section 5.3 with the SVM-RBF classifier. In this sense, Table 6 compares nine hand-engineered feature extractors and ten CNN-based methods in terms of (i) accuracy rates (%) for different datasets, (ii) number of features computed per input image, and (iii) the average feature extraction time (FE TIME).

Regarding the hand-engineered methods, TH-MR8 achieved the highest accuracy rates on the `UIUC` and `CUReT_Q4` datasets, while LBP (*LBP 64f* or *LBP 256f*) reached the best performance for `Outext013` and `MBT`. In the `USPtex` dataset, *GW 8s 6o* outperformed the other methods. In general, *FMS-N* had relatively good performances on `MBT`, `CUReT_Q4`, and `USPtex`. Lastly, the accuracy rates obtained with DTW, FD and *GLCM 128r* were mostly lower than the ones obtained by the other hand-engineered methods. On the other hand, the accuracy rates achieved by the ten CNN-based methods were significantly better than those obtained by their hand-engineered counterparts, especially for the `USPtex` and `CUReT_Q4` datasets. When comparing the five FC-CNN methods, the accuracy rates obtained by FC-EFFICIENTNET-B1 were always higher than the ones reached by the others. As for the five GAP-CNN methods, GAP-DENSENET-121 achieved the highest accuracy rates at all datasets.

Regarding the comparison between FC-CNN and GAP-CNN methods with respect to the same CNN model, DENSENET-121 is the only CNN model that always achieved higher accuracy rates as GAP-DENSENET-121 than as FC-DENSENET-121. Indeed, GAP-DENSENET-121 obtained by far the best results among the ten feature extractors, while the second place is disputed between FC-EFFICIENTNET-B1 and GAP-EFFICIENTNET-B1.

About the number of features computed per input image, most hand-engineered methods

Table 6 – **Comparison between different feature extractors.** In bold, the two highest accuracy rates (%) per dataset, one given to a CNN-based method and the other given to a hand-engineered method. (*) The feature extraction time of TH-MR8 cannot be estimated in the same fashion as the other feature extractors, since it depends not only on the resolution of the image but also on the number of classes found in the analyzed dataset.

	Accuracy rates (%) per dataset						
Feature Extractor	`UIUC`	`Outex013`	`USPtex`	`MBT`	`CUReT_Q4`	# Features	FE Time (ms)
DTW	83.6 ± 3.2	72.4 ± 3.4	67.3 ± 4.2	61.1 ± 4.8	60.5 ± 4.1	56	440.3 ± 11.6
FD	79.5 ± 4.5	74.6 ± 3.3	67.7 ± 5.6	75.2 ± 2.8	64.9 ± 2.0	64 – 79	31.1 ± 2.0
FMS-N	73.0 ± 6.5	84.4 ± 2.3	78.5 ± 5.2	86.7 ± 2.6	77.1 ± 1.7	280	3.9 ± 0.3
FMS-U	75.0 ± 5.7	77.7 ± 2.3	79.3 ± 6.5	89.2 ± 2.4	77.9 ± 2.2	280	3.6 ± 0.2
GLCM 128r	79.1 ± 3.8	78.9 ± 2.7	78.3 ± 6.3	80.2 ± 3.5	43.5 ± 13.4	32	5.4 ± 0.2
GW 8s 6o	92.0 ± 3.7	79.6 ± 2.5	**87.2 ± 5.4**	91.1 ± 1.6	61.7 ± 11.3	96	401.3 ± 15.0
LBP 64f	78.3 ± 6.0	**85.1 ± 1.7**	86.9 ± 6.1	91.3 ± 2.1	63.6 ± 17.6	64	5.9 ± 0.4
LBP 256f	80.7 ± 4.2	83.0 ± 2.3	86.1 ± 6.7	**91.4 ± 1.3**	65.7 ± 16.4	256	5.9 ± 0.5
TH-MR8	**95.5 ± 2.5**	82.3 ± 1.4	84.3 ± 2.6	85.4 ± 2.2	**79.9 ± 2.1**	250 – 1920	(*)
FC-VGG-19	98.6 ± 1.1	84.2 ± 1.3	93.8 ± 2.4	89.0 ± 1.4	89.8 ± 1.2	1000	689.8 ± 10.8
FC-Inception-V3	98.9 ± 0.9	85.2 ± 3.3	95.5 ± 2.0	92.7 ± 1.6	93.4 ± 1.1	1000	225.0 ± 6.2
FC-ResNet-50	99.3 ± 0.6	87.6 ± 2.7	96.3 ± 1.9	92.2 ± 1.0	94.2 ± 1.2	1000	147.2 ± 4.8
FC-DenseNet-121	99.5 ± 0.5	88.2 ± 2.4	96.4 ± 1.3	90.7 ± 1.9	93.8 ± 0.8	1000	141.8 ± 4.7
FC-EfficientNet-B1	99.6 ± 0.7	89.6 ± 2.0	97.2 ± 1.5	94.8 ± 1.2	95.6 ± 1.0	1000	275.8 ± 8.9
GAP-VGG-19	96.1 ± 1.4	83.3 ± 3.5	94.9 ± 1.9	90.6 ± 1.4	89.4 ± 1.2	512	542.3 ± 17.4
GAP-Inception-V3	98.8 ± 1.1	85.4 ± 2.8	96.4 ± 1.6	91.9 ± 1.2	92.2 ± 1.3	2048	224.1 ± 6.7
GAP-ResNet-50	98.8 ± 1.3	88.8 ± 2.5	97.7 ± 1.6	93.1 ± 1.3	94.2 ± 1.2	2048	146.8 ± 4.6
GAP-DenseNet-121	**99.7 ± 0.5**	**91.4 ± 2.1**	**99.4 ± 0.5**	**95.0 ± 1.0**	**97.3 ± 0.8**	1024	139.7 ± 4.3
GAP-EfficientNet-B1	98.5 ± 1.1	89.9 ± 2.2	97.7 ± 1.4	94.8 ± 2.0	94.9 ± 1.1	1280	274.6 ± 7.3

Source: Elaborated by the author.

shown in Table 6 compute fixed-sized feature vectors. The only two exceptions are FD and TH-MR8, where the length of these vectors depends on the input image size or the number of classes in the analyzed dataset. On the other hand, feature vectors originating from CNN-based methods are typically much longer than the ones from hand-engineered methods, the only exception being TH-MR8, which for `CUReT_Q4` and `MBT` yields feature vectors that are almost as large as those computed by GAP-Inception-V3 and GAP-ResNet-50. Such high-dimensional feature vectors can lead to many problems, such: (i) classifiers usually need substantially longer training times as the number of dimensions grows, and (ii) their accuracy rates might be negatively impacted by the "curse of dimensionality" (see subsection 5.4.5). Indeed, this phenomenon might explain the high accuracy rate achieved by TH-MR8 for the `UIUC` dataset (95.5%) and its much lower performance for `USPtex` (82.3%) and `MBT` (85.4%).

As for the FE Time needed to process an image of size 224×224, it was estimated by running the same experiment 1000 times and averaging its execution times. Also, for a fair comparison, all FE Times were computed on a single CPU core belonging to an eight-core laptop (CPU i7-7000HQ at 2.80 GHz) with a 64-bit Ubuntu 18.04 installed. As expected, hand-engineered methods are generally faster than CNN-based feature extractors. Indeed, when comparing the FE Time of the fastest hand-engineered method (*FMS-U*) with that of the fastest CNN-based method (GAP-DenseNet-121), the former is approximately 40 times faster than the latter. In addition, GAP-DenseNet-121 is both the fastest CNN-based method and the one with the best accuracy rates at the analyzed datasets.

5.4.2 Combining pairs of CNN-based methods

In general, two feature extractors M_1 and M_2 are said to be combined when, for each input image, they combine their computed feature vectors into one. Therefore, $M_1 + M_2$ acts as a single feature extractor that computes feature vectors of length $P + Q$, where P and Q are the lengths of the feature vectors computed by M_1 and M_2, respectively.

Here, we verified how much accuracy improvement can be achieved on every CNN-based feature extractor in Table 6 by combining it with GAP-CNN feature extractors. Table 7 presents the results in three subtables, one for each dataset. Also, we ignored the results for `UIUC` and `USPtex`, since the very high accuracy rates reached in subsection 5.4.1 cannot be improved significantly. Again, we used the same evaluation settings as the previous experiment, where $M = M_1 + M_2$. Additionally, we applied the **McNemar** test to validate whether the accuracy improvements were statistically significant or not. The results for each dataset in Table 7 are presented and discussed below.

- `Outex013`. Table 7a reveals the following insights. (i) GAP-VGG-19 failed to significantly increase the accuracy rates for eight of the nine CNN-based methods analyzed. (ii) Similarly, GAP-INCEPTION-V3 and GAP-RESNET-50 could not increase the accuracy rates for seven and four CNN-based methods, respectively. (iii) Although the accuracy improvements accomplished by GAP-EFFICIENTNET-B1 were mostly statistically significant, it failed to improve the results of GAP-DENSENET-121, which is the CNN-based method with the highest baseline accuracy rate (91.4%). (iv) GAP-DENSENET-121 is the only feature extractor that was able to improve the results of the other nine CNN-based methods evaluated. On the contrary, none of these nine methods substantially improved the results of GAP-DENSENET-121. Essentially, there is no combination of two CNN-based methods that led to an accuracy rate statistically superior than the baseline accuracy rates.
- `MBT`. Compared to `Outex013`, Table 7b shows a larger number of combinations that led to significant accuracy improvements with respect to the baseline accuracy rates. Although GAP-DENSENET-121 had the best baseline accuracy rate, GAP-EFFICIENTNET-B1 achieved the best accuracy improvements for eight of the nine CNN-based methods evaluated. Indeed, not only led combination GAP-DENSENET-121 + GAP-EFFICIENTNET-B1 to the best result on the entire dataset (96.4%), but also it was significantly better than the baseline accuracy rates of both GAP-DENSENET-121 (95.0%) and GAP-EFFICIENTNET-B1 (94.8%).
- `CUReT_Q4`. The power of the McNemar test statistic increases as more samples are involved. Therefore, since `CUReT_Q4` has much more samples than the other datasets, there is a tendency of having more significant results in Table 7c than in the previous datasets. As for GAP-DENSENET-121 (97.3%), it was successfully enhanced when used in combination with either FC-INCEPTION-V3, FC-RESNET-50 or FC-EFFICIENTNET-B1. In this

Table 7 – Accuracy rates (%) achieved by combinations of two CNN-based feature extractors. The row "*Baseline*" shows the accuracy rates obtained with each CNN-based method alone. For each column, the best accuracy improvement is emphasized in bold, while the best accuracy rate for each dataset is highlighted in red bold font. Accuracy rates of combinations that are not significantly better than their corresponding baselines are crossed out. This significance was verified with the McNemar test, using $\alpha = 0.05$. EffNet-B1 is the abbreviation for EfficientNet-B1.

(a) `Outex013`

		FC-CNN					GAP-CNN				
		VGG-19	Inception-V3	ResNet-50	DenseNet-121	EffNet-B1	VGG-19	Inception-V3	ResNet-50	DenseNet-121	EffNet-B1
	Baseline →	84.2 ± 1.3	85.2 ± 3.3	87.6 ± 2.7	88.2 ± 2.4	89.6 ± 2.0	83.3 ± 3.5	85.4 ± 2.8	88.8 ± 2.5	91.4 ± 2.1	89.9 ± 2.2
GAP-CNN +	VGG-19	~~85.4 ± 2.5~~	87.9 ± 4.0	~~88.7 ± 2.1~~	~~88.4 ± 2.2~~	~~90.6 ± 1.7~~	–	86.4 ± 2.8	~~89.4 ± 2.5~~	~~91.6 ± 2.1~~	91.0 ± 1.9
	Inception-V3	87.3 ± 2.7	~~85.6 ± 2.8~~	~~87.9 ± 2.3~~	~~88.4 ± 3.2~~	~~89.5 ± 2.1~~	86.4 ± 2.8	–	~~88.1 ± 2.4~~	~~89.2 ± 2.6~~	~~90.2 ± 2.1~~
	ResNet-50	89.6 ± 1.4	88.3 ± 3.0	88.9 ± 2.2	~~89.6 ± 2.9~~	~~90.3 ± 2.7~~	89.4 ± 2.5	88.1 ± 2.4	–	~~90.4 ± 2.6~~	~~90.5 ± 2.5~~
	DenseNet-121	90.8 ± 2.0	90.4 ± 2.3	**91.2 ± 2.4**	90.7 ± 2.0	**92.3 ± 2.6**	**91.6 ± 2.1**	89.2 ± 2.6	90.4 ± 2.6	–	**92.2 ± 2.5**
	EffNet-B1	**91.0 ± 1.9**	**91.2 ± 2.1**	90.4 ± 2.4	**91.2 ± 2.6**	90.9 ± 2.2	91.0 ± 1.9	**90.2 ± 2.1**	**90.5 ± 2.5**	~~**92.2 ± 2.5**~~	–

(b) `MBT`

		FC-CNN					GAP-CNN				
		VGG-19	Inception-V3	ResNet-50	DenseNet-121	EffNet-B1	VGG-19	Inception-V3	ResNet-50	DenseNet-121	EffNet-B1
	Baseline →	89.0 ± 1.4	92.7 ± 1.6	92.2 ± 1.0	90.7 ± 1.9	94.8 ± 1.2	90.6 ± 1.4	91.9 ± 1.2	93.1 ± 1.3	95.0 ± 1.0	94.8 ± 2.0
GAP-CNN +	VGG-19	90.5 ± 1.5	93.6 ± 1.8	93.5 ± 0.6	93.1 ± 1.5	~~95.2 ± 0.7~~	–	92.6 ± 1.3	93.9 ± 1.1	~~94.9 ± 1.2~~	~~95.4 ± 1.5~~
	Inception-V3	93.3 ± 1.4	~~92.4 ± 1.1~~	93.6 ± 1.6	93.2 ± 1.3	~~94.6 ± 1.3~~	92.6 ± 1.3	–	94.4 ± 1.7	~~93.8 ± 1.4~~	~~94.7 ± 1.4~~
	ResNet-50	93.7 ± 1.1	94.6 ± 1.7	93.2 ± 1.3	94.3 ± 1.4	~~95.1 ± 1.4~~	93.9 ± 1.1	94.4 ± 1.7	–	~~95.1 ± 1.5~~	~~95.3 ± 1.2~~
	DenseNet-121	94.3 ± 1.1	94.5 ± 1.7	95.2 ± 1.3	93.9 ± 1.7	**96.2 ± 1.0**	94.9 ± 1.2	93.8 ± 1.4	95.1 ± 1.5	–	**96.4 ± 1.5**
	EffNet-B1	**95.2 ± 1.2**	**95.6 ± 1.7**	**95.6 ± 1.5**	**95.5 ± 1.7**	~~95.7 ± 1.4~~	**95.4 ± 1.5**	**94.7 ± 1.4**	**95.3 ± 1.2**	**96.4 ± 1.5**	–

(c) `CUReT_Q4`

		FC-CNN					GAP-CNN				
		VGG-19	Inception-V3	ResNet-50	DenseNet-121	EffNet-B1	VGG-19	Inception-V3	ResNet-50	DenseNet-121	EffNet-B1
	Baseline →	89.8 ± 1.2	93.4 ± 1.1	94.2 ± 1.2	93.8 ± 0.8	95.6 ± 1.0	89.4 ± 1.2	92.2 ± 1.3	94.2 ± 1.2	97.3 ± 0.8	94.9 ± 1.1
GAP-CNN +	VGG-19	91.2 ± 1.2	95.0 ± 1.1	94.9 ± 1.0	94.7 ± 1.0	96.1 ± 1.0	–	93.6 ± 1.3	94.8 ± 1.1	~~96.6 ± 0.9~~	95.2 ± 1.1
	Inception-V3	94.5 ± 1.1	~~93.1 ± 1.2~~	95.3 ± 1.0	95.1 ± 1.0	95.9 ± 1.0	93.6 ± 1.3	–	95.5 ± 1.1	~~96.2 ± 1.0~~	95.6 ± 1.1
	ResNet-50	95.3 ± 1.2	95.9 ± 1.0	94.4 ± 1.2	95.8 ± 1.1	96.2 ± 1.1	94.8 ± 1.1	95.5 ± 1.1	–	~~96.6 ± 1.1~~	95.9 ± 1.1
	DenseNet-121	**97.0 ± 0.8**	**97.4 ± 0.8**	**97.4 ± 0.8**	**96.5 ± 0.8**	**97.7 ± 0.8**	**96.6 ± 0.9**	**96.2 ± 1.0**	**96.6 ± 1.1**	–	**97.1 ± 1.0**
	EffNet-B1	96.0 ± 1.0	96.4 ± 1.1	96.3 ± 1.2	96.4 ± 1.0	~~95.7 ± 1.2~~	95.2 ± 1.1	95.6 ± 1.1	95.9 ± 1.1	~~**97.1 ± 1.0**~~	–

Source: Elaborated by the author.

regard, the best result (97.7%) was reached by the combination FC-EfficientNet-B1 + GAP-DenseNet-121.

In general, the results of Table 7 show that the best accuracy rates per dataset are achieved when GAP-DenseNet-121 is combined with either FC-EfficientNet-B1 or GAP-EfficientNet-B1. In subsection 5.4.3, we compared these results with the ones achieved by combining CNN-based methods with hand-engineered ones.

5.4.3 Combining CNN-based methods with HE methods

Here we measured the predictive power improvement that a hand-engineered method can imply when used in combination with a CNN-based method. In this regard, we used the same evaluation settings as in subsection 5.4.2. The results are presented in Table 8, and the highlights are discussed in the following paragraphs.

- `Outex013`: DTW only achieved significant accuracy improvements for two of the ten CNN-based feature extractors evaluated, while FD and *FMS-U* both only have six significant improvements. Regarding *FMS-N* and *GW 8s 6o*, they successfully increased the baseline accuracy rates of all CNN-based feature extractors. The situation of TH-MR8 is worth mentioning, because it is very effective when combined with all CNN-based methods except with GAP-DENSENET-121. This undermines the overall performance of TH-MR8, as GAP-DENSENET-121 is the feature extractor with the most promising baseline accuracy rate (91.4%). Fortunately, five of the eight hand-engineered methods were able to substantially enhance the baseline results of GAP-DENSENET-121, from which *LBP 256f* and *FMS-N* achieved respectively the highest (+1.6%) and second highest (+1.4%) accuracy improvement.

- `MBT`: For this dataset, there are four hand-engineered methods that were able to increase the baseline accuracy rates of all CNN-based feature extractors. TH-MR8 and *GLCM 128r* only achieved five successful accuracy improvements each, while DTW and FD respectively led to seven and nine significant improvements. When analyzing the largest accuracy improvements accomplished for each CNN-based method separately, there is a clear rivalry between *FMS-U* and *LBP-256f*. Although *LBP-256f* surpassed *FMS-U* at five of the ten CNN-based methods, the performance of *FMS-U* is equal to or better than *LBP-256f* in the most promising cases, involving: GAP-DENSENET-121 (95.0%), GAP-EFFICIENTNET-B1 (94.8%), and FC-EFFICIENTNET-B1 (94.8%). Indeed, both *FMS-U* and *LBP-256f* lead to a success rate of 96.8% when combined with GAP-DENSENET-121.

- `CUReT_Q4`: Unlike the previous datasets, the results from combinations involving *LBP 256f* and *GW 8s 6o* are unfavorable. This can potentially be explained by the results in Table 6, where both *LBP 256f* and *GW 8s 6o* obtained poor accuracy rates (65.7% and 61.7%, respectively) for `CUReT_Q4`. On the contrary, both DTW and FD led to significant accuracy improvements for all the CNN-based methods evaluated, despite also obtaining poor accuracy rates in Table 6 (60.5% and 64.9%). Again TH-MR8 increased the baseline accuracy rates of most CNN-based methods, with the exception of GAP-DENSENET-121. *FMS-U* ended up having the most significant accuracy improvements. Indeed, the combination GAP-DENSENET-121 + *FMS-U* achieved an accuracy rate of 98%, which is +0.7% greater than the baseline accuracy rate of GAP-DENSENET-121 (97.3%).

In general, when comparing the accuracy improvements of Table 7 and Table 8 for each dataset, there is a clear advantage of hand-engineered methods over CNN-based methods for what concerns the increase in predictive power of GAP-DENSENET-121. Indeed, out of the

Table 8 – **Comparison between combinations of one CNN-based method and one hand-engineered method.** The row "*Baseline*" shows the accuracy rates (%) obtained by each CNN-based method alone. The highest accuracy rate per column is emphasized in bold and the best result per dataset is highlighted in red bold font. The accuracy rates that are not significantly higher than their respective baselines are displayed as strike-through text. The statistical significance was verified with the McNemar test, using $\alpha = 0.05$. EFFNET-B1 is the abbreviation for EFFICIENTNET-B1.

(a) `Outex013`

	FC-CNN					GAP-CNN				
	VGG-19	INCEPTION-V3	RESNET-50	DENSENET-121	EFFNET-B1	VGG-19	INCEPTION-V3	RESNET-50	DENSENET-121	EFFNET-B1
Baseline →	84.2 ± 1.3	85.2 ± 3.3	87.6 ± 2.7	88.2 ± 2.4	89.6 ± 2.0	83.3 ± 3.5	85.4 ± 2.8	88.8 ± 2.5	91.4 ± 2.1	89.9 ± 2.2
DTW	~~84.8 ± 1.5~~	~~85.7 ± 3.0~~	~~87.9 ± 2.8~~	89.2 ± 2.3	~~89.9 ± 2.2~~	85.1 ± 3.4	~~85.9 ± 2.7~~	~~88.8 ± 2.7~~	~~91.6 ± 2.1~~	~~90.2 ± 2.3~~
FD	85.2 ± 1.5	86.2 ± 2.8	~~88.0 ± 2.4~~	89.4 ± 2.6	~~90.2 ± 2.0~~	86.0 ± 3.4	86.1 ± 2.8	~~89.1 ± 2.5~~	~~91.7 ± 1.9~~	90.4 ± 2.3
FMS-N	87.3 ± 2.2	87.9 ± 2.6	89.9 ± 3.0	90.7 ± 1.8	91.0 ± 2.1	**89.4 ± 2.4**	87.1 ± 2.4	90.0 ± 2.3	92.8 ± 1.9	90.9 ± 2.1
FMS-U	86.5 ± 2.1	87.2 ± 2.9	~~88.2 ± 3.0~~	90.4 ± 2.5	~~90.3 ± 2.1~~	86.9 ± 3.7	87.1 ± 2.6	~~89.2 ± 2.4~~	92.7 ± 2.4	~~90.7 ± 2.3~~
GLCM 128r	85.4 ± 1.9	86.5 ± 3.2	88.5 ± 2.6	89.2 ± 2.5	~~89.9 ± 2.2~~	85.7 ± 3.3	86.0 ± 2.6	~~89.3 ± 2.9~~	92.1 ± 2.2	~~90.4 ± 2.2~~
GW 8s 6o	86.0 ± 1.9	86.5 ± 2.9	89.2 ± 3.0	89.7 ± 1.9	90.5 ± 2.0	86.2 ± 3.8	86.1 ± 2.7	89.6 ± 2.4	92.4 ± 1.7	90.7 ± 2.4
LBP 256f	87.1 ± 1.8	87.6 ± 2.4	89.0 ± 2.4	**91.8 ± 2.4**	**91.4 ± 1.8**	87.9 ± 2.4	87.2 ± 2.5	~~89.7 ± 2.6~~	**93.0 ± 1.8**	**91.4 ± 1.6**
TH-MR8	**89.3 ± 1.2**	**88.2 ± 2.4**	**90.0 ± 2.4**	91.3 ± 2.0	91.2 ± 1.8	87.8 ± 3.0	**87.6 ± 2.5**	**90.9 ± 2.7**	~~92.4 ± 2.2~~	91.3 ± 1.9

(b) `MBT`

	FC-CNN					GAP-CNN				
	VGG-19	INCEPTION-V3	RESNET-50	DENSENET-121	EFFNET-B1	VGG-19	INCEPTION-V3	RESNET-50	DENSENET-121	EFFNET-B1
Baseline →	89.0 ± 1.4	92.7 ± 1.6	92.2 ± 1.0	90.7 ± 1.9	94.8 ± 1.2	90.6 ± 1.4	91.9 ± 1.2	93.1 ± 1.3	95.0 ± 1.0	94.8 ± 2.0
DTW	90.6 ± 1.4	~~93.0 ± 1.9~~	93.2 ± 0.9	91.9 ± 1.6	95.4 ± 1.2	92.1 ± 1.1	92.5 ± 1.1	93.5 ± 1.3	~~95.5 ± 1.2~~	~~95.1 ± 1.9~~
FD	90.8 ± 1.7	93.4 ± 1.6	93.6 ± 1.2	92.6 ± 1.3	95.6 ± 1.2	92.9 ± 0.7	92.5 ± 1.1	93.8 ± 1.2	95.9 ± 0.9	~~95.2 ± 2.0~~
FMS-N	92.5 ± 1.3	94.6 ± 1.4	94.4 ± 1.2	93.1 ± 1.7	95.9 ± 0.9	94.1 ± 1.5	93.2 ± 0.9	94.5 ± 1.3	95.8 ± 0.8	95.8 ± 1.6
FMS-U	92.9 ± 1.2	94.6 ± 1.1	94.8 ± 1.1	93.8 ± 1.6	**96.5 ± 1.5**	93.8 ± 1.1	93.2 ± 1.2	**94.9 ± 1.3**	**96.8 ± 1.2**	**96.1 ± 1.8**
GLCM 128r	90.4 ± 1.3	~~92.8 ± 1.6~~	93.1 ± 0.8	91.8 ± 1.6	~~95.0 ± 1.2~~	91.5 ± 1.0	~~92.1 ± 1.2~~	93.4 ± 1.2	~~95.2 ± 1.2~~	~~95.1 ± 2.0~~
GW 8s 6o	92.0 ± 1.6	93.9 ± 1.5	94.1 ± 1.4	93.1 ± 1.6	95.3 ± 1.3	93.8 ± 1.1	92.5 ± 1.2	94.0 ± 1.3	95.9 ± 1.0	95.3 ± 1.6
LBP 256f	**93.1 ± 1.3**	**94.9 ± 1.6**	**95.2 ± 1.1**	**94.7 ± 1.2**	96.4 ± 1.2	**95.7 ± 0.7**	93.5 ± 1.3	94.6 ± 1.4	**96.8 ± 0.6**	95.9 ± 1.6
TH-MR8	91.9 ± 1.9	93.8 ± 2.0	~~93.3 ± 1.5~~	92.9 ± 1.3	~~94.7 ± 1.3~~	~~91.7 ± 2.1~~	**94.3 ± 1.7**	94.3 ± 1.4	~~94.4 ± 1.5~~	~~94.7 ± 2.0~~

(c) `CUReT_Q4`

	FC-CNN					GAP-CNN				
	VGG-19	INCEPTION-V3	RESNET-50	DENSENET-121	EFFNET-B1	VGG-19	INCEPTION-V3	RESNET-50	DENSENET-121	EFFNET-B1
Baseline →	89.8 ± 1.2	93.4 ± 1.1	94.2 ± 1.2	93.8 ± 0.8	95.6 ± 1.0	89.4 ± 1.2	92.2 ± 1.3	94.2 ± 1.2	97.3 ± 0.8	94.9 ± 1.1
DTW	91.5 ± 1.2	94.5 ± 1.0	94.8 ± 1.1	94.8 ± 0.8	96.1 ± 1.0	91.2 ± 1.2	92.9 ± 1.3	94.5 ± 1.2	97.6 ± 0.8	95.2 ± 1.1
FD	92.2 ± 1.3	94.7 ± 1.0	95.1 ± 1.0	95.1 ± 0.8	96.2 ± 0.9	92.3 ± 1.2	93.0 ± 1.2	94.7 ± 1.1	97.7 ± 0.8	95.4 ± 1.1
FMS-N	93.3 ± 1.1	95.2 ± 0.8	95.5 ± 0.9	95.4 ± 0.7	96.6 ± 0.9	93.8 ± 0.9	93.8 ± 1.0	95.3 ± 1.0	97.7 ± 0.6	96.0 ± 1.0
FMS-U	94.1 ± 1.3	**95.9 ± 1.0**	**96.0 ± 1.1**	**96.1 ± 1.0**	**96.8 ± 0.9**	**94.3 ± 1.3**	**94.4 ± 1.2**	95.6 ± 1.1	**98.0 ± 0.8**	**96.2 ± 1.1**
GLCM 128r	90.8 ± 1.3	94.0 ± 0.9	94.4 ± 1.2	94.1 ± 0.8	95.8 ± 1.0	89.9 ± 1.4	92.5 ± 1.3	94.3 ± 1.2	~~97.3 ± 0.7~~	~~95.0 ± 1.1~~
GW 8s 6o	~~88.8 ± 3.7~~	~~92.1 ± 3.2~~	~~92.9 ± 3.2~~	~~92.0 ± 3.5~~	~~94.1 ± 2.9~~	~~86.3 ± 6.3~~	~~91.8 ± 2.1~~	~~93.9 ± 1.7~~	~~95.6 ± 2.9~~	~~93.9 ± 2.5~~
LBP 256f	91.2 ± 3.1	94.0 ± 1.6	~~93.9 ± 2.1~~	~~93.4 ± 2.0~~	~~95.6 ± 1.5~~	~~88.0 ± 5.5~~	93.4 ± 1.1	94.6 ± 1.4	~~96.9 ± 0.8~~	95.2 ± 1.3
TH-MR8	**94.3 ± 0.9**	95.4 ± 0.6	96.0 ± 0.7	95.3 ± 0.7	96.7 ± 0.8	93.1 ± 1.2	**94.4 ± 0.8**	**95.8 ± 1.0**	~~97.1 ± 0.5~~	**96.2 ± 1.1**

Source: Elaborated by the author.

eight hand-engineered feature extractors studied, *FMS-U* and *FMS-N* demonstrated the most consistent performance, increasing the GAP-DENSENET-121 accuracy rate for all datasets evaluated.

These outcomes suggest that the texture information contained in the Fourier spectrum is of high quality and could potentially be exploited to derive feature extractors that, in combination with GAP-DENSENET-121, produce even higher accuracy rates. Additionally, given the very low FE TIME of *FMS-U* and *FMS-N*, their use is recommended. Another good option is *LBP 256f*, which sometimes leads to higher accuracy improvements than *FMS-U*.

5.4.4 The classifier effect

Here, we evaluated different dedicated classifiers to assess whether we still reach the same conclusions as those reached with SVM-RBF. Additionally, we focused these experiments on GAP-DenseNet-121, since it was identified as the best feature extractor in the preceding experiments. Therefore, Table 9 shows the comparison of accuracy improvements achieved by 16 feature extractors when combined with GAP-DenseNet-121 for the `Outex013`, `MBT` and `CUReT_Q4` datasets, and the SVM-RBF, SVM-L and KNN classifiers. In the case of SVM-RBF, it is the same classifier we used in the previous experiments. SVM-L is also a SVM, but with a linear kernel and $C = 10$ instead, while KNN is a k-nearest neighbor classifier with $k = 1$.

The analysis of Table 9 shows that: (i) for every dataset, SVM-RBF led to slightly better accuracy rates than SVM-L, whereas worst results were always achieved with KNN. (ii) Both SVM-RBF and KNN achieved similarly larger accuracy improvements with hand-engineered methods than with CNN-based feature extractors. (iii) The results for `Outex013` and `MBT` datasets show that the baseline accuracy rates of GAP-DenseNet-121 are harder to improve when KNN is used instead of SVM-RBF or SVM-L. (iv) Regardless of the dataset and classifier, three of the eight CNN-based methods never achieved significant accuracy improvements, so their use is not recommended in any situation. (v) For similar reasons, we advise against the use of GAP-VGG-19, FC-ResNet-50 and GAP-ResNet-50. (vi) As for FC-EfficientNet-B1 and GAP-EfficientNet-B1, they achieved successful accuracy improvements when they were evaluated with SVM-L. (vii) Finally, out of the eight hand-engineered feature extractors, *FMS-U* and *FMS-N* were the ones that led to the most stable results across the different datasets and classifiers.

In summary, the results computed by SVM-L indicate that GAP-EfficientNet-B1 may be used as a CNN-based alternative for the hand-engineered methods. However, the results obtained with SVM-RBF still suggest that hand-engineered methods are more suitable to be used in combination with GAP-DenseNet-121 than CNN-based methods. The possible causes for this will be further explored in subsection 5.4.5.

5.4.5 Using feature selection

The previous parts showed that GAP-DenseNet-121 lead to higher accuracy rates in combination with hand-engineered feature extractors than when combined with CNN-based methods. However, this somehow contradicts Table 6, where most CNN-based methods obtained much higher accuracy rates than their hand-engineered counterparts. For example, Table 6 shows that GAP-DenseNet-121 and GAP-ResNet-50 achieved accuracy rates of 95.0% and 93.1% respectively for `MBT`, while *FMS-N* only achieved 89.2% for the same dataset. Yet, even though the accuracy rate achieved by GAP-ResNet-50 is almost 4% higher than that of *FMS-N*, the combination of GAP-DenseNet-121 + GAP-ResNet-50 reached a non-significant result of 95.1%, while GAP-DenseNet-121 + *FMS-N* achieved a much significantly better accuracy

Table 9 – **Comparison of accuracy rates across different classifiers.** The row "*Baseline*" shows the accuracy rates obtained with GAP-DenseNet-121 for each dataset and classifier. The best accuracy rate per column is shown in bold, while the best result for each dataset is displayed in red bold font. The accuracy rates of combinations that did not significantly improve their respective baselines are shown as strike-through text. The McNemar test was used to verify the statistical significance, with $\alpha = 0.05$. EffNet-B1 is the abbreviation for EfficientNet-B1.

		GAP-DenseNet-121								
		Outex013			MBT			CUReT_Q4		
		SVM-RBF	SVM-L	KNN	SVM-RBF	SVM-L	KNN	SVM-RBF	SVM-L	KNN
	Baseline →	91.4 ± 2.1	90.6 ± 2.4	86.9 ± 1.7	95.0 ± 1.0	94.3 ± 1.4	91.6 ± 0.8	97.3 ± 0.8	97.0 ± 0.9	94.0 ± 0.9
FC-CNN	VGG-19	~~90.8 ± 2.0~~	~~90.6 ± 2.4~~	~~86.2 ± 2.4~~	~~94.3 ± 1.1~~	~~94.3 ± 0.8~~	~~90.2 ± 1.9~~	~~97.0 ± 0.8~~	~~96.8 ± 0.8~~	~~92.7 ± 0.9~~
	Inception-V3	~~90.4 ± 2.3~~	~~90.5 ± 2.7~~	~~84.3 ± 2.7~~	~~94.5 ± 1.7~~	~~94.7 ± 1.4~~	~~90.8 ± 1.2~~	~~97.4 ± 0.8~~	~~97.0 ± 0.8~~	~~93.9 ± 1.0~~
	ResNet-50	~~91.2 ± 2.4~~	~~91.5 ± 1.7~~	~~86.8 ± 1.1~~	~~95.2 ± 1.3~~	~~94.8 ± 1.2~~	~~90.7 ± 1.0~~	97.4 ± 0.8	97.1 ± 0.8	~~93.8 ± 1.0~~
	EffNet-B1	~~92.3 ± 2.6~~	92.1 ± 2.2	~~87.4 ± 2.0~~	96.2 ± 1.0	96.3 ± 0.9	~~92.3 ± 1.4~~	97.7 ± 0.8	97.6 ± 0.7	95.1 ± 1.0
GAP-CNN	VGG-19	~~91.6 ± 2.1~~	~~91.3 ± 2.0~~	~~86.6 ± 1.5~~	~~94.9 ± 1.2~~	~~94.8 ± 1.0~~	~~91.4 ± 1.6~~	~~96.6 ± 0.9~~	97.3 ± 0.9	~~93.9 ± 1.0~~
	Inception-V3	~~89.2 ± 2.6~~	~~89.7 ± 3.2~~	~~83.5 ± 1.6~~	~~93.8 ± 1.4~~	~~94.2 ± 1.0~~	~~89.5 ± 1.4~~	~~96.2 ± 1.0~~	~~96.5 ± 0.9~~	~~92.0 ± 1.0~~
	ResNet-50	~~90.4 ± 2.6~~	~~90.8 ± 2.4~~	~~87.2 ± 1.8~~	~~95.1 ± 1.5~~	~~95.0 ± 1.4~~	~~90.2 ± 2.1~~	~~96.6 ± 1.1~~	97.2 ± 0.9	~~93.7 ± 0.9~~
	EffNet-B1	~~92.2 ± 2.5~~	92.3 ± 2.7	~~87.6 ± 1.6~~	96.4 ± 1.5	**96.7 ± 1.2**	~~92.7 ± 1.3~~	~~97.1 ± 1.0~~	**97.8 ± 0.6**	95.3 ± 0.8
	DTW	~~91.6 ± 2.1~~	91.5 ± 2.6	~~87.4 ± 1.2~~	~~95.5 ± 1.2~~	95.2 ± 1.0	~~92.0 ± 0.8~~	97.6 ± 0.8	97.2 ± 0.8	94.7 ± 0.9
	FD	~~91.7 ± 1.9~~	91.5 ± 2.6	~~87.3 ± 1.5~~	95.9 ± 0.9	95.7 ± 1.1	~~91.9 ± 0.5~~	97.7 ± 0.8	97.3 ± 0.8	94.6 ± 0.9
	FMS-N	92.8 ± 1.9	**92.9 ± 2.0**	**88.7 ± 2.5**	95.8 ± 0.8	95.8 ± 0.9	~~92.2 ± 1.0~~	97.7 ± 0.6	97.4 ± 0.7	95.1 ± 0.8
	FMS-U	92.7 ± 2.4	**92.9 ± 2.4**	~~87.7 ± 1.6~~	**96.8 ± 1.2**	96.5 ± 1.0	93.2 ± 0.9	**98.0 ± 0.8**	97.7 ± 0.9	**95.6 ± 0.9**
	GLCM 128r	92.1 ± 2.2	91.5 ± 3.0	~~87.2 ± 1.7~~	~~95.2 ± 1.2~~	95.0 ± 1.2	~~92.0 ± 0.9~~	~~97.3 ± 0.7~~	~~96.9 ± 0.7~~	94.3 ± 0.9
	GW 8s 6o	92.4 ± 1.7	92.4 ± 2.0	88.2 ± 2.0	95.9 ± 1.0	95.6 ± 1.0	92.6 ± 0.7	~~95.6 ± 2.9~~	~~95.9 ± 2.0~~	~~92.8 ± 1.3~~
	LBP 256f	**93.0 ± 1.8**	**92.9 ± 1.7**	88.2 ± 2.1	**96.8 ± 0.6**	96.4 ± 0.8	**93.3 ± 1.2**	~~96.9 ± 0.8~~	~~96.4 ± 1.2~~	~~93.9 ± 0.8~~
	TH-MR8	~~92.4 ± 2.2~~	92.5 ± 2.0	~~87.1 ± 1.9~~	~~94.4 ± 1.5~~	~~94.9 ± 1.5~~	~~90.2 ± 1.7~~	~~97.1 ± 0.5~~	~~97.1 ± 0.6~~	94.5 ± 0.6

Source: Elaborated by the author.

rate of 96.8%. The two most plausible explanations for these contradiction are:

1. Since all CNN-based methods were trained on the basis of the same source domain dataset (ImageNet), there could be a strong correlation between the features computed by different CNN-based methods.

2. Given that all CNN-based methods compute high-dimensional feature vectors, their performance might be adversely affected due to the "curse of dimensionality". Indeed, this is similar to what we observed for TH-MR8, which, among the hand-engineered methods, is the one that gives rise to the largest feature vectors and the worst accuracy improvements when combined with GAP-DenseNet-121.

Here, we addressed the second explanation by using the ANOVA feature selection technique (LOMAX; HAHS-VAUGHN, 2012, Ch. 1.3), which was applied to each combined feature extractor of the form $M = M_1 + M_2$. In more detail, we reduced the number of features computed by M_1, which is always GAP-DenseNet-121, to 768 dimensions, whereas, whenever possible, we reduced the number of features computed by M_2 to 256 dimensions. Hence, M ended up with up to 1024 dimensions.

Table 10 shows the comparison between 16 feature extractors with respect to the accuracy improvement that each of them achieved over the baseline accuracy rates of GAP-DENSENET-121. This comparison was performed for the same datasets and classifiers as in subsection 5.4.4. Comparing Table 9 and Table 10, it is clear that the dimensionality reduction approach used in this part played a interesting role as it led to higher accuracy rates at both GAP-DENSENET-121, and its combinations with other feature extractors. Other key observations in Table 10 are as follows.

- GAP-EFFICIENTNET-B1 is once again the CNN-based method with the highest accuracy rates. However, although most of its results are often better than those presented in Table 9, they were always inferior to those of *FMS-N*, *FMS-U* or *LBP 256f*.

- As for TH-MR8, its situation improved considerably, which suggest that the "curse of dimensionality" indeed had a negative influence in this case.

- The results achieved by *LBP-256*, *FMS-U* and *FMS-N* followed a similar trend as the one inferred from Table 9. *FMS-U* ended up as the hand-engineered feature extractor with the most stable results, and *LBP-256* reached accuracy improvements of 2.2% and 1.9% in the `Outex013` and `MBT` datasets, respectively.

In summary, the results in Table 10 suggest that dimensionality reduction benefits both GAP-DENSENET-121 and hand-engineered methods to some extend, without increasing the computational burden of the evaluation scheme. Therefore, its use is recommended. Besides, as CNN-based methods did not increase the accuracy rates of GAP-DENSENET-121 as much as expected, the "curse of dimensionality" is ruled out as the main reason for the low accuracy improvements.

