UNIVERSIDADE DE SÃO PAULO
Instituto de Ciências Matemáticas e de Computação

Designing convolutional neural network architectures based on dynamical system concepts

Martha Dais Ferreira

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

ICMC USP
SÃO CARLOS

Martha Dais Ferreira

Designing convolutional neural network architectures based on dynamical system concepts

Doctoral dissertation submitted to the Institute of Mathematics and Computer Sciences – ICMC-USP, in partial fulfillment of the requirements for the degree of the Doctorate Program in Computer Science and Computational Mathematics. *FINAL VERSION*

Concentration Area: Computer Science and Computational Mathematics

Advisor: Prof. Dr. Rodrigo Fernandes de Mello

USP – São Carlos
March 2019

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

F383d

Ferreira, Martha Dais
Designing convolutional neural network architectures based on dynamical system concepts / Martha Dais Ferreira; orientador Rodrigo Fernandes de Mello. -- São Carlos, 2019.
135 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, 2019.

1. Deep Learning. 2. Convolutional Neural Networks. 3. Dynamical Systems. 4. Statistical Learning Theory. 5. Image-based False Nearest Neighbors. I. de Mello, Rodrigo Fernandes, 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

Martha Dais Ferreira

Projeto de arquiteturas de redes neurais artificiais convolucionais com o apoio de conceitos oriundos da área de sistemas dinâmicos

Tese apresentada ao Instituto de Ciências Matemáticas e de Computação – ICMC-USP, como parte dos requisitos para obtenção do título de Doutora em Ciências – Ciências de Computação e Matemática Computacional. *VERSÃO REVISADA*

Área de Concentração: Ciências de Computação e Matemática Computacional

Orientador: Prof. Dr. Rodrigo Fernandes de Mello

USP – São Carlos
Março de 2019

This thesis is dedicated to my parents and brother
who have always encouraged me to follow my dreams.

ACKNOWLEDGEMENTS

First of all, I would like to thank my advisor Prof. Rodrigo Fernandes de Mello, who gave me incredible support during my doctorate. He also allowed me to define and explore my own research questions, guiding me to the best direction. In addition, I appreciate his wisdom in other fields of life, which makes me evolve as a person.

I also thank Luis Gustavo Nonato and Moacir Antonelli Ponti for collaborating with my research, reviewing my manuscripts, and providing valuable feedback. My external supervisor Prof. Fernando Vieira Paulovich deserves thanks for receiving me at Dalhousie University and showing me new areas of knowledge.

Special thanks should be given to my family, especially my parents Antonio and Dais, my brother Arthur and my aunt Mera, for their assistance and unconditional support. I am very grateful that they are constantly encouraging me to continue my research to pursue academic career.

I also thank for the structure provided by the University of São Paulo, which was fundamental to develop this doctorate. Finally, it is essential to mention that this study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – Brasil (CAPES) – Finance Code 001.

“This was a triumph.
I’m making a note here:
HUGE SUCCESS!”
(McLain, Ellen (GLaDOS). "Still Alive." by Composer Jonathan Coulton.
The Orange Box Soundtrack. Video game: Portal, 2007.)

RESUMO

FERREIRA, M. D. **Projeto de arquiteturas de redes neurais artificiais convolucionais com o apoio de conceitos oriundos da área de sistemas dinâmicos**. 2019. 135 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, 2019.

Avanços tecnológicos têm permitido e motivado a produção e o armazenamento de grandes volumes de dados e, consequentemente, a necessidade de processamento a fim de obter informações que apoiem processos de tomada de decisão. Neste contexto, a área de Aprendizado Profundo (DL) tem apoiado a resolução de problemas complexos por meio da extração de características implícitas em conteúdos, tais como de imagens, áudios e vídeos, para produzir bons classificadores e regressores. Redes Neurais Convolucionais (CNN) estão entre as estratégias consideradas estado da arte em DL, apresentando ótimo desempenho em tarefas de diferentes domínios. O projeto de arquiteturas de CNNs é um dos maiores desafios envolvidos no uso dessa tecnologia, pois requer considerável conhecimento sobre o domínio da aplicação, bem como sobre transformações algébricas lineares e não-lineares. Atualmente, esses projetos são realizados de forma manual, contando portanto com procedimentos empíricos, ou por meio de algoritmos evolutivos que analisam diferentes arquiteturas candidatas, opção que excessivamente consome recursos computacionais. Em meio ao projeto, surge ainda outra questão que tem atraído a comunidade científica, ela se refere ao uso de arquiteturas profundas e suas relações com *overfitting*, o qual produz memorização dos exemplos de treinamento e, portanto, degradação no processo de aprendizado. Esses dois principais desafios motivaram esta tese de doutorado a trazer uma discussão e uma proposta de abordagem para auxiliar no projeto de arquiteturas de CNNs, bem como permitir a compreensão algébrica de suas operações. Inicialmente, as arquiteturas de CNN foram algebricamente formuladas, o que permitiu concluir que suas relações de imersão espaciais são similares às empregadas pela área de Sistemas Dinâmicos (DS), levando ao desenvolvimento de uma ferramenta de imersão denominada Falsos Vizinhos mais Próximos em Imagem (IFNN). IFNN estima o tamanho das máscaras convolucionais e auxilia na estimação do números de unidades para cada camada de uma arquitetura CNN, a partir do efeito causado pela reconstrução de espaços fase. Essa ferramenta é motivada por outra denominada Falsos Vizinhos mais Próximos (FNN), a qual estima a dimensão de incorporação mínima necessária para representar padrões recorrentes em dados com dependências temporais. Experimentos confirmam que as arquiteturas projetadas com o auxílio da IFNN produziram resultados similares aos reportados por arquiteturas profundas e muito mais complexas. Com base nesses experimentos, conclui-se que a IFNN auxilia no projeto de arquiteturas mais simples, rasas (no sentido de menor profundidade) e eficientes, as quais são mais rapidamente treinadas e fornecem garantias mais justas de aprendizado (necessitam de menor tamanho para as amostras de treinamento). Por fim, as arquiteturas obtidas com o apoio da IFNN foram analisadas com base em seus coeficientes de *Shattering* a fim de verificar suas complexidades relativas, essencialmente a cardinalidade de seus viéses.

Palavras-chave: Aprendizado Profundo, Redes Neurais Convolucionais, Sistemas Dinâmicos, Teoria do Aprendizado Estatístico, Falsos Vizinhos mais Próximos em Imagens.

ABSTRACT

FERREIRA, M. D. **Designing convolutional neural network architectures based on dynamical system concepts**. 2019. 135 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, 2019.

Technology advances have motivated the production and storage of large amounts of data and, consequently, the need for processing them out in order to support decision making. In this context, Deep Learning (DL) has emerged and provided major advances to solve complex supervised tasks through the direct manipulation of raw data content, such as images, audios and videos. Convolutional Neural Networks (CNN) are among the state-of-the-art strategies in DL, confirming relevant performance results in tasks of different domains. Currently, the design of CNN architectures is one of the major challenges involved in the practical use of DL, since it requires considerable knowledge about the application domain, linear and nonlinear algebraic transformations. Architectures are either manually designed, using empirical procedures, or with the support of evolutionary algorithms, an option that excessively consumes computational resources while analyzing candidate solutions. In addition to the architecture design, the possibility of overfitting has attracting the scientific community to study the effect of such complex models and whether they produce some memorization effect on training sets. Those two main challenges motivated this PhD thesis which brings up a proposal to support the automatic design of CNN architectures based on Dynamical System (DS) concepts. Initially, CNN architectures were algebraically formulated, allowing to take conclusions on the relationships of CNN input data organizations and spatial immersions from DS, leading to the development of an immersion tool called Image-based False Nearest Neighbors (IFNN). IFNN estimates the convolutional mask sizes and helps in the process of finding the adequate number of convolutional units per CNN layer by taking advantage of well-known effects caused by the reconstruction of phase spaces. This tool is based on the False-Nearest Neighbors (FNN) method, typically used to estimate the minimal embedding dimension to represent recurrence patterns of time series. Experiments confirm that architectures designed with the support of IFNN mostly usually produce results similar to deeper (and thus more complex) architectures. Based on those experiments, we concluded that IFNN supports the design of simpler, shallower (in the sense of depth) but yet efficient CNN architectures, which are faster to train and provide tighter learning guarantees according to the Statistical Learning Theory (SLT) thus requiring smaller training samples. Finally, the CNN architectures after IFNN were analyzed based on their Shattering coefficients in attempt to verify their relative complexities, and most essentially the cardinalities of their spaces of admissible functions, a.k.a. biases.

Keywords: Deep Learning, Convolutional Neural Networks, Dynamical Systems, Statistical Learning Theory, Image-based False Nearest Neighbors.

LIST OF FIGURES

LIST OF ALGORITHMS

LIST OF TABLES

LIST OF ABBREVIATIONS AND ACRONYMS

Adam	Adaptive Moment Estimation
ANN	Artificial Neural Network
CIFAR-10	Canadian Institute for Advanced Research (10 classes)
CMU	Carnegie Mellon University Face Images
CNN	Convolutional Neural Network
COIL-100	Columbia University Image Library (100 classes)
CSC	Convolutional Space Coding
CSI	Cover Song Identification
DBN	Deep Belief Networks
DCWN	Deep Convolutional Wavelet Network
DL	Deep Learning
DS	Dynamical Systems
ER	Error rates
ERMP	Empirical Risk Minimization Principle
FNN	False Nearest Neighbors
GTSRB	German Traffic Sign Recognition Benchmark
IFNN	Image-based False Nearest Neighbors
IJCNN	International Joint Conference on Neural Networks
LCN	Local Contrast Normalization
LLN	Law of Large Numbers
LRN	Local Response Normalization
MCDNN	Multi-Column Deep Convolutional Neural Networks
MENNDL	Multi-node Evolutionary Neural Networks for Deep Learning
MIR	Music Information Retrieval
ML	Machine Learning
MLP	Multilayer Perceptron
MNIST	Modified National Institute of Standards and Technology
MS-CNN	Multi-Scale Convolutional Neural Network
Nadam	Nesterov-accelerated Adaptive Moment Estimation
NAG	Nesterov Accelerated Gradient
PCA	Principal Components Analysis

PNG	Portable Network Graphics
PP	Preprocessing
PSO	Particle Swarm Optimization
RBF	Radial Basis Function
RBM	Restricted Boltzmann Machine
ReLU	Rectified Linear Unit
RMHC	Random Mutual Hill Climbing
RMS	Root Mean Square
RNN	Recurrent Neural Networks
SGD	Stochastic Gradient Descent
SLT	Statistical Learning Theory
SNoW	Sparse Network of Winnows
SVM	Support Vector Machine
VC	Vapnik-Chervonenkis

CONTENTS

CHAPTER

1

INTRODUCTION

With the technological advances of digital devices and the Internet, several tools have been designed, developed and improved to provide an efficient organization, search, and modeling of big data scenarios, based on the intrinsic knowledge from data content (PAZZANI; BILLSUS, 1997; ROOHULLAH; JAAFAR, 2011; BRIN; PAGE, 2012; GAO, 2013; JING; LIU; TSAI, 2017). In this context, Deep Learning (DL) has received a great attention due to its ability to build up generalized models to proceed with the classification of huge datasets by extracting meaningful information embedded in raw data (FERREIRA *et al.*, 2018). That ability has been widely employed to identify objects in images, transcribe speech into text, textual modeling, e-commerce recommendation, and analyze social networks (KRIZHEVSKY; SUTSKEVER; HINTON, 2012; SCHLUTER; BOCK, 2014; ELKAHKY; SONG; HE, 2015; COVINGTON; ADAMS; SARGIN, 2016).

The main target of DL strategies is to simplify data modeling and classification for general-purpose domains (LECUN; BENGIO; HINTON, 2015), most specially supporting solutions for very complex Machine Learning (ML) problems, such as face recognition (LAWRENCE *et al.*, 1997; ROWLEY; BALUJA; KANADE, 1998; BHATT *et al.*, 2012; LECUN; BENGIO; HINTON, 2015), the investigation of criminal scenes (LIU; LI, 2014), medical diagnosis (ADLER; GUARDO, 1994; CHENG; LIN; MAO, 1996), urban planning (HECHT *et al.*, 2013), weather forecasting (RASOULI; HSIEH; CANNON, 2012), object (KRIZHEVSKY; SUTSKEVER; HINTON, 2012) and vehicle licensing recognition (BROUSSARD *et al.*, 1999). Successful results have also motivated the adoption of DL algorithms by large companies, such as Yahoo!, Google, Bing and Facebook (ELKAHKY; SONG; HE, 2015; COVINGTON; ADAMS; SARGIN, 2016; ROOHULLAH; JAAFAR, 2011; BRIN; PAGE, 2012; GAO, 2013; LECUN; BENGIO; HINTON, 2015), being also included in consumer products, such as cameras and smartphones (LECUN; BENGIO; HINTON, 2015).

In most applications, DL architectures have been currently providing the best results, what has consequently motivated its wide adoption in literature (LECUN; BENGIO; HIN-

TON, 2015; GOODFELLOW; BENGIO; COURVILLE, 2016). The most relevant contribution of DL is the design of algorithms and architectures capable of implicitly (some may say automatically) modeling the relevant features hidden in raw data, bringing this branch to the representation learning area (LECUN; BENGIO; HINTON, 2015). DL architectures correspond to multilayered structures composed of different mathematical operators (or units) in order to learn nonlinear functions from training data, producing relevant features even for very complex datasets (KRIZHEVSKY; SUTSKEVER; HINTON, 2012; SZEGEDY *et al.*, 2015; HE *et al.*, 2016; LECUN *et al.*, 1999; SRIVASTAVA *et al.*, 2014; COGSWELL *et al.*, 2016; PEREZ; WANG, 2017; MELLO; FERREIRA; PONTI, 2017; MELLO; PONTI; FERREIRA, 2018). This process attempts to produce the same result as a kernel in the context of ML (LECUN *et al.*, 1998; LECUN; BENGIO; HINTON, 2015), i.e., some input space transformation that outputs a relevant enough features space on which learning is possible (SCHÖLKOPF; SMOLA, 2002; MELLO; PONTI, 2018). As consequence, DL algorithms are employed to proceed with multiple levels (layers) of data transformations, including linear and nonlinear operators, in an attempt to obtain relevant features for inducing some classification or regression function f.

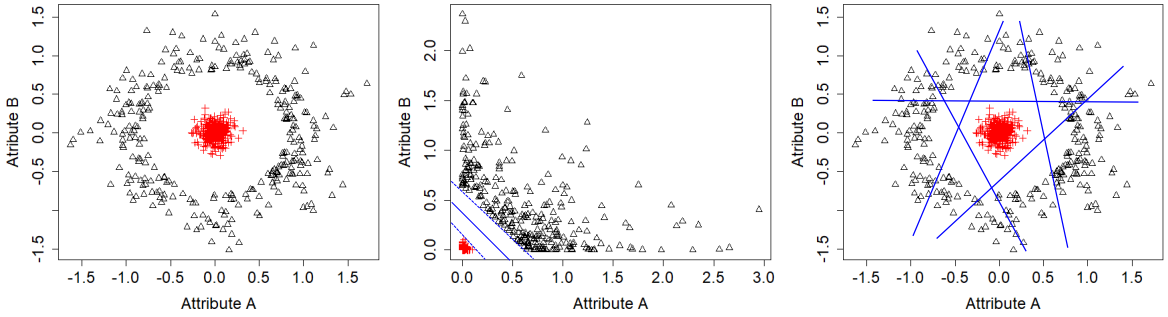
Typically, ML techniques require a careful design and considerable domain expertise to provide an adequate feature extraction in most recognition and classification problems (MELLO; PONTI, 2018; KRIZHEVSKY; SUTSKEVER; HINTON, 2012; SZEGEDY *et al.*, 2015; GARRO; VÁZQUEZ, 2015; MALLAT, 2016; FERREIRA *et al.*, 2018). This requirement also holds for DL algorithms, once they are designed to tackle specific learning tasks (LECUN; BENGIO; HINTON, 2015). The greatest difference of DL solutions in comparison to other ML techniques is the presence of implicit transformations to proceed with the classification task, involving a set of separating hyperplanes to divide the input space. Those transformations are learned along the training process, contrasting to other ML solutions that require some explicit space kernelization, such as Support Vector Machines (SVM) (CORTES; VAPNIK, 1995; MELLO; PONTI, 2018).

SVMs were designed to provide as tight as possible learning guarantees according to the Statistical Learning Theory (SLT) (VAPNIK, 2000; LUXBURG; SCHÖLKOPF, 2011), being considered the best ML supervised algorithm when the correct input space is provided (MELLO; PONTI, 2018) [1]. This is ensured because SVMs induce the best as possible linear separation in some input space by maximizing the margin between the decision boundary (a.k.a. separating hyperplane) and training examples labeled according to different classes. In that sense, SVMs do not simply respect the Empirical Risk Minimization Principle (ERMP), but ensure the greatest as possible theoretical learning guarantees (upper bounds) for it (VAPNIK, 2000).

However, it is worth to highlight that SVMs require the input space to be linearly separable to provide the best results, thus relying on explicit space transformations to obtain an ideal (or close enough) features space (MELLO; PONTI, 2018). Such transformations are mostly

[1] Please, refer to the Large-Margin Bound for the detailed proof.

manually designed by some expert who has enough knowledge about the input space and about the problem itself. Figure 1a illustrates an input space with training examples divided in two classes, in which Figure 1b shows the classification results using SVM after applying a second-order polynomial kernel (explicit transformation), and Figure 1c presents the same classification but using an Artificial Neural Network (ANN) that builds up the implicit transformation using several hyperplanes dividing the input space. In this particular instance, we assume the SVM kernel is known *a priori* (polynomial kernel), while the ANN learns it from training examples.



(a) A two-class input space. (b) An explicit kernel. (c) An implicit kernel.

Figure 1 – Illustrating the difference of an explicit kernel employed by SVM and an implicit one produced by some ANN. Figure (a) shows an example of a dataset with two attributes and two classes (triangles and crosses). Figure (b) illustrates the classification performed by SVM while using a second-order polynomial kernel. The solid line corresponds to the separating hyperplane, and dashed lines represent the margin to be maximized. Figure (c) illustrates the classification performed with some ANN, in which lines represent the separating hyperplanes.

Convolutional Neural Networks (CNNs) are considered a promising tool in the context of DL, given they have been providing relevant results in different application domains and their ease of usage (COATES *et al.*, 2011; COX; PINTO, 2011; NAKASHIKA *et al.*, 2012; BERGSTRA; BENGIO, 2012; SERMANET *et al.*, 2014; KRIZHEVSKY; SUTSKEVER; HINTON, 2012; ZEILER; FERGUS, 2014; CIREGAN; MEIER; SCHMIDHUBER, 2012; LECUN; BENGIO; HINTON, 2015; GOODFELLOW; BENGIO; COURVILLE, 2016). Besides such successful performances, CNNs still do not rely on strong theoretical foundations to prove learning boundaries nor count on formal methodologies to design their architectures, which are empirically assessed. Both drawbacks motivated this thesis to focus on such particular network in an attempt to tackle those challenges but most specially the design of CNN architectures.

CNN architectures have been manually designed for particular tasks, which involve the empirical process of performing several experiments in order to find some good enough parametrization, such as the number of layers, the number of units (operators) per layer, the type of unit at each layer (convolution, pooling, etc.), and the convolutional mask sizes (SZEGEDY *et al.*, 2015; KRIZHEVSKY; SUTSKEVER; HINTON, 2012; GARRO; VÁZQUEZ, 2015). This empirical process makes evident the lack of any formal foundation to support the design of CNN

architectures. As a natural result, the CNN architectures implicitly build up unknown kernels to somehow provide more effective features spaces to perform classification.