5.5 Final considerations

In this chapter, we presented a systematic comparison between the performance of CNN-based methods, hand-engineered methods and combinations of pairs of them for the task of gray-level texture recognition. This comparison was performed at five texture datasets: `UIUC`, `USPtex`, `Outex013`, `MBT` and `CUReT_Q4` and using three different classical machine learning classifiers: SVM-RBF, SVM-L and KNN. The initial experiments confirmed the results of previous works (LIU *et al.*, 2019) that show CNN-based methods reaching higher accuracy rates than hand-engineered methods.

Among the CNN-based methods, GAP-DENSENET-121 led to the best accuracy rates at every dataset and, therefore, it should be the default option for use in all related tasks. Subsequent experiments showed that combinations of two CNN-based methods often obtained non-significant

Table 10 – **Comparison of accuracy rates when dimensionality reduction is used.** The best accuracy improvement per column is emphasized in bold, while the best result for each dataset is highlighted in red bold font. Non-significant accuracy improvements are displayed in a strike-through format. The statistical significance was verified with the McNemar test, using $\alpha = 0.05$. EFFNET-B1 is the abbreviation for EFFICIENTNET-B1.

		GAP-DENSENET-121								
		`Outex013`			`MBT`			`CUReT_Q4`		
		SVM-RBF	SVM-L	KNN	SVM-RBF	SVM-L	KNN	SVM-RBF	SVM-L	KNN
	Baseline →	91.4 ± 2.3	91.1 ± 1.6	88.4 ± 1.3	95.5 ± 0.9	95.2 ± 0.9	92.9 ± 1.1	97.7 ± 0.7	97.1 ± 0.8	95.0 ± 0.8
FC-CNN	VGG-19	~~91.5 ± 1.8~~	~~91.1 ± 1.4~~	~~88.8 ± 1.4~~	~~95.6 ± 1.1~~	~~95.1 ± 1.4~~	~~92.9 ± 1.4~~	~~97.7 ± 0.7~~	~~97.2 ± 0.8~~	95.2 ± 0.9
	INCEPTION-V3	~~91.9 ± 1.3~~	~~90.9 ± 1.6~~	~~86.9 ± 1.9~~	~~95.7 ± 1.3~~	~~95.4 ± 1.1~~	~~92.3 ± 1.3~~	97.9 ± 0.6	97.4 ± 0.7	95.4 ± 0.7
	RESNET-50	~~92.1 ± 1.6~~	~~91.5 ± 1.2~~	~~87.6 ± 1.9~~	~~95.6 ± 1.2~~	~~95.2 ± 1.4~~	~~93.0 ± 1.3~~	97.9 ± 0.7	97.5 ± 0.8	95.4 ± 0.8
	EFFNET-B1	~~91.9 ± 1.7~~	~~91.8 ± 1.8~~	~~88.3 ± 1.8~~	96.3 ± 0.9	95.8 ± 1.0	93.8 ± 1.1	98.0 ± 0.6	97.7 ± 0.7	95.9 ± 0.8
GAP-CNN	VGG-19	~~91.5 ± 2.2~~	~~91.1 ± 1.9~~	~~88.2 ± 1.7~~	~~96.1 ± 1.1~~	~~95.8 ± 1.0~~	~~93.1 ± 1.1~~	~~97.7 ± 0.8~~	97.4 ± 0.8	95.4 ± 0.9
	INCEPTION-V3	~~91.8 ± 1.8~~	~~91.4 ± 1.8~~	~~87.6 ± 1.5~~	~~95.9 ± 0.7~~	96.1 ± 0.8	~~93.0 ± 1.2~~	97.9 ± 0.7	97.4 ± 0.7	95.7 ± 0.7
	RESNET-50	~~91.7 ± 1.8~~	~~91.5 ± 1.3~~	~~87.8 ± 1.9~~	96.0 ± 1.3	~~95.7 ± 1.2~~	~~93.0 ± 1.4~~	97.9 ± 0.7	97.7 ± 0.8	95.6 ± 0.8
	EFFNET-B1	93.0 ± 2.0	~~91.7 ± 1.9~~	~~88.3 ± 1.6~~	96.1 ± 0.8	95.8 ± 0.8	93.9 ± 1.1	98.1 ± 0.7	97.8 ± 0.7	96.1 ± 0.9
	DTW	92.3 ± 2.6	91.8 ± 1.8	~~88.2 ± 1.7~~	~~95.8 ± 0.9~~	95.8 ± 0.9	~~93.1 ± 1.2~~	98.0 ± 0.6	97.4 ± 0.7	95.7 ± 0.7
	FD	92.6 ± 2.7	~~91.8 ± 2.1~~	~~88.8 ± 1.5~~	96.4 ± 0.9	96.3 ± 0.7	~~93.4 ± 0.8~~	98.1 ± 0.7	97.5 ± 0.8	95.7 ± 0.7
	FMS-N	93.4 ± 2.2	**93.2 ± 1.3**	~~89.3 ± 1.6~~	~~96.1 ± 1.0~~	96.4 ± 1.1	~~93.2 ± 1.2~~	98.1 ± 0.6	97.6 ± 0.7	96.1 ± 0.6
	FMS-U	92.6 ± 1.8	92.8 ± 1.6	~~88.3 ± 2.0~~	97.1 ± 0.8	96.9 ± 0.9	**94.5 ± 1.1**	**98.4 ± 0.7**	**97.9 ± 0.8**	**96.5 ± 0.8**
	GLCM 128r	92.2 ± 2.4	~~91.8 ± 2.1~~	~~88.5 ± 1.5~~	96.0 ± 1.0	95.7 ± 1.0	~~93.4 ± 1.0~~	~~97.7 ± 0.6~~	~~97.1 ± 0.6~~	95.2 ± 0.6
	GW 8s 6o	92.9 ± 2.1	92.4 ± 1.4	~~88.5 ± 1.5~~	96.4 ± 0.7	96.2 ± 0.7	94.0 ± 1.0	~~95.3 ± 3.9~~	~~95.7 ± 2.6~~	~~93.1 ± 2.1~~
	LBP 256f	**93.6 ± 2.1**	92.8 ± 2.0	89.3 ± 2.3	**97.4 ± 1.1**	**97.3 ± 0.9**	94.2 ± 1.1	~~96.8 ± 1.0~~	~~96.2 ± 1.6~~	~~94.4 ± 0.8~~
	TH-MR8	92.8 ± 2.2	~~92.2 ± 1.6~~	**90.1 ± 2.0**	~~96.0 ± 1.0~~	96.0 ± 1.1	93.9 ± 1.3	97.9 ± 0.5	97.6 ± 0.5	96.3 ± 0.6

Source: Elaborated by the author.

accuracy rates with respect to those achieved by each separately. On the contrary, some hand-engineered methods significantly improved the accuracy rates of CNN-based methods. Further experiments with different classifiers confirmed these previous observations. The only CNN-based method that competed with its hand-engineered counterparts was GAP-EFFICIENTNET-B1, and it is only recommended when it is used together with a SVM classifier with linear kernel.

Regarding the feature selection approach (ANOVA), it slightly improved most accuracy rates, but the overall pattern of results previously inferred did not change much. Therefore, our final recommendation for texture recognition and related tasks is to use GAP-DENSENET-121 in combination with either *LBP-256f*, *FMS-N* or *FMS-U*, which are very efficient hand-engineered methods with feature extraction times around 38x faster than GAP-DENSENET-121.

Also, it is strongly recommended the use of an SVM with RBF kernel and feature selection. Indeed, feature selection will be heavily used in the next chapter, which will present one of the proposed methods of this thesis.

CHAPTER

6

CNN MODELS AS COLLECTIONS OF DEEP COMPOSITE FUNCTIONS

6.1 Initial considerations

Two conclusions can be drawn from the previous chapter. (i) CNN-based TL methods generate semantically rich feature representations that are typically superior to those of hand-engineered methods. (ii) However, a CNN-based TL method is more likely to obtain substantial performance improvements when combined with a hand-engineered method than when combined with another CNN-based method.

This chapter addresses the second conclusion raised above by proposing a generalized TL strategy that treats each pre-trained CNN model as a collection of robust **Deep Composite Functions (DCFs)**. Indeed, the proposed TL strategy depends on the following hyper-parameters: (i) the global pooling layer that connects the feature extraction part with the dedicated classifier, and (ii) the strategy for selecting a relatively small collection of DCFs from one or more pre-trained CNN models. In more detail, this chapter focuses on answering the questions below.

1. Are there other global pooling layers that can do a better job than the standard Global Average Pooling (GAP) layer?. This chapter proposes (i) the Global Entropy Pooling (GEP) layer, and (ii) the Global Mean Thresholding Pooling (GMTP) layer. More details in subsection 6.4.2.

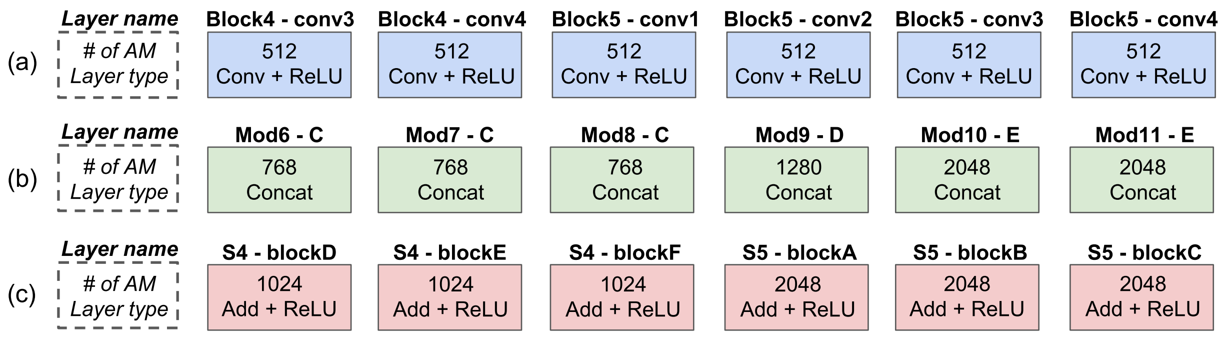
2. What is the best strategy for selecting the most significant DCFs for a given texture dataset?. We first used the layer-by-layer approach (GP-CNN) described in section 6.5. Then we ranked a predefined collection of DCF based on the activation maps they generates for a given meta-dataset (RANKGP-CNN) (see section 6.6), and finally we extend RANKGP-CNN to three pre-trained CNN models (RANKGP-3M-CNN) (see section 6.7).

6.2 Target task, datasets and pre-trained models

As shown in section 3.5 there are two primary types of TL strategies for image classification problems: (i) those that use the CNN model as a pre-trained CNN model, and (ii) those that perform fine-tuning on a modified CNN model. This chapter presents proposed TL strategies of the first type.

However, unlike the previous chapter, we performed experiments on all datasets from Table 2 except `CUReT_Q4`. Hence, this analysis includes both instance-based and category-based texture datasets. Due to computer power limitation, only three CNN models were used in this analysis: BN-VGG-19, INCEPTION-V3, and RESNET-50. In this sense, for each model, we considered the six layers shown in Figure 24. Additionally, we used PyTorch 1.0 (PASZKE *et al.*, 2017) to import the pre-trained weights for BN-VGG-19 and INCEPTION-V3, and we used Keras (CHOLLET *et al.*, 2015) to import the pre-trained weights for RESNET-50.

Figure 24 – **Layers that were considered for each pre-trained CNN model.** Layers are displayed in ascending order of depth level, from left to right. (a) BN-VGG-19, (b) INCEPTION-V3, (c) RESNET-50. *# of AM*: number of activation maps each layer can compute. Layer names come from Keras (CHOLLET *et al.*, 2015).



Source: Adapted from M. Condori and Bruno (2021).

6.3 CNN models treated as collections of deep composite functions

The feature extraction part of a CNN model essentially applies a series of layer transformations to an input image to produce a set of 2D activation maps. These activation maps can uncover many fundamental features of an input image (Springenberg *et al.*, 2015), which are then processed by the classification part to generate a prediction. Indeed, since each activation map results from a unique path of transformations, we treat such a path as a Deep Composite Function (DCF). Therefore, each pre-trained CNN model can be thought as a large collection of powerful deep composite functions.

In more detail, a deep composite function ($\phi_c^{(\ell)}$) is a subnetwork of the CNN model that computes a 2D activation map for each input image I ($\phi_c^{(\ell)}(I) = \mathbf{A}^{(\ell)}[1,c,:,:] = \mathbf{A}_c^{(\ell)}, 1 \leq$

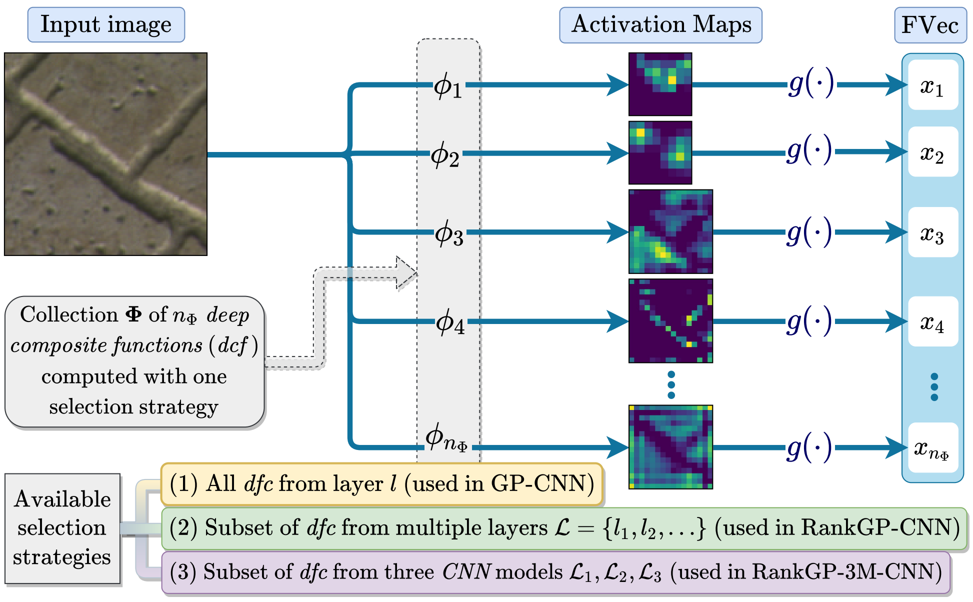
$c \leq C^{(\ell)})$[1]. In this sense, ℓ indicates the layer on which the activation map is calculated, c is a subscript that specify the location of the activation map within ℓ, and $C^{(\ell)}$ indicates the number of activation maps that layer ℓ calculates (the channel dimension). Additionally, i represents the image index, which is one when there is only a single image to process.

6.4 Proposed TL strategy

Based on the notion of deep composite function (DCF), here we propose a generalized TL strategy that uses a pre-trained CNN as a feature extractor. As shown in Figure 25, this TL strategy builds a modified model by making the two decisions below.

1. Use an existing or proposed selection strategy to generate V_{dcf}, a list containing the locations of DCFs within one or more pre-trained CNN models. Then, during training or testing, V_{dcf} will be used to extract a collection Φ of n_Φ selected DCFs. For simplicity, we change the notation of a DCF from $\phi_c^{(\ell)}$ to ϕ_j, where j indicates the position of the j-th DCF in the collection $\Phi = \{\phi_1, \phi_2, \ldots, \phi_{n_\Phi}\}$.
2. Select an existing or proposed global pooling layer (subsection 2.6.3) that converts a set of activation maps into a feature vector.

Figure 25 – **Proposed TL strategy.** The input image is processed by a collection **Φ** of deep composite functions. Then, a global pooling measurement $g(\cdot)$ is computed for each resulting activation map. Finally, the set of n_Φ calculated measurements forms the feature vector (FVec).


Input image
Activation Maps
FVec
ϕ_1
ϕ_2
ϕ_3
ϕ_4
ϕ_{n_Φ}
$g(\cdot)$
x_1
x_2
x_3
x_4
x_{n_Φ}
Collection Φ of n_Φ deep composite functions (dcf) computed with one selection strategy
Available selection strategies
(1) All dfc from layer l (used in GP-CNN)
(2) Subset of dfc from multiple layers $\mathcal{L} = \{l_1, l_2, \ldots\}$ (used in RankGP-CNN)
(3) Subset of dfc from three CNN models $\mathcal{L}_1, \mathcal{L}_2, \mathcal{L}_3$ (used in RankGP-3M-CNN)

Source: Adapted from M. Condori and Bruno (2021).

[1] In subsection 2.4.3, an activation map $\mathbf{A}[i, c, :, :]$ is defined as a 2D slice made along the first and second dimensions of a tensor of size $(N_I, C, W, H); 1 \leq i \leq N_I, 1 \leq c \leq C$.

6.4.1 Feature extraction, training and prediction subroutines

The general evaluation scheme outlined in subsection 4.3.1 applies here. Therefore, this subsection describes the TRAINMODEL and PREDICT subroutines, which are essential for the correct functioning of Algorithm 1.

For the training subroutine, we present Algorithm 4 as the upgraded version of Algorithm 2 (Chapter 5). The highlights of this subroutine are as follows:

- As explained at the beginning of this section, in addition to M_{feat} and M_{clf}, the modified model M is associated with a chosen global pooling layer $g(\cdot)$ and a list of composite indices V_{dcf} that allows locating the selected DCFs. In this sense, if M_{feat} spans only one pre-trained CNN model, each composite index must contain exactly two sub-indices: one to locate the layer, and another to locate the DFC within the layer. On the other hand, if M_{feat} spans two or more pre-trained CNN models, an additional sub-index is necessary to locate the pre-trained model where the DCF resides.
- EXTRACTDCFS is a new sub-routine that uses the list of indices V_{dcf} to extract the collection Φ of selected DCFs from the feature extractor M_{feat}.
- Finally, EXTRACTFEATURES is also a new subroutine that transforms a set of images into a feature matrix (or set of feature vectors) of shape $(n_B \times n_\Phi)$. Its input consists of a set of images $\mathbf{I}$, a collection Φ of n_Φ selected DCFs, and the chosen global pooling layer $g(\cdot)$. The details of the EXTRACTFEATURES subroutine is in Algorithm 5. It is important to note that, although the algorithm appears to be very expensive, it has virtually the same computational cost as GAP-CNN or FC-CNN because, by default, the pre-trained CNN model always computes the entire collection of DCFs.

Algorithm 4 – Deep Composite Functions – Training process

```
 1: procedure TRAINMODEL($M, B_{\text{train}}$)                ▷ Model $M$ and training *split* $B_{\text{train}}$
 2:     $\langle M_{\text{feat}}, g(\cdot), V_{\text{dcf}}, M_{\text{clf}} \rangle \leftarrow M$      ▷ feature extractor $M_{\text{feat}}$, dedicated classifier $M_{\text{clf}}$,
 3:                                   ▷ global pooling layer $g(\cdot)$, list of composite indices $V_{\text{dcf}}$
 4:     $\langle \mathbf{I}_{B_{\text{train}}}, Y_{B_{\text{train}}} \rangle \leftarrow B_{\text{train}}$         ▷ Training images $\mathbf{I}_{B_{\text{train}}}$ and labels $Y_{B_{\text{train}}}$
 5:     $\Phi \leftarrow$ EXTRACTDCFS($M_{\text{feat}}, V_{\text{dcf}}$)
 6:     $X_{B_{\text{train}}} \leftarrow$ EXTRACTFEATURES($\mathbf{I}_{B_{\text{train}}}, \Phi, g(\cdot)$)
 7:     $\langle \widehat{X}_{B_{\text{train}}}, \mathbf{U}, \Sigma \rangle \leftarrow$ STANDARDIZEFVS($X_{B_{\text{train}}}$)       ▷ $\mathbf{U}$ is the set of mean values
 8:                                            ▷ $\Sigma$ is the set of standard deviation values
 9:     $\widehat{M}_{\text{clf}} \leftarrow$ TRAINCLASSIFIER($M_{\text{clf}}, \widehat{X}_{B_{\text{train}}}, Y_{B_{\text{train}}}$)     ▷ $\widehat{M}_{\text{clf}}$ is the trained classifier
10:     $\widehat{M} \leftarrow \langle M_{\text{feat}}, g(\cdot), \widehat{M}_{\text{clf}}, \mathbf{U}, \Sigma \rangle$
11:     return $\widehat{M}$
12: end procedure
```

Similar to the TRAINMODEL subroutine, the PREDICT subroutine is the upgraded version of Algorithm 3. Consequently, it also uses the EXTRACTDCFS and EXTRACTFEATURES

Algorithm 5 – Deep Composite Functions – Feature extraction

```
1: procedure ExtractFeatures($\mathbf{I}_B, \Phi, g(\cdot)$)      ▷ Set of images $\mathbf{I}_B$ from dataset $B$,
2:                                    ▷ collection of deep composite functions $\Phi$,
3:                                    ▷ and global pooling layer $g(\cdot)$
4:     $X_B[1\dots n_B; 1\dots n_\Phi]$      ▷ Let $X$ be a new matrix of size $(n_B \times n_\Phi)$
5:     for all $I_i \in \mathbf{I}_B$ do
6:         fv$[1\dots n_\Phi]$              ▷ Let fv be a new array of $n_\Phi$ elements
7:         for all $\phi_j \in \Phi$ do
8:             fv$[j] \leftarrow g(\phi_i(I_i))$   ▷ Transform $I_i$ into single feature value
9:         end for
10:        $X_B[i,:] \leftarrow$ fv
11:    end for
12:    return $X_B$
13: end procedure
```

subroutines to compute a feature matrix $X_{B_{\text{test}}}$ from a test set of images B_{test}. Finally, the trained classifier predicts the set of labels $\widehat{Y}_{\text{test}}$ from $X_{B_{\text{test}}}$.

6.4.2 Proposed global pooling layers

As explained in subsection 2.6.3, a pooling layer is a powerful layer type, used for reducing the spatial size of a given set of activation maps. In this sense, a Global Pooling (GP) layer is a particular instance of a pooling layer that always reduces the spatial size of every activation map to a single value. In this chapter, in addition to GAP, we experimented with two more types of global average pooling layers. Specifically we propose:

- Global Entropy Pooling (GEP): Rather than computing the average or taking the maximum value, GEP calculates the entropy of each activation map $\mathbf{A}_c^{(\ell)}$ using Equation 6.1. In this regard, $p_c^{(\ell)}$ is a probability distribution that results from the following steps. (i) First, normalize $\mathbf{A}_c^{(\ell)}$ to the range [0, 255]. (ii) Then, calculate the histogram of the resulting $\mathbf{A}_c^{(\ell)}$ using 256 bins. (iii) Lastly, divide the histogram values by its total sum.

$$\text{GEP}(\mathbf{A}_c^{(\ell)}) = -\sum_j p_c^{(\ell)}[j] \ln\left(p_c^{(\ell)}[j]\right) \tag{6.1}$$

- Global Mean Thresholding Pooling (GMTP): It results from an effort of including inter-channel information into the output values. In this regard, Equation 6.2 shows the computation of $T_g^{(\ell)}$, where u and v indicate a spatial position in $\mathbf{A}_c^{(\ell)}$, $n_{\mathbf{A}^{(l)}}$ is the number of activation maps, $h_{\mathbf{A}^{(l)}}$ is the height, and $w_{\mathbf{A}^{(l)}}$ is the width of each activation map. Hence, $T_g^{(\ell)}$ can be seen as the mean value of the set of activation maps in layer ℓ. Finally, for each activation map $\mathbf{A}^{(\ell)}$, GMTP returns the number elements whose values are lower than $T_g^{(\ell)}$.

$$T_g^{(\ell)} = \frac{\sum_c \sum_v \sum_u (\mathbf{A}_c^{(\ell)}[v,u])}{n_{\mathbf{A}^{(\ell)}} * h_{\mathbf{A}^{(\ell)}} * w_{\mathbf{A}^{(\ell)}}}, \qquad \mathrm{GMPT}(\mathbf{A}_c^{(\ell)}) = \sum_{u,v} (\mathbf{A}_c^{(\ell)}[u,v] < T_g^{(\ell)}) \tag{6.2}$$

6.5 GP-CNN: definition and selection strategy

This subsection used the generalized TL strategy proposed in section 6.4 to explore GP-CNN, a straightforward extension of GAP-CNN. In more detail, GP-CNN utilizes the following selection strategy: add to the list V_{dcf}, the locations of all DFCs originating from a particular layer ℓ.

In general, many research works implicitly used the above selection strategy when they conducted layer-by-layer analyses to demonstrate the higher predictive power of deeper layers (RAZAVIAN *et al.*, 2014; CIMPOI *et al.*, 2016). However, since these previous analyses mainly focused on category-based texture datasets (e.g., `DTD`), the following experiments will show a systematic analysis using GP-CNN applied to both category-based and instance-based texture datasets.

6.5.1 Experimental settings

Using the subroutines explained in subsection 6.4.1, we applied the general evaluation scheme described in subsection 4.3.1 to evaluate GP-CNN 702 times. Each evaluation included one combination of the following three parameters.

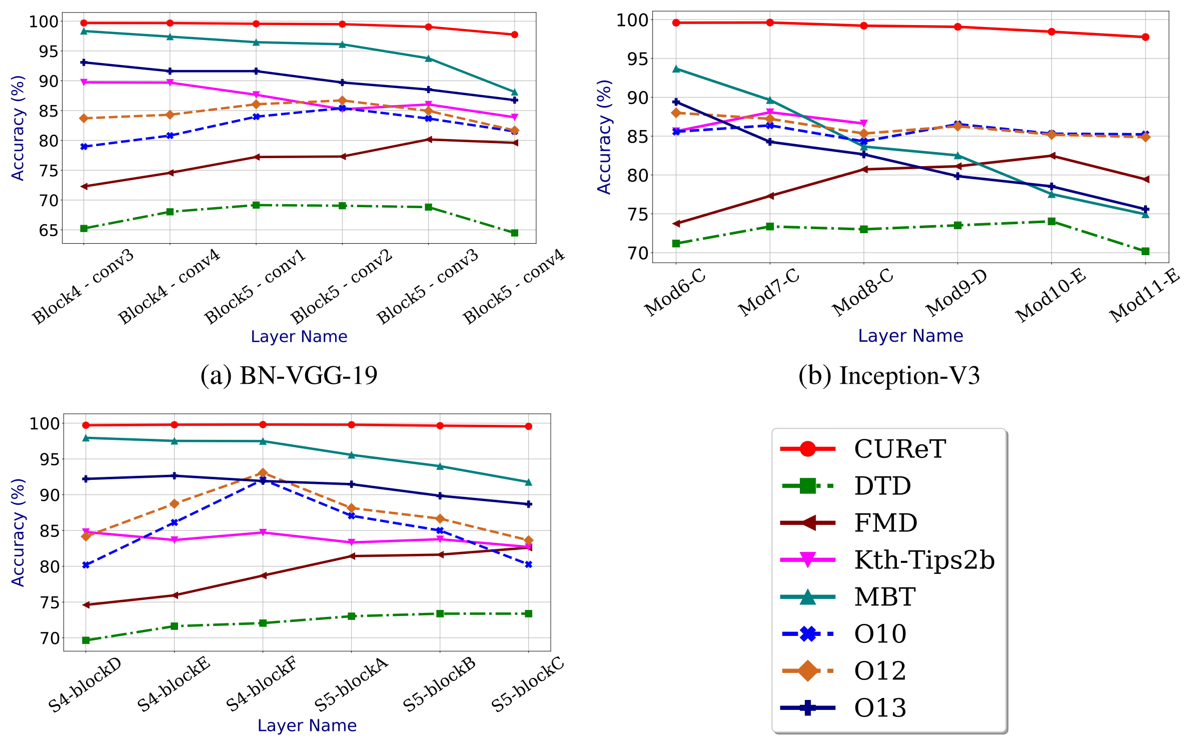
1. **Datasets (B).** All texture datasets from Table 2, except `CUReT_Q4`. There are thirteen datasets in total. Regarding the spliting strategy, datasets without predefined *splits* followed the stratified m-repeated holdout validation (with m=10 and p=50%).
2. **Collections of DCFs (Φ).** Since the selection strategy of GP-CNN generates a list $V_{\mathrm{dcf}}^{(\ell)}$ per layer ℓ, and we considered the eighteen layers shown in Figure 24, there are eighteen collections of DCFs in total.
3. **GP layers $g(\cdot)$.** The three GP layers from subsection 6.4.2 (GAP, GMTP and GEP).

Regarding the dedicated classifier, it is a Support Vector Machine (SVM) classifier with linear kernel and parameter $C = 1$.

6.5.2 Performance of GAP features across multiple depth levels

This analysis was designed to graphically demonstrate how the accuracy rate varies across multiple depth levels within each dataset. In this context, we excluded the results of the

Figure 26 – **Accuracy rates achieved by GP-CNN across multiple layers and datasets**. The horizontal axis of each chart displays the layers in order of increasing depth. Additionally, all charts provide only GAP-related results.



(a) BN-VGG-19 (b) Inception-V3

(c) ResNet-50 (d) datasets

Source: Adapted from M. Condori and Bruno (2021).

Brodatz, UIUC, Vistex, USPtex, and STex datasets because they were very similar to those of CUReT. Additionally, we excluded the results associated with GEP and GMTP layers because they revealed a highly similar trend as GAP. Consequently, of the 702 experiments we ran, Figure 26 shows only the accuracy rates of 144. Specifically, these 144 accuracy rates are distributed into three sub-figures, one for each pre-trained CNN model.

In general, results from category-based texture datasets (especially DTDs and FMDs) support the consensus that the predictive power associated with a particular layer correlates with its depth (RAZAVIAN *et al.*, 2014; CIMPOI *et al.*, 2016). On the other hand, results from instance-based texture datasets, like the Outex suites (O10, O12, O13) and MBT, demonstrates that shallow layers can also generate features with competitive predictive power.

In this regard, the similarity between the source and target domain datasets (ImageNet vs. texture datasets) may explain the accuracy rates of Figure 26. Essentially, we hypothesize that deeper layers produce features that are too abstract (or domain-specific) to be used in challenging instance-based texture datasets, but helpful for category-based datasets. Indeed, Cui *et al.* (2018) have shown that measuring the similarity between the source domain and the target domain is crucial for fine-tuning TL methods since it could determine beforehand whether good results are possible in the target domain.

Table 11 – **Highest accuracy rates (%) achieved by GP-CNN.** This table allows comparisons between CNN models, global pooling layers, and datasets. The highest result per group is emphasized in bold.

		DTD	FMD	Kth-Tips2b	MBT	O10	O12	O13
BN-VGG-19	GAP	69.2 ± 0.8	**80.2 ± 1.5**	89.7 ± 4.7	98.3 ± 0.2	**85.4**	86.7	93.1
	GEP	**69.9 ± 0.7**	79.9 ± 1.5	89.6 ± 5.1	**98.4 ± 0.4**	85.1	**87.2**	93.2
	GMTP	**69.9 ± 0.8**	**80.2 ± 1.7**	**89.8 ± 5.1**	**98.4 ± 0.4**	84.8	**87.2**	**94.0**
INCEPTION-V3	GAP	74.0 ± 0.9	82.5 ± 1.3	88.1 ± 4.3	93.7 ± 0.4	86.5	88	**89.4**
	GEP	74.4 ± 0.6	82.3 ± 1.1	**88.5 ± 3.8**	**94.0 ± 0.4**	86.2	87.4	87.9
	GMTP	**74.5 ± 0.7**	**83.0 ± 1.3**	88.4 ± 4.5	93.8 ± 0.4	**87.9**	**88.2**	88.8
RESNET-50	GAP	73.4 ± 0.8	82.6 ± 1.0	84.8 ± 5.4	**98.0 ± 0.4**	92.1	93	92.6
	GEP	73.2 ± 0.9	83.0 ± 1.5	**86.4 ± 5.6**	96.3 ± 0.5	**93.7**	**93.7**	91.3
	GMTP	**73.5 ± 1.0**	**83.8 ± 2.5**	85.1 ± 6.2	97.6 ± 0.3	93.4	93.5	**92.8**

Source: Adapted from M. Condori and Bruno (2021).

Figure 26c presents an interesting case study on Outex10 (O10) and Outex12 (O12). In particular, it shows a sustained improvement in accuracy from the shallowest layer of RESNET-50 to the S4-blockF layer, followed by a steady decline that ends at the last layer. These results indicate that while layer abstraction increases with layer depth, the appropriate level of abstraction differs from dataset to dataset.

Lastly, it should be noted that the results shown in Figure 26 may vary in several circumstances due to the stochastic nature of the SGD algorithm. Therefore, it would be beneficial to develop advanced selection strategies that filter in the best DCF from each layer.

6.5.3 Comparison of global pooling layers

Table 11 lists the highest accuracy rates obtained per model, dataset, and global pooling layer. Based on the comparison of CNN models, INCEPTION-V3 performs well on all category-based datasets but poorly on all instance-based datasets. Regarding BN-VGG-19 and RESNET-50, the former outperforms the other CNN models in Kth-Tips2b, MBT, and Outex13 datasets, while the latter is successful in most datasets, particularly in Outex10 and Outex12. Despite the lack of clarity regarding which CNN model is best for a given dataset, INCEPTION-V3 seems to be a better fit for category-based datasets, while RESNET-50 seems more robust for instance-based datasets.

Finally, the comparison different global pooling layers shows that GMTP features tend to achieve the best results, especially in INCEPTION-V3 and RESNET-50 models. Nevertheless, it should be noted that the accuracy improvements brought by GMTP are less than 2% in most cases. Therefore, the following experiments will only include GAP features.

6.6 RankGP-CNN: Multi-layer feature extraction using a feature ranking approach

Based on the systematic experiments performed on GP-CNN (subsection 6.5.1), the following patterns emerged: (i) The fact that a DCF originates from a very deep layer does not necessarily make it highly effective for all situations. (ii) We can, however, increase our chances of selecting the right set of DCFs for a given task if we provide further information, such as the type of texture problem at stake.

The above observations led to the development of RANKGP-CNN, a TL strategy that performs a smart selection of DCFs from a pretrained CNN model. For this purpose, RANKGP-CNN uses two complementary approaches: (i) a feature ranking technique that assigns a score value to every DCF and (ii) a meta-dataset $\mathcal{B}$ that provides the necessary information for the correct functioning of the feature ranking technique. In this regard, a meta-dataset can be thought as a collection of one or more distinct datasets.

Essentially, the score assigned to each DCF represents its ability to identify highly discriminating features in the input images. Hence, the selection strategy involves assembling a list V_{dfc} with the locations of the k highest-ranked DCFs. RANKGP-CNN will be described in detail in the following subsections.

6.6.1 *Selection Strategy*

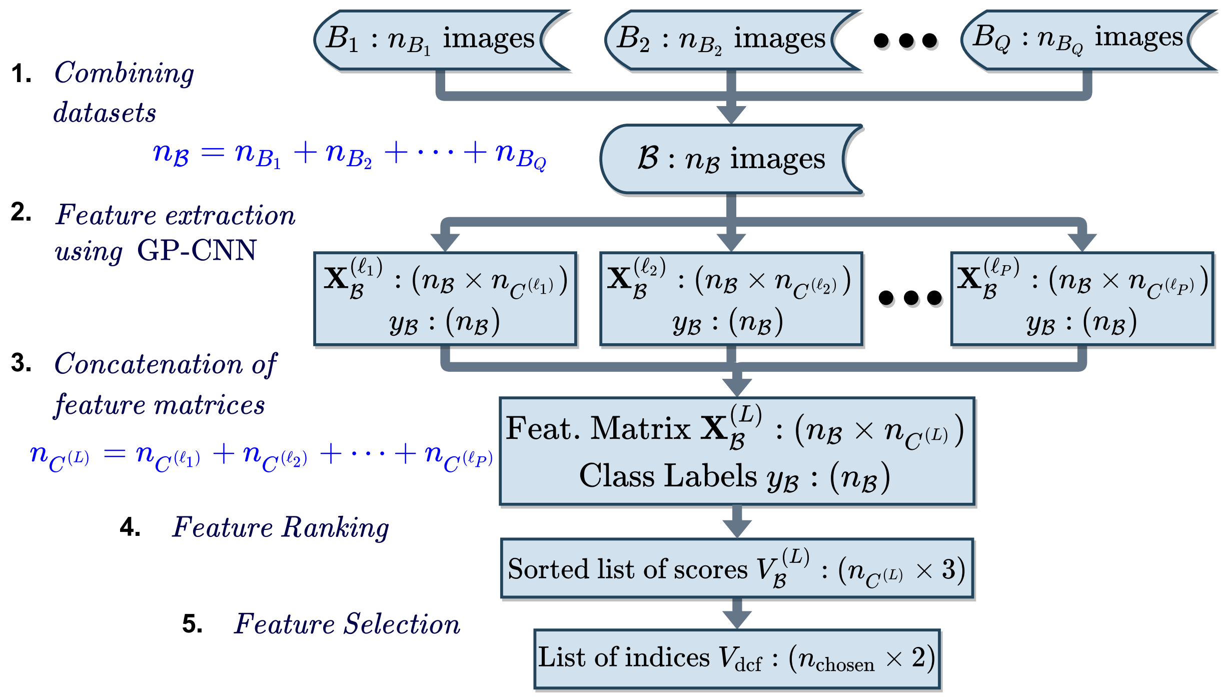
Following are the steps involved in the classical feature selection pipeline. (i) First, with the feature extraction algorithm, compute the feature matrices $\mathbf{X}^{(l)}_{B_{\text{train}}}$ and $\mathbf{X}^{(l)}_{B_{\text{test}}}$ for the training and test sets, respectively. (ii) Then, with the feature ranking technique, process $\mathbf{X}^{(l)}_{B_{\text{train}}}$ to obtain a list of feature scores. (iii) Lastly, remove from both $\mathbf{X}^{(l)}_{B_{\text{train}}}$ and $\mathbf{X}^{(l)}_{B_{\text{test}}}$ the columns that reflect the lowest-ranked features.

Even though the above pipeline is suitable for many applications, it may be counterproductive for datasets that evaluate particular image properties. For instance, the predefined test set of `Outex10` contains the same images as the training set but rotated at various angles (see Figure 18a). Consequently, feature ranking on this training set may result in the removal of relevant rotation-invariant features. Indeed, this situation is particularly troubling for CNN models, where it has been demonstrated that some DCFs can generate rotation- and illumination-invariant features (OLAH; MORDVINTSEV; SCHUBERT, 2017).

The selection strategy of RANKGP-CNN avoids the above problem by applying the feature ranking technique to a meta-dataset $\mathcal{B}$ instead of the input training set B_{train}. As shown in Figure 27, the proposed selection strategy involves the following five stages.

1. **Combining datasets.** Input for this stage is a set of Q datasets $\{B_1, B_2, \ldots, B_Q\}$. These

Figure 27 – General scheme of the proposed approach for ranking the deep composite functions.


$B_1 : n_{B_1}$ images
$B_2 : n_{B_2}$ images
$B_Q : n_{B_Q}$ images
1. Combining datasets
$n_{\mathcal{B}} = n_{B_1} + n_{B_2} + \cdots + n_{B_Q}$
$\mathcal{B} : n_{\mathcal{B}}$ images
2. Feature extraction using GP-CNN
$\mathbf{X}_{\mathcal{B}}^{(\ell_1)} : (n_{\mathcal{B}} \times n_{C^{(\ell_1)}})$ $y_{\mathcal{B}} : (n_{\mathcal{B}})$
$\mathbf{X}_{\mathcal{B}}^{(\ell_2)} : (n_{\mathcal{B}} \times n_{C^{(\ell_2)}})$ $y_{\mathcal{B}} : (n_{\mathcal{B}})$
$\mathbf{X}_{\mathcal{B}}^{(\ell_P)} : (n_{\mathcal{B}} \times n_{C^{(\ell_P)}})$ $y_{\mathcal{B}} : (n_{\mathcal{B}})$
3. Concatenation of feature matrices
$n_{C^{(L)}} = n_{C^{(\ell_1)}} + n_{C^{(\ell_2)}} + \cdots + n_{C^{(\ell_P)}}$
Feat. Matrix $\mathbf{X}_{\mathcal{B}}^{(L)} : (n_{\mathcal{B}} \times n_{C^{(L)}})$
Class Labels $y_{\mathcal{B}} : (n_{\mathcal{B}})$
4. Feature Ranking
Sorted list of scores $V_{\mathcal{B}}^{(L)} : (n_{C^{(L)}} \times 3)$
5. Feature Selection
List of indices $V_{\text{dcf}} : (n_{\text{chosen}} \times 2)$

Source: Adapted from M. Condori and Bruno (2021).

individual datasets form the meta-dataset $\mathcal{B}$. This stage assumes that there is no overlap between member datasets. Consequently, the number of samples in $\mathcal{B}$ is the sum of all the samples in the individual datasets. The same occurs for the number of classes in $\mathcal{B}$.

2. **Feature Extraction using GP-CNN.** The input for this stage is a set of P layers $L = \{\ell_1, \ell_2, \ldots, \ell_P\}$ that come from a pre-trained CNN model. Then, this stage uses the GP-CNN selection strategy to extract the collection $\Phi^{(\ell_p)}$ for each layer $\ell_p \in L$. Next, Algorithm 5 computes a feature matrix $\mathbf{X}_{\mathcal{B}}^{(l_p)}$ of size $(n_{\mathcal{B}} \times n_{\mathbf{C}^{(l_p)}})$ for each layer l_p by processing the set of images from $\mathcal{B}$ with the collection $\Phi^{(\ell_p)}$ and the chosen global pooling layer $g(\cdot)$.

3. **Concatenation of feature matrices.** This stage horizontally concatenates the resulting feature matrices from the previous stage. Therefore, the output of this stage is a large feature matrix $\mathbf{X}_{\mathcal{B}}^{(L)}$ of size $(n_{\mathcal{B}} \times n_{C^{(L)}})$.

4. **Feature Ranking.** Input for this stage is a feature ranking technique ($\mathcal{F}_{\text{rank}}$). Then, this technique is applied to $\mathbf{X}_{\mathcal{B}}^{(L)}$, resulting in a list $V_{\mathcal{B}}^{(L)}$ of shape $(n_{C^{(L)}} \times 3)$. The first two columns of $V_{\mathcal{B}}^{(L)}$ contain the location of each DCF associated with such score, and the third column contains the scores sorted in descending order. The feature ranking technique determines the computational cost of this stage; it could take from a few minutes to several hours to complete.

5. **Feature Selection.** Input for this stage is n_{chosen}, the desired number of DCFs to be

extracted from $V_{\mathcal{B}}^{(L)}$. In this regard, this final stage removes from $V_{\mathcal{B}}^{(L)}$ all DCFs that do not appear in the top n_{chosen} places. Finally, the resulting list is renamed V_{dcf}.

6.6.2 Experimental settings

We used the subroutines explained in subsection 6.4.1 and the general evaluation scheme described in subsection 4.3.1 to systematically evaluate RANKGP-CNN with multiple combinations of the following input parameters.

1. **Datasets (B).** Two category-based datasets (`Kth-Tips2b` and `FMD`) and four instance-based datasets (`Outex10`, `Outex12`, `Outex13`, and `MBT`).
2. **Collections of DCFs (Φ).** As shown in subsection 6.6.1, the selection strategy for RANKGP-CNN requires four input parameters. The possible instances for each input parameter are below.
   - **Meta-datasets** ($\mathcal{B}$). We proposed the following three meta-datasets:
     - $\mathcal{B}_{\text{Ins}}$: it includes all instance-based datasets that were not challenging enough for GP-CNN. Specifically, `USPtex`, `Vistex`, `STex`, `CUReT`, `UIUC`, and `Brodatz`.
     - $\mathcal{B}_{\text{DTD}}$: it is just the `DTD` dataset.
     - $\mathcal{B}_{\text{I+D}}$: it comprises the same datasets as $\mathcal{B}_{\text{Ins}}$, except `CUReT`, which were replaced by `DTD`. Consequently, $\mathcal{B}_{\text{I+D}}$ includes category-based and instance-based texture information.
   - **Predefined sets of layers** (L). Three sets of six layers, each corresponding to a pre-trained CNN model. The three sets are depicted in Figure 24. For instance, $L_{\text{INCEPTION-V3}}$ = {`Mod6-C`, `Mod7-C`, `Mod8-C`, `Mod9-D`, `Mod10-E`, `Mod11-E`}.
   - **Feature ranking techniques** ($\mathcal{F}_{\text{rank}}$). Mutual Information (MI), Extremely Randomized Trees (ET), and One-way Analysis Of Variance (ANOVA). These techniques are briefly described in Appendix B.
   - **Number of selected DCFs** (n_{chosen}). We performed experiments with 250, 500, 1000, 1500, and 2048 chosen deep composite functions.
3. **GP layers $g(\cdot)$.** Only GAP.

6.6.3 Accuracy rates for different sets of input parameters

Table 12 shows the accuracy rates achieved by RANKGP-CNN at multiple (i) meta-datasets, (ii) feature ranking techniques, (iii) pre-trained CNN models, and (iv) texture datasets. In this sense, we did not include the results for $\mathcal{B}_{\text{Ins}}$ since they were mostly lower than those from $\mathcal{B}_{\text{I+D}}$.

Table 12 – **Accuracy results achieved by RANKGP-CNN.** The last column (Mean acc.) presents the accuracy rates averaged across all datasets. Results in **bold** indicate highest accuracy rates per combination of (i) CNN model, (ii) meta-dataset, and (iii) texture dataset. Similarly, results highlighted in **bold red** font represent the highest accuracy rates per combination of (i) CNN model and (ii) dataset.

(a) BN-VGG-19

		FMD	Kth-Tips2b	MBT	O10	O12	O13	Mean acc.
$\mathcal{B}_{\text{DTD}}$	ANOVA	79.5 ± 1.7	89.7 ± 5.1	97.4 ± 0.4	**89.8**	**91.5**	92.9	90.1 ± 5.4
	MI	**80.3 ± 1.3**	**90.8 ± 5.2**	**97.6 ± 0.4**	88.7	90.6	92.2	90.0 ± 5.2
	ET	79.8 ± 1.2	90.6 ± 5.2	97.4 ± 0.4	89.1	91.3	**93.4**	**90.3 ± 5.4**
$\mathcal{B}_{\text{I+D}}$	ANOVA	79.1 ± 1.3	88.5 ± 4.6	98.0 ± 0.4	**92.6**	**93.7**	92.8	**90.8 ± 5.9**
	MI	78.5 ± 1.8	88.0 ± 4.9	**98.3 ± 0.3**	91.1	93.4	**93.8**	90.5 ± 6.2
	ET	78.7 ± 1.4	**88.8 ± 4.7**	98.2 ± 0.3	89.7	92	93.4	90.1 ± 5.9

(b) INCEPTION-V3

		FMD	Kth-Tips2b	MBT	O10	O12	O13	Mean acc.
$\mathcal{B}_{\text{DTD}}$	ANOVA	82.8 ± 0.9	89.0 ± 3.9	89.5 ± 0.7	**94.2**	**93.5**	84.8	89.0 ± 4.2
	MI	82.2 ± 1.0	**90.4 ± 4.7**	**90.9 ± 0.6**	94.2	93.4	**86.9**	**89.7 ± 4.1**
	ET	**83.2 ± 1.4**	88.7 ± 3.2	90.2 ± 0.8	93.1	92.8	84.8	88.8 ± 3.7
$\mathcal{B}_{\text{I+D}}$	ANOVA	**81.8 ± 1.5**	**88.8 ± 3.2**	94.0 ± 0.7	**97.3**	**96.2**	88.1	**91.0 ± 5.4**
	MI	81.4 ± 1.7	88.1 ± 2.5	93.9 ± 0.6	97.2	96.2	88.7	90.9 ± 5.5
	ET	81.6 ± 1.2	87.9 ± 2.4	**94.2 ± 0.3**	96.4	95.6	**89.1**	90.8 ± 5.2

(c) RESNET-50

		FMD	Kth-Tips2b	MBT	O10	O12	O13	Mean acc.
$\mathcal{B}_{\text{DTD}}$	ANOVA	80.3 ± 2.0	87.6 ± 1.5	96.3 ± 0.5	89.4	91.7	91.9	89.5 ± 4.9
	MI	79.2 ± 2.0	87.6 ± 2.6	**96.6 ± 0.5**	90	90	91.6	89.2 ± 5.2
	ET	**80.9 ± 1.2**	**87.7 ± 2.6**	96.4 ± 0.4	**90.9**	**92.4**	92.1	**90.1 ± 4.8**
$\mathcal{B}_{\text{I+D}}$	ANOVA	78.0 ± 1.5	**84.8 ± 1.7**	97.1 ± 0.3	96.4	96.2	91.5	90.7 ± 7.1
	MI	78.0 ± 1.8	82.5 ± 3.1	**98.0 ± 0.3**	97.3	**96.6**	**93.2**	90.9 ± 7.8
	ET	79.0 ± 1.2	83.2 ± 3.4	97.3 ± 0.4	**97.4**	96.6	93.1	**91.1 ± 7.3**

Source: Adapted from M. Condori and Bruno (2021).

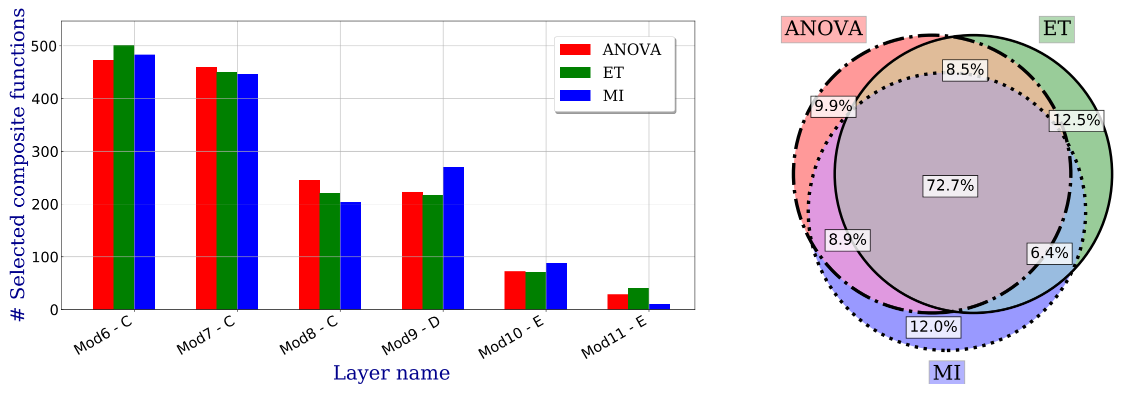
In general, Table 12a, Table 12b, and Table 12c share two interesting patterns. (i) $\mathcal{B}_{\text{DTD}}$ is better for category-based datasets, (ii) and $\mathcal{B}_{\text{I+D}}$ is better for instance-based datasets. As a result, since there are more instance-based datasets than category-based datasets, the mean accuracy rates for $\mathcal{B}_{\text{I+D}}$ are usually higher than those for $\mathcal{B}_{\text{DTD}}$. These patterns exhibit a strong correlation with the concept of **domain similarity**, which suggests that the degree of similarity between the source and target domains has an enormous influence on the performance of a TL strategy. Therefore, a second possible application of RANKGP-CNN is determining the affinity between a pre-trained CNN model and samples from a particular target domain.

The highlights of the comparison of pre-trained CNN models are as follows. (i) BN-VGG-19 surpasses the other CNN models in the Kth-Tips2b, MBT, and Outex13 datasets. However, it failed to perform well in the remaining datasets. (ii) INCEPTION-V3 achieved impressive results in category-based datasets and, compared to other CNN models, it reduced the performance gap in some instance-based datasets, such as `Outex10` and `Outex12`. However, there is still a large performance gap ($\approx 4\%$) between INCEPTION-V3 and the other CNN models in `MBT` and `Outex13`. (iii) As for RESNET-50, its performance leaves much to be desired in category-based datasets. For example, in `Kth-Tips2b`, there is a 2.7% performance gap between the best RESNET-50 accuracy rates and those from the INCEPTION-V3 model.

The comparison of ANOVA, MI, and ET did not reveal a clear winner. Indeed, when using the $\mathcal{B}_{\text{I+D}}$ meta-dataset, the mean accuracy rates in Table 12b and Table 12c are nearly identical for the three techniques. In this context, Figure 28 shows two diagrams that attempt to explain the previous results by comparing the behavior of the three feature ranking techniques for INCEPTION-V3 and $\mathcal{B}_{\text{I+D}}$. In more detail, Figure 28a shows, for each feature ranking method, the distribution of its 1500 top-ranked DCFs across the set of layers of INCEPTION-V3. Despite the slight differences, the three distributions look very similar. Indeed, the number of DCFs always decreases with increasing depth in ANOVA and ET. As for MI, although the decreasing tendency stops at layer `Mod9-D`, it resumes at the next layer.

As for the Venn diagram shown in Figure 28b, it presents the overlap between the top-1500 DCFs generated by ANOVA, MI, and ET. In this regard, the diagram indicates that ANOVA, MI, and ET share 72.7% (1091/1500) of the chosen DCFs. These results suggest that, in addition to having similar DCF distributions, the three feature ranking techniques also generate similar feature vectors. Additionally, 90% of the DCFs chosen by ANOVA also appears on ET and MI. Finally, similar charts can be obtained for BN-VGG-19 and RESNET-50.

Figure 28 – **Comparison of the top-1500 DCFs computed by ANOVA, ET, and MI.** Input parameters: the set of predefined layers from INCEPTION-V3, $\mathcal{B}_{\text{I+D}}$, and $n_{\text{chosen}} = 1500$.



(a) Distribution of DCFs

(b) Overlapping of DCFs

Source: Adapted from M. Condori and Bruno (2021).

6.6.4 *Comparing* RankGP-CNN *with* GP-CNN

Table 13 compares the mean accuracy rates achieved by GP-CNN and RANKGP-CNN. In this sense, the parameter combinations for GP-CNN are the same as those presented in

Table 13 – Accuracy rates averaged across six datasets (`FMD`, `Kth-Tips2b`, `MBT`, `Outex10`, `Outex12` and `Outex13`). In GP-CNN, l_1 to l_6 are layers sorted in increasing depth. RankGP-CNN++ is a variant of RankGP-CNN that use $\mathcal{B}_{\text{DTD}}$ for category-based datasets, and $\mathcal{B}_{\text{I+D}}$ for instance-based datasets. The best result per column is in **bold red** font.

		BN-VGG-19	Inception-V3	ResNet-50
GP-CNN	l_1	86.0 ± 8.8	**86.0 ± 6.1**	85.6 ± 7.6
	l_2	86.4 ± 7.5	85.5 ± 4.0	87.4 ± 6.8
	l_3	**87.2 ± 6.0**	83.9 ± 1.9	**89.7 ± 6.2**
	l_4	86.7 ± 5.6	84.1 ± 3.1	87.8 ± 4.7
	l_5	86.2 ± 4.2	82.9 ± 3.8	86.8 ± 4.1
	l_6	83.6 ± 3.0	81.4 ± 5.0	84.9 ± 4.0
RankGP-CNN ($\mathcal{B}_{\text{I+D}}$)	ANOVA	**90.8 ± 5.9**	**91.0 ± 5.4**	90.7 ± 7.1
	MI	90.5 ± 6.2	90.9 ± 5.5	90.9 ± 7.8
	ET	90.1 ± 5.9	90.8 ± 5.2	**91.1 ± 7.3**
RankGP-CNN++	ANOVA	91.0 ± 5.7	91.2 ± 5.1	91.5 ± 6.0
	MI	**91.3 ± 5.5**	**91.4 ± 5.1**	92.0 ± 6.7
	ET	90.6 ± 5.6	91.2 ± 4.7	**92.2 ± 6.1**

Source: Adapted from M. Condori and Bruno (2021).

subsection 6.5.2. In more detail, it involved (i) the eighteen sets of layers shown in Figure 24, and (ii) the GAP layer. As for RankGP-CNN, we considered (i) the three pre-trained CNN models, (ii) the $\mathcal{B}_{\text{I+D}}$ meta-dataset, (iii) three feature ranking strategies (ANOVA, MI, ET), and (iv) the GAP layer. Lastly, we defined RankGP-CNN++ as the RankGP-CNN that uses $\mathcal{B}_{\text{DTD}}$ for categorical-based datasets and $\mathcal{B}_{\text{I+D}}$ for instance-based datasets.

The highlights of Table 13 are as follows. (i) RankGP-CNN performed better than GP-CNN in all instances, demonstrating that, at least on average, selecting DCFs from multiple layers is better than using those of a single layer. (ii) In every CNN model, the GP-CNN mean accuracy rates of the deepest layer (l_6) were significantly lower than those associated with the other layers ($l_1 - l_5$). (iii) Moreover, RankGP-CNN improved the best results achieved by GP-CNN in the BN-VGG-19, Inception-V3 and ResNet-50 models by +3.6%, +5.0%, and +1.4%, respectively. (iv) Finally, RankGP-CNN++ further improved the previous results, especially for the ResNet-50 model.

6.7 RankGP-3M-CNN: combining BN-VGG-19, Inception-V3, and ResNet-50

Upon analyzing the results in Table 12 and Table 13, two points are evident. (i) Despite RankGP-CNN and GP-CNN computing a similar number of features, RankGP-CNN always produces better results than GP-CNN. (ii) Additionally, the comparison of the results obtained by different CNN models demonstrates that no model outperforms the others in all cases. Therefore, the question is whether it is possible to apply RankGP-CNN to multiple CNN models without damaging the predictive power of each CNN model individually. In response, we propose RANKGP-3M-CNN, a helpful and straightforward variant of RANKGP-CNN that merges the predefined layers from $\{$BN-VGG-19, INCEPTION-V3 and RESNET-50$\}$.

6.7.1 Selection strategy and experimental settings

The selection strategy of RANKGP-3M-CNN is straightforward. It consists of executing three times the RANKGP-CNN selection strategy shown in Figure 27 (one for each pre-trained CNN model). Then, the resulting lists of indices $V_{\text{dcf}}^{(\text{BN-VGG-19})}$, $V_{\text{dcf}}^{(\text{INCEPTION-V3})}$, $V_{\text{dcf}}^{(\text{RESNET-50})}$ are concatenated into a single list V_{dcf}.

As for the experimental settings, we used the subroutines described in subsection 6.4.1 along with the general evaluation scheme presented in subsection 4.3.1 to systematically evaluate RANKGP-3M-CNN with different combinations of the following parameters.

- **Datasets** (B). Two category-based datasets (`FMD` and `Kth-Tips2b`), and four instance-based datasets (`MBT`, `Outex10`, `Outex12`, and `Outex13`).
- **Collections of DCFs** ($\mathbf{\Phi}$). The selection strategy of RANKGP-3M-CNN requires a combination of four parameters. The limits and details for each of them are below.
  - **Meta datasets** $\mathcal{B}$. Only $\mathcal{B}_{\texttt{DTD}}$ and $\mathcal{B}_{\texttt{I+D}}$.
  - **Predefined set of layers** $\mathcal{L}$. One that includes the eighteen layers from Figure 24.
  - **Feature ranking techniques** $\mathcal{F}_{\text{rank}}$. ANOVA, MI, and ET.
  - **Number of selected DCFs** (n_{chosen}). we performed experiments with 400, 560 and 900 DCFs per pre-trained CNN model.
- **GP layers.** Only GAP.

6.7.2 Accuracy rates for different sets of input parameters

Here, we used the experimental settings described in subsection 6.7.1 to evaluate RANKGP-3M-CNN with different parameter combinations. In more detail, for each dataset and combination of (i) one meta-dataset and (ii) one feature ranking technique, Table 14 presents the highest accuracy rate among those computed for different values of n_{chosen}.

Some highlights of Table 14 are as follows. (i) $\mathcal{B}_{\texttt{DTD}}$ is again associated with the highest accuracy rates on category-based datasets, while the best results for instance-based datasets were achieved when using $\mathcal{B}_{\text{I+D}}$. These results support the notion, discussed in subsection 6.6.3, of domain similarity (Cui *et al.*, 2018) being a critical aspect for selecting good collections of DCFs. (ii) ANOVA is associated with the highest mean accuracy rate (92.6%) of all feature

Table 14 – **Accuracy rates (%) achieved by RANKGP-3M-CNN.** The results are available for six texture datasets, three feature ranking approaches and two meta-datasets. The last column has the mean accuracy values.

		FMD	Kth-Tips2b	MBT	010	012	013	Mean acc.
$\mathcal{B}_{\text{DTD}}$	ANOVA	**82.6 ± 1.4**	89.4 ± 4.0	97.4 ± 0.4	94.6	94.6	92.4	91.8 ± 4.8
	MI	80.9 ± 1.7	**91.1 ± 4.5**	97.7 ± 0.2	94.4	95	**92.8**	92.0 ± 5.4
	ET	82.6 ± 2.0	89.9 ± 3.7	**97.8 ± 0.2**	**94.8**	**95.6**	91.9	**92.1 ± 5.0**
$\mathcal{B}_{\text{I+D}}$	ANOVA	**80.4 ± 1.3**	**87.4 ± 3.2**	98.3 ± 0.5	**98.2**	**97.6**	93.8	**92.6 ± 6.7**
	MI	79.9 ± 1.6	85.7 ± 5.2	98.4 ± 0.3	97.0	96.8	**94.6**	92.1 ± 6.9
	ET	79.6 ± 1.5	85.5 ± 5.6	**98.5 ± 0.3**	96.9	96.8	94.4	92.0 ± 7.0

Source: Adapted from M. Condori and Bruno (2021).

ranking techniques analyzed in this thesis. Nevertheless, these results are not conclusive enough to suggest that ANOVA is superior to MI and ET.

Indeed, when considering RANKGP-3M-CNN++ as the TL strategy that uses $\mathcal{B}_{\text{DTD}}$ for category-based datasets, and $\mathcal{B}_{\text{I+D}}$ for instance-based datasets, the mean accuracy rates for ANOVA (93.3%), MI (93.1%), and ET (93.2%) are nearly identical. Nevertheless, MI and ET are both less computationally efficient than ANOVA. For example, ANOVA could process $\mathcal{B}_{\text{I+D}}$ within seconds on a single-core machine, whereas MI would process it within several minutes. In contrast, ET generally requires several hours to complete its work.

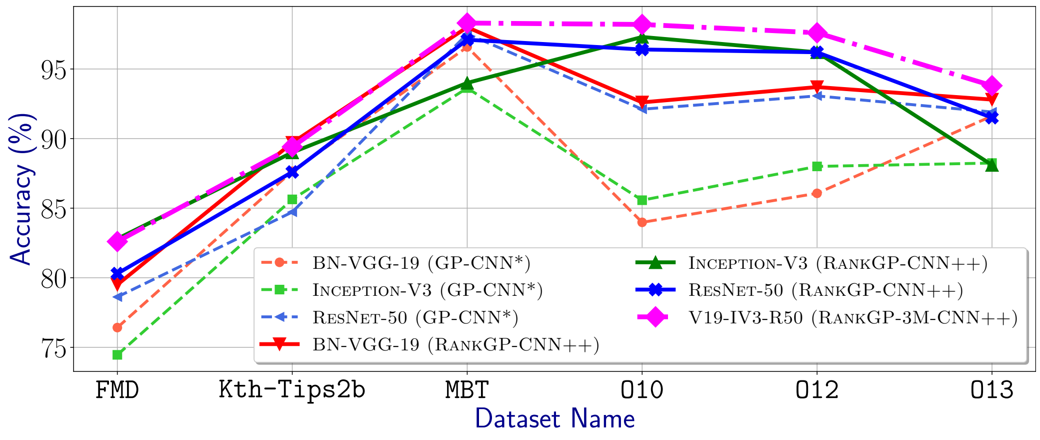
In Figure 29, we can easily observe the progress in the accuracy rates achieved by GP-CNN*, RANKGP-CNN++, and RANKGP-3M-CNN++ for six texture datasets. In this context, we only included the results associated with ANOVA for RANKGP-CNN++ and RANKGP-3M-CNN++. Below are the main points of Figure 29. (i) RANKGP-CNN++ has a distinct advantage over GP-CNN* when comparing the accuracy rates achieved by each pre-trained CNN model, especially when using instance-based datasets (ii) Aside from that, the results achieved by RANKGP-3M-CNN++ are comparable to or even superior to those achieved by RANKGP-CNN++.

The above findings suggest that simultaneously using two or more collections of DCFs is recommended to improve the classification results on each dataset. Similar charts are obtained for other feature ranking techniques (MI and ET), but they are not shown here.

6.8 Comparing RankGP-3M-CNN++ with alternative CNN-based methods

Table 15 shows the comparison of RANKGP-3M-CNN++ with alternative CNN-based methods. Except for the *Training From Scratch* method, these methods are briefly described in section 3.2. Indeed, for a fair comparison, RANKGP-3M-CNN++ and the alternative methods use the dataset *splits* described in subsection 6.5.1. Following are the implementation details for

Figure 29 – **Comparison of accuracy rates between GP-CNN*, RANKGP-CNN++, and RANKGP-3M-CNN++**. GP-CNN* presents the GP-CNN results on the layer, where it achieved the highest average accuracy rate (see Table 13).Regarding RANKGP-CNN++ and RANKGP-3M-CNN++, we only show the results associated with ANOVA.



Source: Adapted from M. Condori and Bruno (2021).

each alternative CNN-based method.

- *Training From Scratch.* We trained three CNN models (BN-VGG-19, INCEPTION-V3, and RESNET-50) with the stochastic gradient descent (SGD) algorithm. Throughout each experiment, SGD was supplied with the following input parameters: (i) a learning rate of 0.001, (ii) a momentum of 0.9, and (iii) a weight decay of zero. Additionally, we used the default random initialization provided by PyTorch 0.4.1 (PASZKE *et al.*, 2017). Finally, the training process consisted of 150 epochs with mini-batches of eight texture images.

- Standard Fine-tuning. We fine-tuned the BN-VGG-19, INCEPTION-V3, and RESNET-50 models, using the same settings as the *Training From Scratch* method (except for the initialization policy). Furthermore, we used the official TorchVision 0.2 library to import the pre-trained CNN models into PyTorch.

- FV-CNN. We included the results published in the original FV-CNN paper (CIMPOI *et al.*, 2016) and those from other research works (LIU *et al.*, 2019) when they employed the same dataset *splits* that we used for RANKGP-3M-CNN++.

- DEEPTEN. We evaluated DEEPTEN (RESNET50) using the code provided by its authors[2]. In this context, although we tried to keep the same hyper-parameters as in the original paper (ZHANG; XUE; DANA, 2017), some changes were still necessary. (i) due to memory constraints, we were unable to use a multi-size training procedure. Rather, we used the standard method of resizing each image to a shortest dimension of 256 pixels (without changing its aspect ratio). Then, to form mini-batches, each resized image is

[2] DeepTen GitHub repository: <https://github.com/zhanghang1989/PyTorch-Encoding>

randomly cropped to N pixels. (ii) In addition to random cropping, we did not apply any data augmentation techniques.

- SPOTTUNE. In the same way as DEEPTEN, we evaluated several texture datasets using SPOTTUNE's source code[3]. In more detail, we performed all experiments on the RESNET-26 model, since it was the only model available within the published code. However, although we tried to maintain the same hyper-parameters as in the original paper (Guo *et al.*, 2019), the following changes were necessary. (i) Because SPOTTUNE is composed of two RESNET copies and a policy network, even on a RESNET-26 model it was not possible to allocate mini-batches greater than 30 samples on a single 11GB GPU core. Therefore, we ran our experiments with two batch sizes: 15 and 30 samples. The highest accuracy among both settings is reported in Table 15. (ii) Because the source code provided different values for the weight decay hyper-parameter, we used the most common one: "zero" throughout the experiments. (iii) Lastly, we resized and cropped the images the same way as in DEEPTEN.

- EASYTL. The evaluation was conducted using the code published by the authors of EASYTL[4]. EASYTL differs from previous TL methods in that its only parameter is the intra-domain alignment algorithm. For this purpose, we used the CORAL algorithm as suggested in the EASYTL original paper (Wang *et al.*, 2019). As EASYTL processes sets of feature vectors rather than sets of activation maps, we also had to choose which sets of feature vectors to provide to EASYTL. Therefore, to find if EASYTL can achieve high accuracy rates in instance-based datasets without having to resort to multi-layer feature extraction, we chose to perform feature extraction from the last convolutional layer of BN-VGG-19, INCEPTION-V3, and RESNET-50 using GP-CNN. Then, the resulting sets of feature vectors became the input for EASYTL.

All the CNN-based methods requiring GPU acceleration were run on a GeForce GTX 1080 Ti. Below are the highlights of Table 15.

1. Regarding the *Training from scratch* and *standard fine-tuning* methods, their results were not as good as those obtained through more sophisticated CNN-based methods.

2. Some alternative CNN-based methods achieved excellent results on category-based datasets. For instance, the accuracy rate achieved by DEEPTEN for the FMD dataset were much better than those reported in its original publication (85.0% vs. 80.2%). Accordingly, it seems that the changes we made to the DEEPTEN's hyper-parameters had a significant impact on its behavior. Additionally, the pre-trained RESNET-50 model we used may be more appropriate for FMD than the one used in the original paper.

[3] SpotTune GitHub repository: <https://github.com/gyhui14/spottune>
[4] EasyTL GitHub repository: <https://github.com/jindongwang/transferlearning/tree/master/code/traditional/EasyTL>

Table 15 – Comparison of accuracy rates (%) between RANKGP-3M-CNN++ and alternative CNN-based methods for texture recognition. (*) Results were extracted from (LIU *et al.*, 2016).

	CUReT	FMD	Kth-Tips2b	MBT	O10	O12	O13
BN-VGG-19 (*scratch*)	96.2 ± 0.5	25.0 ± 2.3	66.6 ± 3.3	31.6 ± 4.0	59.7	58.4	55.9
INCEPTION-V3 (*scratch*)	94.9 ± 1.3	30.5 ± 2.5	67.7 ± 6.1	83.6 ± 1.7	72.5	74.2	66.0
RESNET-50 (*scratch*)	90.4 ± 1.6	26.0 ± 1.4	69.0 ± 4.8	76.2 ± 1.9	70.6	72.4	76.2
BN-VGG-19 (*standard fine-tuning*)	87.7 ± 3.1	18.1 ± 2.7	59.2 ± 3.4	11.3 ± 4.3	30.1	37.8	40.0
INCEPTION-V3 (*standard fine-tuning*)	97.1 ± 1.6	36.5 ± 2.5	77.7 ± 4.6	87.8 ± 3.2	59.4	62.9	68.2
RESNET-50 (*standard fine-tuning*)	96.0 ± 1.3	30.5 ± 3.1	75.3 ± 6.8	82.0 ± 4.4	63.6	67.1	76.3
VGG-19 (FV-CNN)	99.0 ± 0.2	79.8 ± 1.8	88.2 (*)	-	80.0 (*)	82.3 (*)	-
RESNET-50 (DEEPTEN)	98.7 ± 0.7	**85.0 ± 1.2**	82.1 ± 2.3	81.4 ± 2.4	82.2	84.2	76.2
RESNET-26 (SPOTTUNE)	97.6 ± 0.9	30.5 ± 3.7	70.1 ± 7.9	97.3 ± 0.7	63.1	70.0	88.7
BN-VGG-19 (EASYTL)	84.2 ± 1.6	74.6 ± 1.5	84.3 ± 5.6	86.3 ± 1.1	86.2	85.5	84.0
INCEPTION-V3 (EASYTL)	84.9 ± 1.0	77.4 ± 1.4	81.2 ± 6.5	73.9 ± 1.3	87.6	87.0	73.4
RESNET-50 (EASYTL)	89.5 ± 1.5	81.1 ± 1.1	86.2 ± 7.7	89.5 ± 0.8	86.4	84.4	87.4
RANKGP-3M-CNN++ (ANOVA)	**99.8 ± 0.2**	82.6 ± 1.4	89.4 ± 4.0	98.3 ± 0.5	**98.2**	**97.6**	93.8
RANKGP-3M-CNN++ (MI)	**99.8 ± 0.2**	80.9 ± 1.7	**91.1 ± 4.5**	**98.4 ± 0.3**	97.0	96.8	**94.6**

Source: Adapted from M. Condori and Bruno (2021).

3. All alternative CNN-based approaches failed to perform well on two or more different instance-based datasets. These results agree with those described in subsection 6.5.2, which suggested that the final convolutional layer of a pretrained CNN model performs much better for category-based datasets than for instance-based datasets. In contrast, SpotTune achieved high accuracy (97.3%) in the MBT dataset, which indicates that fine-tuning methods can successfully address the previous issue when given the appropriate hyper-parameters. Unfortunately, despite testing two sets of hyper-parameters for SpotTune, they were only successful on MBT. Consequently, we agree with Li *et al.* (2020), who state that identifying the correct hyper-parameters for a dataset is a very resource-intensive and challenging process that requires a well-designed search plan.

4. Based on EASYTL's accuracy rates in instance-based datasets, it appears that semi-supervised learning did not improve the quality of texture information provided by the last convolutional layer of a pre-trained CNN model.

5. Overall, our method performs better on both category-based and instance-based datasets. In this sense, though our method combines the predictive power of three CNN models, the feature extraction part of our method is computationally more efficient than that of the alternative CNN-based methods. Additionally, the computational cost of the selection strategy is very low when using ANOVA.

6.9 Final considerations

For the target task of texture recognition, this chapter examined the quality of different sets of 2D activation maps with global pooling layers. In more detail, we proposed a generalized version of the TL strategy that uses a pre-trained CNN model as a feature extractor (e.g. GAP-CNN, or FC-CNN). In this generalization, each CNN model is viewed as a collection of deep composite functions (DCFs), each responsible for calculating one activation map per input image. Selection strategies were also presented to reduce the large number of DCFs in a CNN model, resulting in the generation of medium-sized feature vectors.

The main findings of this chapter were the following. (i) DCFs that perform well on category-based datasets are likely to perform poorly on instance-based datasets (and vice versa). (ii) The DCFs from shallower layers can have competitive predictive power. Indeed, using these layers, we obtained good accuracy rates on datasets where previous CNN-based methods performed poorly (e.g. `Outex12`) (LIU *et al.*, 2019). (iii) We found some slight evidence that GMPT can extract more relevant texture information than GAP. This point needs further research. (iv) Domain-specific knowledge can be included via a meta-dataset to guide the selection of proper DCFs. (v) Unlike the previous chapter, where combining two CNN models mostly yielded poor performance gains, this chapter successfully combines the predictive power of multiple CNN models, especially on instance-based texture datasets. Therefore, our final model achieved excellent results on both category-based and instance-based datasets, without having to compute huge feature vectors. Furthermore, given the simplicity and generality of our method, it can be easily applied to newer CNN models such as SENet (Hu *et al.*, 2019) or EfficientNet (TAN; LE, 2019).

CHAPTER 7 EXPLORING DETECTION

7.1 Initial considerations

In the previous chapters, we examined TL strategies for image classification tasks. Specifically, we explored how earlier layers can contribute powerful features that allow pre-trained CNN models to reach higher levels of generalization. This chapter will also study the predictive power of layers at multiple depth levels but focusing on object detection tasks. In more detail, this chapter proposes TL strategies for solving two biological tasks: (i) pollen grain detection and (ii) stomata detection.

In general, both tasks are challenging and lead to multiple applications. The detection of pollen grains, for instance, can aid in allergy diagnosis (PABLOS *et al.*, 2016), climate change analysis (D'AMATO *et al.*, 2015), and pesticide monitoring during agronomic seasons (BÖHME *et al.*, 2018). On the other hand, stomata detection can help biologists to study the possible correlations between the stomatal morphological features and the plant productivity/transpiration efficiency across different species (MORALES-NAVARRO *et al.*, 2018; CAINE *et al.*, 2019).

As explained in section 2.3, object detection tasks are more challenging than image classification tasks for the following reasons. (i) There are usually multiple objects within an image, which can overlap. (ii) Objects come in a variety of shapes and sizes. (iii) Objects can differ in size by several orders of magnitude. In this context, the inherent complexity of biological imaging could increase the task difficulty even further. For example, microscopic images of the plant leaf epidermis can contain anywhere from 5 to 400 stomata depending on the magnification factor used. Therefore, the architectural design of the object detector must take into account the morphological characteristics of the micro-structure it wants to detect.

By reviewing the literature for both grain pollen and stomata detection, we observed two patterns: (i) the researchers directly train a popular object detector, or (ii) the researchers build from scratch the design of a new detector. In the first case, the design of the detector disregards the essential morphological characteristics of the pollen grains (or stomata), while in the second case, the design of the detector ignores the new architectural innovations constantly proposed by the scientific community. Therefore, in this chapter, we propose TL strategies that results from an in-depth analysis of the particular target task while taking into consideration the innovative architectural designs of popular object detectors.

7.2 Target task, datasets and pre-trained models

This chapter deals with two target tasks. (i) Pollen grain detection and (ii) Stomata localization. Given an input image I of size (3000×2500), the first target task involves enclosing within bounding boxes all the pollen grains in the image and then assigns each bounding box to a valid target class. As for the stomata localization, since there is only one foreground class, it only requires the object detector to locate and enclose all valid stomata within bounding boxes.

Regarding the datasets, they initially came without bounding boxes, so manual labeling was performed under the supervision of an expert. The details of the pollen and stomata datasets are in subsection 4.2.4 and subsection 4.2.3, respectively. Although pollen grain detection appears to be a more challenging task than stomata localization, the opposite is true since the stomata dataset originates from multiple sources, containing a variety of plant species, magnification factors, sample preparation techniques, and imaging methods.

In both tasks we performed experiments with the following pre-trained CNN models DenseNet-121, DenseNet-201, ResNet-34, ResNet-50, SEResNeXt-50_32x4D, and SEResNeXt-101_32x4D. Additionally, in the pollen grain detection task we included the backbones from SEResNet-50 and SENet-154, while in the stomata localization task, we included the backbones from the EfficientNet family.

The next sections present PollenDet and StomaDet, two simple and fast one-stage fully-convolutional detectors that can process images of up to (3000×2500) within microseconds. Additionally, it is worth mentioning that in this chapter the terms “activation maps” and “feature maps” will be used interchangeably.

7.3 PollenDet

Microscopic images of pollen grains are significantly different than *everyday* photographs such as people faces or buildings. Therefore, the direct application of object detectors such as Faster R-CNN or RetinaNet to detect and classify pollen grains could lead to poor results. In this sense, the main differences between pollen grains and *everyday* objects are:

1. *Square bounding boxes*: Since pollen grains are nearly spherical, they can be enclosed in bounding boxes with aspect ratios of approximately 1:1, while *everyday* objects need a

more extensive range of aspect ratios.

2. *Similar scale*: Given a microscopic image, pollen grains belonging to the same type are approximately of the same scale. On the other hand, *everyday* objects can be found at multiple scales in the same image.

3. *Little intersection*: There is little or not intersection between two or more pollen grains inside a light microscopic image.

Using the information above, we developed POLLENDET, a fast and accurate TL strategy for detecting pollen grains in microscopic images.

7.3.1 General architecture

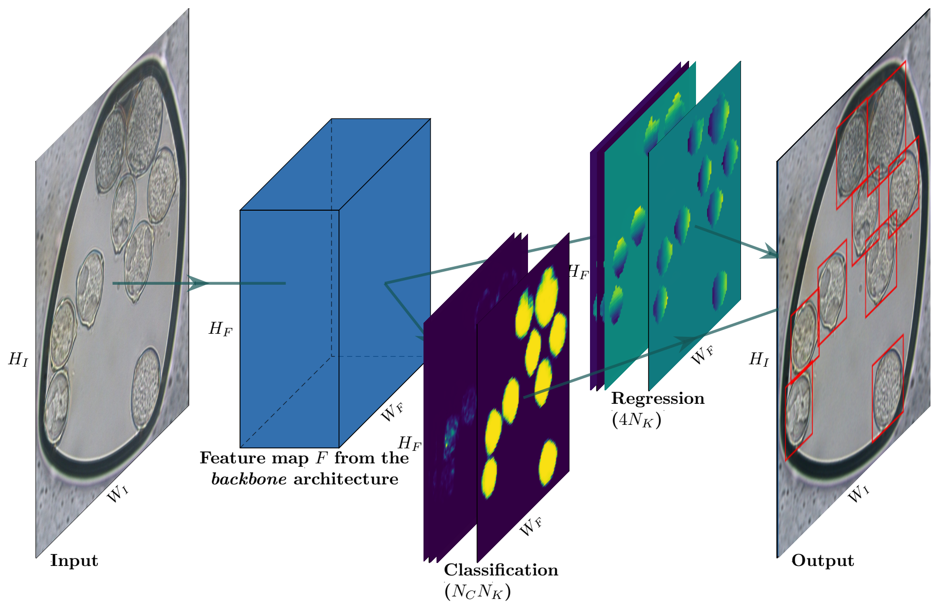
As shown in Figure 30, the architecture of POLLENDET is composed of three main components: (i) The backbone, (ii) the regression head, and (iii) the classification head. The first component is in charge of computing a activation map F from a given input image I. Then, F is processed by the second and third components, which respectively generates $4N_K$ regression coefficients and $N_K * N_C$ classification scores for each spatial position in F. The letters N_K and N_C are respectively the number of different *anchor* box sizes and the number of classes. In this sense, the definition of *anchor* box is the same as in Ref. (REN *et al.*, 2017), that is: the default bounding box associated with one spatial position in F. Since pollen grains are approximately spherical, only three square anchor box sizes are used in POLLENDET: $K = \{128\times128, 181\times181, 256\times256\}$, $N_K = 3$. Regarding the number of classes (N_C), there are two tasks that are addressed by POLLENDET.

1. *Localization* task: It involves the prediction of bounding boxes associated with pollen grains. Therefore, there are only $N_C = 2$ classes, which correspond to the foreground (the pollen grain) and background classes.

2. *Localization + classification* task: In addition to locating the pollen grains in I, it also involves identifying their types. Therefore, N_C is equal to the number of pollen types plus the background class.

The regression and classification components are both constituted by a single convolutional layer of kernel size: (1×1). We tested configurations of two and three convolutional layers but only marginal improvements were obtained. At training time, the regression and classification outputs are each compared against an encoded ground truth. On the other hand, at test time, both outputs are combined and decoded to produce the predicted bounding boxes. Detailed information about the training and test processes can be found in subsection 7.3.2.

Figure 30 – **General architecture of our proposed method.** The input image I is processed by the backbone. Then, the resulting activation map F is further processed by two (1×1) convolutional layers, which produce one map of regression coefficients and one map of labels scores. At test time, both outputs are combined and decoded to obtain the predicted bounding boxes.



Source: Elaborated by the author.

7.3.2 Implementation details

As shown in Figure 31, there is a common architectural pattern among the CNN models above: all of them are composed of six *chunks* or blocks, which are placed one after another. Also, there is one sub-sampling layer between every two contiguous *chunks*. We mean by *chunk*: the specific configuration of different type of layers dictated by the CNN model. Regarding the sub-sampling layers, they reduce the spatial resolution of their input activation maps by a factor of two.