The processing and time costs associated with such empirical designing process have motivated researchers to consider predefined architectures (e.g. LeNet, AlexNet, VGG16) on new datasets, and, in some cases, previously trained models (transfer learning) (BEN-DAVID *et al.*, 2010; HUH; AGRAWAL; EFROS, 2016; CHOI *et al.*, 2017; YANG *et al.*, 2015). As a consequence, predefined CNN architectures may not be enough to provide good results on other application domains, thus leading to underfitting, or may be more complex than necessary, producing overfitting (COGSWELL *et al.*, 2016; MALLAT, 2016; SRIVASTAVA *et al.*, 2014). However, the question that still remains is how to design a good enough CNN architecture to address a specific classification task.

In this research scenario, evolutionary algorithms have been employed to search for adequate architectures for particular tasks (GARRO; VÁZQUEZ, 2015; YOUNG *et al.*, 2015; ALBELWI; MAHMOOD, 2017). Despite such approach avoids the manual design, it still requires training the CNN configurations under analysis, being a time and resource demanding process (HE *et al.*, 2016; KOVALEV; KALINOVSKY; KOVALEV, 2016; SZEGEDY *et al.*, 2015; KRIZHEVSKY; SUTSKEVER; HINTON, 2012). Moreover, deep networks also face issues associated to overfitting, due the large number of functions used to separate the input space; and degradation, once the gradient descent method propagates accumulative errors through layers during the training stage (KRIZHEVSKY; SUTSKEVER; HINTON, 2012; SRIVASTAVA *et al.*, 2014; HE *et al.*, 2016; PEREZ; WANG, 2017; MELLO; FERREIRA; PONTI, 2017; MELLO; PONTI; FERREIRA, 2018).

The lack of theoretical understanding about the CNN learning and the current empirical process involved in the design of CNN architectures are the main motivations for this thesis, which intends to support the design of simpler CNN architectures by estimating convolutional mask sizes as well as the number of units at every CNN layer, without requiring the whole training process for several candidate architectures (FERREIRA *et al.*, 2018). Our approach was based on Dynamical System (DS) concepts used to identify recurrent patterns on time series (time-dependent data), in an attempt to provide the best reconstruction of input data (images or matrices) what directly influences the convolutional kernels as discussed throughout this thesis.

1.1 Objective

In order to contextualize this thesis, it is relevant to mention that its first motivation was a classification problem involving polyphonic audio signals, more precisely the problem of Cover Song Identification (CSI), defined in the context of Music Information Retrieval (MIR) (ELLIS; POLINER, 2007; SERRÀ; ZANIN; ANDRZEJAK, 2009; BOGDANOV *et al.*, 2013; FERREIRA, 2014; OSMALSKY *et al.*, 2015; ONO *et al.*, 2015). At the beginning, we

intended to design some DL algorithm to improve the CSI task, specially reducing learning costs and improving the overall classification performance (FERREIRA, 2014). During our studies, we faced the same DL issues discussed in the previous section, what motivated us to adapt the original research question of this thesis to address such more general problem of designing suitable CNN architectures. In that sense, we attempted to reduce as much as possible the empirical process involved in building up architectures, given its high cost in terms of time and resources used (FERREIRA *et al.*, 2018). Such empirical procedure only confirms researchers still lack in terms of understanding how to optimize the DL design and most specifically the CNN design.

By tackling such general problem, we attempted to improve the understanding and the design of CNNs while addressing different application domains (FERREIRA *et al.*, 2018). At first, we conducted a research and some formalization steps on CNNs in order to understand how features are extracted. In this way, we algebraically formulated the CNN forward process to analyze how architecture parameters impact in building up some implicit kernel. Based on this formulation, we noticed that Dynamical System (DS) concepts could be applied to define the convolutional masks sizes and the number of units per CNN layers, since they support to find adequate input space embeddings, what in our scenario could be used to represent image recurrences in high dimensional spaces (ALLIGOOD; SAUER; YORKE, 1997; TAKENS, 1981; HAJILOO; SALARIEH; ALASTY, 2018; SIVAKUMAR; JAYAWARDENA; FERNANDO, 2002; MA; HAN, 2006). As a direct consequence of embeddings, we could define the size of convolutional kernel masks and the number of such units at each CNN layer. Given such scientific perspective, the hypothesis of this thesis is:

Hypothesis: *Dynamical System embedding tools support the design of specific CNN architectures, helping to estimate the size of convolutional kernel masks and the number of such units per layer, thus improving or providing comparable performances to empirical designing strategies, and directly avoiding the time and resource demands involved in the manual design of CNN architectures.*

In order to confirm our hypothesis, the CNN forward process was analyzed, whose main operation is the convolution (or correlation (GOODFELLOW; BENGIO; COURVILLE, 2016)) followed by subsampling stages to reduce the feature space dimensionality (LECUN *et al.*, 1998). Such analysis motivated us to algebraically formulate the convolution and subsampling operations, both introduced in Chapter 3. Such formulation has given us the insight that the design of CNN architectures is very similar to input space reconstructions provided by DS, leading us to propose an embedding strategy to reconstruct input images and, from that, define convolutional masks. In that sense, the choice of the convolutional mask sizes mathematically corresponds to a space embedding of input images, similarly to Takens' immersion theorem (TAKENS, 1981). From that, we attempted to employ the False Nearest Neighbors (FNN) (KENNEL; BROWN; ABARBANEL, 1992), tool from DS, to support space reconstructions. As FNN could not be

directly used, we studied how it could be adapted to estimate adequate convolutional mask sizes and eventually support the number of units at each convolutional CNN layer. Our claim is that by representing recurrent patterns of input images, we can indeed produce a better discriminative features space and, consequently, induce efficient classifiers. The FNN method was specifically chosen due to its renowned ability to estimate the most adequate embedding dimension for time series, i.e., the number of space dimensions necessary to unfold time recurrences (KENNEL; BROWN; ABARBANEL, 1992; PAGLIOSA; MELLO, 2017).

In this way, we designed the Image-based False Nearest Neighbors (IFNN), inspired by FNN, to address the embedding of input images (FERREIRA *et al.*, 2018). IFNN estimates convolutional mask sizes and the number of units per CNN layer, improving the specific design of CNN architectures for particular learning tasks. As discussed in Chapter 5.3, experiments confirm that architectures designed with IFNN provide similar results with the ones reported in the literature, as presented in (FERREIRA *et al.*, 2018). The architectures resulted from IFNN are typically shallower and with less units, which is a characteristic associated with parsimony and thus Occam's razor principle (VITÁNYI; LI, 1996; BURGESS, 1998). Despite the architectures being more simple, they are still efficient when compared to others from the literature, specially in terms of avoiding overfitting, because they lead to smaller Shattering coefficients (or growth functions) (SRIVASTAVA *et al.*, 2014; COGSWELL *et al.*, 2016; VAPNIK, 2000; MELLO; PONTI, 2018). In addition, we also claim that by having a method to find adequate enough parameters, the final CNN architecture provides some input space kernelization more consistent with data recurrent patterns, what should lead to a representative features space to perform classification.

This simplicity was assessed using the concepts of the Statistical Learning Theory (SLT) (SCHÖLKOPF; SMOLA, 2002; VAPNIK; CHERVONENKIS, 2015) to confirm our claim, specially in terms of the Shattering coefficient estimated for the CNN architectures, therefore providing a complexity measure for the space of admissible functions, a.k.a. the algorithm bias (as detailed in Chapter 6). The Shattering coefficient is necessary to ensure consistency to the Empirical Risk Minimization Principle (ERMP) from SLT, leading to learning guarantees for supervised algorithms (VAPNIK, 2000; MELLO; PONTI, 2018). Our results confirm that IFNN supports the design of simpler and efficient enough CNN architectures, given their fair performance in conjunction with the requirement of smaller training samples than other solutions reported in the literature (LECUN *et al.*, 1999; SZEGEDY *et al.*, 2015; SIMONYAN; ZISSERMAN, 2014; HE *et al.*, 2016; KRIZHEVSKY; SUTSKEVER; HINTON, 2012). We here recall the use of simpler architectures is consistent with the principle of parsimony and Occam's razor problem-solving principle (VITÁNYI; LI, 1996; BURGESS, 1998).

Five commonly used benchmark datasets were selected to evaluate the IFNN method in practical scenarios: CMU, MNIST, CIFAR-10, COIL-100 and GTSRB (more details in Section 5.2). Our results and a detailed comparison with the literature are provided in Section 5.4.

In summary, CMU is a face recognition dataset that was employed considering two different classification tasks, the first intends to identify people in images while the second aims to verify if a person wears sunglasses. Results confirmed that IFNN supports the design of CNN architectures regardless such classification labels, something we regard possible due to IFNN considers the content of images. In both scenarios, IFNN improved the accuracy reported in the literature. The accuracy measured for the handwritten digit dataset MNIST after estimating convolutional masks with the support of the IFNN method was similar to the ones reported in the literature, however allowing the usage of a much simpler architecture.

The CIFAR-10 and COIL-100 datasets were considered to assess IFNN in the context of object recognition tasks. The main difference between those datasets is that CIFAR-10 was designed from Web images, without any preprocessing stage, while COIL-100 was acquired from a controlled environment and images were preprocessed. In case of COIL-100, the architecture designed with IFNN improved the results reported in literature. On the other hand, the results were not satisfactory for CIFAR-10, being significantly worse than other results reported in the literature. While using GTSRB, a benchmark dataset for the traffic sign recognition task (famous after a competition organized in conjunction with the International Joint Conference on Neural Networks (IJCNN) (SOCIETY, 1989 (accessed September 27, 2018))), our CNN architecture estimated with IFNN also produced an accuracy below the literature, however, the difference was not too large, from which we conclude results were still fairly similar.

Finally, the same preprocessing employed by the personnel responsible for designing the COIL-100 dataset was applied to CIFAR-10 and GTSRB in order to analyze the impacts. Such preprocessing simply normalized color intensities. In case of GTSRB, such preprocessing stage improved results, something we expected given images are badly balanced in terms of colors. In case of CIFAR-10, the preprocessing decreased the accuracy, indicating color equalization would not help in any way. Given the worse results obtained for the CIFAR-10, we particularly performed additional experiments considering other CNN architectures built up with the support of IFNN (see Appendix A). This additional analysis allowed to understand why IFNN architectures did not produce satisfactory results for this particular dataset. In brief, we concluded our architectures were not enoughly complex to extract the necessary features to proceed with an efficient classification stage.

Moreover, we observed that the subsampling operation was not efficient enough for CIFAR-10, jeopardizing the CNN generalization, and, in addition, the CNN performances stabilized as the number of units increased, suggesting a minimal and sufficient number of units to reach good performance for a specific architectures. From those conclusions, we notice the need for further studies to build up complex enough architectures as well as to assess the sequence of layer operations, since they impact in the overall space transformation, i.e. input space kernelization, produced by CNNs and, consequently, in their relative classification performances.

All those conclusions motivate some future directions to improve even more the parametrization of CNN architectures, in which we intend to estimate the number of units per layer by working on another optimization criterion, converging to such minimal set of units per CNN layer. Furthermore, we intend to design a tool to estimate the number of layers and the sequence of operations, according to the way they affect CNN classification performances (MALLAT, 2016; PAPYAN; ROMANO; ELAD, 2017; CHEN; GUNTUBOYINA; ZHANG, 2016; HANNEKE, 2016; GARRO; VÁZQUEZ, 2015; YOUNG *et al.*, 2015; ALBELWI; MAHMOOD, 2017).

1.2 Text Organization

This thesis is organized as follows: Chapter 2 introduces the DL area, discusses the related work, the most popular CNN architectures, and the framework used to perform experiments. Next, Chapter 3 describes the CNN forward process, including typical operations on image datasets. In such chapter, we bring one of our contributions in terms of detailing the algebraic formulation of the CNN convolutional and max-pooling operations. Then, learning algorithms are briefly presented in order to provide the advances in the training process, using backpropagation strategies.

Chapter 4 describes the FNN technique, a DS tool used to estimate the embedding dimension in time-dependent data (TAKENS, 1981; KENNEL; BROWN; ABARBANEL, 1992; ALLIGOOD; SAUER; YORKE, 1997). Next, the proposed method, IFNN, is introduced to deal with image data in the context of DL. The relation between concepts of DS and our algebraic formulation are also provided. The setup and experimental results are presented and discussed in Chapter 5, including a detailed description of datasets, approaches used in literature to deal with those datasets, and the CNN architecture results after IFNN. A comparison of our results and those from literature is also given in such chapter.

Chapter 6 presents the main concept of SLT and a theoretical formulation to compute the Shattering coefficient of CNNs. This chapter also discusses about the complexity of CNN architectures from Chapter 5, confirming IFNN supports the design of simpler but efficient CNNs. Conclusions and contributions are listed in Chapter 7. Finally, Appendix A thoroughly lists experiments used to assess the number of CNN units and layers for the CIFAR-10 dataset.

CHAPTER

2

ON DEEP LEARNING

2.1 Initial Considerations

Deep Learning (DL) algorithms extract hierarchical features from input data by using multilayered structures, whose successful performances have motivated their application to address different supervised tasks in domains such as face recognition (BHATT *et al.*, 2012; LECUN; BENGIO; HINTON, 2015), handwritten digit recognition (LECUN *et al.*, 1999; SRIVASTAVA *et al.*, 2014), and object recognition (CIREGAN; MEIER; SCHMIDHUBER, 2012; KRIZHEVSKY; SUTSKEVER; HINTON, 2012). Input data is propagated throughout consecutive linear and nonlinear operations, in order to obtain hierarchical features and support the classification of complex datasets. Such deep representation extracts low-level features such as lines, corners, simple structures, and high levels features as object and shapes when dealing with images (LECUN; BENGIO; HINTON, 2015; GOODFELLOW; BENGIO; COURVILLE, 2016; SRIVASTAVA *et al.*, 2014).

In this context, many researchers drive their studies to improve DL algorithms, including the estimation of hyperparameters, the formulation of DL architectures, and learning processes (ALBELWI; MAHMOOD, 2017; HE *et al.*, 2016; SZEGEDY *et al.*, 2015; PEREZ; WANG, 2017; MELLO; FERREIRA; PONTI, 2017; MELLO; PONTI; FERREIRA, 2018; MALLAT, 2016). Convolutional Neural Networks (CNNs) are considered the state-of-art in the DL area, also comprising the focus of this thesis, due to the good results reported in literature (COATES *et al.*, 2011; SERMANET *et al.*, 2014; KRIZHEVSKY; SUTSKEVER; HINTON, 2012; HE *et al.*, 2016; ZEILER; FERGUS, 2014; CIREGAN; MEIER; SCHMIDHUBER, 2012; LECUN; BENGIO; HINTON, 2015; GOODFELLOW; BENGIO; COURVILLE, 2016). In that sense, this chapter aims to provide a brief introduction on some DL algorithms, and a description of the most popular CNNs applied on image tasks.

2.2 Related Work

Deep Learning (DL) is a branch of the Machine Learning area, which aims to model data in order to extract relevant features to perform learning and classification. Historically, DL finds its roots in the Conexionist learning paradigm, whose first main result comes from the Perceptron (ROSENBLATT, 1958), which was designed to tackle linear separable classification problems inspired in biological neurons (LIN; HUANG; CHEN, 2007). The Perceptron limitations (MINSKY; PAPERT, 1987) to tackle more complex tasks held back the area of Artificial Neural Networks (ANN) for several years.

Continuous research efforts brought ANN back, specially after the design of the Backpropagation training algorithm, developed by Werbos (1988), which allowed the adaptation of multiple hyperplanes in an attempt to classify non-linearly separable datasets. Such evolution eventually supported the design of the Multilayer Perceptron (MLP) (CYBENKO, 1989), which employs multiple hyperplanes (linear functions) to build up nonlinear classifiers and address more complex supervised learning tasks. The Backpropagation algorithm takes advantage of a quasi-convex error function (MELLO; PONTI, 2018) to employ the Gradient Descent (GD) method and update unit (or neuron) parameters in order to reduce model errors, finally building up some mapping of input examples to classes or labels in some output space (YAN *et al.*, 2006; KOSKELA *et al.*, 1996; ORHAN; HEKIM; OZER, 2011).

After that, several other algorithms were designed, experimented and applied in real-world problems, until the proposal of the Support Vector Machines (SVMs) (BOSER; GUYON; VAPNIK, 1992; CORTES; VAPNIK, 1995) which relies on the Large-Margin Bound to find the strongest learning guarantees from literature (VAPNIK, 2000; LUXBURG; SCHÖLKOPF, 2011; MELLO; PONTI, 2018) (see Section 6.2). SVM was designed to address supervised tasks by placing a single hyperplane at the best as possible location while dividing some input space.

In this context, Deep Learning has emerged as a strategy to model and classify large amounts of complex data, by extracting hierarchical features (low level and high level representations) and by using a combination of linear functions such as performed by MLP (MALLAT, 2016). DL algorithms are designed using consecutive layers composed of different types of processing units. A training stage, similar to backpropagation, is applied to update the weights of processing units, in order to better represent input patterns according to the expected labels. Recurrent Neural Networks (RNN), Deep Belief Networks (DBN), and Convolutional Neural Networks (CNN) are among the most employed DL techniques to solve practical problems, all of them considering multilayered architectures (LECUN *et al.*, 1999; LECUN; BENGIO; HINTON, 2015; CHO; RAIKO; ILIN, 2013; SRIVASTAVA; SALAKHUTDINOV; HINTON, 2013; LUKOŠEVIČIUS; JAEGER, 2009; SAK; SENIOR; BEAUFAYS, 2014; SIMONYAN; ZISSERMAN, 2014; KRIZHEVSKY, 2010; KRIZHEVSKY; SUTSKEVER; HINTON, 2012; HE *et al.*, 2016).

DBNs are probabilistic generative models in which every layer is separately trained to improve learning (HINTON; SALAKHUTDINOV, 2006; RANZATO *et al.*, 2007; MOHAMED; DAHL; HINTON, 2012; KRIZHEVSKY, 2010). Such model has provided relevant classification results in several domains, specially in automatic speech recognition, having as main drawback, noisy input patterns that jeopardize its recurrent convergence at each layer (DENG; YU, 2014). DBNs are typically seen as densely connected networks, containing several hidden layers of Restricted Boltzmann Machines (RBM) (SMOLENSKY, 1986). Every RBM is a network by itself, being trained to map inputs to the expected outputs. RBM architecture is illustrated in Figure 2, in which inputs are connected to a hidden layer using symmetrical connections composed of stochastic binary units (HRASKO; PACHECO; KROHLING, 2015; SALAKHUTDINOV; HINTON, 2009). In fact, the DBN training is conducted in two steps: i) each RBM is individually trained; ii) all layers are training together to fine tune the network.

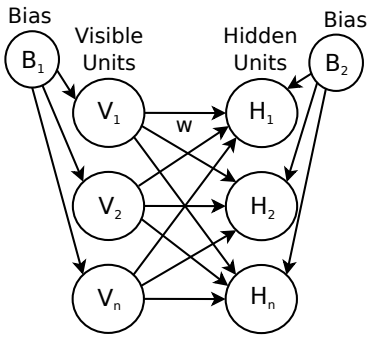


Figure 2 – Illustration of a Restricted Boltzmann Machine architecture, which is composed of visible and hidden units. Each layer has a bias term to adjust the interception point of the separation hyperplane with the input space.

RNN comprises another relevant DL approach, which contains cycled connections as main difference from common feed-forward neural networks (see Figure 3). Those connections provide recurrent behavior to such network, so that it is capable of representing dynamical systems (FUNAHASHI; NAKAMURA, 1993; BARBOUNIS *et al.*, 2006; LUKOŠEVIČIUS; JAEGER, 2009; SAK; SENIOR; BEAUFAYS, 2014; SAK *et al.*, 2015; DU; WANG; WANG, 2015; GREGOR *et al.*, 2015). As a disadvantage, both DBN and RNN contain dense connections, which are computationally intensive during the training stage (BENGIO *et al.*, 2007; LAROCHELLE *et al.*, 2009; BROSCH; TAM, 2015).

CNN has emerged as an alternative to reduce the density of connected units, and it soon became the state-of-art according to several researchers, specially when dealing with object recognition, image segmentation, handwritten digit recognition, and face detection tasks (LECUN *et al.*, 1999; LECUN; BENGIO; HINTON, 2015; SCHERER; SCHULZ; BEHNKE, 2010; SCHERER; MÜLLER; BEHNKE, 2010; OQUAB *et al.*, 2014). Researches claim that CNN is sensitive to topological information contained in the multidimensional inputs, instead of considering term-by-term distances; and it is robust to data shift and scale variations, better supporting the detection of objects (OQUAB *et al.*, 2014; LAUER; SUEN; BLOCH, 2007; LECUN;

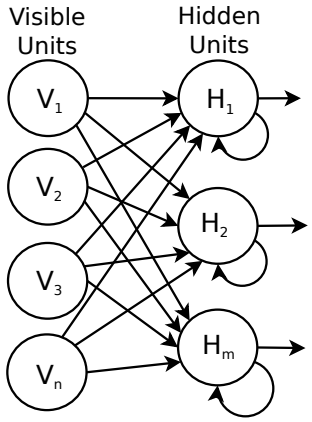


Figure 3 – Illustration of a Recurrent Neural Network architecture, in which hidden units receive their own outputs as new inputs.

BENGIO, 1995; GOODFELLOW; BENGIO; COURVILLE, 2016). In fact, the convolution and subsampling operation performed by CNN is applied onto local regions of the image, providing distortion-invariant features (LECUN; BENGIO, 1995; LECUN *et al.*, 1998; GOODFELLOW; BENGIO; COURVILLE, 2016; SCHLUTER; BOCK, 2014). Other complementary studies also claim that CNNs are robust to extract features from time-dependent data (e.g. audio signals and videos) (OSADCHY; CUN; MILLER, 2007; SCHLUTER; BOCK, 2014; KARPATHY *et al.*, 2014).

Several frameworks to deal with large datasets were designed to support the execution of DL algorithms, such as TensorFlow, Caffe, Torch, Theano, and Deeplearning4j (ABADI *et al.*, 2015; COLLOBERT; BENGIO; MARITHOZ, 2002; JIA *et al.*, 2014; AL-RFOU *et al.*, 2016; TEAM, 2017). The most renowned framework of nowadays is TensorFlow (ABADI *et al.*, 2015), which was designed by Google's Brain Team within Google's Machine Intelligence research organization [1]. Before TensorFlow, the Caffe Deep Learning Framework (JIA *et al.*, 2014) was the most widely employed in literature (SIMONYAN; ZISSERMAN, 2014; GIRSHICK *et al.*, 2014; RUSSAKOVSKY *et al.*, 2015; HE *et al.*, 2016). As matter of fact, both frameworks provide similar resources to perform CNN experiments, including GPU and CPU settings.

Kovalev, Kalinovsky and Kovalev (2016) performed a comparison analysis of TensorFlow, Caffe, Torch, Theano, and Deeplearning4j frameworks under the same scenarios. They reported the processing time and the accuracy obtained as the number of layers and units were increased. According to their conclusions, Theano is the best framework, followed by TensorFlow, Caffe, Torch and, finally, Deeplearning4j, which is the worst in terms of processing time and accuracy results. The Caffe framework was considered in this thesis to perform experiments involving CNNs, mainly because its fair processing time, its accuracy stability, and its widely usage when this research began.

[1] More information about TensorFlow framework is available at <https://www.tensorflow.org>.

2.2.1 Convolutional Neural Networks

Convolutional Neural Networks (CNN), firstly developed by LeCun and Bengio (1995), employs the Stochastic Gradient Descent (SGD) method with its Backpropagation algorithm in order to induce classification models. The first popular CNN algorithm in conjunction with a specific architecture was LeNet-5 (LECUN *et al.*, 1998), which presented promising results while tackling the handwritten digit recognition task through the MNIST dataset (LECUN *et al.*, 1998). The LeNet-5 architecture is composed of a convolutional layer with 6 units and mask size of 5×5, followed by 6 units of an average-pooling with size 2×2 without overlapping, another convolutional layer having 16 units with mask size equals to 5×5, then an average-pooling layer with 16 units of size 2×2 without overlapping. The fifth layer is also convolutional and contains 120 units with mask size equals to 5×5, while the sixth is a fully-connected layer with 84 units, in which a sigmoid function is applied. The last layer is composed of 84 units and computes a Radial Basis Function (RBF) to output the class information (LECUN *et al.*, 1999).

Another popular CNN algorithm plus architecture is AlexNet (KRIZHEVSKY; SUTSKEVER; HINTON, 2012), which outperformed several approaches while dealing with the object recognition task. Its implementation is optimized for GPUs, taking 6 days to train the ImageNet dataset (DENG *et al.*, 2009) (such dataset contains 115 million labeled-images divided in $22,000$ categories). AlexNet architecture is composed of 8 layers, in which the first 5 are convolutional and the last 3 are fully-connected ones. The ReLU function is used as activation function for convolutional layers, and a Local Response Normalization (LRN) is applied after layers 1 and 2. The five convolutional layers are composed of 96, 256, 384, 384 and 285 units in feed-forward order, respectively; the next 2 fully-connected layers contain $4,096$ units, each; and the last one has $1,000$ units. Max-pooling with 3×3 and a stride equals to 2×2 is applied after layers 1, 2 and 3 as subsampling strategy.

AlexNet employs SGD to train its architecture, as well as Data Augmentation and Dropout strategies to avoid overfitting in an attempt to provide an adequate model for object recognition tasks Krizhevsky, Sutskever and Hinton (2012). This network processing stage is performed using 2 GPUs, each of them dealing with half of the total number of units, combining the training at the last layer. According to Krizhevsky, Sutskever and Hinton (2012), large and deep CNNs may improve the results for general-purpose datasets, something that was empirically observed.

In order to explain the AlexNet results, Zeiler and Fergus (2014) proposed a visualization technique called Deconvnet. Such technique was specially developed to understand the good performance of AlexNet when learning the ImageNet dataset. This approach permits to visualize which input information is associated with the individual relevant features mapped at each layer. In addition, it presents the evolution of features extracted during training, and eventual architecture issues. As reported, initial AlexNet layers converge faster and suffer more with input image transformations than the last layers (ZEILER; FERGUS, 2014).

Deconvnet is employed to analyze patterns that are activated by the convolutional layers, mapping the features back to the original space. The original input space is obtained by attempting to reverse the network operations as much as possible, such as the convolutional filters. As discussed by Zeiler and Fergus (2014), max-pooling is not invertible, so an approximation based on a stored maximum localization is used to preserve the structure, and ReLU is applied to guarantee a valid reconstruction. The transposed matrix of the training convolutional filters is employed to compute the reconstruction and highlight the discriminatory elements (ZEILER; FERGUS, 2014). To consider the transpose as an inverse matrix for convolutional filters, the filters produced during training must be orthogonal, however the authors do not explain nor discuss how such orthogonality is ensured.

Another important study about CNNs was conducted by Simonyan and Zisserman (2014), who employed the Caffe framework to assess the impact of the deep CNNs while considering small convolutional filters. They confirmed that CNN architectures with 16 and 19 layers produced good accuracies, resulting in two popular networks: VGG16 and VGG19 (respectively). They employed filters with size 3 and max-pooling of 2×2 without overlapping, whose layers had 64, 128, 256, and 512 units. At the end, two fully-connected layers with $4,096$ units were applied, as well as an additional last fully-connect layer with $1,000$ units (that is the number of classes). All convolutional layers had ReLU as activation function. In that scenario, the authors pointed out some issues, such as problems to initialize filters and that LRN (applied by AlexNet) did not improve results, unlike Data Augmentation and deep architectures.

Following the empirical conclusion that large and deep CNNs improve accuracies, Szegedy *et al.* (2015) proposed GoogLeNet in an attempt to reduce the consumption of computational resources during the training stage by including sparsity in connections between layers. The GoogLeNet architecture employs inception modules to produce such sparse local structures, approximating dense network models. An inception module is a shallow network composed of small-sized filters (1×1, 3×3, 5×5), having a last layer responsible for combining the overall extracted features (see Figure 4). This architecture starts with common convolutional operations with ReLU and max-pooling layers, then inception modules and subsampling layers are included, totalizing 27 layers. It also faces similar problems as other deep architectures, leading the authors to apply auxiliary classifiers at the middle layers to support backpropagation, in an attempt to cope with degradation problems, and Dropout to avoid overfitting. GoogLeNet presents good results and the authors claim it requires less computational resources.

In a similar direction, the Microsoft ResNet was designed to reduce the time spent during training, being currently among the most popular networks (HE *et al.*, 2016). This network applies the residual learning principle to adapt the weights of multiple layers in an attempt to address degradation problems (KRIZHEVSKY; SUTSKEVER; HINTON, 2012; SRIVASTAVA *et al.*, 2014; HE *et al.*, 2016; PEREZ; WANG, 2017; SZEGEDY *et al.*, 2015). According to He *et al.* (2016), ResNet better deals with degradation which is a direct result of the issues faced by

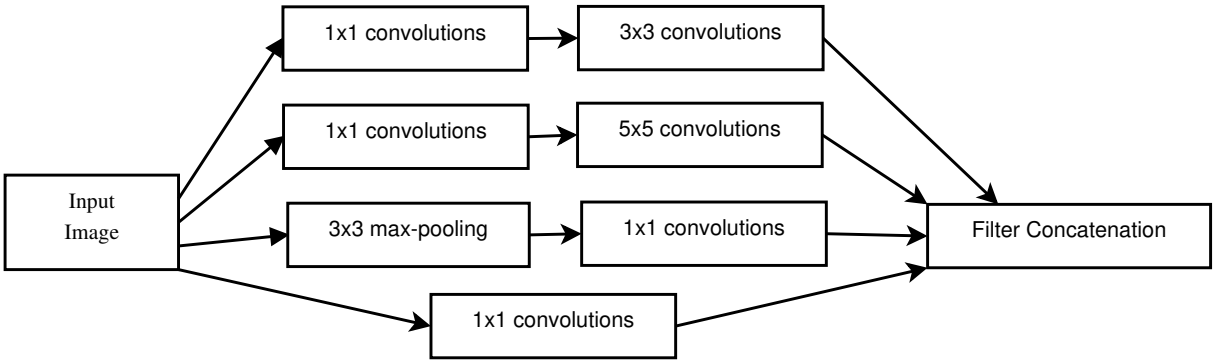


Figure 4 – Representation of an inception module applied by GoogLeNet with dimensionality reduction.

solvers while approximating identity mappings.

The ResNet architecture contains shortcut connections, as illustrated in Figure 5, producing an element-wised sum of input images at the output layer. ReLU and LRN are used at every convolutional layer, right after the shortcut application. Among its characteristics, ResNet has 34 layers in total, including subsampling (average-pooling is employed), zero padding to save image dimensions, and Data Augmentation to avoid overfitting. Authors claim that ResNet produces lower cumulative error during the training stage (also backpropagation) and converges faster.

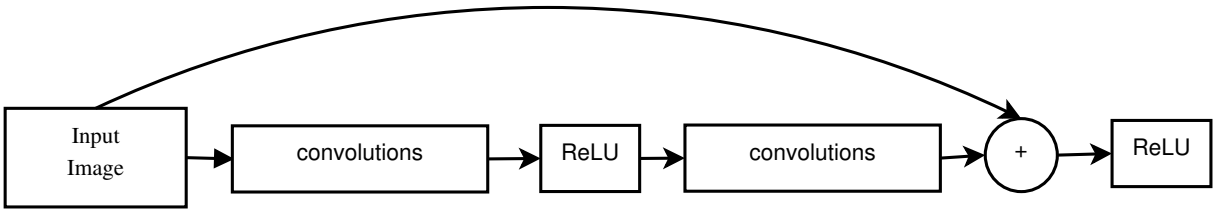


Figure 5 – Representation of a shortcut connection applied by Microsoft ResNet.

Many deep learning architectures from the literature have considered several layers and units at every layer, which are empirically justified by authors who claim that classification error tends to reduce when the network size increases (SZEGEDY *et al.*, 2015). In order to keep the high accuracies and avoid deep network problems, such as overfitting and degradation, researches developed other types of layers, such as the inception module that combines different convolutional mask sizes and operations, and shortcut connections to add up input data into the output layer (HE *et al.*, 2016; SZEGEDY *et al.*, 2015; KRIZHEVSKY; SUTSKEVER; HINTON, 2012; GOODFELLOW *et al.*, 2013). However, deep networks usually consume a lot of computer resources to be trained and provide good enough accuracies (SRIVASTAVA *et al.*, 2014; GOODFELLOW; BENGIO; COURVILLE, 2016; LECUN; BENGIO; HINTON, 2015; KRIZHEVSKY; SUTSKEVER; HINTON, 2012; COATES; NG, 2011). In addition, network architectures are manually designed and their parameters are empirically selected, requiring several experiments until finding a good configuration for a given dataset (FERREIRA *et al.*, 2018).

In this context, some frameworks were proposed in order to simplify the architecture design. Albelwi and Mahmood (2017) developed a framework to assess different architectures using deconvolution networks, which correlate inputs and their corresponding reconstructed images in order to measure information loss. At first, subsets of the training data are selected with Random Mutual Hill Climbing (RMHC) and, then deconvolution networks are used to perform training and compute accuracies. Finally, correlations are used as fitness measurements to select the most adequate architecture design, having such optimizing based on the Nelder-Mead method (NELDER; MEAD, 1965). In spite of the good accuracies, this framework consumes excessive computational resources given full training stages are required to assess each architecture and compute its fitness value.

Another approach to define CNNs parameters is based on evolutionary algorithms, which also requires the training of several different architectures. For example, a comparison among three Particle Swarm Optimization (PSO) algorithms was conducted by Garro and Vázquez (2015) in an attempt to estimate the number of units, connections among units, initial weight and bias values, as well as activation functions. Besides the architecture design improvements, their strategy takes too long to provide architecture estimators, due to the extensive training used to evaluate every candidate solution.

Young *et al.* (2015) proposed Multinode Evolutionary Neural Networks for Deep Learning (MENNDL), a framework to estimate the most adequate convolutional mask size and number of units for deep learning architectures. MENNDL is also driven by the training of every candidate architecture until finding a good parametrization, since the fitness function considers the error rate provided after classification. As consequence, this approach is similar to PSO approaches, being as well as computationally intensive. In summary, evolutionary approaches have been used to estimate good parameters and design adequate CNN architectures (ALBELWI; MAHMOOD, 2017; GARRO; VÁZQUEZ, 2015; YOUNG *et al.*, 2015), however, they demand costly training stages.

Another problem is due to the lack of a full understanding on the learning principles in deep architectures, including which features are extracted, and how the sequence of different operations affect the classification results (MALLAT, 2016; PAPYAN; ROMANO; ELAD, 2017; CHEN; GUNTUBOYINA; ZHANG, 2016; HANNEKE, 2016). Deep architectures are typically employed after the empirical conclusion that the error rate reduces when the network size increases. For example, recent complex and large configurations of networks, such as GoogLeNet (SZEGEDY *et al.*, 2015) and AlexNet (KRIZHEVSKY; SUTSKEVER; HINTON, 2012), provide great accuracies for the object recognition task, however, they take a long time to be trained and, according to the Statistical Learning Theory (VAPNIK; CHERVONENKIS, 2015; MELLO; PONTI, 2018) they are more prone to overfitting, requiring the support of additional strategies such as Dropout (SRIVASTAVA *et al.*, 2014) and DeCov (COGSWELL *et al.*, 2016).

This lack of understanding motivates further studies on the DL process in an attempt to

formalize and so explain how and which features are extracted by consecutive linear and nonlinear operations, as well as to prove DL algorithms indeed learn. In this context, Mallat (2016) provided empirical and theoretical evidences on CNN learning, showing that the hierarchical structure based on convolutional filters is similar to the application of Wavelet transforms. In addition, depending on the CNN architectures, results are susceptible to small perturbations, although the stability and robustness empirically observed for CNNs. Papyan, Romano and Elad (2017) also discuss on the theoretical formalization of CNNs and employ Convolutional Space Coding (CSC) to study the CNN process. Those authors proposed a multi-layer CSC, claiming it is similar to the CNN forward process. Such similarity allowed authors to ensure that CNNs provide unique features and representations for some target problem under analysis.

All the challenges discussed throughout this section motivate the development of a new method to estimate the convolutional mask sizes and number of units per CNN layer, as proposed in this thesis. Our method is based on Dynamical Systems concepts and tools in an attempt to represent recurrent patterns embedded into input images, so that the CNN parametrization is derived from a mathematical foundation instead of an empirical process based on long training stages. In addition, considering studies on the Shattering coefficient (MELLO; PONTI, 2018), we aim to obtain as simple and as efficient as possible CNN architectures, implying lower processing costs and reducing the probability of overfitting (COGSWELL *et al.*, 2016).

2.3 Final Considerations

This chapter briefly introduced the related work, including a historical development of the DL area. The most commonly employed DL algorithms were also summarized in terms of their basic steps used to extract features and perform classification. Convolutional Neural Networks (CNNs) currently comprise the state-of-art in DL, due to their relevant results reported in different domains (KRIZHEVSKY; SUTSKEVER; HINTON, 2012; CIREGAN; MEIER; SCHMIDHUBER, 2012; HE *et al.*, 2016), motivating additional information on how they have been applied throughout the literature.

In spite of their great performances, CNNs have been facing and discussed in terms of issues such as overfitting and degradation (KRIZHEVSKY; SUTSKEVER; HINTON, 2012; SRIVASTAVA *et al.*, 2014; HE *et al.*, 2016; PEREZ; WANG, 2017; MELLO; FERREIRA; PONTI, 2017; MELLO; PONTI; FERREIRA, 2018). Partial solutions to address those issues were also discussed in this chapter, including hyperparameter estimations and theoretical formulations of the CNN learning process. Apart of those attempts, those problems are still open so that the same challenges are still faced on the design of CNN architectures and about a theoretical understanding on how CNNs work, what is the main motivation for this PhD thesis.

CHAPTER

3

CONVOLUTIONAL NEURAL NETWORK

3.1 Initial Considerations

Artificial Neural Networks (ANN) comprise the main techniques in the Deep Learning (DL) scenario, being inspired in biological neurons and designed to extract relevant features in an attempt to address classification tasks (MINSKY; PAPERT, 1987; WERBOS, 1988; HECHT-NIELSEN, 1989). They are composed of hierarchical layers, each of those containing a set of processing units to apply different operations, resulting in a mapping from input examples to output classes. In that context, the Convolutional Neural Networks (CNNs), inspired in the human visual and hearing systems (DENG; YU, 2014), have attracted special attention due to their ability to model and classify complex problems from a wide range of application domains (LECUN *et al.*, 1999; HUANG; LECUN, 2006; SERMANET; LECUN, 2011; CIREGAN; MEIER; SCHMIDHUBER, 2012; KRIZHEVSKY; SUTSKEVER; HINTON, 2012; GOODFELLOW *et al.*, 2013; SCHLUTER; BOCK, 2014; SEVERYN; MOSCHITTI, 2015; ZHU *et al.*, 2016).

This chapter introduces CNNs, specially detailing the forward and the training processes. The functioning of most employed CNN layers is as well explained, including the classical training algorithms used to improve the mapping of inputs into outputs. Moreover, we discuss about methods employed in literature to avoid deep network problems; and about the Caffe framework which was used to perform all experiments reported in this thesis (see Section 5.3).