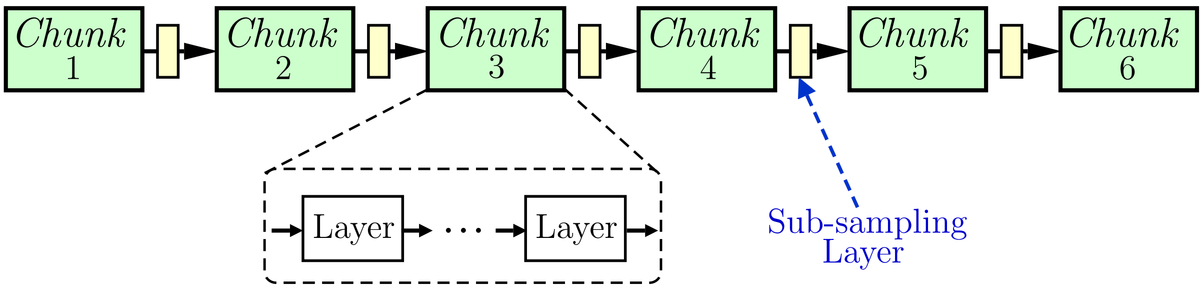
In this context, the backbone of POLLENDET is designed to reuse the same structure and parameter values of a pre-trained CNN model, starting from the first *chunk* up to a finite number S of *chunks*, $1 \leq S \leq 6$. The parameter S also defines the spatial resolution of the activation map F yielded by the backbone as follows:

$$W_F = \frac{W_I}{2^{S-1}}, \qquad H_F = \frac{H_I}{2^{S-1}} \qquad , \tag{7.1}$$

where W_F and H_F are respectively the width and height of F. On the other hand, I is the input image of size $(W_I \times H_I)$, which is to be processed by the backbone. As a consequence, each spatial position in F has a corresponding spatial block of size $(2^{S-1} \times 2^{S-1})$ in I.

Choosing the number of reused *chunks* is very important. For example, in the case of object detectors, the level of semantic information that can be extracted from a particular

Figure 31 – **The common architectural pattern of CNN models**. The convolutional part of all the CNN models considered in this chapter is composed of six *chunks*. Between every two contiguous *chunks*, there is a sub-sampling layer in charge of reducing the width and height of its input activation map by a factor of two.



Source: Elaborated by the author.

backbone, generally increases as it is composed of more *chunks*. However, as indicated in (LIU *et al.*, 2016; Lin *et al.*, 2017), this increment is always as the cost of losing spatial resolution in F, which is detrimental to the detection of small objects. This situation can be repeated in microscopic images, since pollen grains can be pretty small. Therefore, in subsection 7.3.5 we perform several experiments to find the optimal number of *chunks* to be reused.

The convention used for naming any backbone is as follow: <CNN model name>-<number of reused *chunks*>S. For example, a backbone composed by the first three *chunks* of the RESNET152 model is written as RESNET152-3S.

7.3.2.1 Training time: Ground truth encoding

At training time, given an image I of spatial size $(W_I \times H_I)$ and a set of N_G ground truth bounding boxes $G = \{g_1, g_2 \ldots, g_{N_G}\}$, an encoding procedure is performed by projecting each g_i $(1 \leq i \leq N_G)$ into the spatial resolution $(W_F \times H_F)$ of the activation map F yielded by the backbone. The steps of this procedure are as follows:

1. It starts by placing N_K anchor boxes of different sizes at the center of every $(2^{S-1} \times 2^{S-1})$ spatial block in I, where S is the number of reused *chunks* in the backbone. Since there is a one-to-one relationship between the spatial blocks in I and the spatial positions in F, there is a total of $N_K * W_F * H_F$ anchor boxes. In POLLENDET, the following $N_K = 3$ anchor box sizes are considered: $K = \{128 \times 128, 181 \times 181, 256 \times 256\}$.

2. Then, the *Intersection-Over-Union* (IoU) metric is computed between all pairs of ground truth and anchor bounding boxes. Next, each anchor box is linked to the ground truth bounding box with which it has reached the highest IoU overlap.

3. Each anchor box and its linked ground truth box yields one label l and four bounding box regression coefficients $t = [t_x, t_y, t_w, t_h]$. A positive label $(l > 0)$ is assigned when their

IoU overlap is at least 0.5. A negative label ($l = -1$) is assigned when their IoU overlap is between 0.4 and 0.5. In all other scenarios, a label of zero ($l = 0$) is given. In *localization* tasks, all positive labels are always set to number one ($l = 1$), while in *localization + classification* tasks $l \in \{1, 2, ..., N_C - 1\}$. On the other hand, the regression coefficients t are computed by measuring the "distance" between their bounding box coordinates. This "distance" is computed using the same formulas proposed in the Faster R-CNN paper (REN *et al.*, 2017).

4. All labels (l) and regression coefficients (t) are grouped by their associated anchor box size. Therefore, at the end of this encoding procedure, five 2-D tensors of size $(W_F \times H_F)$ are generated for every unique anchor box size k: $\{\mathcal{L}^{(k)}, \mathcal{T}_x^{(k)}, \mathcal{T}_y^{(k)}, \mathcal{T}_w^{(k)}, \mathcal{T}_h^{(k)}\}$, where $k \in K$. The first 2-D tensor ($\mathcal{L}^{(k)}$) is composed of only labels. The second 2-D tensor ($\mathcal{T}_x^{(k)}$) contains the first regression coefficients. The next 2-D tensor contains the second regression coefficients, and so on. Since there are $N_K = 3$ anchor box sizes, a total of fifteen 2-D tensors are computed for every input image I.

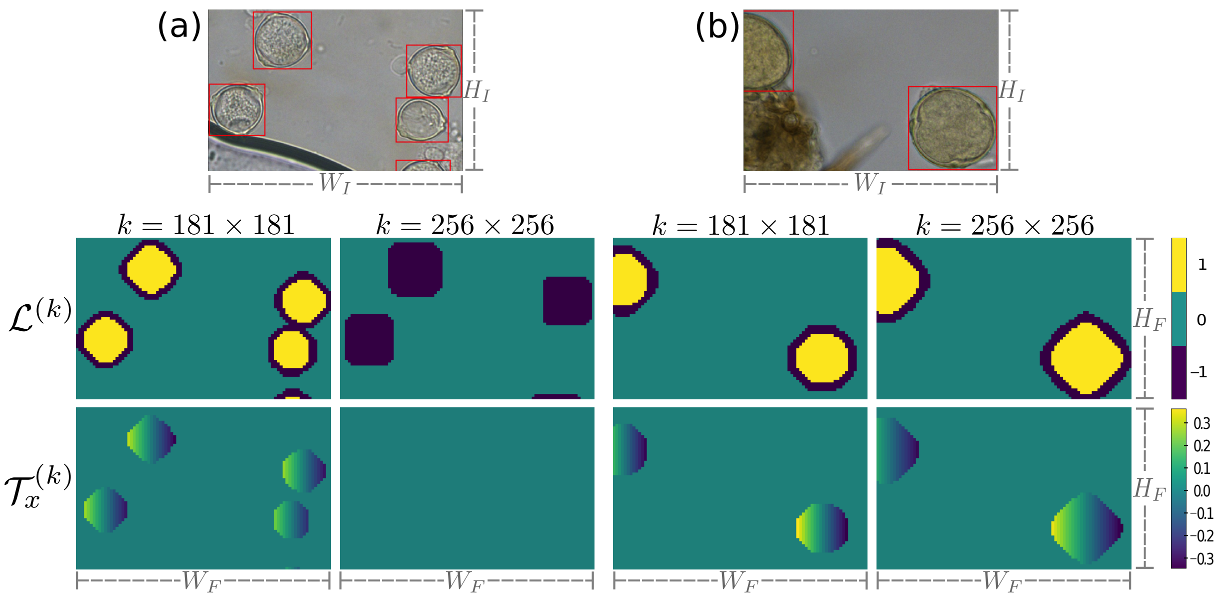
Figure 32 shows examples of $\mathcal{L}^{(k)}$ and $\mathcal{T}_x^{(k)}$ at two different anchor sizes ($k = 181 \times 181$ and $k = 256 \times 256$) for the task of pollen localization. Also, as shown in Figure 32a, $\mathcal{L}^{(k)}$ can sometimes contain only zero and negative labels, which is the result of the anchor size k being too large (or too small) for the given ground truth bounding boxes. This property ensures each anchor size to be only associated to ground truth bounding boxes that are not very different in scale.

7.3.2.2 Training the pollen dataset

Every training sample is represented by three terms: (I, $\mathcal{L}$, $\mathcal{T}$), where I is the original image, $\mathcal{L}$ is the set of N_K tensors with label coefficients and $\mathcal{T}$ is the set of $4N_K$ tensors with regression coefficients. The spatial size of I is denoted by $(W_I \times H_I)$, and the spatial size of every tensor in $\mathcal{L}$ and $\mathcal{T}$ is denoted as $(\frac{H_I}{2^{S-1}} \times \frac{W_I}{2^{S-1}})$ or $(W_F \times H_F)$, where S is the number of reused *chunks* in the backbone.

As shown in Figure 30, for any input I, our model predicts another set of $4N_K$ tensors with regression coefficients (denoted by $\widehat{\mathcal{T}}$) and a set of $N_C * N_K$ score tensors (denoted by $\widehat{\mathcal{L}}$), where N_C is the number of classes in the task at stake. To measure the prediction power of $\widehat{\mathcal{T}}$ and $\widehat{\mathcal{L}}$, they are compared with their ground-truth counterparts through two loss functions: (i) the cross-entropy loss or $CE_{loss}(\mathcal{L}, \widehat{\mathcal{L}})$, which measure the classification prediction of $\widehat{\mathcal{L}}$ and (ii) the smoothed L1 loss or $SL1_{loss}(\mathcal{T}, \widehat{\mathcal{T}})$, which is in charge of measuring the regression prediction of $\widehat{\mathcal{T}}$. Some important implementation details of these loss functions are as follows: (i) the CE_{loss} do not take into account the predicted scores whose corresponding ground truth labels are negative. (ii) The $SL1_{loss}$ only considers the predicted regression coefficients whose corresponding predicted labels are correctly classified as foreground. Then, the total loss T_{loss} is

Figure 32 – **Examples of $\mathcal{L}^{(k)}$ and $\mathcal{T}_x^{(k)}$ at two different anchor box sizes**. The yellow and green areas in $\mathcal{L}$ correspond to the foreground and background labels respectively. The regression coefficients are only active when their corresponding label coefficients are positive. Although it seems the opposite, the spatial size of the original images $(H_I \times W_I)$ is much larger than the spatial size $(W_F \times H_F)$ of their corresponding $\mathcal{L}^{(k)}$ and $\mathcal{T}_x^{(k)}$. (a) Corylus pollen grains, (b) Fagus pollen grains.



Source: Elaborated by the author.

defined as the pondered sum of those previously losses, which empirically is set to:

$$T_{loss} = (0.4)CE_{loss}(\mathcal{L},\widehat{\mathcal{L}}) + SL1_{loss}(\mathcal{T},\widehat{\mathcal{T}}) \tag{7.2}$$

As usually happens in the task of *localization* + *classification* ($N_C > 2$), some foreground classes have significantly more samples than the others. This is commonly known as the *imbalanced class problem* and can be partially solved by adding a vector of N_C class weights coefficients ($\vec{w}_{class}$) to the CE_{loss}. In this sense, $\vec{w}_{class}$ is used to balance the loss contribution of each class by focusing more in the classes with less samples. In classification tasks, the coefficients of $\vec{w}_{class}$ are generally defined as inversely proportional to the number of training samples of their respective classes. However, we found this definition insufficient for tasks that involve localization as well. Therefore, for each foreground class, the total area of its ground truth bounding boxes is also calculated. Without considering the background class, $\vec{w}_{class}$ is computed in the training set as follows.

$$\vec{w}_{class}[c > 0] = \frac{1}{2}\texttt{norm}\left(\frac{1}{\texttt{count}(\mathcal{G})}\right) + \frac{1}{2}\texttt{norm}\left(\frac{1}{\texttt{area}(\mathcal{G})}\right) \quad , \tag{7.3}$$

where the subroutine `count(·)` returns a vector containing the total number of bounding boxes for each foreground class. Similarly, the subroutine `area(·)` returns a vector containing the total area covered by the bounding boxes of each foreground class. In this sense, the symbol $\mathcal{G}$

represents the set of all ground truth bounding boxes in the training set. The term c is the class identifier, where the background class is represented by $c = 0$, and the foreground classes are symbolized as $c > 0$. Therefore, $\vec{w}_{class}[c > 0]$ can be read as the vector of weight coefficients for the foreground classes. In addition, the subroutine `norm(·)` is used to divided an input vector by its mean value. This normalization is important because it ensures that the total sum of coefficients in $\vec{w}_{class}[c > 0]$ is equal to $N_C - 1$. Finally, the weight coefficient for the background class is always set to number one ($\vec{w}_{class}[0] = 1$)

By applying a fine-tuning approach with an ADAM optimizer, PollenDet reduces the loss function T_{loss}. This optimization updates the trainable parameters of PollenDet, including those from the backbone. In this context, before fine-tuning, the training data (a.k.a. training *split*) is divided into two non-overlapping sets. Seventy percent of the training data is put into the training set, and 30 percent goes into the validation set.

At each epoch, instead of using whole images of size (1280×960), random crops of size (640×480) are used to train PollenDet, making sure that there is at least one pollen grain in every random crop. Furthermore, the resulting crops are also horizontally and/or vertically flipped with a 50% chance of probability. The above steps are applied to both the training and validation sets. Then, training and validation losses are computed with Equation 7.2 and Equation 7.3. PollenDet uses the validation loss to monitor the training process by saving a copy of its trainable parameters each time the validation loss reaches a new minimum value. The last saved version is retrieved after the fine-tuning process runs for 800 epochs with two images per batch and a learning rate of 0.001.

7.3.2.3 Testing new pollen images

Similar to the training procedure, our trained model transforms a test image I into a set of $4N_K$ tensors of regression coefficients ($\widehat{\mathcal{T}}$) and $N_C * N_K$ tensors of score coefficients ($\widehat{\mathcal{L}}$) (see Figure 30). Once again, every tensor is bi-dimensional with a spatial size of $(W_F \times H_F)$ or $(\frac{H_I}{2^{S-1}} \times \frac{W_I}{2^{S-1}})$, where S is the number of reused *chunks* in the backbone. To perform the bounding box prediction, both $\widehat{\mathcal{L}}$ and $\widehat{\mathcal{T}}$ are processed as follows:

- **Step 1:** The set of tensors $\widehat{\mathcal{L}}$ and $\widehat{\mathcal{T}}$ are each partitioned into N_K subsets according to their associated anchor box sizes. Therefore, each subset $\widehat{\mathcal{L}}^{(k)}$ and $\widehat{\mathcal{T}}^{(k)}$ $(k \in K)$ contains respectively N_C and four bi-dimensional tensors of spatial size $(W_F \times H_F)$.

- **Step 2:** Each spatial position in $\widehat{\mathcal{L}}^{(k)}$ can be seen as a vector of N_C score coefficients, where the position of each of them indicates the corresponding class label. In this sense, the class probabilities of those N_C scores are computed by using the softmax function. Then, the spatial position is assigned to the label ($\hat{l}$) with the highest probability, if and only if such probability is above a certain threshold t_{sm}, otherwise, the spacial position is assigned a background label ($\hat{l} = 0$). This process is repeated for all spatial positions in $\widehat{\mathcal{L}}^{(k)}$.

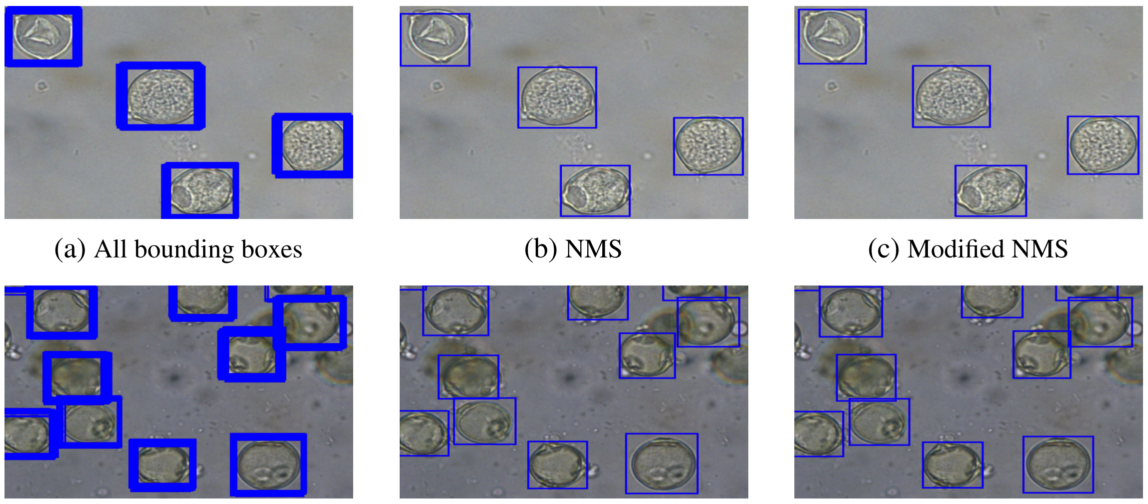
- **Step 3:** Since there is a one-to-one relationship between the spatial positions of $\widehat{\mathcal{L}}^{(k)}$ and $\widehat{\mathcal{T}}^{(k)}$, we retrieve a predicted set of four regression coefficients ($\hat{t} = [\hat{t}_x, \hat{t}_y, \hat{t}_w, \hat{t}_h]$) from $\widehat{\mathcal{T}}^{(k)}$, each time there is a predicted foreground label ($\hat{l} > 0$) in $\widehat{\mathcal{L}}^{(k)}$ at some spatial position (x, y). Then, $\hat{t}$ is used to modify the bounding box coordinates of the anchor box of size k placed at position (x, y). This modification uses the same formulas as in (REN *et al.*, 2017) and the result becomes one bounding box proposal. The set of computed bounding box proposals are stored in a vector $\vec{v}_{bb}$ of size $(N_f \times 4)$, where N_f is the total number of foreground labels computed in step 2. Also, for each bounding box proposal, its computed probability and predicted label are stored in vectors $\vec{v}_p$ and $\vec{v}_{\hat{l}}$ respectively, each of size (N_f).

- **Step 4:** As shown in Figure 33a and Figure 33d, there are lots of bounding box proposals in $\vec{v}_{bb}$ that need to be pruned. Usually, the non-maximum suppression (NMS) algorithm is applied at this stage. NMS first ranks $\vec{v}_{bb}$ and $\vec{v}_{\hat{l}}$ by sorting $\vec{v}_p$ from high to low. Then, it greedily keeps the bounding box proposals with the highest associated probabilities and eliminates all other candidates that overlap with them. In this sense, a lower-ranked bounding box proposal is considered to be overlapping a higher-ranked one if its *Intersection-over-Union* metric (IoU) is greater than a predefined threshold t_{nms}. Two exemplary outputs of the NMS algorithm are presented in Figure 33b and Figure 33e. However, since there is no bounding box refinement at training time as in the Faster RCNN method, the direct application of the NMS algorithm can yield poorly located bounding boxes. Therefore, we leverage the specific pollen characteristics explained at the beginning of section 7.3 to modify the NMS algorithm, so that, instead of removing all the overlapping bounding box proposals, we create a refined bounding box by taking its average. As can be observed in Figure 33c and Figure 33f, the modified NMS usually achieves better localization results than the original NMS algorithm. More about this topic in subsection 7.3.3. In the task of *localization + classification* ($N_C > 2$), the vectors $\vec{v}_{\hat{l}}$ and $\vec{v}_p$ are used to predict the foreground label of every refined bounding box by adding one more functionality to the modified NMS algorithm. This functionality computes, for each set of overlapping bounding box proposals, the sum of their associated probability values for each different predicted label. Then, the refined bounding box is assigned to the predicted label that gets the largest sum.

7.3.3 Experimental Settings

As explained in subsection 4.2.4, we used the stratified holdout validation method with $m = 10$ to create 10 random *splits*. In addition, we also used the general evaluation scheme

Figure 33 – **Comparison between the NMS algorithm and our modified version**. Our modified NMS algorithm generates better-located bounding boxes than the classical NMS algorithm. (a-c) Corylus pollen grains, (d-f) Betula pollen grains.



(a) All bounding boxes (b) NMS (c) Modified NMS

(d) All bounding boxes (e) NMS (f) Modified NMS

Source: Elaborated by the author.

described in subsection 4.3.1 to train and evaluate PollenDet. In this context, the TrainModel subroutine from Algorithm 1 corresponds to the fine-tuning approach described in subsubsection 7.3.2.2, and the Predict subroutine corresponds to the test approach described in subsubsection 7.3.2.3. By default, the thresholds t_{sm} and t_{nms} are respectively set to 0.7 and 0.1. Additionally, the evaluation metric $\mathcal{E}$ used to measure the robustness of PollenDet is the mean Average Precision, which is explained in subsection 4.3.3.

All experiments were performed in a GeForce RTX 2080 Ti graphic card. In addition, our code is mainly written in python 3.6, using the PyTorch 1.0.1 library. The pre-trained CNN models ResNet34, ResNet50, ResNet152, DenseNet121, and DenseNet201 were extracted from the official PyTorch package: Torchvision 0.2.1. On the other hand, SEResNet50, SEResNeXt50_32x4d, SEResNeXt101_32x4d and SENet154 were extracted from a popular alternative PyTorch package[1].

7.3.4 Comparison between the NMS algorithm and our modified version

As explained in subsubsection 7.3.2.3, we propose a modified version of the NMS algorithm to improve the prediction of bounding boxes. In this sense, Table 16 presents the comparison between the mAP (%) values reached by the NMS algorithm and the ones reached by our modified version for the task of *localization* + *classification* of pollen grains. This analysis is performed on 18 backbones, coming from the reuse of between four and five *chunks* of

[1] Cadene repository: <https://github.com/Cadene/pretrained-models.pytorch>

Table 16 – ***Localization + classification* performance comparison between the NMS algorithm and our modified version in 18 backbones.** The performance is measured with $mAP_{0.5}(\%)$ and $mAP_{0.75}(\%)$ values. Therefore, there is a total of 36 comparisons, in which our modified NMS always achieves better results than the original NMS algorithm.

	$S = 4$				$S = 5$			
	$mAP_{0.5}(\%)$		$mAP_{0.75}(\%)$		$mAP_{0.5}(\%)$		$mAP_{0.75}(\%)$	
CNN model	NMS	Modified	NMS	Modified	NMS	Modified	NMS	Modified
RESNET34	80.8 ± 5.4	**85.4 ± 2.5**	59.6 ± 7.6	**76.4 ± 3.4**	89.7 ± 2.8	**90.2 ± 2.6**	83.9 ± 3.8	**88.2 ± 2.0**
RESNET50	74.3 ± 9.0	**79.9 ± 8.7**	42.5 ± 9.6	**66.6 ± 9.4**	86.4 ± 3.5	**87.7 ± 2.8**	72.2 ± 7.3	**83.5 ± 4.3**
RESNET152	76.9 ± 4.9	**81.4 ± 6.2**	51.8 ± 8.5	**73.1 ± 4.6**	87.7 ± 5.4	**88.1 ± 4.3**	76.5 ± 6.2	**85.3 ± 4.1**
DENSENET121	78.9 ± 5.1	**85.6 ± 4.6**	46.8 ± 8.3	**72.4 ± 5.4**	87.4 ± 5.1	**89.3 ± 4.3**	73.7 ± 7.4	**85.1 ± 4.9**
DENSENET201	82.5 ± 6.4	**86.8 ± 4.2**	49.6 ± 9.7	**76.8 ± 4.1**	88.2 ± 5.3	**89.3 ± 4.1**	73.2 ± 6.8	**86.1 ± 4.0**
SERESNET50	75.2 ± 5.8	**82.7 ± 4.4**	39.8 ± 8.2	**66.2 ± 6.5**	81.3 ± 5.3	**84.3 ± 6.0**	60.5 ± 9.0	**77.2 ± 5.9**
SERESNEXT50_32X4D	78.3 ± 5.2	**85.3 ± 3.5**	42.6 ± 9.8	**70.7 ± 6.3**	89.3 ± 2.8	**91.2 ± 2.6**	77.8 ± 4.3	**88.0 ± 3.2**
SERESNEXT101_32X4D	76.0 ± 8.8	**81.3 ± 8.7**	46.0 ± 9.0	**68.1 ± 8.8**	87.3 ± 4.3	**90.2 ± 3.0**	75.1 ± 7.4	**86.0 ± 3.3**
SENET154	83.3 ± 4.6	**88.9 ± 4.3**	57.6 ± 7.9	**79.9 ± 4.9**	84.8 ± 7.2	**86.3 ± 6.5**	70.9 ± 11.3	**81.2 ± 7.4**

Source: Elaborated by the author.

nine pre-trained CNN models. In all cases, our modified version achieved better mAP values than the original NMS algorithm. However, the improvement in the $mAP_{0.5}$ values is not very high in backbones that reuse $S = 5$ *chunks*. Indeed, there is a notorious higher difference in the comparison of $mAP_{0.75}$ values than in the comparison of $mAP_{0.5}$ values, which suggests that our modified version contributes more to ensure that the predicted bounding boxes are better located than to guarantee their better classification.

7.3.5 Analysis of backbones with different number chunks

As explained in subsection 7.3.2, finding good backbones is vital to the overall performance of object detectors. In this regard, the number of reused *chunks* in the backbone is an important parameter to consider. Table 17 shows the *localization* + *classification* mAP results associated with sixteen backbones. These backbones reuse between 3 to 6 *chunks* from four pre-trained CNN models. In all cases, there is a trend which suggest that better mAP values are achieved as more *chunks* are reused. This tendency stops at $S = 6$, in which worse mAP values are obtained. Although it is not shown, similar results were obtained in other pre-trained CNN models such as RESNET152 or SERESNET50. Therefore, from now on, we only present the results of backbones with $S = 5$ *chunks*.

7.3.6 Comparison of backbones from different CNN models

Table 18a and Table 18b respectively show the $mAP_{0.5}(\%)$ and $mAP_{0.75}(\%)$ results associated with nine backbones for the tasks of (i) *detection* ($N_C = 2$) and (ii) *detection* + *classification* ($N_C = 7$) of pollen grains. In this sense, to improve our analysis in the *detection* + *classification* task, Table 18 also includes the AP values achieved in each foreground class. In general, two expected phenomenons are seen at Table 18a and Table 18b.

Table 17 – ***Localization + classification* performance results of POLLENDET with backbones that have 3 to 6 *chunks*.** The performance is measured with $mAP_{0.5}(\%)$ values, in which the backbones with five *chunks* always achieve the best results.

CNN model/# *chunks*	$S = 3$	$S = 4$	$S = 5$	$S = 6$
RESNET34	76.2 ± 8.8	85.4 ± 2.5	**90.2 ± 2.6**	83.4 ± 4.4
SERESNEXT50_32X4D	72.6 ± 5.5	85.3 ± 3.5	**91.2 ± 2.6**	89.2 ± 2.7
DENSENET121	70.4 ± 6.5	85.6 ± 4.6	**89.3 ± 4.3**	81.0 ± 3.7
DENSENET201	72.8 ± 5.1	86.8 ± 4.2	**89.3 ± 4.1**	76.4 ± 9.7

Source: Elaborated by the author.

1. All $mAP_{0.5}$ results are always higher than their corresponding $mAP_{0.75}$ values. The reason for this is in the definition of IoU, in which it is much harder to achieve an IoU above 0.75 than an IoU above 0.5.
2. All $mAP_{0.5}$ and $mAP_{0.75}$ results in the *detection* task are always higher than those of the *detection + classification* task due to the difference in their number of classes.

About the specific results of Table 18a, the best backbone in the *detection + classification* task is SERESNEXT50_32X4D-5S with $mAP_{0.5}$=91.2%. However, there is no clear "winner" in the *detection* task, since most backbones achieve $mAP_{0.5}$ values between 96.1% and 96.5%. The analysis of each individual foreground class is presented in the following lines:

- **Betula pollen**: It is by far the most difficult pollen type to detect. The best $AP_{0.5}$ results are achieved by SERESNEXT101_32X4D-5S and SERESNEXT50_32X4D-5S with 80.4% and 78.3% respectively. However, at both cases, the standard error is high (> 12%), which means that there are some *splits* of the dataset that achieve very good results (above 90%), and other *splits* with no very good results (below 75%).
- **Corylus pollen**: Most methods achieve very accurate results (above 95%) and low standard error (below 2.5%). This could be explained by the relatively high amount of Corylus grains in the training set (approximately 68 grains per *split*). The best $AP_{0.5}$ result is achieved by SERESNET50-5S (97.3%).
- **Carpinus pollen**: There are three backbones with $AP_{0.5}$ results above 90%. However, although it is not as high as in the case of Betula pollen grains, the standard error is still large (> 7%). A possible reason for these outcomes is related to the few Carpinus pollen grains given to the training set at each *split* (approximately 10 grains).
- **Fagus, Quercus and Salix pollen grains**: Most backbones achieved $AP_{0.5}$ results above 90%. In addition, there is a positive correlation between the number of training samples and the computed $AP_{0.5}$ values. In this sense, the best $AP_{0.5}$ results for Fagus, Quercus and Salix are respectively 93.8%, 95.1% and 95.6%, which in turn have corresponding 12, 16

Table 18 – **Detailed analysis of the performance results of PollenDet at different backbones.** This analysis is carried out at the tasks: (i) *localization* and (ii) *localization + classification* of pollen grains. In the second task, the results on each pollen type are also included. (a) The performance is measured with $mAP_{0.5}$ values. (b) The performance is measured with $mAP_{0.75}$ values.

(a) Mean Average Precision with $t_{\mathtt{IoU}} = 0.5$

	Localization	*Localization + classification*						
backbone	$mAP_{0.5}$	Betula	Corylus	Carpinus	Fagus	Quercus	Salix	$mAP_{0.5}$
ResNet34-5S	**96.5** ± **1.9**	75.5 ± 8.7	96.4 ± 2.2	90.8 ± 8.3	93.5 ± 6.2	93.2 ± 4.3	91.7 ± 2.6	90.2 ± 2.6
ResNet50-5S	**96.5** ± **2.9**	73.2 ± 14.1	95.5 ± 2.4	84.3 ± 11.4	86.7 ± 9.4	92.1 ± 4.9	94.4 ± 3.4	87.7 ± 2.8
ResNet152-5S	96.2 ± 1.5	72.1 ± 15.2	93.4 ± 3.8	83.4 ± 10.5	**93.8** ± **4.7**	93.5 ± 4.3	92.6 ± 4.6	88.1 ± 4.3
DenseNet121-5S	95.7 ± 1.8	73.7 ± 12.5	95.5 ± 3.7	86.6 ± 8.7	91.4 ± 8.8	**95.1** ± **4.8**	93.9 ± 2.5	89.3 ± 4.3
DenseNet201-5S	96.3 ± 2.4	71.0 ± 14.2	96.0 ± 1.8	90.3 ± 10.0	92.0 ± 6.3	94.1 ± 5.0	92.5 ± 3.9	89.3 ± 4.1
SEResNet50-5S	93.8 ± 3.1	71.1 ± 13.1	**97.3** ± **2.0**	80.4 ± 21.8	72.9 ± 20.1	88.3 ± 8.9	**95.6** ± **3.0**	84.3 ± 6.0
SEResNeXt50_32x4d-5S	**96.5** ± **2.2**	79.3 ± 12.5	96.1 ± 2.0	**91.2** ± **7.3**	91.0 ± 6.0	94.8 ± 3.4	94.9 ± 3.4	**91.2** ± **2.6**
SEResNeXt101_32x4d-5S	96.2 ± 2.2	**80.4** ± **12.3**	96.6 ± 1.5	87.4 ± 6.6	86.9 ± 8.6	94.6 ± 4.0	95.2 ± 2.7	90.2 ± 3.0
SENet154-5S	96.1 ± 2.6	73.7 ± 12.2	91.6 ± 8.2	84.9 ± 11.3	87.1 ± 20.6	89.8 ± 8.0	90.8 ± 7.7	86.3 ± 6.5

(b) Mean Average Precision with $t_{\mathtt{IoU}} = 0.75$

	Localization	*Localization + classification*						
backbone	$mAP_{0.75}$	Betula	Corylus	Carpinus	Fagus	Quercus	Salix	$mAP_{0.75}$
ResNet34-5S	**94.5** ± **2.4**	73.8 ± 9.3	**93.7** ± **2.0**	**89.3** ± **8.5**	**90.0** ± **7.6**	90.5 ± 4.0	91.7 ± 2.6	**88.2** ± **2.0**
ResNet50-5S	90.6 ± 3.6	69.8 ± 14.3	91.8 ± 3.2	76.0 ± 15.7	81.8 ± 10.2	89.3 ± 6.8	92.1 ± 5.2	83.5 ± 4.3
ResNet152-5S	93.7 ± 2.1	67.8 ± 15.2	90.5 ± 4.8	81.0 ± 9.9	89.7 ± 6.5	91.9 ± 4.2	90.6 ± 6.0	85.3 ± 4.1
DenseNet121-5S	92.1 ± 3.1	61.8 ± 17.6	92.6 ± 3.4	84.5 ± 8.7	88.4 ± 8.7	90.7 ± 4.0	92.9 ± 3.5	85.1 ± 4.9
DenseNet201-5S	92.5 ± 3.0	66.9 ± 14.1	93.2 ± 3.3	84.4 ± 8.7	87.5 ± 6.5	**92.5** ± **5.1**	92.3 ± 3.8	86.1 ± 4.0
SEResNet50-5S	85.7 ± 5.0	60.9 ± 13.1	90.0 ± 4.3	72.1 ± 20.8	65.9 ± 20.8	83.8 ± 9.0	90.7 ± 6.8	77.2 ± 5.9
SEResNeXt50_32x4d-5S	92.3 ± 2.7	74.6 ± 11.4	92.5 ± 2.1	87.1 ± 11.0	87.7 ± 4.5	92.1 ± 4.8	**93.6** ± **5.4**	88.0 ± 3.2
SEResNeXt101_32x4d-5S	92.0 ± 3.2	**74.8** ± **10.9**	93.5 ± 3.3	80.6 ± 7.9	82.9 ± 6.8	91.5 ± 5.7	92.8 ± 3.6	86.0 ± 3.3
SENet154-5S	91.2 ± 4.4	69.2 ± 12.2	85.4 ± 11.8	79.9 ± 10.0	83.1 ± 23.4	81.1 ± 16.3	88.4 ± 11.6	81.2 ± 7.4

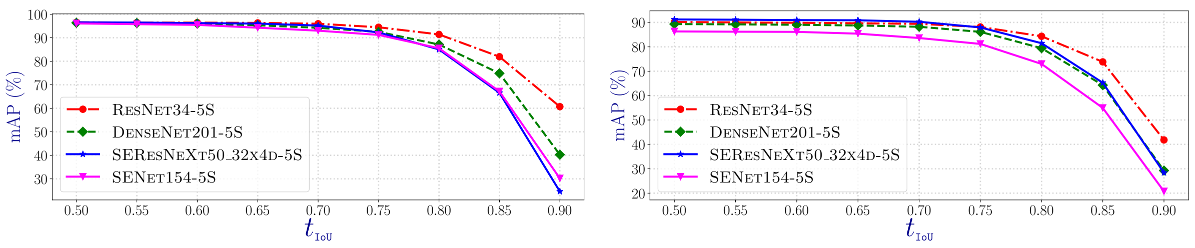
Source: Elaborated by the author.

and 21 training samples at each *split*. A similar correlation can be seen when considering the number of backbones with $AP_{0.5}$ results above 90%.

About the specific results in Table 18b, ResNet34-5S is the best backbone for both *localization* and *localization + classification* tasks with corresponding $mAP_{0.75}$ values of 94.5% and 88.2%. Also, ResNet34-5S obtains the best $AP_{0.75}$ results in the following foreground classes: Corylus (93.7%), Carpinus (89.3%) and Fagus (90%). This suggests that ResNet34-5S is capable of predicting better located bounding boxes than the rest of backbones evaluated (more about this in the following subsection). Another backbone that achieved good results in the *localization + classification* task is SEResNeXt50_32x4d-5S with $mAP_{0.75} = 88\%$. On the other hand, SENet154-5S do not perform well at both tasks. These outcomes contradict the classification results achieved by the pre-trained CNN models in their original domain, in which SENet154 is far superior than ResNet34. A plausible explanation for this contradiction is given by the relatively few samples used to train PollenDet. In this sense, as ResNet34-5S have much less trainable parameters than almost every other backbone, it also needs less samples to be properly trained. This opens up the possibility of further improving PollenDet by increasing the number of samples and thus leverage the prediction capacity of larger backbones.

Figure 34 – **Computed mAPs values at multiple $t_{\mathtt{IoU}}$**. The backbone RESNET34-5S is more robust across the range of IoU thresholds ($0.5 \leq t_{\mathtt{IoU}} \leq 0.9$).

(a) *Localization* (b) *Localization + classification*



Source: Elaborated by the author.

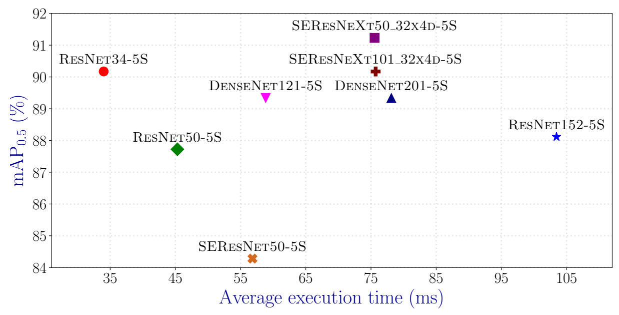
7.3.7 backbone behavior across multiple IoU thresholds

In the previous section, for the task of *localization* + *classification* of pollen grains, we showed that the results of RESNET34-5S fell from $\text{mAP}_{0.5} = 90.2\%$ to $\text{mAP}_{0.75} = 88.2\%$ (difference of 2%), while the results of SERESNEXT50_32X4D-5S fell from $\text{mAP}_{0.5} = 91.2\%$ to $\text{mAP}_{0.75} = 88\%$ (difference of 3.1%). These results suggested that POLLENDET achieves better localization results when RESNET34-5S is used. In Figure 34, we delve into this subject by comparing the behaviour of four backbones across nine different IoU thresholds (from 0.5 to 0.9). In this sense, Figure 34a is focused in the *localization* task and Figure 34b is focused in the *localization* + *classification* task. The results of this experiment clearly confirm the superiority of RESNET34-5S in terms of predicting more precise bounding boxes at both tasks. It is also interesting to observe that DENSENET121-5S achieves more accurate bounding boxes than SERESNEXT50_32X4D-5S in the *localization* task. However, SERESNEXT50_32X4D-5S is still slightly better in the *localization* + *classification* task, which could be explained by its greater classification power.

7.3.8 Execution time comparison

In this final experiment, multiple tests are performed to measure the running time of POLLENDET over microscopic images of size (1280×960). In this sense, Figure 35 shows the average execution time per image vs. $\text{mAP}_{0.5}$ of POLLENDET with different backbones for the task of *localization* + *classification*. This comparison demonstrates that RESNET34-5S achieves the best accuracy/speed trade-off with an average running time of 34 ms and a mAP value of 90.17%. On the other hand, SERESNEXT50_32X4D-5S reaches the best mAP value (91.22%) and has an average speed of 75.55 ms, which is 2.22x slower than RESNNET34-5S. The worst trade-off scenarios were obtained by SERESNET50-5S and RESNET152-5S, in which the former has a mAP value of 84.28% (difference of -6.94% with the best mAP result) and a speed of 56.82 ms (1.67x slower than RESNNET34-5S), and the latter has a mAP value of 88.12% (difference of -3.1% with SERESNEXT50_32X4D-5S) and a speed of 103.46 ms (3.04x slower

Figure 35 – **Average execution time per image vs. mAP$_{0.5}$.** All processed images are of size (1280×960). The execution time of PollenDet was computed with different backbones for the *localization* + *classification* task. In particular, ResNet34-5S offers the best accuracy/speed trade-off. On the contrary, ResNet152-5S and SEResNet50-5S obtain the worst trade-off.



Source: Elaborated by the author.

than ResNet34-5S). Although there is not shown in Figure 35, SENet154-5S reaches a speed of 250.25 ms, which is 7.36x slower than ResNet34-5S.

7.4 StomaDet

StomaDet is an automatic stomata detector inspired by popular object detectors such as Faster RCNN (REN *et al.*, 2017), R-FCN (DAI *et al.*, 2016), and RetinaNet (LIN *et al.*, 2017). Unlike similar research works such as (SAKODA *et al.*, 2019), every aspect in StomaDet was designed by analyzing the stomata found in many microscopic images of the leaf epidermis. These aspects involve: (i) finding a good approach to perform manual stoma annotations (see subsection 4.2.3), (ii) defining the StomaDet architecture that achieves both rapid inference times and high detection rates (see subsection 7.4.2), (iii) creating a solid set of data augmentation strategies (see subsubsection 7.4.3.1), and (iv) defining the post-processing steps to be used at inference time (see subsubsection 7.4.3.3).

7.4.1 Previous Works

With respect to the related works in stomata phenotyping, Higaki, Kutsuna and Hasezawa (2014) proposed a semi-automated approach for the detection of stomata regions on epidermal leaves using a Self Organizing Map clustering approach that retrieve regions of interest (ROI) that are similar to the ones that were manually annotated as stomata. A more automatic approach was proposed in (VIALET-CHABRAND; BRENDEL, 2014), which uses a multi-scale sliding window and a cascade classifier to discriminate between stoma and background ROIs. Another automatic approach is described in (DUARTE; CARVALHO; MARTINS, 2017), which applies the Wavelet Spot Detection method and a series of morphological operations to retrieve stomata regions. The detection of stomata and the subsequent measurement of their opening pores was presented in (JAYAKODY *et al.*, 2017), in which a cascade classifier was used for the detection part and a combination of image processing techniques such as binarization and skeletonization was used for the pore estimation part.