3.2 Architecture

Convolutional Neural Networks (CNNs) have been employed in several application domains, given their results and ability to deal with multidimensional data (HERSHEY *et al.*, 2017; KRIZHEVSKY; SUTSKEVER; HINTON, 2012; KARPATHY *et al.*, 2014). They are organized into multilayered architectures, having convolutional layers interposed by other sorts of layers with different functions, including data dimensionality reduction and normalization (LE-

CUN; BENGIO; HINTON, 2015). The first layer, a.k.a. input layer, is simply responsible for collecting the input data. Next layers, a.k.a. hidden, apply different operations on their inputs, such as convolutions and dimensionality reductions. The last layer outputs features so that a classification algorithm may be connected. It is important to mention that each layer has a set of units (a.k.a. neurons) to compute predefined operations, such as convolution, subsampling, and normalization (GOODFELLOW; BENGIO; COURVILLE, 2016).

The most common CNN architectures are composed as follows: i) input layer, ii) convolution layer with activation function, iii) subsampling layer, iv) repeat layers (ii) and (iii) according to the application task, v) a fully-connected layer to infer classification, and v) an output or classification layer. Figure 6 illustrates a typical architecture employed on the image classification domain, being composed of an input layer, three hidden convolutional layers interposed with subsampling layers, two fully-connected layers and a classification layer.

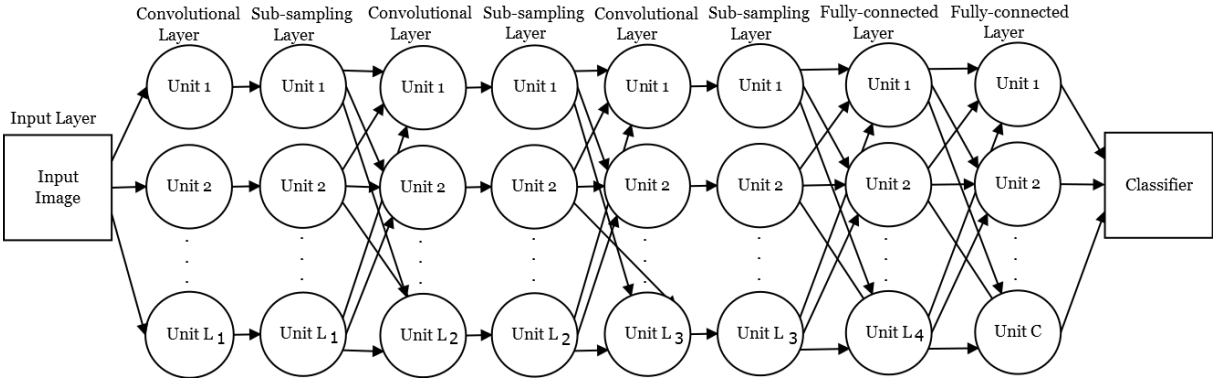


Figure 6 – Illustration of a typical architecture employed on image classification tasks. It is composed of an input layer, three hidden convolutional layers interposed with subsampling layers, two fully-connected layers, and a classification layer.

According to Hubel and Wiesel (1962), CNN processing units are inspired in the mammal's primary visual cortex, in which simple cells select features considering local receptive fields. As consequence, the convolution operation was selected due to its ability of extracting features from local image regions (ABDEL-HAMID *et al.*, 2014), supporting the identification of recurrent patterns from input data (SCHLUTER; BOCK, 2014; GOODFELLOW; BENGIO; COURVILLE, 2016). In this context, recurrences are associated with the most similar and prevalent patterns occurring in a particular input. The convolution operation also produces a representation that is robust to translation variance, i.e., the features produced are still representative after certain input displacements (LECUN; BENGIO, 1995; GOODFELLOW; BENGIO; COURVILLE, 2016).

Equation 3.1 defines the convolution operation, considering the input data as an image, since CNN is typically employed to tackle image classification tasks. In such equation, I is the input image with size $q \times p$, i and j correspond to row and column indices of I, x and y are the row and column indices of the convolutional mask, and Θ is a matrix representing the

convolutional mask with size $m \times n$.

$$s_{i,j} = (I \otimes \Theta)_{i,j} = \sum_{x=1}^{m} \sum_{y=1}^{n} I_{i-x,j-y} \Theta_{x,y} \tag{3.1}$$

It is worth to mention that most CNN implementations replace the convolution for the cross-correlation operation, which is relative to the former when the mask is symmetrical (GOODFELLOW; BENGIO; COURVILLE, 2016; FERREIRA *et al.*, 2018). Convolutional masks are randomly initialized and weights are adapted to better represent the input data, so that the cross-correlation does not affect any learning stage.

Equation 3.2 defines the cross-correlation operation, in which the convolutional mask does not suffer any rotation (GOODFELLOW; BENGIO; COURVILLE, 2016), behaving just like an element-wise multiplication. Both operations are referred to as convolution in the literature (GOODFELLOW; BENGIO; COURVILLE, 2016), so this nomenclature will be used throughout the text. In addition, a zero padding is typically employed on input image before the convolution (or cross-correlation) operation in order to maintain the input image size. Without such zero padding, image dimensions are reduced.

$$s_{i,j} = (I \otimes \Theta)_{i,j} = \sum_{x=1}^{m} \sum_{y=1}^{n} I_{i+x,j+y} \Theta_{x,y} \tag{3.2}$$

Figure 7 illustrates the process performed by one convolutional unit (or neuron), considering an input image I with size 4×4 and zero padding. This figure also shows a convolutional mask Θ with size 3×3, which produces the 4×4-output image s. Considering this example, the convolution operation used to obtain $s_{1,1}$ is defined in Equation 3.3, while the corresponding cross-correlation operation is defined in Equation 3.4. In addition, after the convolution (or cross-correlation), a bias multiplied by some weight may be added as follows $s_{1,1} + bw_0$, changing the interception point of such function with the input space of variables.

$$\begin{aligned} s_{1,1} = & I_{0,0} \times \Theta_{3,3} + I_{0,1} \times \Theta_{3,2} + I_{0,2} \times \Theta_{3,1} + \\ & I_{1,0} \times \Theta_{2,3} + I_{1,1} \times \Theta_{2,2} + I_{1,2} \times \Theta_{2,1} + \\ & I_{2,0} \times \Theta_{1,3} + I_{2,1} \times \Theta_{1,2} + I_{2,2} \times \Theta_{1,1} \end{aligned} \tag{3.3}$$

$$\begin{aligned} s_{1,1} = & I_{0,0} \times \Theta_{1,1} + I_{0,1} \times \Theta_{1,2} + I_{0,2} \times \Theta_{1,3} + \\ & I_{1,0} \times \Theta_{2,1} + I_{1,1} \times \Theta_{2,2} + I_{1,2} \times \Theta_{2,3} + \\ & I_{2,0} \times \Theta_{3,1} + I_{2,1} \times \Theta_{3,2} + I_{2,2} \times \Theta_{3,3} \end{aligned} \tag{3.4}$$

Any intermediary convolutional layer receives all images produced by the previous layer in order to produce a single output image. In that sense, the convolutional mask of such layer is composed of 3 dimensions, having the third as the number of units of the predecessor layer.

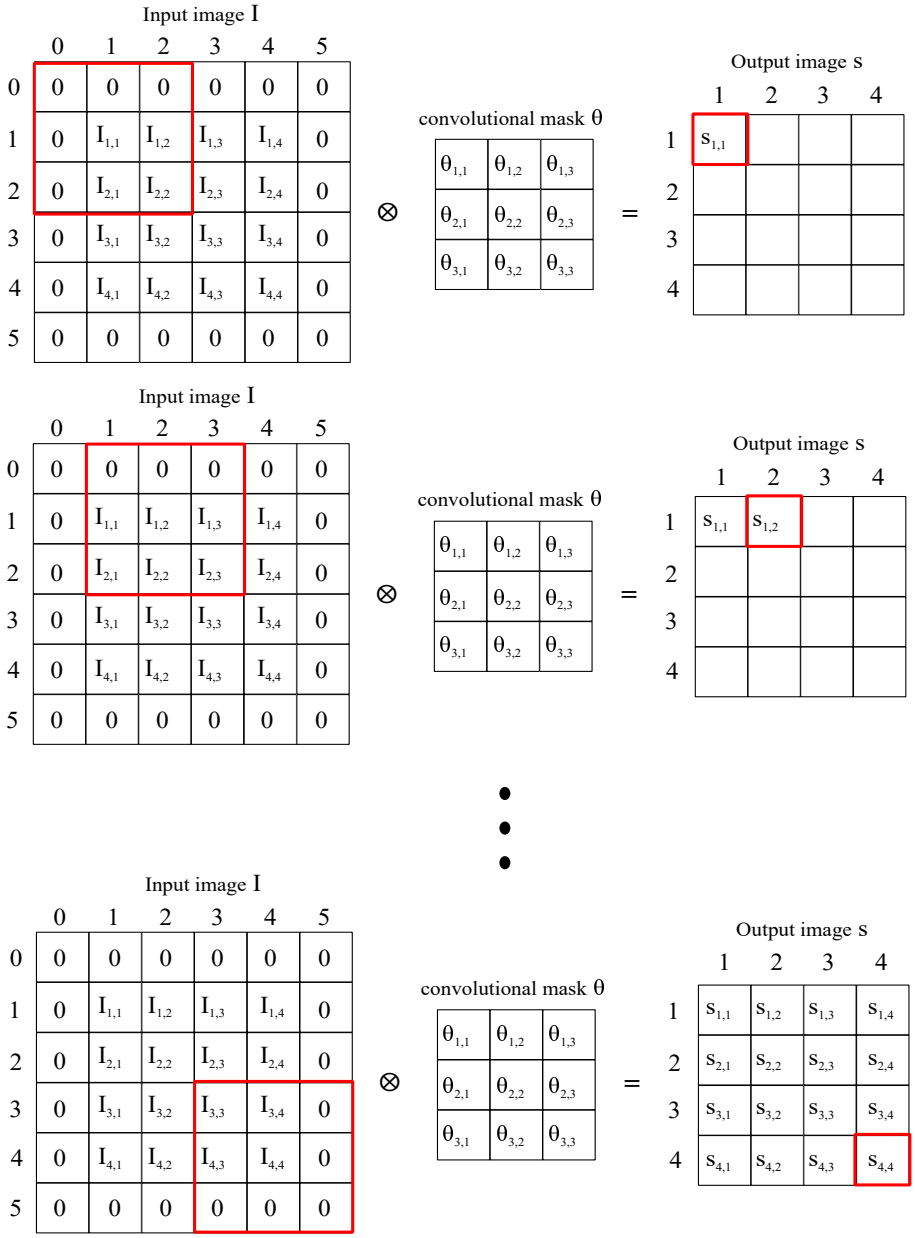


Figure 7 – Illustration of the convolution operation computed by one unit, considering input image I with size 4×4 and zero padding, given the convolutional mask Θ with size 3×3 to produce the 4×4-output image s. The result is an element-wise multiplication of a local region of I with the convolutional mask Θ.

Figure 8 illustrates this process, considering 3 input images with size 4×4 and zero padding as well as a convolutional mask Θ with size $3 \times 3 \times 3$ to generate a 4×4-output image s. The result is an element-wise multiplication of all local receptive fields (of every image) with the convolutional mask Θ. Observe the 2D convolution is still employed, i.e., each mask is applied on an image produced by a correspondent unit at the previous layer. Finally, the images obtained after every convolution operation are summed up to provide a single output image.

An activation function is usually applied right after the convolution operation in order

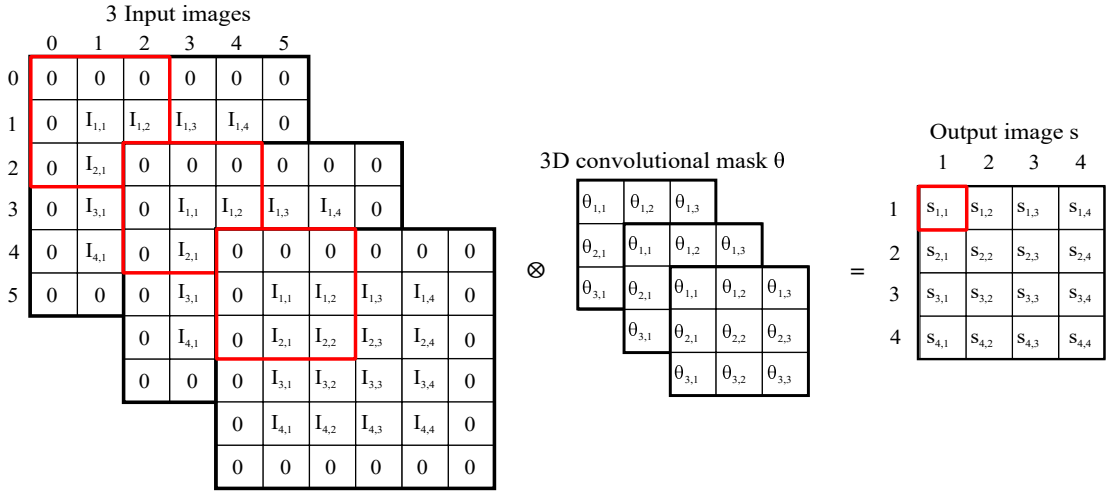


Figure 8 – Illustration of the 3D convolution operation computed by one unit, considering 3 input images with size 4×4 and zero padding as well as a convolutional mask Θ with size $3 \times 3 \times 3$ to produce the 4×4-output image s. The result is an element-wise multiplication of all local receptive fields with the 3D convolutional mask Θ.

to normalize values in a given range. This process is useful in supervised scenarios to define the loss function and to highlight relevant features in the output of the convolution operation. Many activation functions are employed by CNNs, having the sigmoid (Equation 3.6) and the hyperbolic tangent (Equation 3.5) as the most common ones (KRIZHEVSKY; SUTSKEVER; HINTON, 2012; LI *et al.*, 2014; LECUN; BENGIO; HINTON, 2015; NG *et al.*, 2015; LU; JIANG; LIU, 2017). Recently, studies claim the Rectified Linear Unit (ReLU) function improves results under certain scenarios (Equation 3.7) (CIREGAN; MEIER; SCHMIDHUBER, 2012; KRIZHEVSKY; SUTSKEVER; HINTON, 2012; LECUN; BENGIO; HINTON, 2015; IDE; KURITA, 2017) by avoiding negative values and by allowing outputs to be greater than one, differently from the other two functions that bound outputs. In that sense, ReLU better resembles the original magnitudes of input data (SCHLUTER; BOCK, 2014).

$$y(x) = \frac{e^{2x} - 1}{e^{2x} + 1} \tag{3.5}$$

$$y(x) = \frac{1}{1 + e^{-x}} \tag{3.6}$$

$$y(x) = \max(0, x) \tag{3.7}$$

Right after the convolutional layer, a subsampling layer is typically applied in an attempt to maintain the most relevant information as well as to reduce data dimensions, resulting in features that are enoughly robust to local distortions (LECUN *et al.*, 1998). Two main

nonlinear subsampling operations are applied in literature to deal with image tasks, being max-pooling (RANZATO *et al.*, 2007; SEVERYN; MOSCHITTI, 2015; LECUN; BENGIO; HINTON, 2015) the most common, which returns the maximum value of every local receptive field for some processed image. The other subsampling operation is referred to as average-pooling which returns the average of every local receptive field as well (LECUN *et al.*, 1998; POULTNEY; CHOPRA; CUN, 2006; HUANG; LECUN, 2006; JARRETT *et al.*, 2009). Max-pooling is the most used in conjunction with CNN architectures due to it better maintains information magnitudes and, in addition, some authors claim it is more robust to data invariant translations (LECUN; BENGIO; HINTON, 2015). Figure 9 illustrates a subsampling process with size 2×2 and stride 2×2 computed by a single unit on a 4×4-input image I.

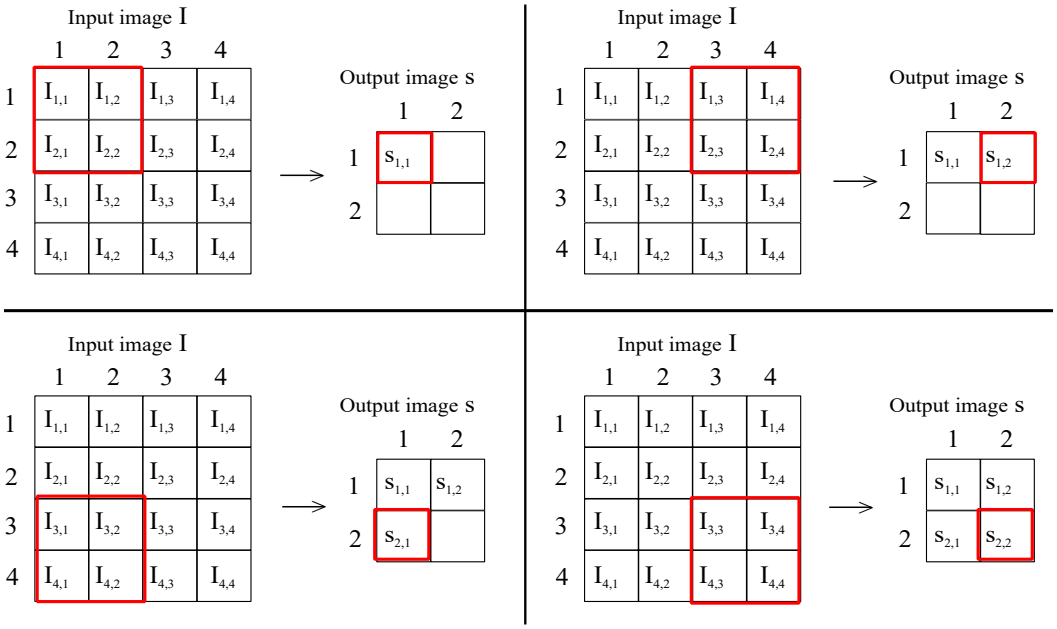


Figure 9 – Illustration of subsampling operation computed by one unit with a mask of size 2×2 and considering a stride of 2×2 applied on a 4×4-input image I.

Local Response Normalization (LRN) is another layer used in conjunction with CNNs to support data generalization through the lateral inhibition of units, something inspired in biological neurons (KRIZHEVSKY; SUTSKEVER; HINTON, 2012). Equation 3.8 defines the normalization applied at such layer, in which l is the number of units of the previous layer; l_s is the number of units that will be normalized (smaller or equal to l); $a^k_{i,j}$ corresponds to the pixel position i, j of some output image provided by unit k from the previous layer; κ, α and β are constants defined to avoid division by zero. Therefore, each unit output is normalized according to the number of neighbors defined by l_s. Typically, hyperparameters are defined as $\kappa = 2$, $n = 5$, $\alpha = 10^{-4}$, $\beta = 0.75$ (KRIZHEVSKY; SUTSKEVER; HINTON, 2012).

$$b^k_{i,j} = \frac{a^k_{i,j}}{(\kappa + \alpha \sum_{p=\max(0,\frac{k-l_s}{2})}^{\min(l-1,\frac{k+l_s}{2})} [a^p_{i,j}]^2)^\beta} \tag{3.8}$$

The last CNN layer is the fully-connected, whose units compute an inner-product as similar as the hidden layer of the Multilayer Perceptron (MLP) algorithm. Equation 3.9 defines the inner-product, in which I corresponds to the input data (image with size $q \times p$), w represents weights, and b is the bias or the term to change the interception of such function with the space of input variables. Pixels of image I and bias b are element-wise multiplied by weights and, afterwards, all values are summed up (LECUN *et al.*, 1998). It is worth to mention that an activation function may be applied on the outputs of this layer.

$$y(x) = \sum_{i=1}^{i=q} \sum_{j=1}^{j=p} (I_{i,j} * w_{i,j}) + b \times w_b \tag{3.9}$$

In this context, the last fully-connected layer outputs the probabilities of every input image to be associated with the corresponding classes. As consequence, the number of units composing such layer is the same as the number of dataset classes. This output is used to compute the loss function and perform training, as well as to classify unknown examples. The most used loss function is the softmax with cross-entropy, which assesses the correlation between probabilities of the correct class and the obtained one. Equation 3.10 defines the softmax function, having C as the number of classes, and x as the output vector with length C (produced by the fully-connected layer). For a classification task, the loss function is defined in Equation 3.11, in which p_j is the probability the input image belongs to class j, and y_j is equal to one when j corresponds to the correct class, or zero otherwise.

$$p_j = \frac{e_j^x}{\sum_{j=1}^{C} e_j^x} \tag{3.10}$$

$$\ell = -\sum_{j=1}^{C} y_j log p_j \tag{3.11}$$

The Euclidean distance and the Radial Basis Function (RBF) are also applied as loss functions after the fully-connected layer. Equation 3.12 defines the loss function calculated using the Euclidean distance (TAIGMAN *et al.*, 2014), in which p_j is the probability produced by the CNN, and y_j is the correct class probability given the input data, as described in softmax. The RBF-loss function is defined in Equation 3.13 (LECUN *et al.*, 1998), having the Gaussian radial basis function as the mostly used, in which $\|\cdot\|$ represents the L2-norm.

$$\ell = \sum_{j=1}^{C} (y_j - p_j)^2 \tag{3.12}$$

$$\ell = \sum_{j=1}^{C} e^{(-\varepsilon \|p_j - y_j\|)^2} \tag{3.13}$$

Finally, some studies also applied Support Vector Machines (SVM) as the last layer of CNNs in an attempt to improve classification (NIU; SUEN, 2012). In that case, the CNN is trained as usual, in which the last layer is fully-connected and responsible for computing the loss-function. After such stage, the last layer is removed and the resulting features are provided to a SVM.

In summary, the CNN architecture can be build with several layers of different operations, in some order, to provide the best performance while tackling a given classification task. However, this process has been manually performed and without any theoretical justification. Such drawback motivated the investigation of mathematical frameworks to formalize the CNN learning process, such as performed in this thesis, whose first contribution is the algebraic formulation presented in the next section.

3.2.1 Algebraic Formulation

in an attempt to understand the forward process of the Convolutional Neural Networks (CNNs), it is essential to mathematically model operations at each layer. In that sense, we started by formulating the convolution operation (Equation 3.1), which employs a mask on each local receptive field of image $I_{i-x,j-y}$, for $x = 1,\ldots,m$ and $y = 1,\ldots,n$. This operation corresponds to a linear transformation of each column vector $\vec{v} = (I_{i-x_0,j-y_0},\ldots,I_{i,j},\ldots,I_{i+x_m,j+y_m})$ using a convolutional mask as a column vector $\vec{\Theta}_k = (\theta_{x_0,y_0},\ldots,\theta_{x_m,y_m})$, as defined in Equation 3.14.

$$s_{i,j} = \vec{\Theta}_k^T \vec{v} \tag{3.14}$$

This linear transformation allows us to conclude that each unit of a convolutional layer linearly divides the space into two regions. Thus, a unit, defined as $\vec{\Theta}_k$ in Equation 3.14, corresponds to a hyperplane that linearly shatters the input data space composed of vectors obtained from local receptive fields (SCHÖLKOPF; SMOLA, 2002). As a convolutional layer has a set of processing units with different convolutional masks (in terms of their values), it produces many linear divisions on the input space. The number of hyperplanes is therefore equal to the number of units at such a layer.

The combination of half-spaces produced by several hyperplanes corresponds to non-linear classification functions, as in case of the Multilayer Perceptron (MLP) (HAYKIN, 1994). Therefore, all units composing a convolutional layer are algebraically defined as in Equation 3.15, in which the number of units is given by l, and each value $s_{i,j}^k$ corresponds to the feature extracted by applying the convolutional mask $\vec{\Theta}_k^T$.

$$\begin{bmatrix} \vec{\Theta}_1^T \\ \vdots \\ \vec{\Theta}_l^T \end{bmatrix} \vec{v}_{i,j} = \begin{bmatrix} s_{i,j}^1 \\ \vdots \\ s_{i,j}^l \end{bmatrix} \tag{3.15}$$

In this context, an input image I is reconstructed into a space containing all possible vectors $\vec{v}$ associated with the convolutional mask displacement, i.e., vectors correspond to local receptive fields. For example, the application of convolutional masks on the image pixels $I_{i,j}$ and $I_{i+1,j+1}$ correspond to such operation on vectors $\vec{v}_{i,j}$ and $\vec{v}_{i+1,j+1}$, respectively (each one related to its local receptive field). In this way, a $q \times p$-image I is transformed into a set of vectors $\mathscr{I} = \{\vec{v}_{1,1}, \ldots, \vec{v}_{q,p}\}$ in space $\mathscr{R}^{m \times n}$, whose dimensions are defined by the dimensions of convolutional masks. As consequence, observe the selected convolutional mask defines an embedding for images in some multidimensional feature space, as performed in the area of Dynamical Systems while modeling time series (TAKENS, 1981; KANTZ; SCHREIBER, 2004).

A single linear transformation of one image is defined in Equation 3.16, having $\vec{v}$ as the column vector of I, vector $\vec{s}^{\,k}$ corresponding to the output image also in a vector representation (considering unit k), and $\vec{\Theta}_k$ is the convolutional mask for unit k. Figure 10 illustrates the embedding applied on a 4×4-image I, considering a convolutional mask with size 3×3. In such figure, each local receptive field becomes a column vector, and the convolutional mask size is a transposed column vector as set in Equation 3.16.

$$\vec{\Theta}_k^T \left[\vec{v}_{0,0} \cdots \vec{v}_{q,p} \right] = \left[s_{0,0}^k \cdots s_{q,p}^k \right] = \vec{s}^{\,k} \tag{3.16}$$

In summary, after embedding, a set of convolutional masks is applied on every image vector $\vec{v}_{i,j}$ as a linear transformation $T(\vec{v}_{i,j}) : \mathscr{R}^{m \times n} \rightarrow \mathscr{R}^l$, leading vectors to another space whose dimensions are defined by the number of convolutional masks, i.e., the number of units at a convolutional layer. As main consequence, the convolutional mask size and the number of units directly influence in the feature space produced by CNN.

It is worth to mention that, an activation function is applied on every $s_{i,j}^k$ (Equation 3.15) as described in Section 3.2, in which the most common is ReLU. This process is responsible for selecting relevant features after the convolutional layer, which divides the input space in several half-spaces according to the hyperplane built by each unit. The activation function ReLU only maintains positive values after this transformation, thus selecting only part of the space to proceed with the next step.

In sequence, the results of a convolutional layer are provided to a next subsampling layer. As described in Section 3.2, the two most common subsampling operations are max-pooling and average-pooling (LECUN; BENGIO; HINTON, 2015; GOODFELLOW; BENGIO; COURVILLE, 2016). Algebraically, max-pooling corresponds to an infinite norm L_∞ applied on a local receptive field, whereas average-pooling is an application of the mean operation, which is not related to any norm. This layer also has a set of units, and each unit is responsible for subsampling the outputs of a previous convolutional layer. So, the number of units in a subsampling layer is equal to the number of units of the previous convolutional layer.

Consider an $m' \times n'$ max-pooling is applied on local receptive fields of image I', rep-

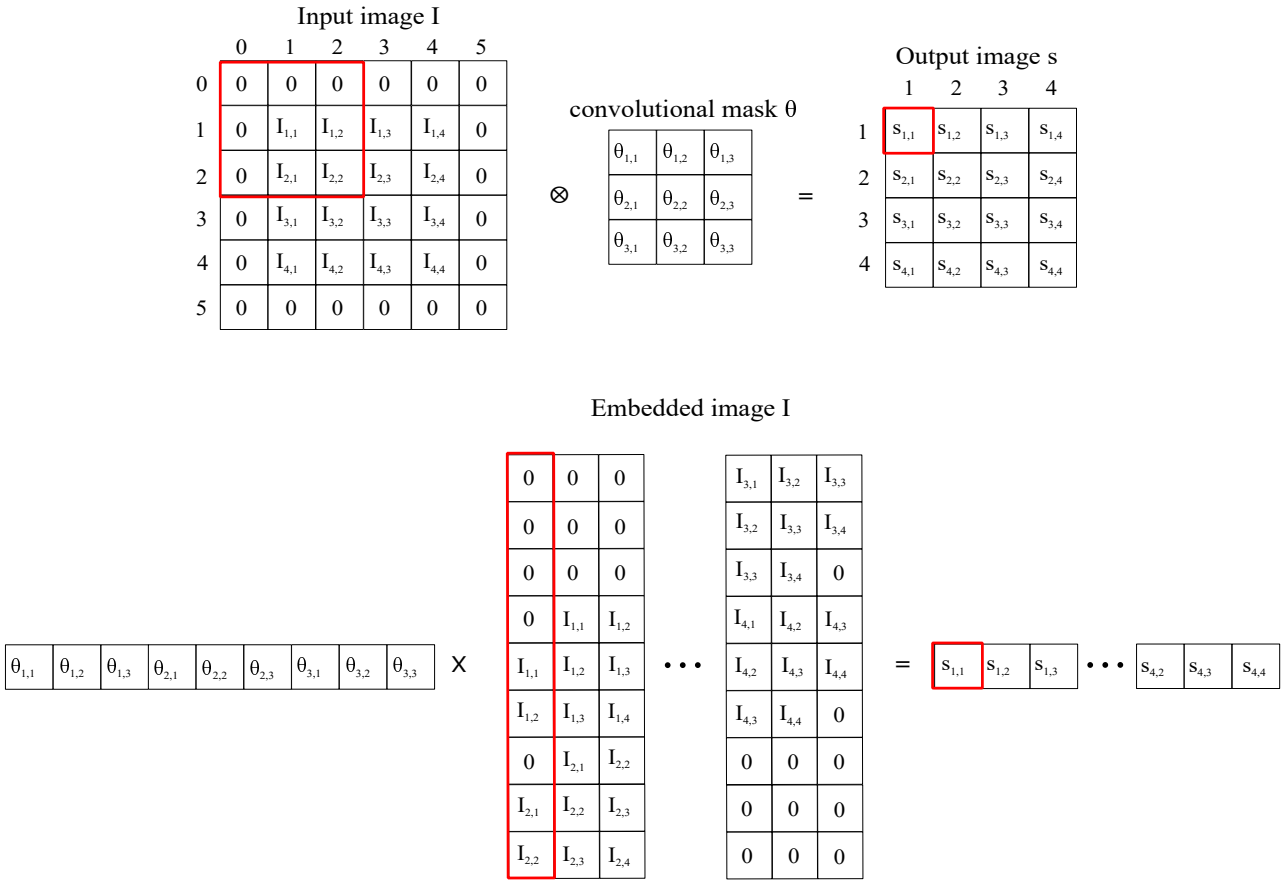


Figure 10 – Illustration of the embedding applied on a 4×4-image I, considering a single convolutional mask with size 3×3. Each local receptive field is represented as a column vector. The convolutional mask size corresponds to a transposed column vector to perform the linear transformation, resulting in a row vector containing the output image.

resented by vectors $\vec{s}^k$, so that image I' should be transformed into a set of vectors $\mathscr{I}' = \{\vec{i}'_{1,1}, \ldots, \vec{i}'_{q',p'}\}$ in space $\mathscr{R}^{m' \times n'}$, since the max-pooling size defines an embedding dimension for it. In this case, the infinite norm is applied on every local receptive field of image I'. It is worth to mention that m' and n' are different from m and n values, also q' and p' differ from q and p when subsampling employs a stride greater than 1 (stride equals to one is referred as no stride in the literature (GOODFELLOW; BENGIO; COURVILLE, 2016; KRIZHEVSKY; SUTSKEVER; HINTON, 2012)). Finally, it is possible to define the max-pooling operation on every local receptive field as $|\vec{i}'_{i,j}|_\infty$.

The fully-connected layer is the last in a CNN architecture, whose units also compute an inner-product among the weights and the values of the local receptive field received from the previous layer. Equation 3.17 defines a processing unit, in which the column vector $\vec{I}' = (I'_{1,1}, \cdots, I'_{q',p'}, b)$ represents a $q' \times p'$-input image I' and the bias; $\vec{w} = (w_{1,1}, \cdots, w_{q',p'}, w_b)$ is a column vector containing weights; and o corresponds to the scalar output value commonly used

to compute the loss function and classify unknown examples.

$$o = \vec{I}'^{T} \vec{w} \tag{3.17}$$

3.3 Training Algorithms

The Statistical Learning Theory (SLT) is responsible for formalizing the supervised learning (LUXBURG; SCHÖLKOPF, 2011; SMOLA; SCHÖLKOPF, 1998), and proving the convergence of classification and regression algorithms (see more details in Section 6.2). In this context, supervised learning algorithms were developed to infer some function that provides good results for a given classification task, what is obtained by adapting the unit weights of a given ANN. Normally, the gradient descent method is used to train ANN architectures as defined in Equation 3.18, in which x_i corresponds to a training example labeled as y_i, $f(x_i, w_t)$ represents the CNN output, η is the learning rate, N is the number of examples, $l(\cdot)$ represents the loss function, w_t is the current weight and w_{t+1} is the same weight after a next adaptation.

$$w_{t+1} = w_t - \eta \frac{1}{N} \sum_{i=1}^{N} \nabla_{w_t} l(f(x_i, w_t), y_i) \tag{3.18}$$

The most widely employed CNN training algorithm is the Stochastic Gradient Descent (SGD), which adapts the convolutional masks and fully-connected weights. SGD uses a deterministic gradient to conduct weights adaptation according to a set of labeled examples, similarly to the traditional gradient descent algorithm (BOTTOU, 2012). The gradient can be estimated considering all training examples at every iteration, or considering a single training example at each iteration as performed using the SGD method defined in Equation 3.19 (BOTTOU, 2012). It is worth to mention that the training examples must be statistically independent to satisfy the SLT assumptions (VAPNIK; CHERVONENKIS, 2015; MELLO; PONTI, 2018).

$$w_{t+1} = w_t - \eta \nabla_{w_t} l(f(x_i, w_t), y_i) \tag{3.19}$$

Since the use of a single example per iteration consumes a lot of processing time, SGD is usually estimated considering a subset of training examples what is referred to as mini-batch. Equation 3.20 defines mini-batch, in which x_i corresponds to an example of the training set with label y_i, $f(x_i, w_t)$ represents the CNN output, η is the learning rate, n represents the number of examples belonging to a training subset (a.k.a. batch size), w_t is the current weight, and w_{t+1} is the weight adapted after an iteration.

$$w_{t+1} = w_t - \eta \frac{1}{n} \sum_{i=1}^{n} \nabla_{w_t} l(f(x_i, w_t), y_i) \tag{3.20}$$

Mini-batch offers some challenges in choosing a suitable learning rate. A small rate may lead to a slow convergence, while a large one may jeopardize the convergence, causing fluctuations around the minima of the loss function. It is possible to adjust the learning rate by reducing it along training iterations, using strategies such as the annealing (ROBBINS; MONRO, 1951). However, such reduction is typically predefined and it is not adapted according to the dataset features (although it could be (ISAACSON; KELLER, 2012)). In addition, the learning rate is the same for all examples in a training batch, but batches may require different learning rates (RUDER, 2016).

In this context, a momentum term is commonly added to support the SGD method in an attempt to avoid oscillations across the loss function slopes and improve convergence, what tends to occur on noisy scenarios (ROBBINS; MONRO, 1951). Equation 3.21 defines the SGD with a momentum term γ, in which v represents the gradient result, $l(f(x_i, w_t), y_i)$ is the loss function, and w corresponds to weights.

$$\begin{aligned} v_t &= \gamma v_{t-1} - \eta \nabla_{w_t} l(f(x_i, w_t), y_i) \\ w_{t+1} &= w_t + v_t \end{aligned} \tag{3.21}$$

In addition, a simple regularization term, known as weight decay, is applied to improve the generalization and also avoid overfitting (KROGH; HERTZ, 1992; NOWLAN; HINTON, 1992). This term is summed up with the loss function in order to keep the values around zero. This is required because the derivative of the error function is quasi-convex around zero. Far from such a region, it forms a flat surface in which derivatives tend to zero so that no adaptation is performed on weights, as consequence learning does not occur in such flat regions.

The weight decay λ is usually added to the loss function as defined in Equation 3.22, in which $E(w_t)$ corresponds to the total error, and $||w_t||^2$ is the L_2-norm of weights.

$$E(w_t) = \frac{\lambda}{2}||w_t||^2 + \sum_{i=1}^{m} l(f(x_i, w_t), y_i) \tag{3.22}$$

Finally, it is worth to mention that ReLU may produce too large values, strongly influencing in unit weights that may need to become too small in order to calculate the derivatives, leading to numerical errors. Thus, the regularization term is typically applied to provide fair derivative computations.

3.3.1 Learning Algorithms

Several learning algorithms were designed to train CNN architectures in an attempt to provide good classification results. In this context, the Nesterov Accelerated Gradient (NAG) backpropagation algorithm (NESTEROV, 1983) was designed to employ a momentum term on the gradient method to accelerate the training convergence process. Equation 3.23 defines the

NAG updating term, in which γ is the momentum factor, v is the gradient result, $l(f(x_i, w_t - \gamma v_{t-1}), y_i)$ is the loss function, $f(x_i, w_t - \gamma v_{t-1})$ represents the weight being applied to x_i, and w corresponds to weights. According to Ruder (2016), this learning algorithm provide a good performance in Recurrent Neural Networks (RNN).

$$\begin{aligned} v_t &= \gamma v_{t-1} - \eta \nabla_{w_t} l(f(x_i, w_t - \gamma v_{t-1}), y_i) \\ w_{t+1} &= w_t + v_t \end{aligned} \tag{3.23}$$

Adagrad (DUCHI; HAZAN; SINGER, 2011) emerges in this context as a complementary approach to adapt parameters such as the learning rate and the momentum, instead of fixing and predefining them. Equation 3.24 defines the weight updating strategy of Adagrad, in which $w_{t,k}$ corresponds to a kth weight at iteration t, and ε is a constant to avoid division by zero. Considering that the architecture contains a total of K weights, $G_{t,kk}$ is the kth $\times$ kth element of a diagonal matrix with size $K \times K$, in which each diagonal element is the sum of squares of all past gradients towards an individual weight. Therefore, $G_{t,kk}$ is the sum of squares of all gradients of past iterations given weight k. In summary, the learning rate is adapted according to previous gradients.

$$w_{t+1,k} = w_{t,k} - \frac{\eta}{\sqrt{G_{t,kk} + \varepsilon}} \nabla_{w_{t,k}} l(f(x_i, w_{t,k}), y_i) \tag{3.24}$$

Equation 3.24 can be rewritten to represent all weights as defined in Equation 3.25, in which an element-wise multiplication $\odot$ is computed considering the gradient of the loss function $\nabla_{w_t} l(f(x_i, w_t), y_i)$, and the adapted learning rate $\frac{\eta}{\sqrt{G_t + \varepsilon}}$.

$$w_{t+1} = w_t - \frac{\eta}{\sqrt{G_t + \varepsilon}} \odot \nabla_{w_t} l(f(x_i, w_t), y_i) \tag{3.25}$$

One disadvantage of Adagrad is that the diagonal matrix G_t accumulates sums of positive terms, increasing their values along training. This process produces small than necessary learning rates, jeopardizing the convergence process. RMSprop (TIELEMAN; HINTON, 2012) emerged as a possible solution to such problem, by regularly decreasing learning rates. In fact, it works by storing the average of all past squared sums of gradients defined in Equation 3.26, having $\bar{g}_t^2$ as the average of past gradients at iteration t, λ is a weight applied on the average of past gradients, and g_t^2 corresponds to the current gradient. Therefore, Equation 3.27 defines the update rule computed by RMSprop, substituting term G_t of Equation 3.25 by the average of past gradients $\bar{g}_t^2$.

$$\bar{g}_t^2 = \lambda \bar{g}_{t-1}^2 + (1 - \lambda) g_t^2 \tag{3.26}$$

$$w_{t+1} = w_t - \frac{\eta}{\sqrt{\bar{g}_t^2 + \varepsilon}} \nabla_{w_t} l(f(x_i, w_t), y_i) \tag{3.27}$$

Observe that the denominator of the learning rate in Equation 3.27 can be represented using a Root Mean Squared (RMS) term, as defined in Equation 3.28, so that the final RMSprop updating rule is formalized in Equation 3.29.

$$\text{RMS}(x) = \sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2} \tag{3.28}$$

$$w_{t+1} = w_t - \frac{\eta}{\text{RMS}(g)_t}\nabla_{w_t} l(f(x_i, w_t), y_i) \tag{3.29}$$

In an attempt to tackle the Adagrad issues, Adadelta (ZEILER, 2012) was designed almost simultaneously to RMSprop. Both algorithms are similar, however Adadelta has another exponentially decaying average of weights, as presented in Equation 3.31, in which $\bar{v}_t^2$ is set in Equation 3.30. Finally, Equation 3.33 defines the updating rule of Adadelta where $RMS(v)_t$ is specified in Equation 3.32.

$$v_t = -\frac{\eta}{\text{RMS}(g)_t}\nabla_{w_t} l(f(x_i, w_t), y_i) \tag{3.30}$$

$$\bar{v}_t^2 = \lambda \bar{v}_{t-1}^2 + (1-\lambda) v_t^2 \tag{3.31}$$

$$\text{RMS}(v)_t = \sqrt{\bar{v}_t^2 + \varepsilon} \tag{3.32}$$

$$w_{t+1} = w_t - \frac{\text{RMS}(v)_{t-1}}{\text{RMS}(g)_t}\nabla_{w_t} l(f(x_i, w_t), y_i) \tag{3.33}$$

RMSprop and Adadelta were simultaneously developed to enhance the Adagrad algorithm which shows a monotonically decreasing behavior for the learning rate. Both algorithms employ the average of the past gradients to improve convergence while performing an exponential decreasing of learning rates (RUDER, 2016).