Indeed, all the above methods use hand-engineered feature representations to characterize the ROIs that are to be classified as stoma or background regions. However, as empirically proved by (AONO *et al.*, 2019), CNN feature representations usually achieve higher classification success rates than their hand-engineered counterparts in the same task. Therefore, very recently, more advance CNN models that use/adapt state-of-the-art object detectors were proposed. For example, for stomata detection, a modified single shot multibox detector (SSD) (LIU *et al.*, 2016) trained from scratch with scanning electron microscopic images (SEM) was proposed in (BHUGRA *et al.*, 2019). Since there are only two classes, namely the `background` and `stoma`, the evaluation in (BHUGRA *et al.*, 2019) was made by computing precision and recall values, which measure how the ground truth bounding boxes relate with the predicted ones in terms of size and location. In the case of stomata counting, two approaches were found: (i) a direct application of SSD in (SAKODA *et al.*, 2019) and (ii) a fine-tuned version of the ALEXNET model (KRIZHEVSKY; SUTSKEVER; HINTON, 2012) in (FETTER *et al.*, 2019). Indeed, as the task of stomata counting do not require finding the exact location and size of each stoma, both approaches compute R^2 statistics to find the correlation between the number stomata counted manually and automatically. Although the approach in (FETTER *et al.*, 2019) also consider precision and recall measures, they were defined for a classification task setting and, thus, can not be compared with the precision and recall values used in a detection task setting.

In general, all of the CNN-based methods described above either directly use an object detector with minimal modifications or implement their own from scratch without considering current trends in object detection.

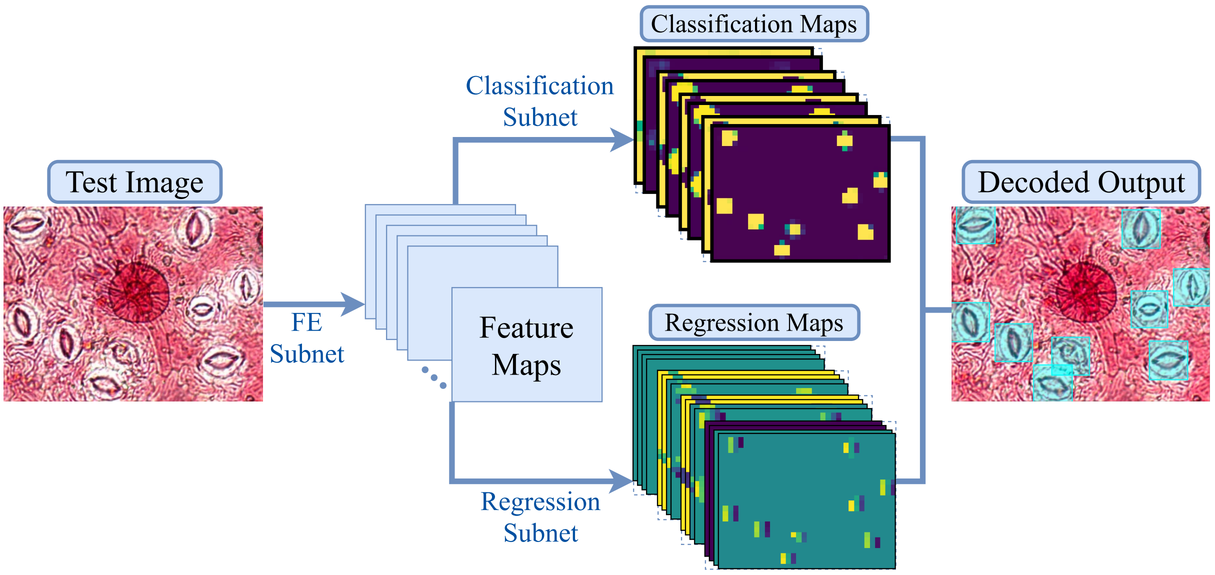
7.4.2 General architecture

Like POLLENDET, the architecture of STOMADET also has three components: (i) the feature extraction (FE) subnet, (ii) the classification subnet, and (iii) the regression subnet (see Figure 36). The details of each component are in the following subsections.

7.4.2.1 The feature extraction subnet (backbone)

The FE subnet of STOMADET computes a collection $\mathcal{F}$ of semantically strong activation maps from an input image I. In this sense, the success of this component is mainly determined by the performance of its backbone network, which is usually represented by the convolutional part of a CNN model pre-trained on the ImageNet dataset (RUSSAKOVSKY *et al.*, 2015). In general, we aim to find the most suitable backbone network for stomata detection. Since there is a large list of pre-trained backbone networks available in deep learning libraries such as PyTorch (PASZKE *et al.*, 2017), we restrict our search space to the following families of backbone networks: (i) RESNET34 and RESNET50 (HE *et al.*, 2016), (ii) DENSENET121 and

Figure 36 – **General architecture of StomaDet, $N_{\mathcal{A}} = 4$.** The input image is processed by the FE subnet. Then, the resulting collection of activation maps is transformed into sets of classification and regression maps. At inference time, these two outputs are decoded into bounding boxes representing the detected stomata. For visualization purposes, we show the `softmax` output for the predicted classification maps.



Source: Elaborated by the author.

DenseNet201 (HUANG *et al.*, 2017), (iii) SEResNeXt50_32x4d and SEResNeXt101_-32x4d (Hu *et al.*, 2019), and (iv) EfficientNetB1, EfficientNetB2, EfficientNetB3, EfficientNetB4 and EfficientNetB5 (TAN; LE, 2019)[2] .

Despite their intrinsic differences, all the above backbone networks share the following architectural pattern: They are organized into six *blocks*, where there is a sub-sampling layer between every two consecutive *blocks*. Each *block* transforms its input set of 2-D activation maps into another semantically stronger set of activation maps. On the other hand, each sub-sampling layer reduce, by a factor of two, the spatial resolution of its input activation maps. This pattern is depicted in Figure 37, where for an input image I of size $(H \times W)$, the backbone i-th *block* $(1 \leq i \leq 6)$ yields a set of activation maps $\mathbf{F}_i$ of spatial resolution $(\frac{H}{2^{i-1}} \times \frac{W}{2^{i-1}})$. Within the context of StomaDet, the fourth, fifth and sixth *blocks* of any backbone network are named as L2, L3 and L4, respectively.

Regardless of the backbone network used, there is a trade-off between (i) the semantic information and (ii) the spatial resolution that a computed set of activation maps can possess. For example, although the set of activation maps computed by L4 is semantically the strongest, its low spatial resolution can make the detection of small stomata impossible. Therefore, the FE subnet must provide to its classification and regression subnets, a collection $\mathcal{F}$ containing one

[2] SENet implementation: <https://github.com/Cadene/pretrained-models.pytorch>. EfficientNet implementation: <https://github.com/rwightman/pytorch-image-models>. The remaining backbone networks were taken from the official TorchVision 0.3 library.

or more sets of activation maps with sufficient spatial resolution to detect small stomata, and semantically strong enough to avoid false positives. In this sense, preliminary experiments over many microscopic images showed that the set of activation maps computed by L3 has sufficient spatial resolution to properly detect stomata of up to a minimum size of (50×50), which is also the smallest size found in the datasets in Table 3. Based on the above information, we explore three adaptation strategies, each in charge of deciding which set or sets of activation maps should be included in $\mathcal{F}$. They are described as follows:

1. **$\mathbf{F}_5$ strategy:** For an input image I of size (H,W), L3 computes a set of activation maps $\mathbf{F}_5$ of spatial resolution $(\frac{H}{16}\times\frac{W}{16})$. Then, $\mathcal{F}=\{\mathbf{F}_5\}$. This adaptation strategy is the one depicted in Figure 36 and Figure 38.

2. **$\mathbf{F}_5$&$\mathbf{F}_6$ strategy:** In this adaptation strategy, for a given image I, the blocks L3 and L4 of the backbone network are used to compute the sets of activation maps $\mathbf{F}_5$ and $\mathbf{F}_6$, respectively (see Figure 37). Then, $\mathcal{F}=\{\mathbf{F}_5,\mathbf{F}_6\}$.

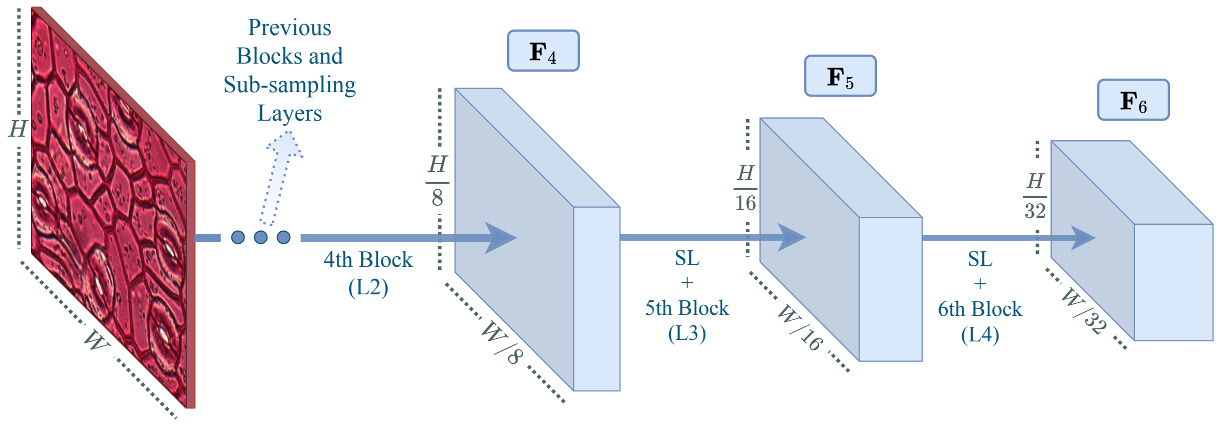
3. **$\widehat{\mathbf{F}}_5$&$\widehat{\mathbf{F}}_6$ strategy:** This strategy is inspired by the Feature Pyramid Network (FPN) (Lin *et al.*, 2017). In this sense, the set of activation maps $\mathbf{F}_6$ computed by L4 is transformed into another set of 256 activation maps $\widehat{\mathbf{F}}_6$ via a lateral connection consisting of a (1×1) convolutional layer. Using the same approach, $\mathbf{F}_5$ is transformed into $\widehat{\mathbf{F}}_5$. Then, $\widehat{\mathbf{F}}_6^{(\times 2)}$ is defined as the up-sampled version of $\widehat{\mathbf{F}}_6$, which through a bilinear interpolation is resized to match the spatial resolution of $\widehat{\mathbf{F}}_5$. Next, $\widehat{\mathbf{F}}_5$ is updated as follows: $\widehat{\mathbf{F}}_5 := \widehat{\mathbf{F}}_5+\widehat{\mathbf{F}}_6^{(\times 2)}$. Finally, $\mathcal{F}=\{\widehat{\mathbf{F}}_5,\widehat{\mathbf{F}}_6\}$.

Therefore, we evaluate 33 different FE subnets, each represented by one pre-trained backbone network and its adaptation strategy.

7.4.2.2 Bounding box encoding

Each training sample given to StomaDet is composed of a microscopic image I of size $(H\times W)$ and its set of square bounding boxes $G=\{g_1,\ldots,g_{N_G}\}$. However, before starting the training process, G is encoded into a more suitable target output, consisting of a set $\mathcal{L}$ of ground truth classification maps and a set $\mathcal{T}$ of ground truth regression maps (see Figure 38). We use the encoding algorithm followed by most detectors of *everyday* objects, that rely on the use of anchor sizes (REN *et al.*, 2017; DAI *et al.*, 2016; He *et al.*, 2017). Formally, an *anchor* is defined as the default bounding box whose size $(h\times w)$ and center position (x,y) can be transformed into those of a ground truth bounding box $g_i\in G$, $(1\leq i\leq N_G)$, via a set of scaling and translation operations.

Figure 37 – **Architectural pattern of backbone networks.** $\mathbf{F}_i$ is the set of 2-D activation maps computed by the i-th *block*. In STOMADET, the fourth, fifth and sixth *blocks* are respectively named as L2, L3, and L4. The term SL is the abbreviation for sub-sampling layer. The proportions between the spatial resolutions of $\mathbf{F}_4$, $\mathbf{F}_5$ and $\mathbf{F}_6$ were readjusted for better visualization.



Source: Elaborated by the author.

Encoding algorithm

Given an input set of $N_{\mathcal{A}}$ anchor sizes $\mathcal{A} = \{\alpha_1, \ldots, \alpha_{N_{\mathcal{A}}}\}$ and a predefined set of $N_{\mathcal{A}}$ *reduction factors* $\{r_1, \ldots, r_{N_{\mathcal{A}}}\}$, each anchor size α_j $(1 \leq j \leq N_{\mathcal{A}})$ is used to encode the set of bounding boxes G as follows: First, the input image I of size (H, W) is partitioned into a regular grid of size $(\frac{H}{r_j} \times \frac{W}{r_j})$. Then, an anchor of size α_j is placed at the center of every cell on the grid. To measure the overlap between each anchor and each bounding box $g_i \in G$, the *Intersection-over-Union* (`IoU`) metric is used, where an `IoU` $= 1$ means complete overlap, and an `IoU` $= 0$ means no overlap. In this context, each cell is classified as *foreground* or *stoma* $(l = 1)$ when its associated anchor achieves an `IoU` ≥ 0.5 with any $g_i \in G$. When the highest `IoU` overlap obtained with an anchor is less than 0.4, its associated cell is classified as *background* $(l = 0)$. In the remaining cases, the associated cell is assigned a classification label of $l = -1$. Each *foreground* cell $(l = 1)$ also stores four regression coefficients $[t_x, t_y, t_w, t_h]$, which are computed with the encoding formulas in (REN *et al.*, 2017). In this sense, the regression coefficients of a given cell contain the information necessary to transform its associated anchor into the $g_i \in G$ with which it has the highest `IoU` overlap. In the case of every *non-foreground* cell $(l \neq 1)$, its regression coefficients are represented by four zeros.

Next, the classification labels and regression coefficients on the grid are organized into five tensors $\{\mathcal{L}^{(j)}, \mathcal{T}_x^{(j)}, \mathcal{T}_y^{(j)}, \mathcal{T}_h^{(j)}, \mathcal{T}_w^{(j)}\}$, each of spatial resolution $(\frac{H}{r_j} \times \frac{W}{r_j})$. As the above process is repeated for every $\alpha_j \in \mathcal{A}$, the set G is encoded into a set $\mathcal{L}$ of $N_{\mathcal{A}}$ ground truth classification maps, and a set $\mathcal{T}$ of $4N_{\mathcal{A}}$ ground truth regression maps (see Figure 38).

Set of anchor sizes for stomata detection

Due to the IoU metric, only the bounding boxes whose sizes are similar to a given $\alpha_j \in \mathcal{A}$ are encoded into $\mathcal{L}^{(j)}$ and $\mathcal{T}^{(j)}$. Therefore, when α_j is too big (or too small) with respect to every bounding box in G, the resulting $\mathcal{L}^{(j)}$ and $\mathcal{T}^{(j)}$ will contain only zeros. In this context, given a training *split* $B_{\mathtt{train}}$, a well chosen set $\mathcal{A}$ must satisfy two conditions: (i) it includes enough anchor sizes to properly encode all possible sets of bounding boxes from $B_{\mathtt{train}}$, and (ii) every $\alpha_j \in \mathcal{A}$ is capable of encoding a substantial number of bounding boxes from $B_{\mathtt{train}}$.

Most detectors of *everyday* objects (LIN *et al.*, 2017) use similar sets of anchor sizes. However, since the bounding boxes employed to annotate stomata differ from those used to annotate *everyday* objects, in StomaDet we created the following sets of anchor sizes:

- $\mathcal{A}_{\mathtt{WoodyL}} = \{(56\times56),(88\times88),(120\times120),(152\times152)\}$. It was created by analyzing the square bounding boxes from the WoodyL dataset, whose minimum and maximum areas are 50^2 and 160^2, respectively.
- $\mathcal{A}_{\mathtt{All}} = \{(56\times56),(88\times88),(120\times120),(152\times152),(184\times184),(216\times216)\}$. It was created by analyzing the $B_{\mathtt{All}}$ dataset, which revealed the existence of square bounding boxes with areas of up to 240^2 pixels. Therefore, $\mathcal{A}_{\mathtt{All}}$ has two more anchors sizes than $\mathcal{A}_{\mathtt{WoodyL}}$.

The set of reduction factors $\{r_1,\ldots,r_{N_\mathcal{A}}\}$, that dictate the spatial resolutions in $\mathcal{L}$ and $\mathcal{T}$, is defined for every pair of one adaptation strategy ($\mathbf{F}_5$, $\mathbf{F}_5$&$\mathbf{F}_6$, $\widehat{\mathbf{F}}_5$&$\widehat{\mathbf{F}}_6$) and one set of anchor sizes ($\mathcal{A}_{\mathtt{WoodyL}}$, $\mathcal{A}_{\mathtt{All}}$) as follows: (i) When $\mathbf{F}_5$ is used, all reduction factors associated with $\mathcal{A}_{\mathtt{WoodyL}}$ and $\mathcal{A}_{\mathtt{All}}$ are set to be $\{16,16,16,16\}$ and $\{16,16,16,16,16,16\}$, respectively. (ii) When either $\mathbf{F}_5$&$\mathbf{F}_6$ or $\widehat{\mathbf{F}}_5$&$\widehat{\mathbf{F}}_6$ is used, the set of reduction factors associated with $\mathcal{A}_{\mathtt{WoodyL}}$ and $\mathcal{A}_{\mathtt{All}}$ are established in $\{16,16,16,32\}$ and $\{16,16,16,16,32,32\}$, respectively[3].

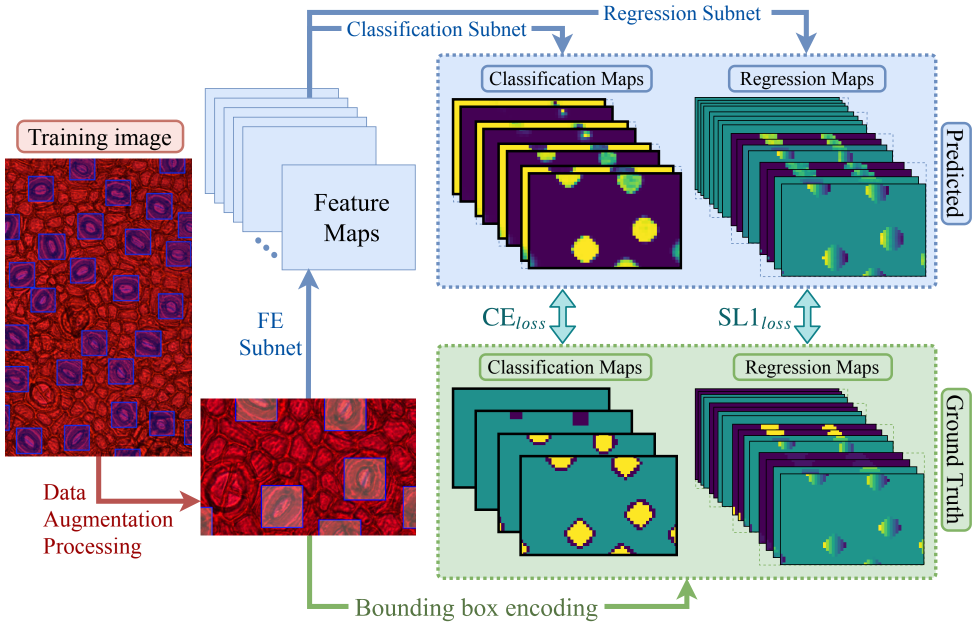
7.4.2.3 Classification and regression subnets

The classification and regression subnets of StomaDet consist of one or more 1×1 convolutional layers, which are configured to yield a set $\widehat{\mathcal{L}}$ of $2N_\mathcal{A}$ classification maps and a set $\widehat{\mathcal{T}}$ of $4N_\mathcal{A}$ regression maps of similar characteristics to their ground truth counterparts $(\mathcal{L},\mathcal{T})$. This includes reusing the same values previously defined for $\{r_1,\ldots,r_{N_\mathcal{A}}\}$. For example, when $\widehat{\mathbf{F}}_5$&$\widehat{\mathbf{F}}_6$ is used, StomaDet predicts two classification maps and four regression maps of size $(\frac{H}{16}\times\frac{W}{16})$ for each of the first three anchor sizes in $\mathcal{A}_{\mathtt{WoodyL}}$, and two classification maps and four regression maps of size $(\frac{H}{32}\times\frac{W}{32})$ for the last anchor size in $\mathcal{A}_{\mathtt{WoodyL}}$.

Therefore, as shown in Figure 38, there is a one-to-one correspondence between the encoded ground truth $(\mathcal{L},\mathcal{T})$ and the predicted output $(\widehat{\mathcal{L}},\widehat{\mathcal{T}})$. More specifically, for each anchor,

[3] We adapted and extended the AnchorBoxesManager class from <https://github.com/warmspringwinds/pytorch-segmentation-detection/blob/master/pytorch_segmentation_detection/utils/detection.py>.

Figure 38 – **Simplified training process of StomaDet, $N_{\mathcal{A}} = 4$.** Each training image is processed with seven data augmentation strategies. The resulting image is processed by StomaDet yielding the predicted sets of classification and regression maps, while the resulting bounding boxes are encoded into sets of ground truth classification and regression maps. The training loss computed between predicted and ground truth sets is back-propagated and all parameters are updated with the ADAM optimizer. CE_{loss} = Cross Entropy loss, and SL1_{loss} = Smooth L1 loss. For better visualization, we show the `softmax` output of the predicted classification maps.



Source: Elaborated by the author.

in addition to its classification label $l \in \mathcal{L}$ and its set of regression coefficients $t = \{t_x, t_y, t_h, t_w\}$, ($t \in \mathcal{T}$), there are also a set of predicted label scores $\hat{l} = \{\hat{l}_b, \hat{l}_f\}$ and a set of predicted regression coefficients $\hat{t} = \{\hat{t}_x, \hat{t}_y, \hat{t}_h, \hat{t}_w\}$, ($\hat{l} \in \widehat{\mathcal{L}}, \hat{t} \in \widehat{\mathcal{T}}$), where $\hat{l}_b$ and $\hat{l}_f$ are respectively the predicted label scores for the *background* and the *foreground* (stoma) classes.

7.4.3 Implementation details

Now that we have defined the architecture details, we are ready to define the implementation details. There are three main topics: (i) data augmentation, (ii) training policy, and (iii) inference. The following subsections will discuss each particular topic in detail.

7.4.3.1 Data Augmentation

As demonstrated in (ZOPH *et al.*, 2020b), when data augmentation (DA) strategies are applied correctly, the performance of any detector can be significantly increased. In StomaDet, we propose seven DA strategies, which together with their probabilities of occurrence P were chosen on the basis of: (i) the observed morphological characteristics of stomata on microscopic images, and (ii) preliminary experiments using a reduced training set. The details for each DA strategy are below.

- **Multiscale Resizing** ($P = 50\%$). Using the bicubic interpolation algorithm, the image and its associated ground truth bounding boxes are enlarged or reduced by a scaling factor that is chosen randomly from the set $M_{\text{sca}} = [0.5, 0.6, 0.7, 0.8, 0.9,\ 1.1, 1.2, 1.3, 1.4, 1.5]$. However, if at least one resized bounding box has an area that covers less than 50^2 pixels or more than 240^2 pixels, everything is restored to its original sizes, and this DA strategy is recomputed with another scaling factor.

- **Horizonal/Vertical Flipping**. Each DA strategy is applied with a P=50% chance.

- **Image Rotation** ($P = 50\%$). When activated, the image I and the central points of its associated bounding boxes rotate θ degrees with respect to the center of I, where θ is chosen randomly from $M_{\Theta} = [5°, 10°, 15°, 20°, 25°,\ 30°, 35°, 40°, 45°]$. Unlike the rotation strategy used in (ZOPH *et al.*, 2020b), square bounding boxes do not need to be expanded after the rotation.

- **Random Cropping** ($P = 100\%$). The input image is randomly cropped to the size of (512×800) at training time, and to the size of (900×900) at validation time (subsubsection 7.4.3.2). This cropping process is recomputed when the resulting image does not contain at least one stoma.

- **Equalization and gray-level transformation**. Each DA strategy is applied with probability P=40%. As for the equalization strategy, it is applied per image channel as in (ZOPH *et al.*, 2020b). Regarding the gray-level transformation, the resulting single-channel image is copied into a three-channel one.

All the seven DA strategies are executed in the same order as listed above.

7.4.3.2 Training

Before initiating the training process, the training set is divided into two subsets, each with 50% of the samples. The first subset ($B_{\texttt{train}}$) is used to train StomaDet and the other, also known as the validation set ($B_{\texttt{val}}$), to select the best trained model that will be used at inference time. Every training sample is represented by the pair (I, G), where I is the input image chosen randomly from $B_{\texttt{train}}$, and G is its set of bounding boxes. The training process of StomaDet starts by applying the data augmentation strategies described in subsubsection 7.4.3.1 to transform (I, G) into (I', G'). Then, G' is encoded into $(\mathcal{L}, \mathcal{T})$, and I' is transformed into a collection of activation maps $\mathcal{F}$ by the FE subnet. Next, $\mathcal{F}$ is processed by the classification and regression subnets, which predict $(\widehat{\mathcal{L}}, \widehat{\mathcal{T}})$.

For each anchor, the quality of its associated predicted output $(\hat{l} \in \widehat{\mathcal{L}}, \hat{t} \in \widehat{\mathcal{T}})$ with respect to its ground truth counterpart $(l \in \mathcal{L}, t \in \mathcal{T})$ is measured with the Cross Entropy loss (CE_{loss})

and the Smooth L1 loss ($\mathrm{SL1}_{loss}$) as follows:

$$T_{loss}(\hat{l},l,\hat{t},t) = \lambda[l \geq 0]CE_{loss}(\hat{l},l) + [(l=1) \ \texttt{AND} \ (\hat{l}_f > \hat{l}_b)]\mathrm{SL1}_{loss}(\hat{t},t) \tag{7.4}$$

where the brackets $[\cdot]$ indicate a conditional operation that returns 1 when the condition is true and zero otherwise. For example, $[l \geq 0]$ outputs 1 when the anchor has a non-negative classification label. Also, we set the weight λ=1 when $[(l=1) \ \texttt{AND} \ (\hat{l}_f > \hat{l}_b)]$ evaluates to zero, and λ=0.4 otherwise. In this sense, the *training loss* is the average of the T_{loss} values computed at all anchors.

At each epoch, the *training loss* is minimized with the ADAM optimizer (KINGMA; BA, 2015) using a learning rate of 0.001 and mini-batches of two training samples per iteration. After completing the epoch, all samples from $B_{\texttt{val}}$ are used to compute the *validation loss*, once again with Equation 7.4. Whenever this *validation loss* reaches a minimum value, we store the trainable parameters of STOMADET on the disk. After completing all epochs, the parameters associated with the minimum validation loss are restored to be used at inference time. Regarding the number of epochs, in some experiments STOMADET is trained for 500 epochs, and in others, for 1k epochs. Also, we consider the following training policies: (i) the “Train Everything” policy, in which ADAM updates all STOMADET trainable parameters, and (ii) the “Train From L3” policy, in which ADAM only updates the STOMADET parameters in layers from block L3 and onwards.

7.4.3.3 Inference

Once trained, STOMADET predicts the classification and regression maps ($\widehat{\mathcal{L}}$, $\widehat{\mathcal{T}}$) for a given microscopic image I (see Figure 36). Then, the `softmax` function transforms the predicted classification scores of every anchor ($\{\hat{l}_b, \hat{l}_f\} \in \widehat{\mathcal{L}}$) into probabilities $\{\hat{p}_b, \hat{p}_f\}$, where $\hat{p}_b + \hat{p}_f = 1$. Every time the foreground probability of an anchor is equal to or greater than a certain threshold ($\hat{p}_f \geq t_{\texttt{st}}$), the formulas from (REN *et al.*, 2017) are used to decode its associated $\hat{t} \in \widehat{\mathcal{T}}$ into a bounding box. In this sense, higher values for $t_{\texttt{st}}$ means fewer bounding boxes decoded. Next, all bounding boxes are ranked in descending order of their associated foreground probabilities, forming the set $\widehat{G}$.

Popular object detectors such as (LIN *et al.*, 2017; ZHANG *et al.*, 2018b) use the non-maximum suppression (NMS) algorithm to compute the final set of bounding boxes $\hat{G}_{\texttt{final}}$ from $\widehat{G}$. This NMS algorithm iterates over the following three steps until $\widehat{G}$ is empty: (i) The best-ranked bounding box $\hat{b}_{\texttt{best}}$ is taken out from $\widehat{G}$. (ii) $\hat{b}_{\texttt{best}}$ is added to the final set of bounding boxes $\hat{G}_{\texttt{final}}$. (iii) Every bounding box that reaches an $\texttt{IoU} \geq 0.25$ with $\hat{b}_{\texttt{best}}$ is permanently removed from $\widehat{G}$.

In STOMADET, we propose the mNMS algorithm, which modifies the step (ii) of the original NMS algorithm as follows: $\hat{b}_{\texttt{best}}$ is only added to $\hat{G}_{\texttt{final}}$ when there is at least two bounding boxes in $\widehat{G}$ that overlap $\hat{b}_{\texttt{best}}$ with an $\texttt{IoU} \geq 0.5$.

7.4.4 Experimental Settings

StomaDet uses almost the same experimental settings as PollenDet (see subsection 7.3.3). There are only two differences

- While PollenDet created ten *splits*, StomaDet only created five *splits* due to computational limitations.
- Since StomaDet only deals with object localization (one foreground class), the evaluation metrics $\mathcal{E}$ we used are the: Precision (Pr), Recall (Re) and F1-Score ($\mathit{F1}$) (see subsection 4.3.3).

7.4.5 Comparison of feature extraction subnets and training policies on `WoodyL`

Here, we compare the precision, recall and F1-scores associated with 22 FE subnets and two training policies on the `WoodyL` dataset. In these experiments, we used the set of anchor sizes $\mathcal{A}_{\texttt{WoodyL}}$, and 500 epochs per training process.

The analysis of Table 19 indicates that most precision rates are higher than their corresponding recall rates, which suggests that the inference process prioritizes having fewer erroneous detections over having fewer undetected stomata. An extreme case of this is given by the precision (99.4%) and recall (8.1%) rates obtained with SEResNeXt101_32x4d ($\mathbf{F}_5$&$\mathbf{F}_6$).

The comparison of training policies reveals the following situations: (i) all DenseNet, ResNet and SEResNeXt backbone networks benefits the most from the “Train From L3” policy, while (ii) EfficientNet backbone networks usually benefits the most from the “Train Everything” policy. However, regardless of the training policy used, the best F1-scores are mainly achieved by EfficientNet-based FE subnets, followed by SEResNeXt50_32x4d ($\mathbf{F}_5$&$\mathbf{F}_6$) and SEResNeXt50_32x4d ($\mathbf{F}_5$). When comparing the two adaptation strategies ($\mathbf{F}_5$ and $\mathbf{F}_5$&$\mathbf{F}_6$) for each EfficientNet backbone network, the F1-scores at the “Train Everything” column tend to be larger for the $\mathbf{F}_5$ strategy than for the $\mathbf{F}_5$&$\mathbf{F}_6$, whereas the opposite happens in the “Train From L3” column. Finally, the comparison between inference times (IT) shows that deeper backbone networks lead to higher processing times, and that $\mathbf{F}_5$&$\mathbf{F}_6$ requires slightly more IT than $\mathbf{F}_5$.

All subsequent experiments will be focused on EfficientNetB1, EfficientNetB3 and EfficientNetB5 backbone networks as they offered the best trade-off between F1-score and IT in the preceding experiments.

Table 19 – **F1-scores, precision and recall rates for combinations of FE subnets and training policies at the WoodyL dataset.** The best F1-scores (%) for each family of backbone networks is emphasized in bold font. The last column displays the average inference times (IT) in seconds (s) for images of size (2000×1500).

FE subnet	Train Everything			Train From L3			IT
	Precision (%)	Recall (%)	F1 (%)	Precision (%)	Recall (%)	F1 (%)	(s) ± 0.02
DenseNet121 ($\mathbf{F}_5$)	93.3 ± 3.8	67.9 ± 9.4	**78.3 ± 6.3**	92.6 ± 9.7	86.0 ± 2.6	88.9 ± 4.0	0.23
DenseNet121 ($\mathbf{F}_5$&$\mathbf{F}_6$)	95.5 ± 5.4	12.4 ± 9.3	20.9 ± 12.7	94.9 ± 3.6	86.3 ± 3.0	**90.3 ± 1.3**	0.24
DenseNet201 ($\mathbf{F}_5$)	81.0 ± 20.3	60.1 ± 17.9	67.7 ± 17.0	93.2 ± 5.8	86.2 ± 4.1	89.4 ± 3.3	0.28
DenseNet201 ($\mathbf{F}_5$&$\mathbf{F}_6$)	89.2 ± 17.2	8.2 ± 0.1	14.9 ± 0.4	90.3 ± 4.5	82.9 ± 5.9	86.4 ± 5.0	0.32
EfficientNetB1 ($\mathbf{F}_5$)	98.0 ± 0.4	97.6 ± 0.2	97.8 ± 0.1	97.7 ± 0.6	95.0 ± 1.4	96.3 ± 0.9	0.19
EfficientNetB1 ($\mathbf{F}_5$&$\mathbf{F}_6$)	98.5 ± 0.5	96.8 ± 1.5	97.6 ± 0.6	97.5 ± 0.9	95.9 ± 0.8	96.7 ± 0.3	0.20
EfficientNetB2 ($\mathbf{F}_5$)	97.9 ± 0.8	96.3 ± 1.0	97.1 ± 0.6	97.5 ± 0.8	96.5 ± 1.9	97.0 ± 0.7	0.19
EfficientNetB2 ($\mathbf{F}_5$&$\mathbf{F}_6$)	97.8 ± 0.7	96.3 ± 1.3	97.1 ± 0.7	97.4 ± 0.7	95.9 ± 0.7	96.6 ± 0.6	0.20
EfficientNetB3 ($\mathbf{F}_5$)	98.1 ± 0.5	97.5 ± 0.7	97.8 ± 0.2	97.0 ± 0.4	97.7 ± 0.9	97.3 ± 0.4	0.21
EfficientNetB3 ($\mathbf{F}_5$&$\mathbf{F}_6$)	97.8 ± 1.1	97.4 ± 1.5	97.6 ± 0.5	97.9 ± 0.7	96.9 ± 0.9	97.4 ± 0.5	0.22
EfficientNetB4 ($\mathbf{F}_5$)	98.2 ± 0.2	97.4 ± 0.7	97.8 ± 0.3	98.0 ± 0.7	96.4 ± 1.1	97.2 ± 0.7	0.24
EfficientNetB4 ($\mathbf{F}_5$&$\mathbf{F}_6$)	96.2 ± 4.7	97.7 ± 0.3	96.9 ± 2.5	97.9 ± 0.8	97.0 ± 1.0	97.5 ± 0.7	0.25
EfficientNetB5 ($\mathbf{F}_5$)	98.5 ± 0.2	98.0 ± 0.3	**98.3 ± 0.2**	98.1 ± 0.5	97.2 ± 0.7	97.6 ± 0.4	0.28
EfficientNetB5 ($\mathbf{F}_5$&$\mathbf{F}_6$)	98.1 ± 0.5	98.1 ± 0.6	98.1 ± 0.2	97.9 ± 0.5	97.6 ± 0.3	**97.8 ± 0.2**	0.31
ResNet34 ($\mathbf{F}_5$)	96.8 ± 1.1	84.1 ± 4.8	**89.9 ± 2.6**	97.0 ± 1.5	95.1 ± 1.2	**96.1 ± 0.5**	0.17
ResNet34 ($\mathbf{F}_5$&$\mathbf{F}_6$)	78.0 ± 32.3	27.9 ± 19.7	38.2 ± 23.9	97.0 ± 0.7	90.9 ± 1.3	93.9 ± 0.7	0.17
ResNet50 ($\mathbf{F}_5$)	96.6 ± 1.8	70.1 ± 11.2	80.8 ± 7.5	94.8 ± 2.9	89.2 ± 5.6	91.8 ± 2.9	0.20
ResNet50 ($\mathbf{F}_5$&$\mathbf{F}_6$)	66.6 ± 41.9	9.5 ± 1.7	14.9 ± 4.5	95.8 ± 0.8	85.4 ± 5.2	90.2 ± 2.9	0.21
SEResNeXt50_32x4d ($\mathbf{F}_5$)	94.9 ± 3.7	63.7 ± 18.3	**75.4 ± 13.3**	96.3 ± 1.8	96.7 ± 0.9	96.5 ± 0.6	0.25
SEResNeXt50_32x4d ($\mathbf{F}_5$&$\mathbf{F}_6$)	93.0 ± 8.9	8.2 ± 0.1	15.0 ± 0.2	97.8 ± 0.9	96.0 ± 0.6	**96.9 ± 0.2**	0.26
SEResNeXt101_32x4d ($\mathbf{F}_5$)	82.7 ± 30.0	9.2 ± 2.2	16.2 ± 3.7	94.6 ± 2.8	94.6 ± 3.2	94.6 ± 2.9	0.33
SEResNeXt101_32x4d ($\mathbf{F}_5$&$\mathbf{F}_6$)	99.4 ± 0.8	8.1 ± 0.1	15.1 ± 0.2	92.8 ± 1.5	91.1 ± 2.5	91.9 ± 1.5	0.36

Source: Elaborated by the author.

7.4.6 mNMS versus NMS

Here, we analyze how much the detection power of StomaDet changes when, at inference time, our mNMS algorithm is replaced by the original NMS algorithm on the $\mathcal{B}_{\texttt{All}}$ dataset. More specifically, for each training policy and FE subnet, Table 20 shows the comparison between the F1-scores achieved by the NMS algorithm for three different thresholds $t_{\texttt{st}}$ and by the mNMS algorithm for $t_{\texttt{st}}$=0.5. Unlike the preceding experiments, we use the set of anchor sizes $\mathcal{A}_{\texttt{All}}$ and 1000 epochs per training process.

Table 20 shows no clear distinction between the F1-scores obtained with the NMS algorithm for $t_{\texttt{st}}$=0.5 and $t_{\texttt{st}}$=0.6. This can be explained by analyzing the positive correlation between $t_{\texttt{st}}$ and the precision rate (Pr), and the negative correlation between $t_{\texttt{st}}$ and the recall rate (Re). A good example of this is given by EfficientNetB1 ($\hat{\mathbf{F}}_5$&$\hat{\mathbf{F}}_6$), whose Pr=95.6% and Re=94.9% for $t_{\texttt{st}}$=0.5, changed to Pr=96.0% and Re=94.3% for $t_{\texttt{st}}$=0.6. Consequently, the F1-score is similar for both $t_{\texttt{st}}$=0.5 and $t_{\texttt{st}}$=0.6. Regarding $t_{\texttt{st}}$=0.7, it typically leads to lower F1-scores as the trade-off between Pr and Re becomes very unbalanced.

Table 20 – **F1 scores (%) achieved by NMS and mNMS on the $\mathcal{B}_{\texttt{All}}$ dataset.** The best result for each pair of one FE subnet and one training policy is emphasized in bold font. EFFNET: abbreviation for EFFICIENTNET.

	Train Everything				Train From L3			
	NMS			mNMS	NMS			mNMS
FE subnet	$t_{\texttt{st}} = 0.5$	$t_{\texttt{st}} = 0.6$	$t_{\texttt{st}} = 0.7$	$t_{\texttt{st}} = 0.5$	$t_{\texttt{st}} = 0.5$	$t_{\texttt{st}} = 0.6$	$t_{\texttt{st}} = 0.7$	$t_{\texttt{st}} = 0.5$
EFFNETB1 ($\mathbf{F}_5$)	93.4 ± 1.6	93.4 ± 1.5	93.2 ± 1.5	**93.5 ± 1.3**	94.0 ± 1.8	94.1 ± 1.7	94.1 ± 1.7	**94.6 ± 0.8**
EFFNETB1 ($\mathbf{F}_5$&$\mathbf{F}_6$)	94.0 ± 2.3	94.1 ± 2.1	94.2 ± 1.9	**95.3 ± 0.7**	94.8 ± 0.8	94.9 ± 0.8	94.9 ± 0.8	**95.0 ± 0.9**
EFFNETB1 ($\widehat{\mathbf{F}}_5$&$\widehat{\mathbf{F}}_6$)	95.2 ± 0.5	95.2 ± 0.5	95.0 ± 0.5	**95.4 ± 0.5**	95.0 ± 0.8	94.9 ± 0.9	94.7 ± 1.1	**95.1 ± 0.7**
EFFNETB3 ($\mathbf{F}_5$)	94.4 ± 0.6	94.4 ± 0.7	94.2 ± 0.7	**94.7 ± 0.9**	93.7 ± 2.3	93.6 ± 2.4	93.4 ± 2.5	**94.3 ± 2.3**
EFFNETB3 ($\mathbf{F}_5$&$\mathbf{F}_6$)	92.8 ± 4.3	92.8 ± 4.4	92.7 ± 4.4	**94.2 ± 2.0**	93.5 ± 3.5	93.5 ± 3.5	93.3 ± 3.6	**94.0 ± 3.7**
EFFNETB3 ($\widehat{\mathbf{F}}_5$&$\widehat{\mathbf{F}}_6$)	94.1 ± 1.7	93.9 ± 1.7	93.6 ± 1.8	**94.5 ± 1.5**	95.5 ± 0.8	95.4 ± 1.1	95.2 ± 1.5	**95.6 ± 1.3**
EFFNETB5 ($\mathbf{F}_5$)	95.0 ± 1.4	94.9 ± 1.4	94.7 ± 1.3	**95.1 ± 1.0**	92.8 ± 3.7	92.8 ± 3.7	92.6 ± 3.7	**94.7 ± 1.2**
EFFNETB5 ($\mathbf{F}_5$&$\mathbf{F}_6$)	**93.6 ± 1.0**	93.4 ± 1.2	92.7 ± 1.6	**93.6 ± 1.3**	95.8 ± 0.4	95.9 ± 0.5	95.8 ± 0.5	**96.2 ± 0.5**
EFFNETB5 ($\widehat{\mathbf{F}}_5$&$\widehat{\mathbf{F}}_6$)	94.3 ± 1.2	94.3 ± 1.2	94.0 ± 1.1	**94.8 ± 0.8**	95.7 ± 0.6	95.7 ± 0.6	95.5 ± 0.7	**95.9 ± 0.7**

Source: Elaborated by the author.