The Adaptive Moment Estimation (Adam) (KINGMA; BA, 2014) learning algorithm, designed based on Adadelta, attempts to adapt learning rates considering the decaying average of past squared gradients in conjunction with a momentum term. Adam adds up a momentum term to the updating rule in order to keep an exponentially decaying of the average of past gradients, as defined in Equation 3.34 in which λ_1 and λ_2 are the decay rates. In sequence, Equation 3.35 is added to avoid the convergence of vectors m and v towards zero, leading to a null derivative. Finally, Adam updating rule is defined in Equation 3.36.

$$\begin{aligned} m_t &= \lambda_1 m_{t-1} + (1-\lambda_1)\nabla_{w_t} l(f(x_i, w_t), y_i) \\ v_t &= \lambda_2 v_{t-1} + (1-\lambda_2)(\nabla_{w_t} l(f(x_i, w_t), y_i))^2 \end{aligned} \tag{3.34}$$

$$\hat{m}_t = \frac{m_t}{1-\lambda_1^t}$$
$$\hat{v}_t = \frac{v_t}{1-\lambda_2^t} \tag{3.35}$$

$$w_{t+1} = w_t - \frac{\eta}{\sqrt{\hat{v}_t}+\varepsilon}\hat{m}_t \tag{3.36}$$

In summary, Adam employs a combination of the strategies applied by the previously described algorithms, using the momentum term with an adaptation of the learning rate to faster converge learning. Other algorithms were also designed to improve Adam, such as Nesterov-accelerated Adaptive Moment Estimation (Nadam) (DOZAT, 2016) and AdaMax (KINGMA; BA, 2014). Furthermore, a bias correction term was included in Adam to avoid zero to be used as decay rate (RUDER, 2016). All those mechanisms attempt to approximate a rough loss function, improving the SGD convergence and letting the ANN training faster, however some parameters are still fixed or difficult to initialize.

3.3.2 Reducing Overfitting

One of the main problems faced by deep learning is the probability of overfitting training samples along iterations, given a network performs well on the training set but it does not properly work on unseen examples (see Section 6.2 for more information). This problem arises when training is performed using an architecture with an excessive number of layers and units operating on some small training set (MELLO; PONTI, 2018). According to the concepts of the SLT, the training set should have an adequate size in order to provide learning guarantees for a given architecture. Hence, as more complex the network is, more samples are required to ensure convergence (see Section 6.3.1). Based on those aspects, two strategies have been applied to deep networks in an attempt to avoid overfitting: i) the increase of training sets, and ii) the reduction of the network complexity.

In this context, the Data Augmentation method artificially enlarges the training set by making transformations on input examples while preserving the associated labels. In that scenario, translations and horizontal reflections are among the most common transformations applied on small image regions in an attempt to produce new useful instances (KRIZHEVSKY; SUTSKEVER; HINTON, 2012; PEREZ; WANG, 2017). Another type of Data Augmentation consists in modifying color intensities of images with the support of the Principal Component Analysis (PCA) (PEARSON, 1901; GROTH *et al.*, 2013). PCA is applied on RGB pixel values to extract relevant components that are added to images as defined in Equation 3.37, in which $I_{i,j}$ represents a pixel containing RGB channels, α corresponds to a random value, and, finally v and

λ are the eigenvectors and their corresponding eigenvalues.

$$I_{i,j} = [I^R_{i,j}, I^G_{i,j}, I^B_{i,j}] + [v_1, v_2, v_3] \begin{bmatrix} \alpha_1\lambda_1 \\ \alpha_2\lambda_2 \\ \alpha_3\lambda_3 \end{bmatrix} \tag{3.37}$$

Perez and Wang (2017) performed experiments to evaluate and explore different Data Augmentation techniques, confirming their usefulness to enhance classification results when deep networks already deal with overfitting and regularization problems. This was somehow expected once stronger learning guarantees should be obtained in the presence of augmentation, as also formalized by the SLT when dealing with greater samples (VAPNIK; CHERVONENKIS, 2015; MELLO; PONTI, 2018). Data Augmentation techniques were developed to support datasets without enough data, however, they have also been used to avoid overfitting of deep architectures that already deal with large datasets. Such aspect motivates this thesis that attempts to design shallow and efficient architectures for specific learning tasks (MELLO; FERREIRA; PONTI, 2017; MELLO; PONTI; FERREIRA, 2018).

Complementarily, Dropout (SRIVASTAVA *et al.*, 2014) has become a popular technique to reduce the network complexity and, consequently, avoid overfitting. Such approach consists in multiplying the output of each unit of a hidden layer by a vector of independent random variables from a Bernoulli distribution with a probability $p = 0.5$ (HOGG; CRAIG, 1995; CHO; RAIKO; ILIN, 2013). In such a way, an architecture is trained on each input data and part of the units are inhibited (outputs are multiplied by zero), so they do not participate in the forward nor in the backpropagation process. In addition, during the test stage, all units are considered and their outputs are multiplied by a vector with all probabilities equal to $wp = 0.5$, given this is an approximation of a geometric mean (KRIZHEVSKY; SUTSKEVER; HINTON, 2012; SRIVASTAVA *et al.*, 2014). The authors claim that Dropout provides a combination of different architectures once the set of active units form different configurations at a time; and it reduces the dependence among units which might be created by the backpropagation algorithm. However, according to Shang *et al.* (2016), the ReLU function creates a correlation between units in order to provide complementary features, which might be untied by Dropout, leading to the production of poor features or increasing even more the processing time to extract adequate features.

In summary, Data Augmentation and Dropout have been applied on most deep learning architectures because they reduce the overfitting and also support convergence guarantees according to the SLT (MELLO; PONTI, 2018; VAPNIK; CHERVONENKIS, 2015). However, both techniques increase the processing time and require more resources to perform training.

3.3.3 Caffe Deep Learning Framework

Several frameworks were developed to support the execution of Deep Learning (DL) algorithms when dealing with large datasets (ABADI *et al.*, 2015; COLLOBERT; BENGIO;

MARITHOZ, 2002; JIA *et al.*, 2014; AL-RFOU *et al.*, 2016; TEAM, 2017). The Caffe Deep Learning Framework [1] (JIA *et al.*, 2014) was selected to perform the experiments in this thesis, given it was the most relevant tool when this study began (SIMONYAN; ZISSERMAN, 2014; GIRSHICK *et al.*, 2014; RUSSAKOVSKY *et al.*, 2015; HE *et al.*, 2016).

Initially, we designed and implemented the CNN forward process in order to support the study of all operations involved in such a network [2]. Our source code was written in the C++ Language, helping us to understand details and ask additional questions about the learning convergence, proofs and the CNN efficiency on different application domains. Those stages were essential to conduct this thesis, which started with the mathematical formulation of CNNs, as detailed in Section 3.2.1.

After a throughout understanding of CNN, we decided to employ the Caffe Deep Learning Framework to perform experiments (SIMONYAN; ZISSERMAN, 2014; GIRSHICK *et al.*, 2014; RUSSAKOVSKY *et al.*, 2015; HE *et al.*, 2016), being written in C++ and CUDA to take advantage of GPU operations. It also provides interfaces to connect with other languages such as Python and MATLAB, and additional support to various types of images such as PNG, JPEG, TIFF, and LevelDB (compressed format by Google [3]).

Caffe modularizes operations, allowing the manipulation of several types of layers, such as convolution, max-pooling, average-pooling, local response normalization, ReLU, and Softmax with Loss [4]. It also allows users to initialize weights and biases for hidden layers, as well as it shuffles input data to improve training. Furthermore, a list of learning algorithms is available to perform weight updating, such as SGD and Adam [5]. Finally, Caffe permits to adapt learning rates along time, supporting convergence improvements under certain scenarios (JIA *et al.*, 2014; GOODFELLOW; BENGIO; COURVILLE, 2016).

3.4 Final Considerations

This chapter discussed the Convolutional Neural Network (CNN), their respective operations, and typical loss functions. Our algebraic formulation was introduced, which provided the basic foundation to study the CNN parameters, such as convolutional mask sizes and number of units. The main CNN learning algorithms were also described, including their approaches to reduce the chance of overfitting. In this context, the Caffe Deep Learning Framework was mentioned, given it was employed in this PhD thesis to perform experiments and validate our strategy of estimating CNN parameters in an attempt to reduce the overall computational costs while maintaining efficient results.

[1] The source code is available at <http://caffe.berkeleyvision.org/>
[2] The source code is available at <https://github.com/marthadais/studyForwardCNN.git>.
[3] LevelDB is available at <https://github.com/google/leveldb>.
[4] All layers and data formats are available at <http://caffe.berkeleyvision.org/tutorial/layers.html>.
[5] All learning algorithms are available at <http://caffe.berkeleyvision.org/tutorial/solver.html>.

CHAPTER

4

RECONSTRUCTING IMAGE RECURRENCES

4.1 Initial Considerations

Our algebraic formulation for Convolutional Neural Networks, described in Section 3.2.1, is the foundation used to employ Dynamical System concepts to estimate architecture parameters. Dynamical Systems (DS) provide tools to model the motion of time-dependent physical systems in terms of differential equations which represent some generation process (KANTZ; SCHREIBER, 2004). At first, the most usual DS approach involves the data reconstruction in another space, a.k.a. phase space, in an attempt to represent the system states and how they evolve along time iterations (ALLIGOOD; SAUER; YORKE, 1997).

The most typical approach to reconstruct data into phase spaces is Takens' immersion theorem, which unfolds temporal recurrences into such multidimensional space (TAKENS, 1981). In this scenario, recurrences correspond to temporal patterns or, simply, in sequences of observations which present some sort of repetition along time. This reconstruction intends to represent the dynamical system trajectory, from which the generation process can be estimated, this is the way of finding the set of equations responsible for producing next system states based on the current ones.

In this context, False Nearest Neighbors (FNN) is the most well-known technique to estimate one of the parameters of Takens' immersion theorem, which is referred to as the embedding dimension (KENNEL; BROWN; ABARBANEL, 1992). Such parameter sets the number of dimensions required to unfold the system trajectory in phase space, so that it represents the coupled dependence of observations along time, thus supporting the unveiling of data recurrences.

In this PhD thesis, image recurrences are a key factor to define the convolutional mask sizes and the number of convolutional units per layer, consequently we took advantage of the knowledge on the FNN method to propose a new approach to support the design of CNN

architectures in a similar fashion, i.e., by exploring data recurrences. The original FNN and our new approach are described next. Results, discussed in Section 5.3, confirm good enough estimations of CNN architectures for most of the datasets considered.

4.2 False Nearest Neighbors

In Dynamical Systems, a nonlinear time series is often analyzed using the time-delay reconstruction theorem proposed by Takens (1981), which is referred to as Takens' immersion theorem. It maps a time series $X = \{x_0, \ldots, x_n\}$ into some phase space in an attempt to produce system states $x_i(m, \tau) = (x_i, x_{i+\tau}, \ldots, x_{i+(m-1)\tau})$, having m as the embedding dimension and τ as the time delay.

Takens' immersion theorem is very useful to deal with time series in real-world problems, because the reconstructed trajectories usually provide relevant temporal features, allowing the system modeling and prediction (RIOS; MELLO, 2013). As main requirement, adequate values for parameters m and τ must be estimated, what is not a simple task.

Several studies are devoted to estimate the embedding dimension and the time delay for general-purpose time series (ALLIGOOD; SAUER; YORKE, 1997; KANTZ; SCHREIBER, 2004). Mutual Information is the mostly used method to estimate the time delay by using the correlation of joint probabilities in an attempt to identify dependencies among system states (COVER; THOMAS, 2012). In this thesis, we consider the time delay is always $\tau = 1$, given the CNN convolutional operation is applied along the neighborhood of image pixels, something that helps to represent objects of interest. As consequence, there is no need to estimate parameter τ.

On the other hand, the False Nearest Neighbors (FNN) method is the most employed approach from the literature to estimate the adequate embedding dimension to unfold time-dependent data (KENNEL; BROWN; ABARBANEL, 1992; RHODES; MORARI, 1997). FNN identifies the number of observations lying in some neighborhood, as the number of space dimensions increase. When the space dimension is smaller than necessary, nearby observations will not belong to the same neighborhood when an additional dimension is included, being those observations classified as false neighbors.

The process of identifying false neighbors is iterative and it measures changes in neighborhood relationships as the embedding dimension increases. When the number of false nearest neighbors is minimum or equal to zero, we say the ideal embedding dimension was found. After such estimation, any additional dimension will not significantly impact on the system states (KENNEL; BROWN; ABARBANEL, 1992; PAGLIOSA; MELLO, 2017; RIOS; MELLO, 2013).

Figure 11 illustrates a false neighborhood confirmed after reconstructing two different

phase spaces from the Hénon Map (HÉNON, 1976), in which two system states are highlighted by green and red points. The system states appear nearest neighbors at a single-dimensional phase space (red point is at (-0.3845454) and the green point is at (-0.3658502)), but after adding another dimension, the states become more distant from each other (red point is at $(-0.3845454, 1.6690975)$ and the green point is at $(-0.3658502, 0.9317967)$). This scenario indicates both system states are false neighbors at least in the one-dimensional phase space. Of course this process has to be repeated until small or no changes are observed.

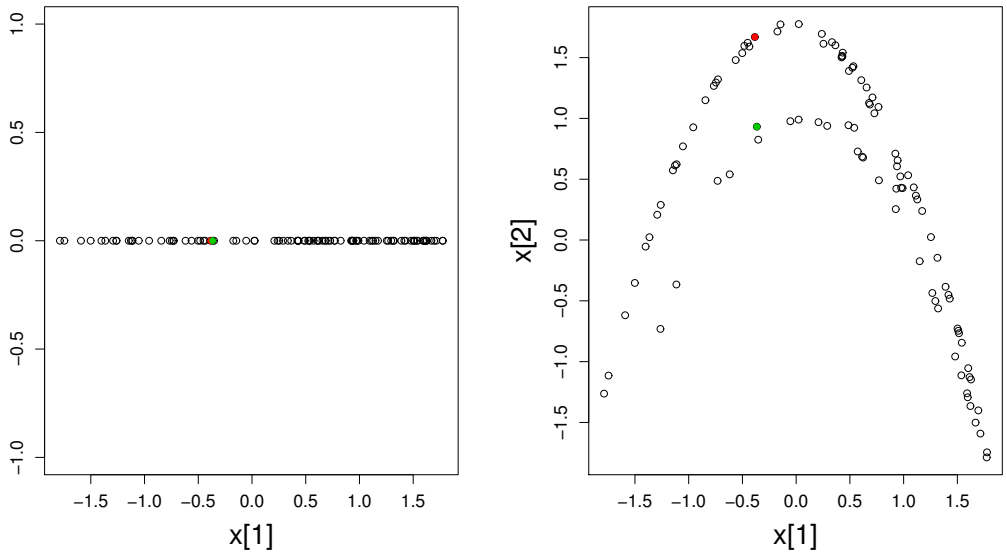


Figure 11 – Illustration of two phase spaces produced using the Hénon Map after Takens's immersion theorem with time delay $\tau = 1$. The space at the left is unidimensional ($m = 1$), while the right-most is bidimensional ($m = 2$). Observe the two states (green and red) considered as neighbors in the unidimensional space are indeed false neighbors when analyzed in the next space configuration.

FNN reconstructs the original data into phase spaces by increasing the embedding dimension m, starting with a small value (usually $m = 2$). Considering this iterative process, an adequate phase space is in $\mathbb{R}^m$. The squared Euclidean distance is computed from state x_i and its r-th nearest neighbor x_i^r to assess the false nearest neighbors criterion, as defined in Equation 4.1. After adding a new dimension $m+1$, such a distance is updated as in Equation 4.2.

$$D_m^2(i,r) = \sum_{k=0}^{m-1} [x_{i+k\tau} - x_{i+k\tau}^r]^2 \tag{4.1}$$

$$D_{m+1}^2(i,r) = D_m^2(i,r) + \sum_{k=0}^{(m+1)-1} [x_{i+k\tau} - x_{i+k\tau}^r]^2 \tag{4.2}$$

During the FNN execution, the time delay τ is kept fixed (here it is always $\tau = 1$). The variation of distances from nearest states is computed after the addition of a new dimension.

Equation 4.3 defines the criterion to assess whether a neighbor is taken as false, in which the r-th nearest neighbor of state $x_i(m,\tau)$ is $x_i^r(m,\tau)$, and $D_m^2(i,r)$ corresponds to the square of the Euclidean distance between $x_i(m,\tau)$ and $x_i^r(m,\tau)$. If the variation is greater than R_t, then states are considered as false neighbors. R_t is a user-defined tolerance and r is the number of neighbors that will be considered along iterations. Nonetheless, Kennel, Brown and Abarbanel (1992) suggest $R_t = 10$ and mention that $r = 1$ is enough to compute the estimation.

$$\sqrt{\frac{D_{m+1}^2(i,r) - D_m^2(i,r)}{D_m^2(i,r)}} > R_t \tag{4.3}$$

After analyzing this criterion, Kennel, Brown and Abarbanel (1992) observed that FNN provides bad estimations for time series containing white noise, therefore reporting that a small embedding dimension is sufficient to unfold noisy series. Noise impacts in neighboring observations, making they look like more distant than they indeed are. Such situation is attenuated when a time series contains more observations.

Considering practical problems, a new criterion was developed to deal with limited data. Equation 4.5 defines the new criterion, in which $D_{m+1}^2(i,r)$ represents the squared Euclidean distance between nearest observations for $r = 1$, A_t is a user-defined tolerance, and D_A is the size or diameter of system trajectories (see Equation 4.4, in which $\bar{x}$ is the average of observations). This criterion may identify whether observations are truly nearest neighbors even though distant from each other, since false nearest neighbors will be stretched to the extremities of the space domain when unfolded. It is worth to mention that Kennel, Brown and Abarbanel (1992) suggest to use both criteria in conjunction when estimating the false nearest neighbors, however just the second criterion was employed in our experiments, since it provided good results and convergence.

$$D_A^2 = \frac{1}{N}\sum_{n=1}^{N}(x(i) - \bar{x})^2 \tag{4.4}$$

$$\frac{D_{m+1}^2(i,r)}{D_A} > A_t \tag{4.5}$$

Finally, FNN computes the fraction of false neighbors for all data, providing a cutoff point employed to estimate the embedding dimension m. As an instance, FNN was applied on the dataset built using the Lorenz System [1]. The fraction of false nearest neighbors is illustrated in Figure 12, in which the estimated value for m is equal to 3, confirming the adequate dimension for such phase space as reported in the literature (PAGLIOSA; MELLO, 2017; RIOS; MELLO, 2013).

[1] This dataset, based on the Lorenz System, is available at <http://people.virginia.edu/~smb3u/psych611/lor63.dat>.

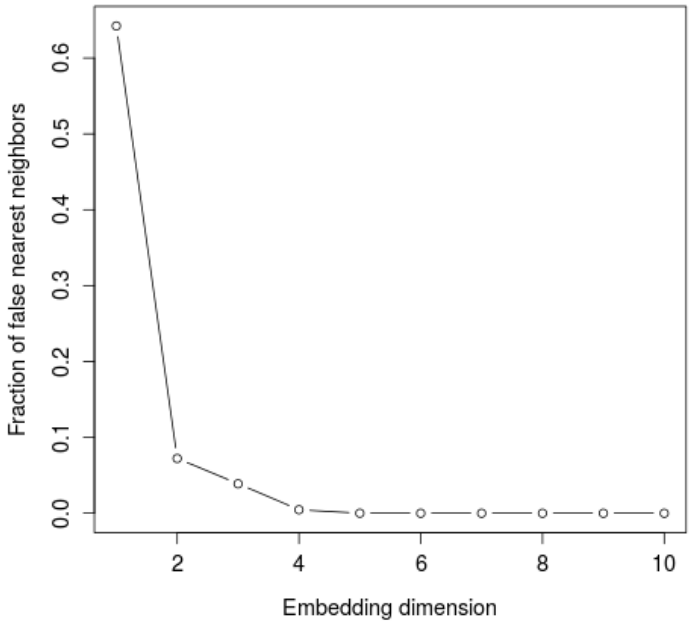


Figure 12 – Illustration of the False Nearest Neighbors method considering a dataset based on the Lorenz System. The estimated embedding dimension is $m = 3$, which is the most adequate according to the literature (PAGLIOSA; MELLO, 2017; ALLIGOOD; SAUER; YORKE, 1997).

4.3 Image-based False Nearest Neighbors

Considering the algebraic CNN formulation detailed in Section 3.2.1, we observed images are, in fact, embedded into phase spaces before proceeding with the application of any linear transformation. This embedding process is quite similar to the embedding proposed by Takens (1981), the greatest difference is that images require two embedding parameters, one along rows (m) and another along columns (n).

In case of convolution, a mask is applied on every local receptive field of the image $I[i-x, j-y]$, for $x = 1,\ldots,m$ and $y = 1,\ldots,n$. This local field can be represented as a column vector $\vec{v} = (I_{i-x_1,j-y_1},\ldots,I_{i,j},\ldots,I_{i+x_m,j+y_n})$, in which the mask size $m \times n$ defines the dimensionality of vectors $\vec{v}_{i,j}$. Therefore, each image pixel is represented by a vector with two embedding dimensions m and n, having a static time delay $\tau = 1$.

In this context, the Image-based False Nearest Neighbors (IFNN) method consists in modifying the original FNN to deal with images, in an attempt to estimate the most adequate embedding dimensions m and n that correspond to the convolutional mask sizes of CNNs (FERREIRA *et al.*, 2018).

Algorithm 1 introduces all modifications made in the original FNN, now requiring the set of vectors $\mathscr{I}$ of some image I for a given $m \times n$, the number of nearest neighbors r, a value for the threshold (or tolerance) A_t, and also a recurrence offset T. Parameter A_t characterizes when

two vectors are false neighbors and parameter T defines an offset in which we expect recurrences to happen, similarly to the recurrent trajectories of Dynamical Systems (ALLIGOOD; SAUER; YORKE, 1997; KANTZ; SCHREIBER, 2004). The most adequate value for T depends on the target problem. Thus, in case of images, $T = 1$ was used, once local image regions usually maintain similar characteristics. In further experiments, parameters r and A_t were set as suggested by Kennel, Brown and Abarbanel (1992), thus $r = 1$ and $A_t = 10$.

Algorithm 1 – Analysis of the fraction of false nearest neighbors for the set of vectors $\mathscr{I}$, after some phase space reconstruction.

INPUT: $\mathscr{I}$: set of vectors $\vec{v}_{i,j}$ for a given input data I; r: number of nearest neighbors; A_t: radius threshold for neighbors; T: offset of vectors contained in $\mathscr{I}$.
OUTPUT: ffn: fraction of false nearest neighbors

function ADAPTEDFNNANALYSIS($\mathscr{I}$, r, A_t, T)
  Set $counter = 0$
  Set $denom = 0$
  ▷ the positions of vectors $\vec{v}_{i,j} \in \mathscr{I}$ are used as index idx. Thus, $\mathscr{I} = \{\vec{v}_{1,1}, \vec{v}_{1,2}, \ldots, \vec{v}_{q,p}\}$ will be here represented using a simplified notation as follows $\mathscr{I} = \{\vec{v}_1, \vec{v}_2, \ldots, \vec{v}_{idx}, \ldots, \vec{v}_{q \times p}\}$
  for every vector $\vec{v}_{idx} \in \mathscr{I}$ **do**
    Finds the r nearest vectors of $\{\vec{v}_{idx}\}$ in $\mathscr{I}$
    for each neighbor $\vec{n}_{idx}$ of vector $\vec{v}_{idx}$ **do**
      $d =$ **Euclidean**$(\vec{n}_{idx}, \vec{v}_{idx})$
      $d = d +$ **Euclidean**$(\vec{n}_{idx+T}, \vec{v}_{idx+T})$
      if $d > A_t$ **then**
        counter++
      end if
    end for
    $denom = denom + r$
  end for

  $ffn = counter/denom$ **return** ffn
end function

At the end, Algorithm 1 provides the average value of false nearest neighbors for a given phase space $\mathscr{I}$ estimated from image I. When a small fraction of neighbors is provided by the algorithm, $\mathscr{I}$ behaves in the same way as a set of states of a well-formed attractor in phase space (ALLIGOOD; SAUER; YORKE, 1997; KANTZ; SCHREIBER, 2004). On the other hand, a bad-formed attractor will result in large output values for the algorithm. It is worth to mention that an attractor is considered as an object of a dynamical system, in which a set of points from a phase space represents a trajectory (behavior) of a particular time series (PAGLIOSA; MELLO, 2017; RIOS; MELLO, 2013). Lastly, adequate embedding dimensions are achieved when the fraction of false nearest neighbors becomes stable around zero.

Algorithm 1 analyzes a phase space produced for a given pair m and n of embedding dimensions. Next, Algorithm 2 varies the two embedding dimensions m (number of rows) and n (number of columns) along iterations. Actually, it considers a range, thus embedding dimensions in rows vary in interval $[1, maxM]$ (maximum number of rows) and along columns

in $[1, maxN]$ (maximum number of columns). In summary, an input image I is embedded into a multidimensional space according to each mask size $m \times n$, and then the fraction of false nearest neighbors is computed and stored in a matrix F. The rows and columns indices of matrix F correspond to the m and n used to perform the underlying embedding.

Algorithm 2 – Algorithm to estimate adequate mask sizes for the convolutional units of a CNN layer.

INPUT: I: input data; $maxM, maxN$: maximum number of rows and columns that a convolutional mask can assume, respectively; r: number of nearest neighbors; A_t: radius threshold for neighbors; T: offset of vectors contained in $\mathscr{I}$.
OUTPUT: F: matrix containing the fraction of false nearest neighbors for all mask sizes evaluated.

function MASKEVALUATION(I, $maxM$, $maxN$, r, A_t, T)
  Initializes matrix F of size $maxM \times maxN$
  for $m = 1$ to $maxM$ **do**
    for $n = 1$ to $maxN$ **do**
      $\mathscr{I} =$ **embed**(I, m, n); ▷ The function embed returns a matrix containing all vector produced by the embedding according to dimension $m \times n$ (more details in Section 3.2.1)
      $F[m,n] =$ **adaptedFNNAnalysis**($\mathscr{I}, r, A_t, T$)
    end for
  end for
return $F[m,n]$
end function

The algorithm is executed on every individual image from the training set, providing a matrix F for each image that contains a fraction of false neighbors for different mask sizes. Therefore, an element-wise average is computed using all matrices F produced after analyzing each image. This results in a single matrix $\bar{F}$ of false neighbors for the entire dataset.

According to previous experiments, we noticed that after some iterations matrix $\bar{F}$ becomes stable. As consequence, we designed a stop criterion to avoid the execution for all images contained in the training set, significantly improving the processing time. The norm of the average matrix $\| \bar{F} \|$ is used as convergence criterion for our algorithm. It is worth to mention that the input images need to be stratified in order to represent all necessary features for a dataset.

The stop criterion analyzes the difference between current and previous norm matrices ($\| \bar{F} \|_t$ and $\| \bar{F} \|_{t+1}$). If such difference is smaller than a threshold, the execution is stopped. The suggested threshold is 0.001, which was empirically set after an experimental analysis.

Finally, a histogram is computed considering the average matrix $\bar{F}$ to rank the most adequate convolutional mask sizes. For each row in matrix $\bar{F}$, the column containing the smallest fraction of false neighbors is counted. Next, the same is performed along columns, that is, for each column in $\bar{F}$, the row having the smallest fraction of false neighbors is counted. At the end, the histogram is represented by a matrix indicating which row and column is the most adequate, having the maximum value equals to 2. This representation was used because it presents the best estimation for a given row and the best estimation for a given column, as indicated by the counter.

Thus, when the row and column match for both orientations, the number in the histogram is equal to 2, suggesting that such row and column are the best combination for the dataset.

In order to illustrate our IFNN method, Algorithm 2 was performed on the MNIST dataset (LECUN *et al.*, 1998). The (already available) training set of MNIST was used as input data, and a new image was given to IFNN until the stop criterion was satisfied. This is the same process performed in Section 5.3, in which the threshold value was set as 0.001, the maximum number of rows and columns was configured as $maxM = maxN = 15$, the number of nearest neighbors was fixed as $r = 1$, and the radius threshold for neighbors was defined as $A_t = 10$. In such situation, IFNN produced a matrix F whose axes represent mask sizes and values correspond to the fraction of false nearest neighbors. An individual image from the training set and its corresponding matrix F are illustrated in Figure 13.

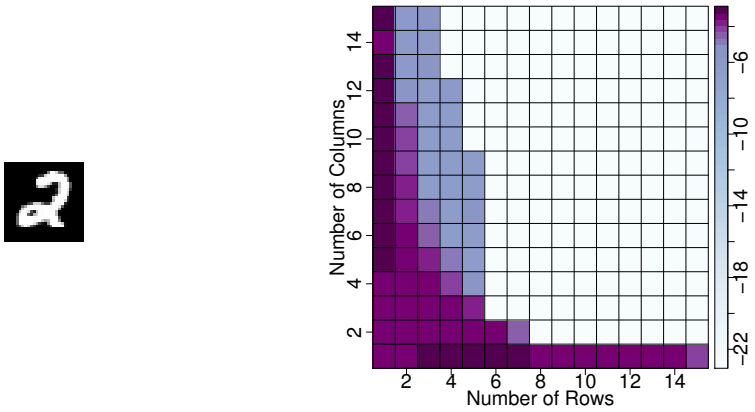


Figure 13 – Sample of the results obtained after IFNN using a single input image from the MNIST dataset to select an adequate convolutional mask size: Figure (a) illustrates an image of digit "2" (32×32) which belongs to the training set; and Figure (b) corresponds to the matrix with the fraction of false nearest neighbors produced for the digit from Figure (a), in which a logarithm function was used to better illustrate the results. In Figure (b), the x-axis represents the size of masks in terms of columns, while the y-axis is associated to the number of mask rows.

After finishing the execution, the average matrix $\bar{F}$ and its histogram were obtained. Figure 14 illustrates both matrices ($\bar{F}$ and its histogram). It is worth to mention that the input images were randomly selected from the training set, in which 198 images were necessary to meet the IFNN stop criterion.

In order to validate the IFNN results, a CNN architecture was parametrized and assessed having all convolutional mask sizes in range $[1, 15]$ for both rows and columns. The maximum value for the range was defined as close as possible to half of the image size, which has already provided good results. The architecture employed was the same used in Section 5.3, being a CNN with one convolutional layer with ReLU as activation function, followed by a max-pooling layer of size 3×3 with a stride 2×2. In such situation, the number of units was fixed in 10 simply in order to assess the impact of the convolutional mask sizes.

This experiment was performed using the Caffe framework (described in Section 3.3.3)

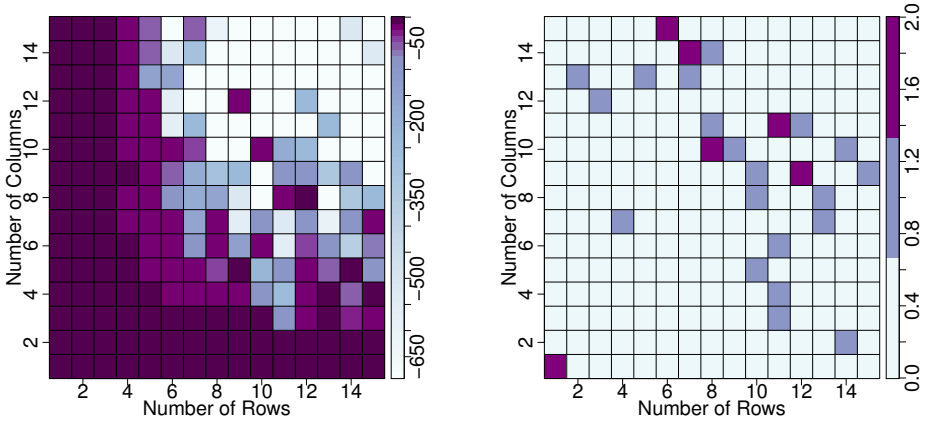


Figure 14 – Results obtained with IFNN while selecting an adequate convolutional mask size for the MNIST dataset: Figure (a) represents the element-wise average matrix $\bar{F}$ produced after the IFNN, given 198 randomly selected images from the training set; and Figure (b) corresponds to the histogram used to select the best five mask sizes. In both figures, the x-axis represents the size of masks in terms of columns, while the y-axis is associated with the number of mask rows.

with the SGD method (described in Section 3.3) as discussed in Section 5.3, having the learning rate as 0.001 (fixed – without reduction); momentum as 0.9; weight decay as 0.004 (regularization term), and the training batch as 100 samples. Error rates confirmed that the convolutional mask sizes impacted on the classification performances, corroborating the results provided by IFNN (see Figure 15). Such rates started to decrease at the same point the fraction of false nearest neighbors achieved values close to zero.

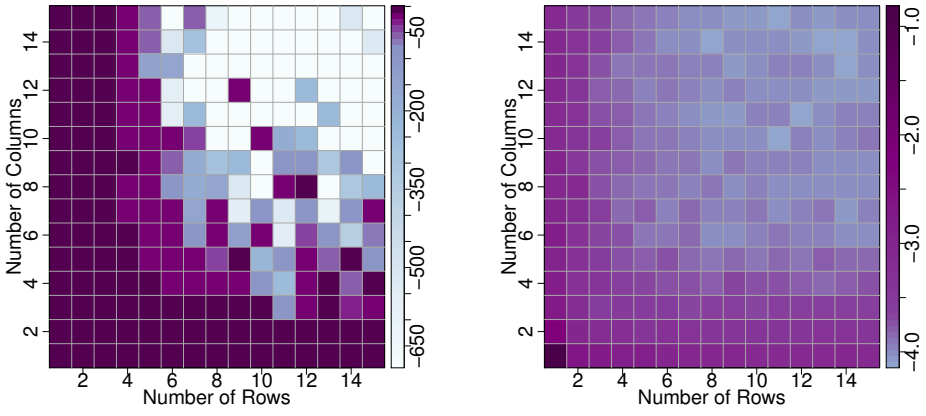


Figure 15 – Illustration of the IFNN method and the CNN results for the MNIST dataset: Figure (a) represents the element-wise average matrix produced by the IFNN given random images from the training set; and Figure (b) corresponds to the error rates, considering all convolutional mask sizes under analyses. In case of Figure (b), a logarithm function was applied to improve the visualization. In both figures, the x-axis represents the size of masks in terms of columns, while the y-axis is associated with the number of mask rows.

This benchmark supported the conclusion that IFNN supports the detection of image patterns, producing a fair estimation of the convolutional mask sizes. In addition, IFNN does not require long CNN training stages to assess possible mask sizes, just taking few hours to perform

our estimation approach.

4.3.1 Number of Convolutional Units

Another great challenge discussed in the literature involves the definition of an adequate number of convolutional units per CNN layer (KRIZHEVSKY; SUTSKEVER; HINTON, 2012; HE *et al.*, 2016; SZEGEDY *et al.*, 2015; GARRO; VÁZQUEZ, 2015; YOUNG *et al.*, 2015; ALBELWI; MAHMOOD, 2017). In this thesis, we adapted our previously described approach to address such a goal. According to Section 3.2.1, a convolutional layer contains l units that apply the convolution operation on some local receptive field $\vec{v}_{i,j}$ of image I. Such a process is equivalent to map $\vec{v}_{i,j}$ into an l-dimensional space.

Observe that if $l > m \times n$, the produced space will have a higher dimension than the input space, bringing no additional information. When $l = m \times n$, an inverse transformation may exist (STRANG, 1988), simplifying the equations of a training stage. Finally, considering $l < m \times n$, the input space suffers a dimensionality reduction. Therefore, the goal is to estimate l, having $l \leq m \times n$.

Firstly, the most adequate convolutional mask size is estimated using Algorithm 2. The mask size estimated is then provided as input to Algorithm 3, which is responsible for analyzing the number of convolutional units and how they impact linear transformations. Thus, we embedded images with the mask size estimated and, next, l random linear transformations were applied to produce a new space S to be analyzed using IFNN.

Algorithm 3 receives the input image I, the convolutional mask size $m \times n$, the number of neighbors r, the tolerance A_t, the offset T, and number maximum of units $maxL$. The obtained result is a vector $\vec{u}$ with the fraction of false neighbors from 1 to $maxL$ units.

This algorithm was executed on 200 random input images from a given training set, providing a vector $\vec{u}$ of fraction of false neighbors for each one. The fixed amount of images (200) was empirically chosen after previous experiments, once this value has already presented a good result stabilization. As new units are added, a reduction of the fraction false nearest neighbors is expected.

In order to obtain a single vector as result, the element-wise average vector $\vec{u}'$ was computed. The most adequate number of convolution units was represented by the index of vector $\vec{u}'$ that firstly reached zero (or near enough) for the fraction of false nearest neighbors. As main conclusion, the space produced by the linear transformation is then enough to represent the input space.