For what concerns mNMS, Table 20 reveals that in 17 out of 18 cases, it achieved higher F1-scores than their NMS counterparts. These results demonstrate that mNMS can successfully mitigate the bad effects caused by the Pr/Re trade-off. Finally, the top three highest F1-scores come from the "Train From L3" policy, where EFFICIENTNETB5 ($\mathbf{F}_5$&$\mathbf{F}_6$) is at the top (96.2%).

7.4.7 Detection of small, medium-sized and large stomata

Here, for each FE subnet and training policy used, we analyze the performance of STOMADET at detecting small, medium-sized and large stomata on the $\mathcal{B}_{\texttt{All}}$ dataset. In this context, we defined as F1$_s$, F1$_m$ and F1$_l$, the F1-scores for small (area $\leq 90^2$), medium-sized ($90^2 <$ area $\leq 180^2$) and large stomata (area $\leq 90^2$), respectively. To compute the above metrics, we slightly redefined the terms: true positive (tp), false positive (fp) and false negative (fn). For example, in the case of F1$_s$, tp becomes the amount of small stomata that were correctly detected, fp becomes the number of wrong detections whose areas match that of a small stoma, and fn becomes the amount of small stomata that were never detected. By extension, tp, fp and fn are similarly redefined for F1$_m$ and F1$_l$. All other evaluation settings were the same as those used in subsection 7.4.6.

The results in Table 21 show that there is a 2.4% gap between the best results obtained with F1$_m$ and F1$_s$, and a 0.4% gap between the best results obtained with F1$_l$ and F1$_m$. These suggest that STOMADET struggles more against small stomata than it does against medium-sized or large stomata. When comparing the two training policies, similar to Table 20, the F1-scores in Table 21 tend to be higher in the "Train From L3" columns than in the "Train Everything" columns.

Table 21 – **F1-scores for small ($F1_s$), medium-sized ($F1_m$) and large ($F1_l$) stomata on the $\mathcal{B}_{\texttt{All}}$ dataset.** For each column, the best result is emphasized in bold font. EFFNET: abbreviation for EFFICIENTNET.

FE subnet	Train Everything			Train From L3		
	$F1_s$	$F1_m$	$F1_l$	$F1_s$	$F1_m$	$F1_l$
EFFNETB1 ($\mathbf{F}_5$)	89.6 ± 2.8	96.2 ± 1.2	96.7 ± 0.5	92.2 ± 2.2	96.3 ± 0.4	94.5 ± 4.5
EFFNETB1 ($\mathbf{F}_5$&$\mathbf{F}_6$)	**93.6 ± 1.2**	96.5 ± 0.5	96.6 ± 1.1	92.9 ± 2.0	96.5 ± 0.5	**97.6 ± 0.5**
EFFNETB1 ($\hat{\mathbf{F}}_5$&$\hat{\mathbf{F}}_6$)	93.2 ± 1.4	**96.8 ± 0.4**	97.2 ± 0.8	92.7 ± 1.3	96.8 ± 0.6	97.1 ± 0.7
EFFNETB3 ($\mathbf{F}_5$)	91.9 ± 2.5	96.7 ± 0.4	95.0 ± 1.7	90.3 ± 6.2	96.9 ± 0.5	97.1 ± 1.1
EFFNETB3 ($\mathbf{F}_5$&$\mathbf{F}_6$)	91.0 ± 4.2	96.5 ± 0.7	95.5 ± 2.1	89.7 ± 9.7	96.7 ± 0.5	97.4 ± 0.7
EFFNETB3 ($\hat{\mathbf{F}}_5$&$\hat{\mathbf{F}}_6$)	91.6 ± 3.6	96.5 ± 0.6	96.7 ± 0.8	93.3 ± 3.3	**97.2 ± 0.3**	**97.6 ± 0.8**
EFFNETB5 ($\mathbf{F}_5$)	92.7 ± 2.2	**96.8 ± 0.3**	96.5 ± 0.4	92.2 ± 2.5	96.5 ± 0.5	96.5 ± 1.4
EFFNETB5 ($\mathbf{F}_5$&$\mathbf{F}_6$)	89.7 ± 2.2	96.3 ± 1.0	**97.3 ± 0.4**	**94.8 ± 1.0**	**97.2 ± 0.2**	97.5 ± 0.8
EFFNETB5 ($\hat{\mathbf{F}}_5$&$\hat{\mathbf{F}}_6$)	92.5 ± 1.6	96.4 ± 0.7	97.2 ± 1.0	94.3 ± 0.9	97.0 ± 0.5	97.5 ± 0.9

Source: Elaborated by the author.

Other interesting results in Table 21 are as follows: (i) When observing the results associated with different adaptation strategies, the $F1_l$-scores achieved by $\mathbf{F}_5$ are mostly surpassed by those obtained with $\mathbf{F}_5$&$\mathbf{F}_6$ and $\widehat{\mathbf{F}}_5$&$\widehat{\mathbf{F}}_6$. This is explained by analyzing the grid resolution from which STOMADET make predictions, where in the case of $\mathbf{F}_5$, it is enough for the detection of small and medium-sized stomata, but it is too small for the detection of large stomata. (ii) In general, the standard deviations associated with the $F1_s$-scores are substantially higher than those associated with $F1_m$ and $F1_l$, which suggest that there are some unknown problems in the training process that prevents STOMADET to always achieve good performances at detecting small stomata. (iii) STOMADET reached its best results for $F1_s$ and $F1_m$, and its second best result for $F1_l$, when it was trained with the "Train From L3" policy and used the EFFICIENTNETB5 ($\mathbf{F}_5$&$\mathbf{F}_6$) FE subnet.

7.4.8 Performance across different datasets

Table 22 shows the performance comparison of nine FE subnets on $\mathcal{B}_{\texttt{All}}$ and its constituent datasets (see Table 3). Here, we only focus on the results associated with the "Train From L3" policy since it led to the highest F1-scores in the preceding experiments. The evaluation settings are the same as those in subsection 7.4.6 and subsection 7.4.7. However, in addition to computing the global F1-score for $\mathcal{B}_{\texttt{All}}$, the F1-scores for each of its constituent datasets were also computed.

The analysis of Table 22 shows that: (i) the highest and the second highest F1-scores are mostly concentrated in the EFFICIENTNETB5-based FE subnets. This agree with the fact that deeper architectures generally lead to better results. (ii) There is a 4.3% gap between the most

Table 22 – **F1-scores for all datasets included in $\mathcal{B}_{\texttt{All}}$.** The best result for each dataset is highlighted in red bold font, and the second best F1-score is emphasized in bold font. EFFNET: abbreviation for EFFICIENTNET.

FE subnet	`SimoneDB`	`WoodyL`	`Poplar`	`Ginkgo`	`CuticleDB`	`USNM/USBG`	`Ctenanthe`	$\mathcal{B}_{\texttt{All}}$
EFFNETB1 ($\mathbf{F}_5$)	97.7 ± 0.3	95.4 ± 1.0	97.5 ± 0.2	96.2 ± 0.6	93.1 ± 0.9	92.2 ± 2.1	93.5 ± 6.9	94.6 ± 0.8
EFFNETB1 ($\mathbf{F}_5$&$\mathbf{F}_6$)	**97.9 ± 0.2**	95.7 ± 1.5	97.7 ± 0.2	**96.7 ± 0.4**	93.9 ± 0.7	92.0 ± 0.9	94.9 ± 4.3	95.0 ± 0.9
EFFNETB1 ($\hat{\mathbf{F}}_5$&$\hat{\mathbf{F}}_6$)	**97.9 ± 0.2**	96.6 ± 1.2	97.7 ± 0.3	96.4 ± 0.5	94.4 ± 0.6	91.8 ± 1.5	94.2 ± 3.2	95.1 ± 0.7
EFFNETB3 ($\mathbf{F}_5$)	**97.9 ± 0.3**	96.2 ± 1.4	**97.9 ± 0.2**	95.4 ± 1.1	94.7 ± 0.7	91.6 ± 1.6	89.3 ± 12.5	94.3 ± 2.3
EFFNETB3 ($\mathbf{F}_5$&$\mathbf{F}_6$)	97.5 ± 0.6	96.1 ± 1.6	97.7 ± 0.1	96.4 ± 0.9	93.9 ± 1.0	**94.0 ± 1.1**	82.3 ± 28.3	94.0 ± 3.7
EFFNETB3 ($\hat{\mathbf{F}}_5$&$\hat{\mathbf{F}}_6$)	97.7 ± 0.4	**97.5 ± 0.4**	97.6 ± 0.4	95.9 ± 0.5	94.8 ± 0.8	93.1 ± 1.4	94.2 ± 6.8	95.6 ± 1.3
EFFNETB5 ($\mathbf{F}_5$)	97.7 ± 0.3	94.6 ± 1.7	**97.9 ± 0.2**	95.3 ± 1.6	94.6 ± 1.5	92.5 ± 0.7	93.5 ± 6.0	94.7 ± 1.2
EFFNETB5 ($\mathbf{F}_5$&$\mathbf{F}_6$)	97.6 ± 0.8	**97.2 ± 0.9**	**97.8 ± 0.2**	96.0 ± 1.1	**95.3 ± 0.2**	**93.7 ± 0.5**	**96.8 ± 1.9**	**96.2 ± 0.5**
EFFNETB5 ($\hat{\mathbf{F}}_5$&$\hat{\mathbf{F}}_6$)	**98.3 ± 0.1**	97.0 ± 0.9	97.7 ± 0.3	**96.5 ± 0.2**	**95.0 ± 0.3**	93.2 ± 1.4	**95.6 ± 2.1**	**95.9 ± 0.7**

Source: Elaborated by the author.

difficult dataset (`USNM/USBG`) and the easiest one (`SimoneDB`). (iii) It seems that STOMADET struggles more with the `Ctenanthe` dataset, as its computed F1-scores are usually associated with very high standard deviations. (iv) Although STOMADET mainly achieved its best results with EFFICIENTNETB5 ($\mathbf{F}_5$&$\mathbf{F}_6$), there are at least two datasets in which significantly higher F1-scores were achieved by other FE subnets (+0.7% in the `SimoneDB` dataset and +0.7% in the `Ginkgo` dataset). Therefore, future research works will be oriented in proposing methods for solving this issue.

In summary, STOMADET achieves its best results when it uses the following settings: (i) **FE subnet:** EFFICIENTNETB5 ($\mathbf{F}_5$&$\mathbf{F}_6$), (ii) **Training policy:** "Train From L3", and (iii) **Inference settings:** t_{st}=0.5 + mNMS. Furthermore, using these previous settings, the best STOMADET model achieved the following F1-scores: `SimoneDB` (97.6%), `WoodyL` (97.8%), `Poplar` (98.2%), `Ginkgo` (96.7%), `CuticleDB` (95.3%), `USNM/USBG` (94.4%), `Ctenanthe` (97.7%), $\mathcal{B}_{\texttt{All}}$ (96.7%), F1$_s$=95.5%, F1$_m$=97.5%, and F1$_l$=97.6%.

7.5 Final considerations

In this chapter, we proposed two object detectors, one for detecting pollen grains (POLLENDET), and the other for locating stomata (STOMADET). Both POLLENDET and STOMADET are one-stage detectors whose inference time is mostly driven by the size of the backbone.

In POLLENDET, we systematically compare multiple backbones from nine pre-trained CNN models, which empirically demonstrates that the optimal choice is always reusing the first five *chunks* of the pre-trained *CNN* model. Then, we modified the NMS algorithm to improve the localization of bounding box proposals as shown in subsection 7.3.4. In addition, the analysis shows that POLLENDET with SERESNEXT50_32X4D-5S achieves the best results at both tasks (mAP$_{0.5}$ = 96.5% and mAP$_{0.5}$ = 91.2% respectively) for an IoU threshold of $t_{\texttt{IoU}}$ = 0.5.

However, as higher IoU thresholds are employed, the use of RESNET34-5S emerges as a better option. This suggests that: (i) SERESNEXT50_32X4D-5S is the best in producing activation maps that benefit the classification of pollen grains, but (ii) RESNET34-5S makes possible the generation of bounding boxes that better enclose such pollen grains. This scenario could change as more samples and more pollen types are considered to train the larger backbones such as SENET154-5S or SERESNEXT101_32X4D-5S. Finally, the running time analysis reinforces the superiority of RESNET34-5S (34 ms per image) over the other backbones.

Due to its relatively large training set, STOMADET achieved high detection rates in many types of microscopic images of the foliar epidermis of multiple plants species. Since the backbone plays a fundamental role in the detection of stomata, several experiments were focused on finding its best sub-components, which ended up being the EFFICIENTNETB5 backbone network and the $\mathbf{F}_5$&$\mathbf{F}_6$ adaptation strategy. In addition, as demonstrated by the "Train From L3" policy, freezing the early layers of the backbone network at training time helps STOMADET to achieve higher detection performances. Also, we proposed a slight modification to the NMS algorithm, which helps STOMADET to achieve even higher detection rates. Finally, the average inference time of STOMADET is 310 milliseconds per (2000×1500) microscopic image.

CHAPTER 8 CONCLUSIONS

This project presented novel multilayer TL strategies for image classification and object detection tasks. It has two well-defined parts,where the first part was dedicated to providing the theoretical background for understanding the second part of the project.

Chapter 2 introduced the basic concepts of deep learning and convolutional neural networks. In more detail, this chapter began with a discussion of the differences between traditional machine learning and deep learning. Then, we presented a historical perspective of deep learning and convolutional neural networks. Next, we discussed the main visual recognition tasks in the field of computer vision, with particular attention to the two primary tasks we face in this project: (i) image classification and (ii) object detection. Following, we explained the basic definitions around a CNN model, such as tensors, the division of a CNN model into a feature extraction part and classification part, layers, blocks, composite functions, activation maps, and feature vectors. We ended the chapter explaining the training and inference cycle and the most relevant layer types.

In Chapter 3, we presented more advanced concepts that built upon those presented in Chapter 2. In this regard, we first described the notion of transfer learning (TL) and its necessity in domains where data is scarce. Then, this chapter introduced formally the concept of TL, and its main element: the pre-trained CNN model. Next, we presented the most relevant pre-trained CNN models and discussed their main advantages and disadvantages. The chapter also presented the two types of TL strategies for image classification tasks: (i) strategies that use the pre-trained model as a feature extractor and (ii) strategies that fine-tune the pre-trained model. The chapter ends by describing popular TL strategies for image classification and object detection tasks.

Chapter 4 presented the datasets we used in the experimental part of this project. We mainly focused on three target tasks: (i) texture recognition, (ii) pollen grain detection, and (iii) stomata detection. We also provided in the chapter the notion of splits and splitting strategies. Next, we presented the common evaluation metrics used in image classification and object detection. Finally, we presented the most important concept of the chapter: the general evaluation scheme, which every experiment in the next chapter follows.

Chapter 5 is the first chapter of the experimental part of this project. Here, we identified some situations where traditional hand-engineered feature extractors achieved better results than CNN-based TL methods. In more detail, we found out that the predictive power of most CNN-based feature extractors increased more when combined with a hand-engineered method than when combined with another CNN-based method. The situation was particularly interesting because, individually, most CNN-based TL methods obtained accuracy rates that surpassed those from the hand-engineered methods by at least 4%. The same situation happened across dedicated classifiers and with feature selection approaches.

In Chapter 6 we performed a layer-by-layer approach to determine whether earlier layers of a pre-trained CNN model can provide features with competitive predictive power. The results somewhat contradict the literary review in texture analysis, which stated that the last layer of a pre-trained CNN model has the most predictive power. Then, we realized that previous works only performed layer-by-layer analysis on category-based datasets. With this information, we proposed a novel multi-layer TL approach that treats pre-trained CNN models as deep composite functions. In more detail, we proposed RANKGP-CNN, a TL strategy that selects the best deep composite functions based on a defined global pooling layer and a feature ranking approach. Finally, we extended the TL strategy to multiple pre-trained models (RANKGP-3M-CNN) and compared its results with alternative CNN-based models. In general, the results of RANKGP-3M-CNN surpass all the alternative CNN-based TL strategies in instance-based texture datasets.

Finally, Chapter 7 presented two TL strategies for two target tasks. The first TL strategy was designed for the pollen detection task, and the second was designed for the stomata localization task. Both approaches take into consideration the morphological characteristics of the target tasks and the innovative architectural ideas of popular object detectors such as RETINADET and FASTER-FCNN.

8.1 Futures Perspectives

Although RANKGP-3M-CNN achieved excellent results, we known that fine-tuning methods can be powerful when given the correct hyper-parameters. Therefore, we want to include in the near future, the concept of deep composite functions into a fine-tuning process. The basic idea is to include a RankDropOut layer into the modified model which will work as a dropout layer, removing randomly the connections, but instead of assigning the same drop probability to every connection, the probability will be determined by the feature ranking approach in the mini-batch.

Furthermore, we are particularly interested on applying RankGP-CNN to CNN models trained in a self-supervised fashion (e.g., SIMCLR (Chen *et al.*, 2020)) to see if they encode texture information differently. Additionally, we want to expand this analysis to real-life applications that involve texture recognition (e.g., biological and medical datasets) and then extend this idea to object detectors, such as StomaDet and PollenDet.

8.2 Bibliographical Production

Articles in scientific journals

- **Condori, R. H.**, & Bruno, O. M. (2021). Analysis of activation maps through global pooling measurements for texture classification. Information Sciences, 555, 260-279.
- Scabini, L. F., **Condori, R. H.**, Gonçalves, W. N., & Bruno, O. M. (2019). Multilayer complex network descriptors for color–texture characterization. Information Sciences, 491, 30-47.

Articles in Preparation

- **Condori, R. H.**, Baetens, J. M., Martinez, A. S., & Bruno, O, M. (2022) Enhancing CNN-based methods with hand-engineered features for gray-level texture recognition.
- **Condori, R. H.**, Ribas, L. C., Biagioni S., Behling H., & Bruno, O, M. (2022) Fast detection and classification of pollen grains through convolutional neural networks.
- **Condori, R. H.**, & Bruno, O. M. (2021). StomaDet: A fast and proficient Convolutional Neural Network for automatic stomata detection at different plant species.

Articles published in conference proceedings

- **Condori, R. H.**, Romualdo, L. M., Bruno, O. M., & de Cerqueira Luz, P. H. (2017, October). Comparison between traditional texture methods and deep learning descriptors for detection of nitrogen deficiency in maize crops. In 2017 Workshop of Computer Vision (WVC) (pp. 7-12). IEEE.
- Scabini, L. F., **Condori, R. H.**, Ribas, L. C., & Bruno, O. M. (2019, September). Evaluating Deep Convolutional Neural Networks as Texture Feature Extractors. In International Conference on Image Analysis and Processing (pp. 192-202). Springer, Cham.
- Scabini, L. F., **Condori, R. M.**, Munhoz, I. C., & Bruno, O. M. (2019, September). Deep Convolutional Neural Networks for Plant Species Characterization Based on Leaf Midrib. In International Conference on Computer Analysis of Images and Patterns (pp. 389-401). Springer, Cham.

BIBLIOGRAPHY

A. R. Backes and W. N. Gonçalves and A. S. Martinez and O. M. Bruno. Texture analysis and classification using deterministic tourist walk. Pattern Recognition, Elsevier, v. 43, n. 3, p. 685 – 694, 2010. ISSN 0031-3203. Available: <https://doi.org/10.1016/j.patcog.2009.07.017>. Citations on pages 112 and 207.

ABDELMOUNAIME, S.; DONG-CHEN, H. New Brodatz-Based Image Databases for Grayscale Color and Multiband Texture Analysis. **ISRN Machine Vision**, v. 2013, p. 1–14, 2013. Available: <https://doi.org/10.1155/2013/876386>. Citation on page 94.

ALI, Z.; KEFALAS, P.; MUHAMMAD, K.; ALI, B.; IMRAN, M. Deep learning in citation recommendation models survey. **Expert Systems with Applications**, v. 162, p. 113790, 2020. ISSN 0957-4174. Available: <https://doi.org/10.1016/j.eswa.2020.113790>. Citation on page 40.

AMOROSI, A.; COLALONGO, M.; FIORINI, F.; FUSCO, F.; PASINI, G.; VAIANI, S.; SARTI, G. Palaeogeographic and palaeoclimatic evolution of the Po Plain from 150-ky core records. **Global and Planetary Change**, Elsevier, v. 40, n. 1, p. 55–78, 2004. ISSN 0921-8181. Available: <https://doi.org/10.1016/S0921-8181(03)00098-5>. Citation on page 103.

ANDREARCZYK, V.; WHELAN, P. F. Using filter banks in Convolutional Neural Networks for texture classification. **Pattern Recognition Letters**, v. 84, p. 63 – 69, 2016. ISSN 0167-8655. Available: <https://doi.org/10.1016/j.patrec.2016.08.016>. Citations on pages 41, 83, and 85.

AONO, A. H.; NAGAI, J. S.; DICKEL, G. d. S. M.; MARINHO, R. C.; OLIVEIRA, P. E. A. M. de; FARIA, F. A. A Stomata Classification and Detection System in Microscope Images of Maize Cultivars. **bioRxiv**, Cold Spring Harbor Laboratory, 2019. Available: <https://doi.org/10.1101/538165>. Citations on pages 42 and 162.

APICELLA, A.; DONNARUMMA, F.; ISGRò, F.; PREVETE, R. A survey on modern trainable activation functions. **Neural Networks**, v. 138, p. 14–32, 2021. ISSN 0893-6080. Available: <https://doi.org/10.1016/j.neunet.2021.01.026>. Citation on page 67.

ARPIT, D.; CAMPOS, V.; BENGIO, Y. How to Initialize your Network? Robust Initialization for WeightNorm &amp; ResNets. In: **Advances in Neural Information Processing Systems**. Curran Associates, Inc., 2019. v. 32, p. 1–10. Available: <https://proceedings.neurips.cc/paper/2019/file/e520f70ac3930490458892665cda6620-Paper.pdf>. Citation on page 56.

ASCOLI, S. d'; SAGUN, L.; BIROLI, G. Triple descent and the two kinds of overfitting: where &amp; why do they appear? In: LAROCHELLE, H.; RANZATO, M.; HADSELL, R.; BALCAN, M. F.; LIN, H. (Ed.). **Advances in Neural Information Processing Systems**. Curran Associates, Inc., 2020. v. 33, p. 3058–3069. Available: <https://proceedings.neurips.cc/paper/2020/file/1fd09c5f59a8ff35d499c0ee25a1d47e-Paper.pdf>. Citation on page 62.

AUDEBERT, N.; SAUX, B. L.; LEFEVRE, S. Deep Learning for Classification of Hyperspectral Data: A Comparative Review. **IEEE Geoscience and Remote Sensing Magazine**, v. 7, n. 2, p.

159–173, 2019. ISSN 2168-6831. Available: <https://doi.org/10.1109/MGRS.2019.2912563>. Citation on page 52.

BACKES, A. R.; CASANOVA, D.; BRUNO, O. M. Color texture analysis based on fractal descriptors. **Pattern Recognition**, v. 45, n. 5, p. 1984 – 1992, 2012. ISSN 0031-3203. Available: <https://doi.org/10.1016/j.patcog.2011.11.009>. Citation on page 94.

Bansal, A.; Nanduri, A.; Castillo, C. D.; Ranjan, R.; Chellappa, R. UMDFaces: An annotated face dataset for training deep networks. In: **2017 IEEE International Joint Conference on Biometrics (IJCB)**. IEEE, 2017. p. 464–473. ISBN 978-1-5386-1124-1. Available: <https://doi.org/10.1109/BTAS.2017.8272731>. Citation on page 91.

BARBER, D.; TORRE, F. D. L.; FEO, F.; FLORIDO, F.; GUARDIA, P.; MORENO, C.; QUIRALTE, J.; LOMBARDERO, M.; VILLALBA, M.; SALCEDO, G.; RODRíGUEZ, R. Understanding patient sensitization profiles in complex pollen areas: a molecular epidemiological study. **Allergy**, v. 63, n. 11, p. 1550–1558, 2008. Available: <https://doi.org/10.1111/j.1398-9995.2008.01807.x>. Citation on page 103.

BARCLAY, R.; MCELWAIN, J.; DILCHER, D.; SAGEMAN, B. The Cuticle Database: Developing an interactive tool for taxonomic and paleoenvironmental study of the fossil cuticle record. **Courier-Forschungsinstitut Senckenberg**, Schweizerbart Science Publishers, v. 258, p. 39–55, 2007. Available: <http://www.schweizerbart.de//publications/detail/isbn/9783510613885/CFS_Courier_Forschungsinstitut_Senckenbe>. Citations on pages 99 and 101.

BARCLAY, R. S.; WING, S. L. Improving the Ginkgo CO2 barometer: Implications for the early Cenozoic atmosphere. **Earth and Planetary Science Letters**, v. 439, p. 158–171, 2016. ISSN 0012-821X. Available: <https://doi.org/10.1016/j.epsl.2016.01.012>. Citations on pages 99 and 101.

Bello, I.; Fedus, W.; Du, X.; Cubuk, E. D.; Srinivas, A.; Lin, T.-Y.; Shlens, J.; Zoph, B. Revisiting ResNets: Improved Training and Scaling Strategies. **arXiv e-prints**, p. 1–18, Mar. 2021. Available: <https://arxiv.org/abs/2103.07579>. Citations on pages 56, 75, and 82.

BERTINI, A. Pliocene to Pleistocene palynoflora and vegetation in Italy: State of the art. **Quaternary International**, Elsevier, v. 225, n. 1, p. 5–24, 2010. ISSN 1040-6182. Available: <https://doi.org/10.1016/j.quaint.2010.04.025>. Citation on page 103.

BHUGRA, S.; MISHRA, D.; ANUPAMA, A.; CHAUDHURY, S.; LALL, B.; CHUGH, A.; CHINNUSAMY, V. Deep Convolutional Neural Networks Based Framework for Estimation of Stomata Density and Structure from Microscopic Images. In: **Computer Vision – ECCV 2018 Workshops**. Cham: Springer International Publishing, 2019. p. 412–423. ISBN 978-3-030-11024-6. Available: <https://doi.org/10.1007/978-3-030-11024-6_31>. Citations on pages 42 and 162.

BÖHME, F.; BISCHOFF, G.; ZEBITZ, C. P. W.; ROSENKRANZ, P.; WALLNER, K. Pesticide residue survey of pollen loads collected by honeybees (Apis mellifera) in daily intervals at three agricultural sites in South Germany. **PLOS ONE**, Public Library of Science, v. 13, n. 7, p. 1–21, Jul 2018. Available: <https://doi.org/10.1371/journal.pone.0199995>. Citations on pages 103 and 147.

BOSER, B. E.; GUYON, I. M.; VAPNIK, V. N. A Training Algorithm for Optimal Margin Classifiers. In: **Proceedings of the Fifth Annual Workshop on Computational Learning**

Theory. New York, NY, USA: Association for Computing Machinery, 1992. p. 144–152. ISBN 089791497X. Available: <https://doi.org/10.1145/130385.130401>. Citation on page 49.

BRODATZ, P. **Textures: a photographic album for artists and designers**. New York, USA: Dover Publications, 1966. Citations on pages 94 and 114.

Brown, T. B.; Mann, B.; Ryder, N.; Subbiah, M.; Kaplan, J.; Dhariwal, P.; Neelakantan, A.; Shyam, P.; Sastry, G.; Askell, A.; Agarwal, S.; Herbert-Voss, A.; Krueger, G.; Henighan, T.; Child, R.; Ramesh, A.; Ziegler, D. M.; Wu, J.; Winter, C.; Hesse, C.; Chen, M.; Sigler, E.; Litwin, M.; Gray, S.; Chess, B.; Clark, J.; Berner, C.; McCand lish, S.; Radford, A.; Sutskever, I.; Amodei, D. Language Models are Few-Shot Learners. **arXiv e-prints**, p. arXiv:2005.14165, May 2020. Available: <https://arxiv.org/abs/2005.14165>. Citation on page 39.

CAI, Z.; VASCONCELOS, N. Cascade R-CNN: Delving Into High Quality Object Detection. In: **2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition**. IEEE, 2018. p. 6154–6162. ISBN 978-1-5386-6420-9. Available: <https://doi.org/10.1109/cvpr.2018.00644>. Citations on pages 42, 73, and 88.

CAINE, R. S.; YIN, X.; SLOAN, J.; HARRISON, E. L.; MOHAMMED, U.; FULTON, T.; BISWAL, A. K.; DIONORA, J.; CHATER, C. C.; COE, R. A.; BANDYOPADHYAY, A.; MURCHIE, E. H.; SWARUP, R.; QUICK, W. P.; GRAY, J. E. Rice with reduced stomatal density conserves water and has improved drought tolerance under future climate conditions. **New Phytologist**, John Wiley & Sons, Ltd, v. 221, n. 1, p. 371–384, Jan 2019. ISSN 0028-646X. Available: <https://doi.org/10.1111/nph.15344>. Citations on pages 97 and 147.

CAPUTO, B.; HAYMAN, E.; MALLIKARJUNA, P. Class-specific material categorisation. In: **Tenth IEEE International Conference on Computer Vision (ICCV'05)**. [s.n.], 2005. v. 2, p. 1597–1604. ISSN 1550-5499. Available: <https://doi.org/10.1109/ICCV.2005.54>. Citation on page 94.

Carion, N.; Massa, F.; Synnaeve, G.; Usunier, N.; Kirillov, A.; Zagoruyko, S. End-to-End Object Detection with Transformers. In: **The European Conference on Computer Vision (ECCV)**. [s.n.], 2020. p. 1–17. Available: <https://arxiv.org/abs/2005.12872>. Citations on pages 40 and 48.

CARPENTER, K. J. Stomatal architecture and evolution in basal angiosperms. **American Journal of Botany**, John Wiley & Sons, Ltd, v. 92, n. 10, p. 1595–1615, Oct 2005. ISSN 0002-9122. Available: <https://doi.org/10.3732/ajb.92.10.1595>. Citation on page 97.

CASTRO, F. M.; MARÍN-JIMÉNEZ, M. J.; GUIL, N.; SCHMID, C.; ALAHARI, K. End-to-End Incremental Learning. In: **Computer Vision – ECCV 2018**. Springer International Publishing, 2018. p. 241–257. ISBN 978-3-030-01258-8. Available: <https://doi.org/10.1007/978-3-030-01258-8_15>. Citation on page 41.

CAVALIN, P.; OLIVEIRA, L. S. A Review of Texture Classification Methods and Databases. In: **2017 30th SIBGRAPI Conference on Graphics, Patterns and Images Tutorials (SIBGRAPI-T)**. [s.n.], 2017. p. 1–8. ISSN 2474-0705. Available: <https://doi.org/10.1109/SIBGRAPI-T.2017.10>. Citation on page 93.

Chen, K.; Pang, J.; Wang, J.; Xiong, Y.; Li, X.; Sun, S.; Feng, W.; Liu, Z.; Shi, J.; Ouyang, W.; Loy, C. C.; Lin, D. Hybrid Task Cascade for Instance Segmentation. In: **2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)**. IEEE, 2019. p. 4969–4978.

ISBN 978-1-7281-3293-8. Available: <https://doi.org/10.1109/CVPR.2019.00511>. Citation on page 88.

Chen, T.; Kornblith, S.; Norouzi, M.; Hinton, G. A Simple Framework for Contrastive Learning of Visual Representations. In: **Proceedings of the International Conference on Machine Learning**. [s.n.], 2020. p. 10709–10719. Available: <https://arxiv.org/abs/2002.05709>. Citations on pages 40 and 178.

CHEN, Y.; YANG, T.; ZHANG, X.; MENG, G.; XIAO, X.; SUN, J. DetNAS: Backbone Search for Object Detection. In: **Advances in Neural Information Processing Systems**. Curran Associates, Inc., 2019. v. 32, p. 6642–6652. Available: <https://proceedings.neurips.cc/paper/2019/file/228b25587479f2fc7570428e8bcbabdc-Paper.pdf>. Citation on page 42.

CHOLLET, F. **Deep Learning with Python**. 1. ed. Manning Publications, 2017. ISBN 1617294438,9781617294433. Available: <https://www.manning.com/books/deep-learning-with-python>. Citations on pages 39, 47, 53, 62, 63, and 68.

____. Xception: Deep learning with depthwise separable convolutions. In: **2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)**. IEEE, 2017. p. 1800–1807. Available: <https://doi.org/10.1109/CVPR.2017.195>. Citations on pages 56, 75, and 79.

CHOLLET, F. *et al.* **Keras**. [S.l.]: GitHub, 2015. Available online: <https://github.com/fchollet/keras>. Accessed: 2019-01-21. Citations on pages 17, 39, 52, 74, 83, and 128.

CIMPOI, M.; MAJI, S.; KOKKINOS, I.; VEDALDI, A. Deep Filter Banks for Texture Recognition, Description, and Segmentation. **International Journal of Computer Vision**, v. 118, n. 1, p. 65–94, May 2016. ISSN 1573-1405. Available: <https://doi.org/10.1007/s11263-015-0872-3>. Citations on pages 40, 42, 74, 84, 94, 96, 132, 133, and 143.

CLEVERT, D.; UNTERTHINER, T.; HOCHREITER, S. Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs). In: **4th International Conference on Learning Representations, ICLR 2016**. [s.n.], 2016. Available: <http://arxiv.org/abs/1511.07289>. Citation on page 67.

CORTES, C.; VAPNIK, V. Support-vector networks. **Machine Learning**, v. 20, n. 3, p. 273–297, Sep 1995. ISSN 1573-0565. Available: <https://doi.org/10.1007/BF00994018>. Citation on page 112.

CRESWELL, A.; WHITE, T.; DUMOULIN, V.; ARULKUMARAN, K.; SENGUPTA, B.; BHARATH, A. A. Generative Adversarial Networks: An Overview. **IEEE Signal Processing Magazine**, IEEE, v. 35, n. 1, p. 53–65, 2018. ISSN 1558-0792. Available: <https://doi.org/10.1109/MSP.2017.2765202>. Citation on page 47.

Cui, Y.; Song, Y.; Sun, C.; Howard, A.; Belongie, S. Large Scale Fine-Grained Categorization and Domain-Specific Transfer Learning. In: **IEEE/CVF Conference on Computer Vision and Pattern Recognition**. [s.n.], 2018. p. 4109–4118. ISSN 2575-7075. Available: <https://doi.org/10.1109/CVPR.2018.00432>. Citations on pages 43, 133, and 141.

DAI, j.; LI, Y.; HE, K.; SUN, J. R-FCN: Object Detection via Region-based Fully Convolutional Networks. In: **Proceedings of the 30th Conference on Neural Information Processing Systems**. USA: Curran Associates, Inc., 2016. p. 379–387. Available: <http://papers.nips.cc/paper/6465-r-fcn-object-detection-via-region-based-fully-convolutional-networks.pdf>. Citations on pages 88, 161, and 164.

D'AMATO, G.; CECCHI, L.; BONINI, S.; NUNES, C.; ANNESI-MAESANO, I.; BEHRENDT, H.; LICCARDI, G.; POPOV, T.; CAUWENBERGE, P. V. Allergenic pollen and pollen allergy in Europe. **Allergy**, John Wiley & Sons, Ltd, v. 62, n. 9, p. 976–990, Sep 2007. ISSN 0105-4538. Available: <https://doi.org/10.1111/j.1398-9995.2007.01393.x>. Citations on pages 103 and 104.

D'AMATO, G.; HOLGATE, S. T.; PAWANKAR, R.; LEDFORD, D. K.; CECCHI, L.; AL-AHMAD, M.; AL-ENEZI, F.; AL-MUHSEN, S.; ANSOTEGUI, I.; BAENA-CAGNANI, C. E.; BAKER, D. J.; BAYRAM, H.; BERGMANN, K. C.; BOULET, L.-P.; BUTERS, J. T.; D'AMATO, M.; DORSANO, S.; DOUWES, J.; FINLAY, S. E.; GARRASI, D.; GÓMEZ, M.; HAAHTELA, T.; HALWANI, R.; HASSANI, Y.; MAHBOUB, B.; MARKS, G.; MICHELOZZI, P.; MONTAGNI, M.; NUNES, C.; OH, J. J.-W.; POPOV, T. A.; PORTNOY, J.; RIDOLO, E.; ROSÁRIO, N.; ROTTEM, M.; SÁNCHEZ-BORGES, M.; SIBANDA, E.; SIENRA-MONGE, J. J.; VITALE, C.; ANNESI-MAESANO, I. Meteorological conditions, climate change, new emerging factors, and asthma and related allergic disorders. A statement of the World Allergy Organization. **World Allergy Organization Journal**, Elsevier, v. 8, Jan 2015. ISSN 1939-4551. Available: <https://doi.org/10.1186/s40413-015-0073-0>. Citations on pages 103 and 147.

DANA, K. J.; GINNEKEN, B. van; NAYAR, S. K.; KOENDERINK, J. J. Reflectance and Texture of Real-world Surfaces. **ACM Transactions on Graphics**, ACM, New York, NY, USA, v. 18, n. 1, p. 1–34, jan 1999. ISSN 0730-0301. Available: <https://doi.org/10.1145/300776.300778>. Citation on page 94.

DHILLON, A.; VERMA, G. K. Convolutional neural network: a review of models, methodologies and applications to object detection. **Progress in Artificial Intelligence**, v. 9, n. 2, p. 85–112, jun 2020. ISSN 2192-6360. Available: <https://doi.org/10.1007/s13748-019-00203-0>. Citations on pages 40, 48, and 98.

DOSOVITSKIY, A.; BEYER, L.; KOLESNIKOV, A.; WEISSENBORN, D.; ZHAI, X.; UNTERTHINER, T.; DEHGHANI, M.; MINDERER, M.; HEIGOLD, G.; GELLY, S.; USZKOREIT, J.; HOULSBY, N. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. In: **International Conference on Learning Representations**. [s.n.], 2021. Available: <https://openreview.net/forum?id=YicbFdNTTy>. Citations on pages 40 and 48.

DRBOHLAV, O.; LEONARDIS, A. Towards correct and informative evaluation methodology for texture classification under varying viewpoint and illumination. **Computer Vision and Image Understanding**, v. 114, n. 4, p. 439 – 449, 2010. ISSN 1077-3142. Available: <https://doi.org/10.1016/j.cviu.2009.08.006>. Citations on pages 94 and 95.

Du, X.; Zoph, B.; Hung, W.-C.; Lin, T.-Y. Simple Training Strategies and Model Scaling for Object Detection. **arXiv e-prints**, p. 1–9, Jun. 2021. Available: <https://arxiv.org/abs/2107.00057v1>. Citations on pages 88 and 89.

DUARTE, K. T.; CARVALHO, M. A. G. de; MARTINS, P. S. Segmenting high-quality digital images of stomata using the wavelet spot detection and the watershed transform. In: **VISIGRAPP (4: VISAPP)**. [s.n.], 2017. p. 540–547. Available: <https://www.scitepress.org/Papers/2017/61681/61681.pdf>. Citation on page 161.

DUCHI, J.; HAZAN, E.; SINGER, Y. Adaptive Subgradient Methods for Online Learning and Stochastic Optimization. **Journal of Machine Learning Research**, v. 12, p. 2121–2159, Jul. 2011. ISSN 1532-4435. Available: <https://dl.acm.org/doi/10.5555/1953048.2021068>. Citation on page 58.

EISELE, J. F.; FÄSSLER, F.; BÜRGEL, P. F.; CHABAN, C. A Rapid and Simple Method for Microscopy-Based Stomata Analyses. **PLOS ONE**, Public Library of Science, v. 11, n. 10, p. 1–13, Oct 2016. Available: <https://doi.org/10.1371/journal.pone.0164576>. Citation on page 97.

EVERINGHAM, M.; GOOL, L. V.; WILLIAMS, C. K. I.; WINN, J.; ZISSERMAN, A. The Pascal Visual Object Classes (VOC) Challenge. **International Journal of Computer Vision**, v. 88, n. 2, p. 303–338, Jun 2010. ISSN 1573-1405. Available: <https://doi.org/10.1007/s11263-009-0275-4>. Citations on pages 42 and 73.

Fatourechi, M.; Ward, R. K.; Mason, S. G.; Huggins, J.; Schlögl, A.; Birch, G. E. Comparison of Evaluation Metrics in Classification Applications with Imbalanced Datasets. In: **2008 Seventh International Conference on Machine Learning and Applications**. IEEE, 2008. p. 777–782. ISBN 978-0-7695-3495-4. Available: <https://doi.org/10.1109/ICMLA.2008.34>. Citation on page 91.

FETTER, K. C.; EBERHARDT, S.; BARCLAY, R. S.; WING, S.; KELLER, S. R. Stomata-Counter: a neural network for automatic stomata identification and counting. **New Phytologist**, v. 223, n. 3, p. 1671–1681, Aug 2019. ISSN 0028-646X. Available: <https://doi.org/10.1111/nph.15892>. Citations on pages 42, 99, 100, 101, and 162.

FILHO, A.; DE, H. A. **Redes complexas em sistemas celulares e moleculares de plantas**. Phd Thesis (text) — Universidade de São Paulo, May 2018. Available: <https://doi.org/10.11606/T.76.2018.tde-17092018-102042>. Citations on pages 99 and 100.

FILHO, H. A. d. A.; MACHICAO, J.; BRUNO, O. M. Geometric plasticity at leaves from Ctenanthe oppenheimiana probed by measure of distances between stomata. **Journal of Physics: Conference Series**, IOP Publishing, v. 936, p. 012094, Dec 2017. ISSN 1742-6588. Available: <https://doi.org/10.1088/1742-6596/936/1/012094>. Citation on page 100.

FLORINDO, J. B.; BRUNO, O. M. Fractal descriptors based on Fourier spectrum applied to texture analysis. **Physica A: Statistical Mechanics and its Applications**, Elsevier, v. 391, n. 20, p. 4909–4922, 2012. ISSN 0378-4371. Available: <https://doi.org/10.1016/j.physa.2012.03.039>. Citations on pages 112 and 207.

FRÖHLICH-NOWOISKY, J.; KAMPF, C. J.; WEBER, B.; HUFFMAN, J. A.; PÖHLKER, C.; ANDREAE, M. O.; LANG-YONA, N.; BURROWS, S. M.; GUNTHE, S. S.; ELBERT, W.; SU, H.; HOOR, P.; THINES, E.; HOFFMANN, T.; DESPRÉS, V. R.; PÖSCHL, U. Bioaerosols in the Earth system: Climate, health, and ecosystem interactions. **Atmospheric Research**, v. 182, p. 346–376, Dec 2016. ISSN 0169-8095. Available: <https://doi.org/10.1016/j.atmosres.2016.07.018>. Citation on page 102.

FUENTES, A.; YOON, S.; KIM, S. C.; PARK, D. S. A Robust Deep-Learning-Based Detector for Real-Time Tomato Plant Diseases and Pests Recognition. **Sensors**, v. 17, n. 9, 2017. ISSN 1424-8220. Available: <https://doi.org/10.3390/s17092022>. Citation on page 42.

FUENTES, A. F.; YOON, S.; LEE, J.; PARK, D. S. High-Performance Deep Neural Network-Based Tomato Plant Diseases and Pests Diagnosis System With Refinement Filter Bank. **Frontiers in Plant Science**, v. 9, p. 1–15, 2018. ISSN 1664-462X. Available: <https://doi.org/10.3389/fpls.2018.01162>. Citation on page 42.

FUKUSHIMA, K.; MIYAKE, S. Neocognitron: A Self-Organizing Neural Network Model for a Mechanism of Visual Pattern Recognition. In: **Competition and Cooperation in Neural Nets**. [s.n.], 1982. p. 267–285. ISBN 978-3-642-46466-9. Available: <https://doi.org/10.1007/978-3-642-46466-9_18>. Citation on page 49.

GAO, Z.; XIE, J.; WANG, Q.; LI, P. Global Second-Order Pooling Convolutional Networks. In: **2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)**. IEEE, 2019. (2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)), p. 3019–3028. ISSN 2575-7075. Available: <https://doi.org/10.1109/CVPR.2019.00314>. Citation on page 65.

GEURTS, P.; ERNST, D.; WEHENKEL, L. Extremely randomized trees. **Machine Learning**, v. 63, n. 1, p. 3–42, Apr 2006. ISSN 1573-0565. Available: <https://doi.org/10.1007/s10994-006-6226-1>. Citation on page 212.

Ghiasi, G.; Lin, T.; Le, Q. V. NAS-FPN: Learning Scalable Feature Pyramid Architecture for Object Detection. In: **2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)**. [s.n.], 2019. p. 7029–7038. ISSN 2575-7075. Available: <https://doi.org/10.1109/CVPR.2019.00720>. Citation on page 87.

Girshick, R. Fast R-CNN. In: **2015 IEEE International Conference on Computer Vision (ICCV)**. IEEE, 2015. p. 1440–1448. ISSN 2380-7504. Available: <https://doi.org/10.1109/ICCV.2015.169>. Citations on pages 42, 61, and 87.

Girshick, R.; Donahue, J.; Darrell, T.; Malik, J. Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation. In: **2014 IEEE Conference on Computer Vision and Pattern Recognition**. IEEE, 2014. p. 580–587. ISBN 978-1-4799-5118-5. Available: <https://doi.org/10.1109/CVPR.2014.81>. Citations on pages 42 and 87.

GLOROT, X.; BORDES, A.; BENGIO, Y. Deep Sparse Rectifier Neural Networks. In: **Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics**. PMLR, 2011. v. 15, p. 315–323. Available: <https://proceedings.mlr.press/v15/glorot11a.html>. Citations on pages 56 and 67.

GONZALEZ, R. C.; WOODS, R. E.; EDDINS, S. L. **Digital Image Processing Using MATLAB**. 2nd edition. ed. [S.l.]: Gatesmark Publishing, 2009. ISBN 0982085400. Citations on pages 112 and 208.

GONçALVES, W. N. **Caminhadas Determinísticas em Redes Complexas Aplicadas em Visão Computacional**. Phd Thesis (PhD Thesis) — Universidade de São Paulo, Brazil, 2010. Available: <https://doi.org/10.11606/D.55.2010.tde-08042010-112016>. Citation on page 114.

GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A. **Deep Learning**. MIT Press, 2016. ISBN 0262035618, 9780262035613. Available: <http://www.deeplearningbook.org>. Citations on pages 39, 47, 49, 58, 59, 62, 66, and 68.

Guo, Y.; Shi, H.; Kumar, A.; Grauman, K.; Rosing, T.; Feris, R. SpotTune: Transfer Learning Through Adaptive Fine-Tuning. In: **The IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)**. [s.n.], 2019. p. 4800–4809. ISSN 2575-7075. Available: <https://doi.org/10.1109/CVPR.2019.00494>. Citations on pages 41, 73, 74, 83, 86, and 144.

Ha, Q.; Liu, B.; Liu, F. Identifying Melanoma Images using EfficientNet Ensemble: Winning Solution to the SIIM-ISIC Melanoma Classification Challenge. **arXiv e-prints**, p. 1–6, Oct. 2020. Available: <https://arxiv.org/abs/2010.05351>. Citations on pages 62, 72, and 81.

HAN, J.; MA, K.-K. Rotation-invariant and scale-invariant Gabor features for texture image retrieval. **Image and Vision Computing**, Elsevier, v. 25, n. 9, p. 1474 – 1481, 2007. ISSN 0262-8856. Available: <https://doi.org/10.1016/j.imavis.2006.12.015>. Citation on page 206.

HANIN, B.; ROLNICK, D. How to Start Training: The Effect of Initialization and Architecture. In: **Advances in Neural Information Processing Systems**. Curran Associates, Inc., 2018. v. 31, p. 1–11. Available: <https://proceedings.neurips.cc/paper/2018/file/d81f9c1be2e08964bf9f24b15f0e4900-Paper.pdf>. Citations on pages 55 and 56.

HARALICK, R. M.; SHANMUGAM, K.; DINSTEIN, I. Textural features for image classification. **IEEE Transactions on Systems, Man, and Cybernetics**, IEEE, SMC-3, n. 6, p. 610–621, Nov 1973. ISSN 0018-9472. Available: <https://doi.org/10.1109/TSMC.1973.4309314>. Citations on pages 112 and 205.

He, K.; Gkioxari, G.; Dollár, P.; Girshick, R. Mask R-CNN. In: **2017 IEEE International Conference on Computer Vision (ICCV)**. IEEE, 2017. p. 2980–2988. ISBN 978-1-5386-1032-9. Available: <https://doi.org/10.1109/ICCV.2017.322>. Citations on pages 107 and 164.

He, K.; Zhang, X.; Ren, S.; Sun, J. Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification. In: **2015 IEEE International Conference on Computer Vision (ICCV)**. IEEE, 2015. p. 1026–1034. ISBN 978-1-4673-8391-2. Available: <https://doi.org/10.1109/ICCV.2015.123>. Citations on pages 56, 66, 67, and 80.

HE, K.; ZHANG, X.; REN, S.; SUN, J. Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition. **IEEE Transactions on Pattern Analysis and Machine Intelligence**, v. 37, n. 9, p. 1904–1916, Sept 2015. ISSN 0162-8828. Available: <https://doi.org/10.1109/TPAMI.2015.2389824>. Citation on page 65.

____. Deep Residual Learning for Image Recognition. In: **2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)**. [s.n.], 2016. p. 770–778. Available: <https://doi.org/10.1109/CVPR.2016.90>. Citations on pages 49, 54, 56, 75, 79, 112, and 162.

HE, T.; ZHANG, Z.; ZHANG, H.; ZHANG, Z.; XIE, J.; LI, M. Bag of Tricks for Image Classification with Convolutional Neural Networks. In: **2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)**. IEEE, 2019. p. 558–567. ISSN 2575-7075. Available: <https://doi.org/10.1109/CVPR.2019.00065>. Citation on page 56.

HIGAKI, T.; KUTSUNA, N.; HASEZAWA, S. Carta-based semi-automatic detection of stomatal regions on an arabidopsis cotyledon surface. **Plant Morphology**, The Japanese Society of Plant Morphology, v. 26, n. 1, p. 9–12, 2014. Available: <https://doi.org/10.5685/plmorphol.26.9>. Citation on page 161.

HINTON, G. E.; OSINDERO, S.; TEH, Y.-W. A Fast Learning Algorithm for Deep Belief Nets. **Neural Computation**, MIT Press, Cambridge, MA, USA, v. 18, n. 7, p. 1527–1554, Jul. 2006. ISSN 0899-7667. Available: <https://doi.org/10.1162/neco.2006.18.7.1527>. Citation on page 49.

HOCHREITER, S.; SCHMIDHUBER, J. Long Short-Term Memory. **Neural computation**, MIT Press, Cambridge, MA, USA, v. 9, n. 8, p. 1735–1780, Nov. 1997. ISSN 0899-7667. Available: <https://doi.org/10.1162/neco.1997.9.8.1735>. Citation on page 49.

HOLT, K. A.; BENNETT, K. D. Principles and methods for automated palynology. **New Phytologist**, John Wiley & Sons, Ltd, v. 203, n. 3, p. 735–742, Aug 2014. ISSN 0028-646X. Available: <https://doi.org/10.1111/nph.12848>. Citation on page 103.

HOSSAIN, S.; SERIKAWA, S. Texture databases - A comprehensive survey. **Pattern Recognition Letters**, v. 34, n. 15, p. 2007–2022, 2013. ISSN 0167-8655. Available: <https://doi.org/10.1016/j.patrec.2013.02.009>. Citation on page 93.

HOSSIN, M.; SULAIMAN, M. A Review on Evaluation Metrics for Data Classification Evaluations. **International Journal of Data Mining & Knowledge Management Process**, v. 5, n. 2, p. 01–11, Mar. 2015. ISSN 2231007X, 22309608. Available: <https://doi.org/10.5121/ijdkp.2015.5201>. Citations on pages 91 and 105.

HOU, R.; CHEN, C.; SHAH, M. Tube Convolutional Neural Network (T-CNN) for Action Detection in Videos. In: **2017 IEEE International Conference on Computer Vision (ICCV)**. IEEE, 2017. p. 5823–5832. ISSN 2380-7504. Available: <https://doi.org/10.1109/ICCV.2017.620>. Citation on page 51.

Hu, J.; Shen, L.; Albanie, S.; Sun, G.; Wu, E. Squeeze-and-Excitation Networks. **IEEE Transactions on Pattern Analysis and Machine Intelligence**, p. 1–13, 2019. Available: <https://doi.org/10.1109/TPAMI.2019.2913372>. Citations on pages 40, 54, 56, 66, 75, 81, 146, and 163.

HUANG, G.; LIU, Z.; MAATEN, L. v. d.; WEINBERGER, K. Q. Densely Connected Convolutional Networks. In: **2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)**. [s.n.], 2017. p. 2261–2269. ISSN 1063-6919. Available: <https://doi.org/10.1109/CVPR.2017.243>. Citations on pages 40, 54, 56, 75, 80, 112, and 163.

HUBEL, D. H.; WIESEL, T. N. Receptive fields of single neurones in the cat's striate cortex. **The Journal of Physiology**, John Wiley & Sons, Ltd, v. 148, n. 3, p. 574–591, Oct 1959. ISSN 0022-3751. Available: <https://doi.org/10.1113/jphysiol.1959.sp006308>. Citation on page 48.

HUMEAU-HEURTIER, A. Texture Feature Extraction Methods: A Survey. **IEEE Access**, IEEE, v. 7, p. 8975–9000, 2019. ISSN 2169-3536. Available: <https://doi.org/10.1109/ACCESS.2018.2890743>. Citation on page 93.

Hurt, J. A.; Scott, G. J.; Anderson, D. T.; Davis, C. H. Benchmark Meta-Dataset of High-Resolution Remote Sensing Imagery for Training Robust Deep Learning Models in Machine-Assisted Visual Analytics. In: **2018 IEEE Applied Imagery Pattern Recognition Workshop (AIPR)**. IEEE, 2018. p. 1–9. ISBN 978-1-5386-9306-3. Available: <https://doi.org/10.1109/AIPR.2018.8707433>. Citation on page 91.

IOFFE, S.; SZEGEDY, C. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In: **Proceedings of the 32nd International Conference on Machine Learning**. PMLR, 2015. v. 37, p. 448–456. Available: <http://proceedings.mlr.press/v37/ioffe15.html>. Citations on pages 41, 68, 78, and 80.

JAYAKODY, H.; LIU, S.; WHITTY, M.; PETRIE, P. Microscope image based fully automated stomata detection and pore measurement method for grapevines. **Plant methods**, BioMed Central, v. 13, p. 94–94, Nov 2017. ISSN 1746-4811. Available: <https://doi.org/10.1186/s13007-017-0244-9>. Citation on page 161.

JOUPPI, N. P.; YOUNG, C.; PATIL, N.; PATTERSON, D.; AGRAWAL, G.; BAJWA, R.; BATES, S.; BHATIA, S.; BODEN, N.; BORCHERS, A.; BOYLE, R.; CANTIN, P.-l.; CHAO, C.; CLARK, C.; CORIELL, J.; DALEY, M.; DAU, M.; DEAN, J.; GELB, B.; GHAEMMAGHAMI, T. V.; GOTTIPATI, R.; GULLAND, W.; HAGMANN, R.; HO, C. R.; HOGBERG, D.; HU, J.; HUNDT, R.; HURT, D.; IBARZ, J.; JAFFEY, A.; JAWORSKI, A.; KAPLAN, A.; KHAITAN, H.; KILLEBREW, D.; KOCH, A.; KUMAR, N.; LACY, S.; LAUDON, J.; LAW, J.; LE, D.; LEARY, C.; LIU, Z.; LUCKE, K.; LUNDIN, A.; MACKEAN, G.; MAGGIORE, A.; MAHONY, M.; MILLER, K.; NAGARAJAN, R.; NARAYANASWAMI, R.; NI, R.; NIX, K.; NORRIE, T.; OMERNICK, M.; PENUKONDA, N.; PHELPS, A.; ROSS, J.; ROSS, M.; SALEK, A.; SAMADIANI, E.; SEVERN, C.; SIZIKOV, G.; SNELHAM, M.; SOUTER, J.; STEINBERG, D.; SWING, A.; TAN, M.; THORSON, G.; TIAN, B.; TOMA, H.; TUTTLE, E.; VASUDEVAN, V.; WALTER, R.; WANG, W.; WILCOX, E.; YOON, D. H. In-Datacenter Performance Analysis of a Tensor Processing Unit. In: **Proceedings of the 44th Annual International Symposium on Computer Architecture**. Association for Computing Machinery, 2017. p. 1–12. ISBN 978-1-4503-4892-8. Available: <https://doi.org/10.1145/3079856.3080246>. Citations on pages 40 and 50.

KANDASWAMY, U.; SCHUCKERS, S. A.; ADJEROH, D. Comparison of texture analysis schemes under nonideal conditions. **IEEE Transactions on Image Processing**, v. 20, n. 8, p. 2260–2275, Aug 2011. ISSN 1057-7149. Available: <https://doi.org/10.1109/TIP.2010.2101612>. Citation on page 205.

KHAN, A.; SOHAIL, A.; ZAHOORA, U.; QURESHI, A. S. A survey of the recent architectures of deep convolutional neural networks. **Artificial Intelligence Review**, v. 53, n. 8, p. 5455–5516, Dec 2020. ISSN 1573-7462. Available: <https://doi.org/10.1007/s10462-020-09825-6>. Citations on pages 40, 65, 69, and 74.

KINGMA, D. P.; BA, J. Adam: A Method for Stochastic Optimization. In: **3rd International Conference on Learning Representations, ICLR**. [s.n.], 2015. Available: <http://arxiv.org/abs/1412.6980>. Citations on pages 58 and 169.

Kolesnikov, A.; Beyer, L.; Zhai, X.; Puigcerver, J.; Yung, J.; Gelly, S.; Houlsby, N. Big Transfer (BiT): General Visual Representation Learning. **arXiv e-prints**, p. arXiv:1912.11370, Dec. 2019. Available: <https://arxiv.org/abs/1912.11370>. Citation on page 39.

Kornblith, S.; Shlens, J.; Le, Q. V. Do Better ImageNet Models Transfer Better? In: **2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)**. [s.n.], 2019. p. 2656–2666. ISSN 2575-7075. Available: <https://doi.org/10.1109/CVPR.2019.00277>. Citations on pages 41 and 74.

Krause, J.; Stark, M.; Deng, J.; Fei-Fei, L. 3D Object Representations for Fine-Grained Categorization. In: **2013 IEEE International Conference on Computer Vision Workshops**. IEEE, 2013. p. 554–561. ISBN 978-1-4799-3022-7. Available: <https://doi.org/10.1109/ICCVW.2013.77>. Citation on page 73.

KRIZHEVSKY, A.; SUTSKEVER, I.; HINTON, G. E. ImageNet Classification with Deep Convolutional Neural Networks. In: **Proceedings of the 25th International Conference on**

Neural Information Processing Systems. USA: Curran Associates Inc., 2012. p. 1097–1105. Available: <http://dl.acm.org/citation.cfm?id=2999134.2999257>. Citations on pages 40, 49, 75, 76, and 162.

KWITT, P. M. R. **Salzburg texture image database (STex).** 2010. Available online: <http://wavelab.at/sources/STex/>. Accessed: 2021-11-01. Citation on page 94.

LAWSON, T.; BLATT, M. R. Stomatal Size, Speed, and Responsiveness Impact on Photosynthesis and Water Use Efficiency. **Plant Physiology**, v. 164, n. 4, p. 1556–1570, Apr 2014. ISSN 0032-0889. Available: <https://doi.org/10.1104/pp.114.237107>. Citation on page 97.

LAZEBNIK, S.; SCHMID, C.; PONCE, J. A sparse texture representation using local affine regions. **IEEE Transactions on Pattern Analysis and Machine Intelligence**, IEEE, v. 27, n. 8, p. 1265–1278, Aug 2005. ISSN 0162-8828. Available: <https://doi.org/10.1109/TPAMI.2005.151>. Citation on page 94.

Le, Q. V. Building high-level features using large scale unsupervised learning. In: **2013 IEEE International Conference on Acoustics, Speech and Signal Processing**. [s.n.], 2013. p. 8595–8598. ISBN 978-1-4799-0356-6. Available: <https://doi.org/10.1109/ICASSP.2013.6639343>. Citation on page 47.

LECUN, Y.; BENGIO, Y.; HINTON, G. Deep learning. **Nature**, v. 521, p. 436–444, May 2015. Available: <https://doi.org/10.1038/nature14539>. Citations on pages 39, 47, and 52.

LECUN, Y.; BOSER, B.; DENKER, J. S.; HENDERSON, D.; HOWARD, R. E.; HUBBARD, W.; JACKEL, L. D. Backpropagation Applied to Handwritten Zip Code Recognition. **Neural Computation**, v. 1, n. 4, p. 541–551, Dec 1989. ISSN 0899-7667. Available: <https://doi.org/10.1162/neco.1989.1.4.541>. Citations on pages 49 and 51.

LECUN, Y.; BOTTOU, L.; BENGIO, Y.; HAFFNER, P. Gradient-based learning applied to document recognition. **Proceedings of the IEEE**, v. 86, n. 11, p. 2278–2324, Nov 1998. ISSN 0018-9219. Available: <https://doi.org/10.1109/5.726791>. Citations on pages 49, 52, 65, and 75.

LEE, C.-Y.; GALLAGHER, P. W.; TU, Z. Generalizing Pooling Functions in Convolutional Neural Networks: Mixed, Gated, and Tree. In: **Proceedings of the 19th International Conference on Artificial Intelligence and Statistics**. PMLR, 2016. v. 51, p. 464–472. Available: <http://proceedings.mlr.press/v51/lee16a.html>. Citation on page 65.

LEUNG, T.; MALIK, J. Representing and recognizing the visual appearance of materials using three-dimensional textons. **International Journal of Computer Vision**, v. 43, n. 1, p. 29–44, Jun 2001. ISSN 1573-1405. Available: <https://doi.org/10.1023/A:1011126920638>. Citation on page 208.

LI, H.; CHAUDHARI, P.; YANG, H.; LAM, M.; RAVICHANDRAN, A.; BHOTIKA, R.; SOATTO, S. Rethinking the Hyperparameters for Fine-tuning. In: **International Conference on Learning Representations**. [s.n.], 2020. p. 1–20. Available: <https://openreview.net/forum?id=B1g8VkHFPH>. Citations on pages 41, 85, and 145.

LI, Y.; CHEN, Y.; WANG, N.; ZHANG, Z. Scale-Aware Trident Networks for Object Detection. In: **2019 IEEE/CVF International Conference on Computer Vision (ICCV)**. IEEE, 2019. p. 6053–6062. ISSN 2380-7504. Available: <https://doi.org/10.1109/ICCV.2019.00615>. Citations on pages 64 and 74.

LI, Z.; HOIEM, D. Learning without Forgetting. **IEEE Transactions on Pattern Analysis and Machine Intelligence**, v. 40, n. 12, p. 2935–2947, Dec. 2018. ISSN 1939-3539. Available: <https://doi.org/10.1109/TPAMI.2017.2773081>. Citation on page 41.

LIN, H.; JEGELKA, S. ResNet with One-Neuron Hidden Layers is a Universal Approximator. In: **Proceedings of the 32nd International Conference on Neural Information Processing Systems**. Curran Associates Inc., 2018. p. 6172–6181. Available: <https://dl.acm.org/doi/10.5555/3327345.3327515>. Citation on page 67.

LIN, M.; CHEN, Q.; YAN, S. Network In Network. **CoRR**, abs/1312.4400, 2013. Available: <http://arxiv.org/abs/1312.4400>. Citations on pages 65 and 77.

Lin, T.; Dollár, P.; Girshick, R.; He, K.; Hariharan, B.; Belongie, S. Feature Pyramid Networks for Object Detection. In: **2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)**. IEEE, 2017. p. 936–944. ISBN 978-1-5386-0457-1. Available: <https://doi.org/10.1109/CVPR.2017.106>. Citations on pages 88, 151, and 164.

LIN, T.; GOYAL, P.; GIRSHICK, R.; HE, K.; DOLLáR, P. Focal Loss for Dense Object Detection. In: **2017 IEEE International Conference on Computer Vision (ICCV)**. IEEE, 2017. p. 2999–3007. ISSN 2380-7504. Available: <https://doi.org/10.1109/iccv.2017.324>. Citations on pages 42, 60, 61, 88, 161, 166, and 169.

Lin, T.-Y.; Maire, M.; Belongie, S.; Bourdev, L.; Girshick, R.; Hays, J.; Perona, P.; Ramanan, D.; Zitnick, C. L.; Dollár, P. Microsoft COCO: Common Objects in Context. **arXiv e-prints**, p. arXiv:1405.0312, May 2014. Available: <https://arxiv.org/abs/1405.0312>. Citations on pages 42, 50, 73, 91, 97, and 106.

LIN, T.-Y.; ROYCHOWDHURY, A.; MAJI, S. Bilinear Convolutional Neural Networks for Fine-Grained Visual Recognition. **IEEE Transactions on Pattern Analysis and Machine Intelligence**, v. 40, n. 6, p. 1309–1322, Jun. 2018. ISSN 1939-3539. Available: <https://doi.org/10.1109/TPAMI.2017.2723400>. Citation on page 42.

LIU, H.; SIMONYAN, K.; YANG, Y. DARTS: Differentiable Architecture Search. In: **International Conference on Learning Representations**. [s.n.], 2019. p. 1–13. Available: <https://openreview.net/forum?id=S1eYHoC5FX>. Citations on pages 56 and 81.

LIU, L.; CHEN, J.; FIEGUTH, P.; ZHAO, G.; CHELLAPPA, R.; PIETIKÄINEN, M. From BoW to CNN: Two Decades of Texture Representation for Texture Classification. **International Journal of Computer Vision**, v. 127, n. 1, p. 74–109, Jan 2019. ISSN 1573-1405. Available: <https://doi.org/10.1007/s11263-018-1125-z>. Citations on pages 40, 93, 94, 96, 111, 124, 143, and 146.

LIU, L.; FIEGUTH, P.; WANG, X.; PIETIKÄINEN, M.; HU, D. Evaluation of LBP and Deep Texture Descriptors with a New Robustness Benchmark. In: **Computer Vision – ECCV 2016: 14th European Conference, Amsterdam, The Netherlands**. Cham: [s.n.], 2016. p. 69–86. ISBN 978-3-319-46487-9. Available: <https://doi.org/10.1007/978-3-319-46487-9_5>. Citations on pages 27, 96, and 145.

LIU, L.; OUYANG, W.; WANG, X.; FIEGUTH, P.; CHEN, J.; LIU, X.; PIETIKÄINEN, M. Deep Learning for Generic Object Detection: A Survey. **International Journal of Computer Vision**, v. 128, n. 2, p. 261–318, Feb 2020. ISSN 1573-1405. Available: <https://doi.org/10.1007/s11263-019-01247-4>. Citations on pages 42, 86, 87, 91, and 98.

Liu, P.; Han, S.; Meng, Z.; Tong, Y. Facial Expression Recognition via a Boosted Deep Belief Network. In: **2014 IEEE Conference on Computer Vision and Pattern Recognition**. [s.n.], 2014. p. 1805–1812. ISBN 978-1-4799-5118-5. Available: <https://doi.org/10.1109/CVPR.2014.233>. Citation on page 47.

LIU, W.; ANGUELOV, D.; ERHAN, D.; SZEGEDY, C.; REED, S.; FU, C.-Y.; BERG, A. C. SSD: Single Shot MultiBox Detector. In: **Computer Vision – ECCV 2016**. Springer International Publishing, 2016. p. 21–37. ISBN 978-3-319-46448-0. Available: <https://doi.org/10.1007/978-3-319-46448-0_2>. Citations on pages 42, 88, 151, and 162.

LOMAX, R. G.; HAHS-VAUGHN, D. L. **Statistical Concepts: A Second Course**. 4th ed. ed. Routledge, 2012. ISBN 9781136490064. Available: <https://books.google.com.br/books?id=coz_Fss1tY8C>. Citations on pages 123 and 211.

LU, H.; CAO, Z.; XIAO, Y.; FANG, Z.; ZHU, Y. Toward Good Practices for Fine-Grained Maize Cultivar Identification With Filter-Specific Convolutional Activations. **IEEE Transactions on Automation Science and Engineering**, v. 15, n. 2, p. 430–442, April 2018. ISSN 1545-5955. Available: <https://doi.org/10.1109/TASE.2016.2616485>. Citation on page 73.

LUOR, D.-C. A comparative assessment of data standardization on support vector machine for classification problems. **Intelligent Data Analysis**, IOS Press, v. 19, p. 529–546, 2015. ISSN 1571-4128. Available: <https://doi.org/10.3233/IDA-150730>. Citation on page 68.

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. Available: <https://doi.org/10.1016/j.ins.2020.09.058>. Citations on pages 128, 129, 133, 134, 136, 138, 139, 140, 142, 143, and 145.

MAAS, A. L.; HANNUN, A. Y.; NG, A. Y. Rectifier nonlinearities improve neural network acoustic models. In: **in ICML Workshop on Deep Learning for Audio, Speech and Language Processing**. [s.n.], 2013. Available: <https://ai.stanford.edu/~amaas/papers/relu_hybrid_icml2013_final.pdf>. Citation on page 67.

MAGRI, D.; Di Rita, F.; ARANBARRI, J.; FLETCHER, W.; GONZáLEZ-SAMPéRIZ, P. Quaternary disappearance of tree taxa from Southern Europe: Timing and trends. **Quaternary Science Reviews**, Elsevier, v. 163, p. 23–55, 2017. ISSN 0277-3791. Available: <https://doi.org/10.1016/j.quascirev.2017.02.014>. Citation on page 103.

MANJUNATH, B. S.; MA, W. Y. Texture features for browsing and retrieval of image data. **IEEE Transactions on Pattern Analysis and Machine Intelligence**, v. 18, n. 8, p. 837–842, Aug 1996. ISSN 0162-8828. Available: <https://doi.org/10.1109/34.531803>. Citations on pages 112, 205, and 206.

MCART, S. H.; FERSCH, A. A.; MILANO, N. J.; TRUITT, L. L.; BÖRÖCZKY, K. High pesticide risk to honey bees despite low focal crop pollen collection during pollination of a mass blooming crop. **Scientific Reports**, v. 7, n. 1, p. 46554, Apr 2017. ISSN 2045-2322. Available: <https://doi.org/10.1038/srep46554>. Citation on page 103.

MCCULLOCH, W. S.; PITTS, W. A logical calculus of the ideas immanent in nervous activity. **The bulletin of mathematical biophysics**, v. 5, n. 4, p. 115–133, Dec 1943. ISSN 1522-9602. Available: <https://doi.org/10.1007/BF02478259>. Citation on page 48.

MEYER, P.; NOBLET, V.; MAZZARA, C.; LALLEMENT, A. Survey on deep learning for radiotherapy. **Computers in Biology and Medicine**, v. 98, p. 126–146, Jul. 2018. ISSN 00104825. Available: <https://doi.org/10.1016/j.compbiomed.2018.05.018>. Citation on page 40.

MIKOłAJCZYK, A.; GROCHOWSKI, M. Data augmentation for improving deep learning in image classification problem. In: **2018 International Interdisciplinary PhD Workshop (IIPhDW)**. IEEE, 2018. p. 117–122. ISBN 978-1-5386-6143-7. Available: <https://doi.org/10.1109/IIPHDW.2018.8388338>. Citation on page 62.

MINSKY, M.; PAPERT, S. **Perceptrons: An introduction to computational geometry.** MIT press, 1969. ISBN 978-0-262-13043-1. Available: <https://mitpress.mit.edu/books/perceptrons>. Citation on page 48.

MISRA, D. Mish: A Self Regularized Non-Monotonic Activation Function. In: **31st British Machine Vision Conference 2020**. BMVA Press, 2020. Available: <https://www.bmvc2020-conference.com/assets/papers/0928.pdf>. Citation on page 67.

MOCCIA, S.; De Momi, E.; El Hadji, S.; MATTOS, L. S. Blood vessel segmentation algorithms — review of methods, datasets and evaluation metrics. **Computer Methods and Programs in Biomedicine**, v. 158, p. 71–91, 2018. ISSN 0169-2607. Available: <https://doi.org/10.1016/j.cmpb.2018.02.001>. Citation on page 91.

MOCHIDA, K.; KODA, S.; INOUE, K.; HIRAYAMA, T.; TANAKA, S.; NISHII, R.; MELGANI, F. Computer vision-based phenotyping for improvement of plant productivity: a machine learning perspective. **GigaScience**, v. 8, n. 1, 12 2019. ISSN 2047-217X. Available: <https://doi.org/10.1093/gigascience/giy153>. Citation on page 40.

MORALES-NAVARRO, S.; PéREZ-DíAZ, R.; ORTEGA, A.; MARCOS, A. de; MENA, M.; FENOLL, C.; GONZáLEZ-VILLANUEVA, E.; RUIZ-LARA, S. Overexpression of a SDD1-Like Gene From Wild Tomato Decreases Stomatal Density and Enhances Dehydration Avoidance in Arabidopsis and Cultivated Tomato. **Frontiers in Plant Science**, v. 9, p. 940, 2018. ISSN 1664-462X. Available: <https://doi.org/10.3389/fpls.2018.00940>. Citations on pages 97 and 147.

Mormont, R.; Geurts, P.; Marée, R. Comparison of Deep Transfer Learning Strategies for Digital Pathology. In: **2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)**. [s.n.], 2018. p. 2343–234309. ISSN 2160-7516. Available: <https://doi.org/10.1109/CVPRW.2018.00303>. Citations on pages 42, 74, and 84.

NAKKIRAN, P.; KAPLUN, G.; BANSAL, Y.; YANG, T.; BARAK, B.; SUTSKEVER, I. Deep Double Descent: Where Bigger Models and More Data Hurt. In: **International Conference on Learning Representations**. [s.n.], 2020. p. 1–24. Available: <https://openreview.net/forum?id=B1g5sA4twr>. Citation on page 62.

Nilsback, M.; Zisserman, A. Automated Flower Classification over a Large Number of Classes. In: **2008 Sixth Indian Conference on Computer Vision, Graphics Image Processing**. IEEE, 2008. p. 722–729. ISBN 978-0-7695-3476-3. Available: <https://doi.org/10.1109/ICVGIP.2008.47>. Citations on pages 15, 16, 54, 66, and 73.

OJALA, T.; MAENPAA, T.; PIETIKAINEN, M.; VIERTOLA, J.; KYLLONEN, J.; HUOVINEN, S. Outex - new framework for empirical evaluation of texture analysis algorithms. In: **Object recognition supported by user interaction for service robots**. [s.n.], 2002. v. 1, p. 701–706.

ISSN 1051-4651. Available: <https://doi.org/10.1109/ICPR.2002.1044854>. Citation on page 94.

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. ISSN 0162-8828. Available: <https://doi.org/10.1109/TPAMI.2002.1017623>. Citations on pages 94, 112, and 206.

OLAH, C.; CAMMARATA, N.; SCHUBERT, L.; GOH, G.; PETROV, M.; CARTER, S. Zoom In: An Introduction to Circuits. **Distill**, 2020. Available: <https://doi.org/10.23915/distill.00024.001>. Citation on page 39.

OLAH, C.; MORDVINTSEV, A.; SCHUBERT, L. Feature Visualization. **Distill**, 2017. Available: <https://doi.org/10.23915/distill.00007>. Citation on page 135.

O'MAHONY, N.; CAMPBELL, S.; CARVALHO, A.; HARAPANAHALLI, S.; HERNANDEZ, G. V.; KRPALKOVA, L.; RIORDAN, D.; WALSH, J. Deep Learning vs. Traditional Computer Vision. In: **Advances in Computer Vision**. Springer International Publishing, 2020. p. 128–144. ISBN 978-3-030-17795-9. Available: <https://doi.org/10.1007/978-3-030-17795-9_10>. Citation on page 47.

OQUAB, M.; BOTTOU, L.; LAPTEV, I.; SIVIC, J. Learning and Transferring Mid-level Image Representations Using Convolutional Neural Networks. In: **2014 IEEE Conference on Computer Vision and Pattern Recognition**. [s.n.], 2014. p. 1717–1724. ISSN 1063-6919. Available: <https://doi.org/10.1109/CVPR.2014.222>. Citations on pages 40, 72, and 73.

PABLOS, I.; WILDNER, S.; ASAM, C.; WALLNER, M.; GADERMAIER, G. Pollen Allergens for Molecular Diagnosis. **Current Allergy and Asthma Reports**, v. 16, n. 4, p. 31, Mar 2016. ISSN 1534-6315. Available: <https://doi.org/10.1007/s11882-016-0603-z>. Citations on pages 103, 104, and 147.

PAN, S. J.; YANG, Q. A Survey on Transfer Learning. **IEEE Transactions on Knowledge and Data Engineering**, v. 22, n. 10, p. 1345–1359, Oct 2010. ISSN 1041-4347. Available: <https://doi.org/10.1109/TKDE.2009.191>. Citation on page 71.

PASZKE, A.; GROSS, S.; CHINTALA, S.; CHANAN, G.; YANG, E.; DEVITO, Z.; LIN, Z.; DESMAISON, A.; ANTIGA, L.; LERER, A. Automatic differentiation in PyTorch. In: **NIPS 2017 Workshop Autodiff**. [S.l.: s.n.], 2017. Citations on pages 39, 52, 56, 68, 74, 115, 128, 143, and 162.

PERRONNIN, F.; SáNCHEZ, J.; MENSINK, T. Improving the Fisher Kernel for Large-Scale Image Classification. In: **Computer Vision – ECCV 2010**. Springer, 2010. p. 143–156. ISBN 978-3-642-15561-1. Available: <https://doi.org/10.1007/978-3-642-15561-1_11>. Citation on page 84.

PICARD, R.; GRACZYK, C.; MANN, S.; WACHMAN, J.; PICARD, L.; CAMPBELL, L.; NEGROPONTE, N. **Vision texture database**. [S.l.]: the Media Laboratory, MIT, Cambridge, Massachusetts, 1995. Available online: <https://vismod.media.mit.edu/vismod/imagery/VisionTexture/vistex.html>. Accessed: 2020-12-01. Citation on page 94.

PICCIALLI, F.; SOMMA, V. D.; GIAMPAOLO, F.; CUOMO, S.; FORTINO, G. A survey on deep learning in medicine: Why, how and when? **Information Fusion**, v. 66, p. 111 – 137, 2021.

ISSN 1566-2535. Available: <https://doi.org/10.1016/j.inffus.2020.09.006>. Citations on pages 40 and 72.

QIAN, N. On the momentum term in gradient descent learning algorithms. **Neural Networks**, v. 12, n. 1, p. 145 – 151, 1999. ISSN 0893-6080. Available: <https://doi.org/10.1016/S0893-6080(98)00116-6>. Citation on page 58.

RAGHU, M.; SCHMIDT, E. W. A Survey of Deep Learning for Scientific Discovery. **arXiv e-prints**, abs/2003.11755, 2020. Available: <https://arxiv.org/abs/2003.11755>. Citation on page 40.

RAKHLIN, A.; SHVETS, A.; IGLOVIKOV, V.; KALININ, A. A. Deep Convolutional Neural Networks for Breast Cancer Histology Image Analysis. In: **Image Analysis and Recognition**. Cham: Springer International Publishing, 2018. p. 737–744. ISBN 978-3-319-93000-8. Available: <https://doi.org/10.1007/978-3-319-93000-8_83>. Citation on page 42.

RAMACHANDRAN, P.; ZOPH, B.; LE, Q. V. Searching for Activation Functions. In: **6th International Conference on Learning Representations, ICLR 2018**. OpenReview.net, 2018. Available: <https://openreview.net/forum?id=Hkuq2EkPf>. Citation on page 67.

Raschka, S. Model Evaluation, Model Selection, and Algorithm Selection in Machine Learning. **arXiv e-prints**, p. 1–49, Nov. 2018. Available: <https://arxiv.org/abs/1811.12808>. Citations on pages 92, 96, and 104.

RAZAVIAN, A. S.; AZIZPOUR, H.; SULLIVAN, J.; CARLSSON, S. CNN Features Off-the-Shelf: An Astounding Baseline for Recognition. In: **2014 IEEE Conference on Computer Vision and Pattern Recognition Workshops**. [s.n.], 2014. p. 512–519. ISSN 2160-7516. Available: <https://doi.org/10.1109/CVPRW.2014.131>. Citations on pages 41, 74, 83, 132, and 133.

REAL, E.; AGGARWAL, A.; HUANG, Y.; LE, Q. V. Regularized Evolution for Image Classifier Architecture Search. **Proceedings of the AAAI Conference on Artificial Intelligence**, v. 33, n. 01, p. 4780–4789, Jul 2019. ISSN 2374-3468. Available: <https://doi.org/10.1609/aaai.v33i01.33014780>. Citations on pages 56 and 81.

REDMON, J.; FARHADI, A. YOLO9000: Better, Faster, Stronger. In: **2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)**. [s.n.], 2017. p. 6517–6525. ISSN 1063-6919. Available: <https://doi.org/10.1109/CVPR.2017.690>. Citations on pages 42 and 88.

Redmon, J.; Farhadi, A. YOLOv3: An Incremental Improvement. **arXiv e-prints**, p. arXiv:1804.02767, Apr. 2018. Available: <https://arxiv.org/abs/1804.02767>. Citations on pages 42, 88, and 107.

REN, S.; HE, K.; GIRSHICK, R.; SUN, J. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. **IEEE Transactions on Pattern Analysis and Machine Intelligence**, v. 39, n. 6, p. 1137–1149, June 2017. ISSN 0162-8828. Available: <https://doi.org/10.1109/TPAMI.2016.2577031>. Citations on pages 42, 52, 59, 87, 149, 152, 155, 161, 164, 165, and 169.

Ribani, R.; Marengoni, M. A Survey of Transfer Learning for Convolutional Neural Networks. In: **32nd SIBGRAPI Conference on Graphics, Patterns and Images Tutorials (SIBGRAPI-T)**. [s.n.], 2019. p. 47–57. ISSN 2474-0705. Available: <https://doi.org/10.1109/SIBGRAPI-T.2019.00010>. Citations on pages 40, 56, 71, and 72.

RODRÍGUEZ, P.; BAUTISTA, M. A.; GONZàLEZ, J.; ESCALERA, S. Beyond one-hot encoding: Lower dimensional target embedding. **Image and Vision Computing**, Elsevier, v. 75, p. 21–31, 2018. ISSN 0262-8856. Available: <https://doi.org/10.1016/j.imavis.2018.04.004>. Citation on page 59.

ROSENBLATT, F. **The Perceptron - A Perceiving and Recognizing Automaton.** [S.l.], 1957. Available: <https://blogs.umass.edu/brain-wars/files/2016/03/rosenblatt-1957.pdf>. Citation on page 48.

ROSS, B. C. Mutual information between discrete and continuous data sets. **PLoS ONE**, v. 9, n. 2, p. e87357, Feb. 2014. ISSN 1932-6203. Available: <https://doi.org/10.1371/journal.pone.0087357>. Citation on page 211.

RUDALL, P. J.; CHEN, E. D.; CULLEN, E. Evolution and development of monocot stomata. **American Journal of Botany**, John Wiley & Sons, Ltd, v. 104, n. 8, p. 1122–1141, Aug 2017. ISSN 0002-9122. Available: <https://doi.org/10.3732/ajb.1700086>. Citation on page 97.

Ruder, S. An overview of gradient descent optimization algorithms. **arXiv e-prints**, p. 1–14, Sep. 2016. Available: <https://arxiv.org/abs/1609.04747>. Citation on page 58.

RUMELHART, D. E.; HINTON, G. E.; WILLIAMS, R. J. Learning representations by back-propagating errors. **Nature**, v. 323, p. 533–536, Oct 1986. ISSN 1476-4687. Available: <https://doi.org/10.1038/323533a0>. Citations on pages 49, 57, and 59.

RUSS, J. C. **Fractal surfaces**. New York, US: Springer, 1994. Citation on page 207.

RUSSAKOVSKY, O.; DENG, J.; SU, H.; KRAUSE, J.; SATHEESH, S.; MA, S.; HUANG, Z.; KARPATHY, A.; KHOSLA, A.; BERNSTEIN, M.; BERG, A. C.; FEI-FEI, L. ImageNet Large Scale Visual Recognition Challenge. **International Journal of Computer Vision (IJCV)**, v. 115, n. 3, p. 211–252, 2015. Available: <https://doi.org/10.1007/s11263-015-0816-y>. Citations on pages 40, 50, 73, 76, 91, and 162.

Rusu, A. A.; Rabinowitz, N. C.; Desjardins, G.; Soyer, H.; Kirkpatrick, J.; Kavukcuoglu, K.; Pascanu, R.; Hadsell, R. Progressive Neural Networks. **arXiv e-prints**, Jun. 2016. Available: <https://arxiv.org/abs/1606.04671>. Citations on pages 42 and 74.

SAHA, B.; GUPTA, S.; PHUNG, D.; VENKATESH, S. Multiple task transfer learning with small sample sizes. **Knowledge and Information Systems**, v. 46, n. 2, p. 315–342, Feb 2016. ISSN 0219-3116. Available: <https://doi.org/10.1007/s10115-015-0821-z>. Citation on page 73.

SAKODA, K.; WATANABE, T.; SUKEMURA, S.; KOBAYASHI, S.; NAGASAKI, Y.; TANAKA, Y.; SHIRAIWA, T. Genetic Diversity in Stomatal Density among Soybeans Elucidated Using High-throughput Technique Based on an Algorithm for Object Detection. **Scientific Reports**, v. 9, n. 1, p. 7610, May 2019. ISSN 2045-2322. Available: <https://doi.org/10.1038/s41598-019-44127-0>. Citations on pages 42, 161, and 162.

SANTELIA, D.; LAWSON, T. Rethinking Guard Cell Metabolism. **Plant Physiology**, v. 172, n. 3, p. 1371–1392, 09 2016. ISSN 0032-0889. Available: <https://doi.org/10.1104/pp.16.00767>. Citation on page 97.

SANTURKAR, S.; TSIPRAS, D.; ILYAS, A.; Mądry, A. How Does Batch Normalization Help Optimization? In: **Proceedings of the 32nd International Conference on Neural Information Processing Systems**. Curran Associates Inc., 2018. p. 2488–2498. Available: <https://dl.acm.org/doi/10.5555/3327144.3327174>. Citation on page 68.

ŠAULIENĖ, I.; ŠUKIENĖ, L.; KAINOV, D.; GREIČIUVIENĖ, J. The impact of pollen load on quality of life: a questionnaire-based study in Lithuania. **Aerobiologia**, v. 32, n. 2, p. 157–170, Jun 2016. ISSN 1573-3025. Available: <https://doi.org/10.1007/s10453-015-9387-1>. Citation on page 104.

SCABINI, L. F. S.; CONDORI, R. H. M.; RIBAS, L. C.; BRUNO, O. M. Evaluating Deep Convolutional Neural Networks as Texture Feature Extractors. In: **Image Analysis and Processing – ICIAP 2019**. Springer International Publishing, 2019. p. 192–202. ISBN 978-3-030-30645-8. Available: <https://doi.org/10.1007/978-3-030-30645-8_18>. Citations on pages 41 and 84.

SHARAN, L.; ROSENHOLTZ, R.; ADELSON, E. Material perception: What can you see in a brief glance? **Journal of Vision**, v. 9, n. 8, p. 784–784, Aug. 2009. ISSN 1534-7362. Available: <https://doi.org/10.1167/9.8.784>. Citation on page 94.

SILVA, N. R. da; OLIVEIRA, M. W. d. S.; FILHO, H. A. d. A.; PINHEIRO, L. F. S.; ROSSATTO, D. R.; KOLB, R. M.; BRUNO, O. M. Leaf epidermis images for robust identification of plants. **Scientific Reports**, v. 6, n. 1, p. 1–10, May 2016. ISSN 2045-2322. Available: <https://doi.org/10.1038/srep25994>. Citations on pages 99 and 100.

SILVA, N. R. da; OLIVEIRA, M. W. d. S.; FILHO, H. A. d. A.; PINHEIRO, L. F. S.; KOLB, R. M.; BRUNO, O. M. Automatic Leaf Epidermis Assessment Using Fourier Descriptors in Texture Images. **Bio-protocol**, Bio-protocol LLC., v. 7, n. 23, p. 1–13, Dec 2017. ISSN 2331-8325. Available: <https://doi.org/10.21769/BioProtoc.2630>. Citation on page 100.

SIMONYAN, K.; ZISSERMAN, A. Very Deep Convolutional Networks for Large-Scale Image Recognition. In: **3rd International Conference on Learning Representations, ICLR**. [s.n.], 2015. Available: <http://arxiv.org/abs/1409.1556>. Citations on pages 49, 75, 76, and 112.

SINGH, A. K.; GANAPATHYSUBRAMANIAN, B.; SARKAR, S.; SINGH, A. Deep Learning for Plant Stress Phenotyping: Trends and Future Perspectives. **Trends in Plant Science**, v. 23, n. 10, p. 883 – 898, 2018. ISSN 1360-1385. Available: <https://doi.org/10.1016/j.tplants.2018.07.004>. Citations on pages 40 and 72.

SINGH, S.; KRISHNAN, S. Filter response normalization layer: Eliminating batch dependence in the training of deep neural networks. In: **2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)**. IEEE, 2020. p. 11234–11243. ISBN 978-1-7281-7168-5. Available: <https://doi.org/10.1109/CVPR42600.2020.01125>. Citation on page 69.

SONG, Y.; LI, Q.; FENG, D.; ZOU, J. J.; CAI, W. Texture image classification with discriminative neural networks. **Computational Visual Media**, v. 2, n. 4, p. 367–377, Dec 2016. ISSN 2096-0662. Available: <https://doi.org/10.1007/s41095-016-0060-6>. Citations on pages 41, 42, 74, 83, and 84.

Springenberg, J. T.; Dosovitskiy, A.; Brox, T.; Riedmiller, M. Striving for Simplicity: The All Convolutional Net. In: **International Conference on Learning Representations (workshop track)**. [s.n.], 2015. p. 1–14. Available: <https://arxiv.org/abs/1412.6806>. Citations on pages 41, 64, and 128.

SRIVASTAVA, N.; HINTON, G.; KRIZHEVSKY, A.; SUTSKEVER, I.; SALAKHUTDINOV, R. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. **The journal of machine learning research**, v. 15, n. 1, p. 1929–1958, 2014. ISSN 1532-4435. Available: <http://jmlr.org/papers/v15/srivastava14a.html>. Citations on pages 41, 62, and 76.

SRIVASTAVA, N.; SALAKHUTDINOV, R. Multimodal Learning with Deep Boltzmann Machines. **The Journal of Machine Learning Research**, JMLR.org, v. 15, n. 1, p. 2949–2980, Jan. 2014. ISSN 1532-4435. Available: <https://dl.acm.org/doi/10.5555/2627435.2697059>. Citation on page 47.

Sumbul, G.; Charfuelan, M.; Demir, B.; Markl, V. Bigearthnet: A Large-Scale Benchmark Archive for Remote Sensing Image Understanding. In: **IGARSS 2019 - 2019 IEEE International Geoscience and Remote Sensing Symposium**. IEEE, 2019. p. 5901–5904. ISBN 978-1-5386-9154-0. Available: <https://doi.org/10.1109/IGARSS.2019.8900532>. Citations on pages 72 and 91.

SUN, C.; SHRIVASTAVA, A.; SINGH, S.; GUPTA, A. Revisiting Unreasonable Effectiveness of Data in Deep Learning Era. In: **2017 IEEE International Conference on Computer Vision (ICCV)**. [s.n.], 2017. p. 843–852. ISSN 2380-7504. Available: <https://doi.org/10.1109/ICCV.2017.97>. Citations on pages 40, 72, and 73.

SUTSKEVER, I.; MARTENS, J.; DAHL, G.; HINTON, G. On the importance of initialization and momentum in deep learning. In: **Proceedings of the 30th International Conference on Machine Learning**. PMLR, 2013. v. 28, n. 3, p. 1139–1147. Available: <https://proceedings.mlr.press/v28/sutskever13.html>. Citation on page 58.

SZEGEDY, C.; IOFFE, S.; VANHOUCKE, V.; ALEMI, A. A. Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning. In: **Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence (AAAI-17)**. AAAI Press, 2017. p. 4278–4284. Available: <https://dl.acm.org/doi/10.5555/3298023.3298188>. Citations on pages 49, 56, 75, 79, and 82.

SZEGEDY, C.; LIU, W.; JIA, Y.; SERMANET, P.; REED, S.; ANGUELOV, D.; ERHAN, D.; VANHOUCKE, V.; RABINOVICH, A. Going deeper with convolutions. In: **2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)**. [s.n.], 2015. p. 1–9. ISSN 1063-6919. Available: <https://doi.org/10.1109/CVPR.2015.7298594>. Citations on pages 49, 54, 55, 65, 78, 82, and 112.

SZEGEDY, C.; VANHOUCKE, V.; IOFFE, S.; SHLENS, J.; WOJNA, Z. Rethinking the Inception Architecture for Computer Vision. In: **IEEE Conference on Computer Vision and Pattern Recognition (CVPR)**. [s.n.], 2016. p. 2818–2826. ISSN 1063-6919. Available: <https://doi.org/10.1109/CVPR.2016.308>. Citations on pages 66, 75, and 78.

SZELISKI, R. **Computer Vision: Algorithms and Applications**. 1. ed. Springer-Verlag London, 2011. 812 p. (Texts in Computer Science). ISBN 978-1-84882-935-0. Available: <https://doi.org/10.1007/978-1-84882-935-0>. Citation on page 50.

Tan, M.; Chen, B.; Pang, R.; Vasudevan, V.; Sandler, M.; Howard, A.; Le, Q. V. MnasNet: Platform-Aware Neural Architecture Search for Mobile. In: **The IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)**. [s.n.], 2019. p. 2815–2823. Available: <https://doi.org/10.1109/CVPR.2019.00293>. Citations on pages 52, 56, 81, and 86.

TAN, M.; LE, Q. EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks. In: **Proceedings of the 36th International Conference on Machine Learning**. PMLR, 2019. v. 97, p. 6105–6114. Available: <http://proceedings.mlr.press/v97/tan19a.html>. Citations on pages 40, 49, 55, 66, 72, 75, 81, 112, 146, and 163.

____. Efficientnetv2: Smaller models and faster training. In: MEILA, M.; ZHANG, T. (Ed.). **Proceedings of the 38th International Conference on Machine Learning**. PMLR, 2021. v. 139, p. 10096–10106. Available: <https://proceedings.mlr.press/v139/tan21a.html>. Citation on page 81.

Tan, M.; Pang, R.; Le, Q. V. EfficientDet: Scalable and Efficient Object Detection. In: **2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)**. IEEE, 2020. p. 10778–10787. ISBN 978-1-7281-7168-5. Available: <https://doi.org/10.1109/CVPR42600.2020.01079>. Citations on pages 42, 61, 73, and 89.

TANG, Z.; WANG, Y.; WANG, Q.; CHU, X. The Impact of GPU DVFS on the Energy and Performance of Deep Learning: An Empirical Study. In: **Proceedings of the Tenth ACM International Conference on Future Energy Systems**. New York, NY, USA: Association for Computing Machinery, 2019. p. 315–325. ISBN 9781450366717. Available: <https://doi.org/10.1145/3307772.3328315>. Citations on pages 40 and 50.

TONG, K.; WU, Y.; ZHOU, F. Recent advances in small object detection based on deep learning: A review. **Image and Vision Computing**, v. 97, p. 1–14, 2020. ISSN 0262-8856. Available: <https://doi.org/10.1016/j.imavis.2020.103910>. Citations on pages 40, 42, 48, 53, 72, and 87.

TRIANTAFILLOU, E.; ZHU, T.; DUMOULIN, V.; LAMBLIN, P.; EVCI, U.; XU, K.; GOROSHIN, R.; GELADA, C.; SWERSKY, K.; MANZAGOL, P.-A.; LAROCHELLE, H. Meta-Dataset: A Dataset of Datasets for Learning to Learn from Few Examples. In: **International Conference on Learning Representations**. [s.n.], 2020. Available: <https://openreview.net/forum?id=rkgAGAVKPr>. Citations on pages 50 and 91.

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, Apr 2005. ISSN 1573-1405. Available: <https://doi.org/10.1023/B:VISI.0000046589.39864.ee>. Citations on pages 112, 114, and 208.

VIALET-CHABRAND, S.; BRENDEL, O. Automatic measurement of stomatal density from microphotographs. **Trees**, v. 28, n. 6, p. 1859–1865, Dec 2014. ISSN 1432-2285. Available: <https://doi.org/10.1007/s00468-014-1063-5>. Citation on page 161.

VIDYA, K. S.; NG, E.; ACHARYA, U. R.; CHOU, S. M.; TAN, R. S.; GHISTA, D. N. Computer-aided diagnosis of Myocardial Infarction using ultrasound images with DWT, GLCM and HOS methods: A comparative study. **Computers in Biology and Medicine**, Elsevier, v. 62, p. 86 – 93, 2015. ISSN 0010-4825. Available: <https://doi.org/10.1016/j.compbiomed.2015.03.033>. Citation on page 93.

VOULODIMOS, A.; DOULAMIS, N.; DOULAMIS, A.; PROTOPAPADAKIS, E. Deep Learning for Computer Vision: A Brief Review. **Computational Intelligence and Neuroscience**, Hindawi, v. 2018, p. 1–13, Feb 2018. ISSN 1687-5265. Available: <https://doi.org/10.1155/2018/7068349>. Citation on page 48.

Wang, J.; Chen, Y.; Yu, H.; Huang, M.; Yang, Q. Easy Transfer Learning By Exploiting Intra-Domain Structures. In: **The IEEE International Conference on Multimedia and Expo (ICME)**. IEEE, 2019. p. 1210–1215. ISSN 1945-788X. Available: <https://doi.org/10.1109/ICME.2019.00211>. Citations on pages 83, 84, and 144.

WANG, Q.; WU, B.; ZHU, P.; LI, P.; ZUO, W.; HU, Q. Eca-net: Efficient channel attention for deep convolutional neural networks. In: **2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)**. [S.l.]: IEEE, 2020. p. 11531–11539. ISBN 978-1-7281-7168-5. Citation on page 86.

WANG, Z.; DONG, N.; ROSARIO, S. D.; XU, M.; XIE, P.; XING, E. P. Ellipse Detection of Optic Disc-and-Cup Boundary in Fundus Images. In: **2019 IEEE 16th International Symposium on Biomedical Imaging (ISBI 2019)**. IEEE, 2019. p. 601–604. ISSN 1945-8452. Available: <https://doi.org/10.1109/ISBI.2019.8759173>. Citation on page 98.

Wu, B.; Dai, X.; Zhang, P.; Wang, Y.; Sun, F.; Wu, Y.; Tian, Y.; Vajda, P.; Jia, Y.; Keutzer, K. FBNet: Hardware-Aware Efficient ConvNet Design via Differentiable Neural Architecture Search. In: **2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)**. [s.n.], 2019. p. 10726–10734. ISSN 2575-7075. Available: <https://doi.org/10.1109/CVPR.2019.01099>. Citations on pages 56 and 81.

WU, Y.; HE, K. Group Normalization. In: **Computer Vision – ECCV 2018**. Springer International Publishing, 2018. p. 3–19. ISBN 978-3-030-01261-8. Available: <https://doi.org/10.1007/978-3-030-01261-8_1>. Citations on pages 41 and 69.

Xie, S.; Girshick, R.; Dollár, P.; Tu, Z.; He, K. Aggregated Residual Transformations for Deep Neural Networks. In: **2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)**. IEEE, 2017. p. 5987–5995. ISBN 978-1-5386-0457-1. Available: <https://doi.org/10.1109/CVPR.2017.634>. Citations on pages 49, 64, 72, 75, and 81.

XU, Y.; KONG, Q.; WANG, W.; PLUMBLEY, M. D. Large-Scale Weakly Supervised Audio Classification Using Gated Convolutional Neural Network. In: **2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)**. IEEE, 2018. p. 121–125. ISSN 2379-190X. Available: <https://doi.org/10.1109/ICASSP.2018.8461975>. Citation on page 51.

YOO, D.; PARK, S.; LEE, J.; KWEON, I. S. Multi-scale pyramid pooling for deep convolutional representation. In: **2015 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)**. [s.n.], 2015. p. 71–80. ISSN 2160-7516. Available: <https://doi.org/10.1109/CVPRW.2015.7301274>. Citation on page 41.

YOSINSKI, J.; CLUNE, J.; BENGIO, Y.; LIPSON, H. How Transferable Are Features in Deep Neural Networks? In: **Advances in Neural Information Processing Systems 27**. Curran Associates, Inc., 2014. p. 3320–3328. Available: <http://papers.nips.cc/paper/5347-how-transferable-are-features-in-deep-neural-networks.pdf>. Citations on pages 42, 73, and 85.

Yosinski, J.; Clune, J.; Nguyen, A.; Fuchs, T.; Lipson, H. Understanding Neural Networks Through Deep Visualization. **arXiv e-prints**, p. 1–12, Jun. 2015. Available: <https://arxiv.org/abs/1506.06579>. Citation on page 53.

YU, D.; WANG, H.; CHEN, P.; WEI, Z. Mixed Pooling for Convolutional Neural Networks. In: **Rough Sets and Knowledge Technology**. Springer International Publishing, 2014. p. 364–375. ISBN 978-3-319-11740-9. Available: <https://doi.org/10.1007/978-3-319-11740-9_34>. Citation on page 65.

YU, F.; KOLTUN, V. Multi-Scale Context Aggregation by Dilated Convolutions. In: **4th International Conference on Learning Representations, ICLR**. [s.n.], 2016. p. 1–13. Available: <http://arxiv.org/abs/1511.07122>. Citation on page 64.

YUAN, J.; WANG, X.; ZHOU, H.; LI, Y.; ZHANG, J.; YU, S.; WANG, M.; HAO, M.; ZHAO, Q.; LIU, L.; LI, M.; LI, J. Comparison of Sample Preparation Techniques for Inspection of Leaf Epidermises Using Light Microscopy and Scanning Electronic Microscopy. **Frontiers in Plant Science**, v. 11, p. 133, 2020. ISSN 1664-462X. Available: <https://doi.org/10.3389/fpls.2020.00133>. Citation on page 97.

ZARITSKY, A.; NATAN, S.; HOREV, J.; HECHT, I.; WOLF, L.; BEN-JACOB, E.; TSARFATY, I. Cell Motility Dynamics: A Novel Segmentation Algorithm to Quantify Multi-Cellular Bright Field Microscopy Images. **PLoS ONE**, Public Library of Science, v. 6, n. 11, p. 1–10, 11 2011. ISSN 1932-6203. Available: <https://doi.org/10.1371/journal.pone.0027593>. Citation on page 93.

Zeiler, M. D. ADADELTA: An Adaptive Learning Rate Method. **arXiv e-prints**, p. arXiv:1212.5701, Dec. 2012. Available: <https://arxiv.org/abs/1212.5701>. Citation on page 58.

ZEILER, M. D.; FERGUS, R. Visualizing and Understanding Convolutional Networks. In: **Computer Vision – ECCV 2014**. Springer International Publishing, 2014. p. 818–833. ISBN 978-3-319-10590-1. Available: <https://doi.org/10.1007/978-3-319-10590-1_53>. Citations on pages 39, 53, and 76.

ZHANG, H.; XUE, J.; DANA, K. Deep TEN: Texture Encoding Network. In: **The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)**. [s.n.], 2017. p. 2896–2905. ISSN 1063-6919. Available: <https://doi.org/10.1109/CVPR.2017.309>. Citations on pages 41, 74, 83, 85, 96, and 143.

ZHANG, Q.; YANG, L. T.; CHEN, Z.; LI, P. A survey on deep learning for big data. **Information Fusion**, v. 42, p. 146 – 157, 2018. ISSN 1566-2535. Available: <https://doi.org/10.1016/j.inffus.2017.10.006>. Citation on page 39.

ZHANG, S.; WEN, L.; BIAN, X.; LEI, Z.; LI, S. Z. Single-shot refinement neural network for object detection. In: **2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition**. IEEE, 2018. p. 4203–4212. ISBN 978-1-5386-6420-9. Available: <https://doi.org/10.1109/CVPR.2018.00442>. Citations on pages 89 and 169.

Zhang, X.; Li, Z.; Change Loy, C.; Lin, D. PolyNet: A Pursuit of Structural Diversity in Very Deep Networks. **arXiv e-prints**, p. arXiv:1611.05725, Nov. 2017. Available: <http://arxiv.org/abs/1611.05725>. Citations on pages 52, 55, 66, 75, and 82.

ZHANG, Y.; DAVISON, B. D. Impact of imagenet model selection on domain adaptation. In: **2020 IEEE Winter Applications of Computer Vision Workshops (WACVW)**. IEEE, 2020. p. 173–182. ISBN 978-1-7281-7163-0. Available: <https://doi.org/10.1109/WACVW50321.2020.9096945>. Citation on page 74.

ZHANG, Z.; BATSELIER, K.; LIU, H.; DANIEL, L.; WONG, N. Tensor Computation: A New Framework for High-Dimensional Problems in EDA. **IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems**, IEEE, v. 36, n. 4, p. 521–536, 2017. ISSN 1937-4151. Available: <https://doi.org/10.1109/TCAD.2016.2618879>. Citation on page 51.

ZHAO, Q.; SHENG, T.; WANG, Y.; TANG, Z.; CHEN, Y.; CAI, L.; LING, H. M2Det: A Single-Shot Object Detector Based on Multi-Level Feature Pyramid Network. **Proceedings of the AAAI Conference on Artificial Intelligence**, v. 33, n. 01, p. 9259–9266, Jul. 2019. ISSN 2374-3468. Available: <https://doi.org/10.1609/aaai.v33i01.33019259>. Citations on pages 42 and 89.

ZHENG, H.; FU, J.; MEI, T.; LUO, J. Learning Multi-attention Convolutional Neural Network for Fine-Grained Image Recognition. In: **2017 IEEE International Conference on Computer Vision (ICCV)**. Venice: [s.n.], 2017. p. 5219–5227. ISBN 978-1-5386-1032-9. Available: <https://doi.org/10.1109/ICCV.2017.557>. Citation on page 41.

ZHOU, C.; PAFFENROTH, R. C. Anomaly Detection with Robust Deep Autoencoders. In: **Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining**. Association for Computing Machinery, 2017. p. 665–674. ISBN 978-1-4503-4887-4. Available: <https://doi.org/10.1145/3097983.3098052>. Citation on page 47.

ZISKA, L.; KNOWLTON, K.; ROGERS, C.; DALAN, D.; TIERNEY, N.; ELDER, M. A.; FILLEY, W.; SHROPSHIRE, J.; FORD, L. B.; HEDBERG, C.; FLEETWOOD, P.; HOVANKY, K. T.; KAVANAUGH, T.; FULFORD, G.; VRTIS, R. F.; PATZ, J. A.; PORTNOY, J.; COATES, F.; BIELORY, L.; FRENZ, D. Recent warming by latitude associated with increased length of ragweed pollen season in central North America. **Proceedings of the National Academy of Sciences**, National Academy of Sciences, v. 108, n. 10, p. 4248–4251, 2011. ISSN 0027-8424. Available: <https://doi.org/10.1073/pnas.1014107108>. Citation on page 103.

ZOPH, B.; CUBUK, E. D.; GHIASI, G.; LIN, T.-Y.; SHLENS, J.; LE, Q. V. Learning Data Augmentation Strategies for Object Detection. In: **Computer Vision – ECCV 2020**. Cham: Springer International Publishing, 2020. p. 566–583. ISBN 978-3-030-58583-9. Available: <https://doi.org/10.1007/978-3-030-58583-9_34>. Citations on pages 62 and 72.

____. Learning data augmentation strategies for object detection. In: VEDALDI, A.; BISCHOF, H.; BROX, T.; FRAHM, J.-M. (Ed.). **Computer Vision – ECCV 2020**. Cham: Springer International Publishing, 2020. p. 566–583. ISBN 978-3-030-58583-9. Available: <https://doi.org/10.1007/978-3-030-58583-9_34>. Citations on pages 167 and 168.

ZOPH, B.; VASUDEVAN, V.; SHLENS, J.; LE, Q. V. Learning transferable architectures for scalable image recognition. In: **2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition**. IEEE, 2018. p. 8697–8710. ISSN 2575-7075. Available: <https://doi.org/10.1109/CVPR.2018.00907>. Citations on pages 56, 75, 81, and 86.

APPENDIX

A

HAND-ENGINEERED FEATURE EXTRACTORS FOR TEXTURE ANALYSIS

This appendix presents a list of hand-engineered feature extractors for texture analysis. We refer the reader to (KANDASWAMY; SCHUCKERS; ADJEROH, 2011) for an extensive comparison between most of the traditional methods.

A.1 Gray Level Co-occurrence Matrix (GLCM)

The GLCM, originally proposed in 1973 (HARALICK; SHANMUGAM; DINSTEIN, 1973), has become a well-established method for texture characterization. Essentially, a GLCM is a square matrix of size L, corresponding to the number of gray levels in an image I. Each entry in a GLCM may be interpreted as a probability $p_{d,\theta}(i,j)$, reflecting the relative frequency of all pairs of pixels in an image I with gray levels i and j that are separated by a given distance d and an angle θ. Typically, $d = \{1,2\}$ and $\theta = \{0°, 45°, 90°, 135°\}$.

After setting up the GLCMs for different combinations of d and θ, second-order statistics are computed, such as the contrast, homogeneity, energy and correlation. Consequently, the number of descriptors in the resulting feature vector is given by the number of GLCMs times the number of second-order statistics.

A.2 Gabor wavelets (GW)

A two-dimensional Gabor filter $g(x,y)$, also referred to as a mother wavelet (MANJUNATH; MA, 1996), is given by:

$$g(x,y) = \frac{1}{2\pi\sigma_x\sigma_y} \exp\left[-\frac{1}{2}\left(\frac{x^2}{\sigma_x^2} + \frac{y^2}{\sigma_y^2}\right) + 2\pi jWx\right], \tag{A.1}$$

while its Fourier transformation $G(u,v)$ reads:

$$G(u,v) = \exp\left[\frac{1}{2}\left[\frac{(u-W)^2}{\sigma_u^2} + \frac{v^2}{\sigma_v^2}\right]\right]. \tag{A.2}$$

The parameters σ_x and σ_y control the filter bandwidth along the x and y axes respectively, $\sigma_u = \frac{1}{2\pi\sigma_x}$, $\sigma_v = \frac{1}{2\pi\sigma_y}$ and the pair $(W,0)$ represents the center frequency of the filter, when plotted with rectangular coordinates (u,v) in the frequency domain (HAN; MA, 2007).

Upon choosing N_s scales and N_o orientations, a bank of self-similar Gabor filters can be obtained by dilating and rotating $g(x,y)$. More specifically, the Gabor filter for a scale s and orientation o is given by:

$$g_{s,o}(x,y) = a^{-s} g(x',y'), \tag{A.3}$$

where $s = 1\ldots N_s$, $o = 1\ldots N_o$, $a > 1$ is a parameter, $x' = a^{-s}(x\cos\theta + y\sin\theta)$, $y' = a^{-s}(-x\sin\theta + y\cos\theta)$ and $\theta = o\pi/N_o$. The scale factor is a^{-s}.

In order to compute the parameters a, σ_u, σ_v and W, the following formula can be used (MANJUNATH; MA, 1996):

$$a = \left(\frac{u_h}{u_l}\right)^{\frac{1}{S-1}}, \ \sigma_u = \frac{(a-1)u_h}{(a+1)\sqrt{2\ln 2}}, \ W = u_h,$$

$$\sigma_v = \tan\left(\frac{\pi}{2k}\right)\left[u_h - 2\ln\left(\frac{2\sigma_u^2}{u_h}\right)\right]\left[2\ln 2 - \frac{(2\ln 2)^2\sigma_u^2}{u_h^2}\right]^{-1/2},$$

where u_l and u_h denote the lower and upper center frequencies of interest, respectively.

An image I can now be convolved with each Gabor filter in the filter bank, and statistics such as the average and standard deviation of the pixel values of the filtered images can be computed. Finally, a feature vector is created by concatenating these statistics. Thus, the number of descriptors is $N_s \times N_o$ times the number of statistics.

A.3 Local Binary Patterns (LBP)

Local Binary Patterns were introduced in (OJALA; PIETIKAINEN; MAENPAA, 2002). This texture method transforms an image I into an image I' by replacing every pixel value p_{ij} of I by a binary pattern that consists of eight bits reflecting whether or not the pixel value of the concerning neighboring pixel is smaller or larger than p_{ij}. A bit one is added to the binary pattern for every neighboring pixel value that is equal or larger than p_{ij}, whereas a zero is added if the opposite is true. Then, the resulting 8-bit patterns are replaced by their integer number representations p'_{ij}. Finally, the feature vector represents the histogram of these integer numbers in I' for a given number of bins.

A.4 Fractal Descriptors (FDs)

First of all, the power spectrum P_F of an image I is computed. In this sense, any position in P_F is expressed in polar coordinates $P_F(f,\phi)$. Then, $\widehat{P_F}(f)$ is defined as the average value of all pixels in P_F lying at the frequency f. Finally, a log−log scale is used for $\widehat{P_F}(f)$. Since the power spectrum curve obeys a power law (RUSS, 1994), $\widehat{P_F}(f)$ can be approximated on a log-log scale by a straight line. The slope of this line is the corresponding fractal dimension. In (FLORINDO; BRUNO, 2012), a multiscale transformation of the log-log curve is proposed to account for the intrinsic multiscale nature of the previous approach. It implies analyzing the log-log curve at multiple scales. Therefore, instead of having one fractal dimension, in the end, a set is obtained, one for each scale. The resulting values compose the feature vector of an image I.

A.5 Deterministic Tourist Walks (DTWs)

An image I is mapped onto an undirected regular graph $G(V,E)$, in such a way that the pixel with coordinates (x,y) in I corresponds to the vertex $v_{x,y} \in V$ (A. R. Backes and W. N. Gonçalves and A. S. Martinez and O. M. Bruno, 2010). The edges $e(v,v') \in E$ are established by connecting the vertices $v'_{x',y'} \in V$ whose Euclidean distance from the vertex $v_{x,y}$ is less or equal than a given value R. Moreover, a weighting function is defined $w : E \longrightarrow \mathbb{R} : w(v,v') = |I(x,y) - I(x',y')|$. Now, a set of thresholds T is defined, i.e. $T = \{t_0, t_0 + t_{inc}, t_0 + 2\,t_{inc}, \ldots, t_0 + (N_T - 1)\,t_{inc}\}$, after which every threshold $t_i \in T$ can be used to convert the original graph $G(V,E)$ to the graph $G_i(V,E')$ by removing all edges $e(v,v')$ whose $w(v,v')$ is greater than the threshold t_i. Clearly, $E' \subset E$.

Each graph $G_i(V,E')$ can subsequently be analyzed by using DTWs. A DTW can be understood as the computation of a path of vertices p. Let p_k indicate a vertex at position k in the path p. The vertex p_{k+1} is computed by choosing the minimum (or maximum) weight $w(p_k,v')$ for which there is an edge $e(p_k,v') \in E'$ that is not yet among the last μ vertices of the path p. In this context, μ is known as the *memory* of the DTW. The path p will eventually get trapped inside a cycle that is called the *attractor*. The number of vertices that make up the *attractor* is denoted by a.

The set of vertices that lies between the initial vertex of p and the last point before the *attractor* is called the *transient*, and its length is denoted by b.

Every vertex $v_{x,y} \in V$ in the graph $G_i(V,E')$ is considered as the starting point to initiate a DTW, so there are as many DTWs as there are vertices in the graph $G_i(V,E')$ for each memory μ. Then, histograms representing the number of DTWs with memory μ, that have a length equal to $a+b$ are constructed. Finally, all histograms resulting from the different thresholds t_i and memories μ are concatenated into a feature vector.

A.6 Texton histograms

This method was introduced in (LEUNG; MALIK, 2001) and then improved in (VARMA; ZISSERMAN, 2005). In contrast to the previous methods, this method requires the generation of a *visual dictionary* before computing a feature vector from a given image I. This *dictionary* is assembled using N_c image sets $\mathcal{T} = \{\mathcal{T}_1 \dots \mathcal{T}_{N_c}\}$, where N_c corresponds to the number of classes of the image dataset being analyzed. Each image I from a particular set $\mathcal{T}_j$ is convolved with a filter bank $\mathcal{F}$ creating a set of *filter responses*. These *filter responses* are stacked to N_f-dimensional vectors, where N_f is the number of *filter responses*. The N_f-dimensional vectors of all images in $\mathcal{T}_j$ are aggregated and clustered using the *K-Means* algorithm. The resulting k-cluster centers are called *textons* and are added to the *visual dictionary*. The same procedure is repeated for each image set in $\mathcal{T}$. Since each image set $\mathcal{T}_j$ contributes k textons, a *visual dictionary* of length $N_c \times k$ is created. The procedure for computing a feature vector from a given image I is as follows: (i) I is convolved with $\mathcal{F}$. (ii) The Euclidean distances between the resulting N_f-dimensional vectors of I and the *textons* in the *visual dictionary* are computed. (iii) A label is assigned to each N_f-dimensional vector indicating the position of the *texton* that lies closest to it. (iv) Finally, the feature vector of I is represented by a histogram of $N_c \times k$ bins of those labels.

In this project, we chose the MR8 filter bank (VARMA; ZISSERMAN, 2005), which consists of 38 filters. The first two filters are respectively a Gaussian filter and a Laplacian of the Gaussian filter, both with $\sigma = 10$. Edge and bar filters at three scales and six orientations complete the set. All *filter responses* from edge filters that are on the same scale are merged by taking the maximum response value across all orientations. The same procedure is repeated with the *filter responses* from bar filters. Thus, only 8 filter responses are computed in the MR8 filter bank. This method is rotation invariant since all rotational responses are merged. In Section **??**, this method is referred to as TH-MR8.

A.7 Fourier Magnitude Sampling

Fourier Magnitude Sampling (FMS) was adapted from (GONZALEZ; WOODS; EDDINS, 2009). FMS analyzes a given image I by estimating the behaviour of its Fourier magnitude spectrum F through two different sampling schemes: radial and circular. More specifically, let us define $F(r,\theta)$ as the Fourier magnitude spectrum of I expressed in polar coordinates, where r is the radial component $(0 \le r \le 1)$ and θ is the angular component $(0 \le \theta \le 2\pi)$. By convention, the point located at the center of F is its origin $(r = 0)$. In this context, $r = 1$ corresponds to the maximum radius position that can be considered in F from every angle θ. Then, a radial sample $F_{r_s \to r_f}(\hat{\theta})$ is defined as the vector that contains all the values from $F(r_s,\hat{\theta})$ to $F(r_f,\hat{\theta})$, where $0 \le r_s < r_f \le 1$. Similarly, a circular sample $F_{\theta_s \to \theta_f}(\hat{r})$ consists of all values from $F(\hat{r},\theta_s)$ to $F(\hat{r},\theta_f)$, where $0 \le \theta_s < \theta_f \le 2\pi$.

Let us define as $\hat{\Theta} = \{\hat{\theta}_1, \hat{\theta}_2, \ldots, \hat{\theta}_M\}$, $(M \geq 2)$, the set of angles from which radial samples are generated in such a way that: $\hat{\theta}_1 = \theta_s, \hat{\theta}_M = \theta_f$ and $\hat{\theta}_i = \theta_s + (i-1)\theta_{step}$, where $\theta_{step} = \frac{\theta_f - \theta_s}{M-1}$. Likewise, the set of radii $\hat{R} = \{\hat{r}_1, \hat{r}_2, \ldots, \hat{r}_N\}$, $(N \geq 2)$ needed to generate the N circular samples, are specified as follows: $\hat{r}_1 = r_s, \hat{r}_N = r_f$ and $\hat{r}_j = r_s + (j-1)r_{step}$, where $r_{step} = \frac{r_f - r_s}{N-1}$. Finally, for each retrieved sample (circular and/or radial), its average is computed and a feature vector of size $M + N$ is composed by those average values.

APPENDIX

B

TRADITIONAL FEATURE RANKING APPROACHES

The feature ranking approaches described here take as input: (i) a feature matrix (N) of size (N), and (ii) a set of N class labels (N). The term N indicates the number of samples in the texture dataset N, and N indicates the number of extracted features at the layer N of a pre-trained *CNN* model (similar to GP-CNN). Additionally, N represents the set of N unique class labels, and N indicates the N-th feature candidate located at the N-th column of N, N. A feature ranking approach then assigns one score N to every candidate N, where higher scores mean better feature candidates. The following lines contain a brief description of each feature ranking approach used in this paper.

The feature ranking approaches described here take as input: (i) a feature matrix ($\mathbf{X}_B^{(l)}$) of size ($n_B \times n_{A^{(l)}}$), and (ii) a set of n_B class labels (y_B). The term n_B indicates the number of samples in the texture dataset B, and $n_{A^{(l)}}$ indicates the number of extracted features at the layer l of a pre-trained *CNN* model (similar to GP-CNN). Additionally, C_B represents the set of n_{C_B} unique class labels, and $\vec{u_j}$ indicates the j-th feature candidate located at the j-th column of $\mathbf{X}_B^{(l)}$, $1 \leq j \leq n_{A^{(l)}}$. A feature ranking approach then assigns one score $s(\vec{u_j})$ to every candidate $\vec{u_j}$, where higher scores mean better feature candidates. The following lines contain a brief description of each feature ranking approach used in this paper.

- **Mutual Information** (MI): It ranks each feature candidate ($\vec{u_j}$) according to its high or low relationship with y_B. Given that $\vec{u_j}$ is a continuous variable and y_B is discrete, we estimate MI using the approach described in (ROSS, 2014).

- **One-way ANOVA F-test** (ANOVA): It ranks the features in $\mathbf{X}_B^{(l)}$ according to their F-ratios. In this sense, ANOVA uses y_B to estimate the inter-class *vs.* intra-class variability for each $\vec{u_j}$ (LOMAX; HAHS-VAUGHN, 2012). More specifically, we compute the F-ratio as $MS_{\text{betw}}/MS_{\text{with}}$, where $MS_{\text{betw}} = SS_{\text{betw}}/(n_{C_B} - 1)$ and $MS_{\text{with}} = SS_{\text{with}}/(n_B - n_{C_B})$. In this sense, the sum of SS_{betw} and SS_{with} is the same as computing the sample variance of $\vec{u_j}$, and their relationship provides valuable information about the distribution of the samples from $\vec{u_j}$ among the class labels (y_B). Therefore, we obtain a high F-ratio when SS_{betw} is high and SS_{with} is low, which means that samples belonging to different class labels are far apart, and samples within the same class label are very close.

- **Extremely Randomized Decision Trees** (ET): It is a tree-based ensemble method that ranks the $n_{A^{(l)}}$ features candidates from $\mathbf{X}_B^{(l)}$ by computing their *Gini importances*. In more detail, this method grows n_{tree} independent trees. The growing algorithm (GEURTS; ERNST; WEHENKEL, 2006) of every tree starts when its root node creates two child nodes by *splitting* the n_B samples from $\mathbf{X}_B^{(l)}$ into two non-overlapping subsets. More specifically, this *splitting* procedure involves choosing $\sqrt[2]{n_{A^{(l)}}}$ feature candidates randomly. Then, it computes one splitting point (p_j) per chosen feature candidate ($\vec{u_j}$) as a random value that falls within the range of $\vec{u_j}$. Next, ET uses the feature candidate with the highest *decrease of impurity* ($\vec{u}_{\text{best}}$) to split the set of samples. A set of zero impurity contains only samples that belong to the same class label. By contrast, a set with a high degree of *impurity* contains similar proportions of samples belonging to different class labels. ET recursively applies the same *splitting* procedure to every child node until: (i) it already achieved zero impurity or (ii) it consists of only constant features. When all trees stop growing, ET estimates the importance of each feature ($\vec{u_j}$) by computing its total *decrease of impurity* per tree and then averaging these results.

ICMC USP
SÃO CARLOS