To illustrate this process, the mask size 8×10 estimated in a previous example (Figure 14) is given as input to Algorithm 3, which is used to find the adequate number of units for a single CNN layer. The element-wise average vector $\vec{u}'$ obtained is presented in Table 1, in which 9 was found as the best number of units, since it is the index of the first zero in $\vec{u}'$.

Algorithm 3 – Algorithm to estimate the number of convolutional units for a given CNN layer.

INPUT: I: input data; m,n: fixed number of rows and columns for every convolutional mask, respectively; $maxL$: maximum number of units to be considered; r: number of nearest neighbors; A_t: radius threshold for neighbors; T: offset of vectors contained in $\mathscr{I}$.

OUTPUT: $\vec{u}$: vector containing the fraction of false neighbors as each unit is added to the convolutional layer.

function NUMBEROFUNITSEVALUATION(I,m,n,$maxL$,r,A_t,T)
  Initialize vector $\vec{u}$ with $maxL$ elements
  Initialize matrix Θ as empty
  $\mathscr{I} =$ **embed**(I, m, n) ▷ The function embed returns a matrix containing all vector produced by the embedding according to dimension $m \times n$ (more details in Section 3.2.1)
  for $k = 1$ to $maxL$ **do**
    Generate a random convolutional mask Θ_k with dimensions $m \times n$
    Add $\vec{\Theta}_k^T$ as the next row of matrix Θ
    Apply the convolution on vectors in $\mathscr{I}$ to obtain the new space S in form $S = (\Theta \mathscr{I}^T)^T$
    $\vec{u}[k] =$ **adaptedFNNAnalysis**(S, r, A_t, T)
  end for
return $\vec{u}$
end function

Table 1 – Vector obtained while analyzing the number of units for a single-layer CNN given the MNIST dataset with the convolutional mask size 8×10. Vector indices correspond to the number of units being assessed. The first zero is used to estimate the number of units, being equal to 9 in this case.

Index	Fraction of false neighbors
1	1
2	0.1354
3	0.0212
4	0.0076
5	0.0053
6	0.0015
7	0.0007
8	0.0007
9	0
10	0

Algorithms 2 and 3 were implemented in the R Statistical Language (IHAKA; GENTLEMAN, 1996), while Algorithm 1 was implemented using the C Language. An R package containing all source codes of IFNN is available at <https://github.com/marthadais/Estimate-Parameters>, so that the reader can perform experiments to confirm our results as well as address further classification tasks (FERREIRA *et al.*, 2018).

4.4 Final Considerations

In this chapter, we employed Dynamical System concepts to support the estimation of Convolutional Neural Networks architecture parameters, to mention: the convolutional mask sizes and the number of convolutional units per CNN layer. The original False Nearest Neighbors (FNN) was described, since it motivated our Image-based False Nearest Neighbors (IFNN) method, which employs dynamical system concepts to reconstruct image recurrences and estimate convolutional mask sizes and the number of convolutional units per CNN layer, as discussed. The evaluation of IFNN is presented in Section 5.3, which provides a discussion on the obtained results. As the number of convolutional units estimation was not enough for some datasets, we devoted an additional study (see Appendix A) on such parameter and analyzed it in terms of the Statistical Learning Theory (SLT) (VAPNIK, 2000; MELLO; PONTI, 2018).

CHAPTER

5

EXPERIMENTS

5.1 Initial Considerations

Based on the algebraic formulation presented in Section 3.2.1, we observed that convolutional operations can be written as linear transformations, having input images reconstructed into some space after an embedding. This reconstruction is similar to the ones employed in the context of Dynamical Systems, which are mostly based on Takens' immersion theorem (TAKENS, 1981).

In this context, False Nearest Neighbors (FNN) is a well known method from Dynamical Systems to estimate the embedding dimension, producing the most adequate space for some time-dependent dataset. As described in Section 4.3, FNN motivated us to estimate the convolutional mask sizes and number of units per CNN layer. This chapter presents our experimental results obtained using IFNN to design simple but yet efficient CNN architectures while addressing different classification tasks, including handwritten digit, face detection and object recognition.

5.2 Datasets

Five datasets were considered to evaluate the Image-based False Nearest Neighbors (IFNN) method (Section 4.3), being the Carnegie Mellon University Face Images (CMU) the first one (ROWEIS; SAUL, 2000) [1], which contains $1,872$-grayscale face images in four variations for the same person so that different classification tasks can be performed. The four variations are pose (left, right, up, straight), expression (happy, sad, angry, neutral), eyes (wearing sunglasses or not), and size (128×120, 64×60, 32×30). The smallest images provided (32×30) were used to perform the experiments described in this chapter, composing a total of 640 images. In such experiments, CMU was analyzed considering two classification possibilities: i) which

[1] CMU is available at <https://archive.ics.uci.edu/ml/datasets/CMU+Face+Images>.

person is present in pictures (20 classes); and ii) if the person wears sunglasses or not (2 classes).

The second dataset used to analyze IFNN is the Modified National Institute of Standards and Technology (MNIST) dataset (LECUN *et al.*, 1998) [2], which is composed of binary images of handwritten digits. It contains 70,000 images in total, being divided in 60,000 as training set and 10,000 as test set. MNIST is taken as a Deep Learning benchmark, and it has been widely used in the literature (LECUN *et al.*, 1999; GOODFELLOW *et al.*, 2013; CIREGAN; MEIER; SCHMIDHUBER, 2012).

The next dataset was the Canadian Institute for Advanced Research (CIFAR-10) dataset (KRIZHEVSKY; HINTON, 2009) [3] which supports the object recognition task. It is composed of small RGB images (32×32) divided into 10 classes: airplane, automobile, bird, cat, deer, dog, frog, horse, ship and truck. This dataset contains 60,000 images in total, each class containing 6,000 images (stratified dataset). The training set of CIFAR-10 is formed with 50,000 images and the remaining 10,000 are part of the test set.

Another dataset employed in our experiments, also in the object recognition task, is the Columbia University Image Library (COIL-100) (NENE; NAYAR; MURASE, 1996) [4], which is formed by 100 classes. It contains 7,200 RGB images, each class composed of 355 images with pose variations (stratified dataset). A preprocessed version of this dataset was considered in experiments, in which the brightness and the intensity of the pixels were scaled (NENE; NAYAR; MURASE, 1996).

The fifth dataset is the German Traffic Sign Recognition Benchmark (GTSRB) (STALLKAMP *et al.*, 2012) [5], which is taken as a multiclass benchmark. It was used in a classification competition at the International Joint Conference on Neural Networks (IJCNN) in 2011 and 2013 (HOUBEN *et al.*, 2013; SOCIETY, 1989 (accessed September 27, 2018)). This dataset contains 144,769-labeled traffic sign images under different sizes. The training set used in experiments has 39,209 images and the test set contains the remaining 12,630 images. GTSRB is divided into 43 classes.

In order to perform the CNN experiments, datasets were organized in training and test sets, as performed in similar studies (GOODFELLOW *et al.*, 2013; LECUN *et al.*, 1999; SRIVASTAVA *et al.*, 2014; HINTON; SALAKHUTDINOV, 2006; WESTON *et al.*, 2012; HASSAIRI; EJBALI; ZAIED, 2015; CIREGAN; MEIER; SCHMIDHUBER, 2012; SERMANET; LECUN, 2011). In the case of MNIST, CIFAR-10 and GTSRB, the training and test sets were already provided. CMU and COIL-100 are not organized into those sets, thus we decided to follow a similar approach by randomly selecting 70% of samples to compose the training set and the remaining for testing. In addition, images from COIL-100 and GTRSB datasets were

[2] MNIST is available at <http://yann.lecun.com/exdb/MNIST/>.
[3] CIFAR-10 is available at <http://www.cs.utoronto.ca/~kriz/CIFAR.html>.
[4] COIL-100 is available at <http://www1.cs.columbia.edu/CAVE/software/softlib/coil-100.php>.
[5] GTSRB is available at <http://benchmark.ini.rub.de/index.php?section=gtsrb&subsection=dataset>.

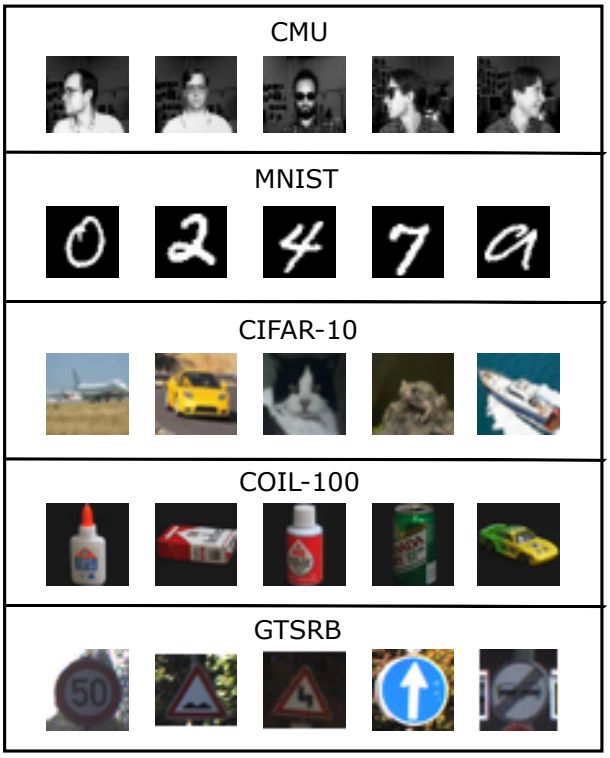


Figure 16 – Five samples of the images contained in each one of the five datasets. Those images were already resized.

resized to 32×32 pixels in order to improve the computational performance and to ensure that all images had the same size (KRIZHEVSKY; SUTSKEVER; HINTON, 2012; LECUN *et al.*, 1999; SRIVASTAVA *et al.*, 2014; CIREGAN; MEIER; SCHMIDHUBER, 2012). It is worth to mention that the training sets were used as input to the IFNN method, even though it does not require label information. Table 2 summarizes all datasets information and Figure 16 illustrates some samples from datasets.

Table 2 – Summarizing the information on datasets, including the size, number of classes, if images are colorful, if the set is already provided in terms of training and test sets.

Datasets	Number of Samples	Number of Classes	Colorful Images	Separated Sets Provided	Resized
CMU	640	20 and 2	No	No	No
MNIST	70,000	10	No	Yes	No
CIFAR-10	60,000	10	Yes	Yes	No
COIL-100	7,200	100	Yes	No	Yes
GTSRB	51,839	43	Yes	Yes	Yes

5.2.1 Literature Results

Deep architectures are typically employed because have some indications that the error rates reduce when the network sizes increase. For examples, recent complex and large configura-

tions of networks, such as GoogLeNet (SZEGEDY *et al.*, 2015) and AlexNet (KRIZHEVSKY; SUTSKEVER; HINTON, 2012), provide high accuracies for the object recognition task, however they take much time to be trained and they are much more prone to overfitting, requiring the usage of techniques as Dropout (SRIVASTAVA *et al.*, 2014) and DeCov (COGSWELL *et al.*, 2016).

The same five datasets previously discussed were already used to assess the performance of DL algorithms (LECUN *et al.*, 1999; HINTON; SALAKHUTDINOV, 2006; CIREGAN; MEIER; SCHMIDHUBER, 2012; KRIZHEVSKY, 2010; GOODFELLOW *et al.*, 2013; SERMANET; LECUN, 2011). For instance, MNIST is taken as the main benchmark for handwritten digit recognition task [6]. The best result reported for such dataset was obtained by Ciregan, Meier and Schmidhuber (2012) who achieved the error rate of 0.23%. Those authors developed the Multi-Column Deep Convolutional Neural Networks (MCDNN), which is composed of several simultaneous CNNs. The first layer of MCDNN is responsible for applying a different normalization on the image. Therefore, a set of CNNs is designated to perform the classification for each neuron normalization, and the results of all CNNs are combined to provide a single result. In that scenario, MCDNN was composed of 35 columns, using 7 different normalizations (units on the first layer) and 5 CNNs to classify each normalization, i.e. each normalization neuron feeds 5 CNNs. Each CNN was composed of 6 layers in the following sequence: i) convolutional with 20 units of size 4×4; ii) max-pooling of size 2×2; iii) convolutional with 40 units of size 5×5; iv) max-pooling of size 2×2; v) fully-connected with 150 units; and vi) fully-connected with 10 units.

Another good result for the MNIST dataset was obtained by LeCun *et al.* (1999), who reported an error of 0.95% with LeNet-5, reducing it to 0.8% after applying a Data Augmentation strategy (described in Section 3.3.2), that includes distorted data as part of the training set. In this way, a combination of three LeNet-4 architectures, trained with different subsets of the MNIST dataset, reduced even more such error, achieving 0.7%. Both architectures, LeNet-4 and LeNet-5, are composed of 7 layers as follows: i) convolutional with 6 units; ii) average-pooling of size 2×2; iii) convolutional with 16 units; iv) average-pooling of size 2×2; v) convolutional with 120 units; vi) fully-connected with 84 units; and vii) fully-connected 10 units. The difference between them is the convolutional mask size in convolutional layers, in which LeNet-4 is build up with 4×4 mask size, while LeNet-5 is 5×5.

Other two important benchmark datasets are GTSRB and CIFAR-10, on which Ciregan, Meier and Schmidhuber (2012) employed a MCDNN to reach error rates of 0.54% and 11.21%, respectively. The MCDNN applied on GTSRB was composed of 25 columns, being 5 normalizations and 5 CNNs, each one composed of 8 layers as follows: i) convolutional with 100 units of size 7×7; ii) max-pooling of size 2×2; iii) convolutional with 150 units of size 4×4; iv) max-pooling of size 2×2; v) convolutional with 250 units of size 4×4; vi) max-pooling of size

[6] The main results for the MNIST dataset are available at <http://yann.lecun.com/exdb/MNIST/>.

2×2; vii) fully-connected with 300 units; and viii) fully-connected with 43 units. In case of CIFAR-10, MCDNN was composed of 8 columns, being 2 normalization and 4 CNNs, each one with 9 layers and 300 units as follows: i) a convolutional of size 3×3; ii) a max-pooling of size 2×2; iii) a convolution of size 2×2; iv) a max-pooling of size 2×2; v) a convolutional of size 3×3; vi) a max-pooling of size 2×2; vii) a convolutional of size 2×2; viii) a max-pooling of size 2×2; ix) a fully-connected; x) a fully-connected with 100 units; and xi) fully-connected with 10 units.

In order to classify CIFAR-10, Coates and Ng (2011) applied CNN in conjunction with unsupervised learning to extract relevant features and the Support Vector Machine (SVM). Different CNN architectures were employed to validate the most adequate configuration, in which the best architecture had 6 layers as follows: i) convolutional of size 6×6 with $1,600$ units; ii) average-pooling of size 3×3; iii) convolutional of size 2×2 with $3,200$ units; iv) average-pooling of size 2×2; v) convolutional of size 2×2 with $3,200$ units; and vi) average-pooling of size 2×2. This configuration achieved an error rate of 18.0%.

Srivastava *et al.* (2014) used a CNN with Dropout to also deal with CIFAR-10, obtaining an error rate of 12.61%. The architecture used was composed of 8 layers, in which convolutional layers had 5×5 filters and 3×3 max-pooling units with stride equals to 2×2. The sequence of layers was: i) convolutional with 96 units; ii) max-pooling; iii) convolutional with 128 units; iv) max-pooling; v) convolutional with 256 units; vi) max-pooling; vii) fully-connected with $2,048$ units; and viii) fully-connected with $2,048$ units.

In an attempt to improve CNN accuracies, MNIST and CIFAR-10 were used to evaluate a new activation function called Maxout, as proposed by Goodfellow *et al.* (2013). Maxout was applied on units of convolutional layers, in which the convolution output was multiplied by a diagonal weight matrix and summed to a bias vector. This approach produced 0.45% as error rate for MNIST and 9.38% for CIFAR-10, using a CNN architecture composed of 3 layers with 64 3×3-units each, and Dropout to improve the classification.

Sermanet and LeCun (2011) proposed a Multi-Scale Convolutional Neural Network (MS-CNN) with two-level features to classify the GTSRB, achieving 0.83% as error rate. The CNN architecture was composed of 6 layers, being 3 convolutionals with hyperbolic tangents as activation functions, 2 subsampling and 1 fully-connected, however the authors do not provide additional information about the number of units per layer and mask sizes. This network used the output of the second CNN layer to compose the first-level features, and the fifth CNN layer output as second-level features. The two-level features were given as input to a sequence of fully-connected layers to perform classification.

COIL-100 is another object recognition dataset used as benchmark to validate classification techniques. Hassairi, Ejbali and Zaied (2015) reported an error rate equals to 0.8%, using a Deep Convolutional Wavelet Network (DCWN) to identify individual classes. The DCWN is an architecture composed of wavelets, RBMs, convolutional and Local Contrast Normalization

(LCN) layers in a complex configuration. This model uses a Gaussian kernel to compute the subsampling and sigmoid as activation function. On the other hand, Zou *et al.* (2012) converted COIL-100 in a video dataset to proceed with the classification. A 2-layer Deep Neural Network was employed in which the first layer was responsible for convolutions, followed by a L2-pooling, and the second applied the Principal Component Analysis (PCA) method. With this approach, Zou *et al.* (2012) obtained an error rate equals to 13% for COIL-100.

The CMU dataset was used to conduct classification tasks involving the identification of people, and if they were wearing sunglasses. Zhu *et al.* (2016) reported an error rate of 5.68% using CNN to identify people wearing sunglasses. That architecture was composed of 4 convolutional layers with 92, 192, 256 and 384 units, respectively, one fully connected layer with 512 units, and a modified SVM to perform the classification at the last stage. On the other hand, Teller and Veloso (1995) used genetic programming to identify the person at each image, reaching an error rate of 8%.

Considering Deep Belief Networks (DBN), Hinton and Salakhutdinov (2006) achieved an error rate equals to 1.25% for the MNIST dataset. Their architecture was composed of 5 Restricted Boltzmann Machines (RBM) with 784, 500, 500, $2,000$ and 10 units, respectively. A greedy learning algorithm was used in such DBN implementation to support training. Srivastava *et al.* (2014) also applied a DBM network with Dropout in order to classify MNIST, considering an architecture with 3 layers with 500, 500 and $2,000$ units, respectively. Such DBN used a logistic as activation function, reporting an error rate equals to 0.79%.

DBN was also employed by Krizhevsky (2010) to classify CIFAR-10 using 9×9 convolutional filters to extract edges. Such architecture was composed of 2 convolutional RBM layers, having the first containing 64 units and the second with $1,024$ units. It achieved an error rate of 21.1%.

Considering the COIL-100 dataset, Yang, Roth and Ahuja (2000) studied the Sparse Network of Winnows (SNoW) to represent image edges and recognize objects, reporting an error rate of 7.69%. Each neuron of SNoW was used to represent a subnetwork responsible for classifying a single class, therefore the architecture had a number of units equal to the number of dataset classes. A detailed description was not provided for subnetworks, what also occurs with the network proposed by Sermanet and LeCun (2011). This lack of information brought up some difficulties on the reproduction of results, jeopardizing the development of the areas of Artificial Neural Networks (ANN) and Deep Learning.

Most of the Deep Leaning networks from literature were built up with several layers and units per layer (SZEGEDY *et al.*, 2015), reporting good results as presented in here. All those results are used as baseline in this thesis to compare against the error rates provided by IFNN.

5.3 Estimation Experiments

Experiments were performed (datasets are described in Section 5.2) in order to assess the Image-based False Nearest Neighbors (IFNN) method, proposed in this thesis. As reported next, our results are similar to the ones from the literature (discussed in Section 5.2.1), supporting our hypothesis that simpler CNN architectures can provide good performances. It is worth to mention that data augmentation was not used.

Firstly, the most adequate convolutional mask size and convolutional number of units per layer was obtained considering IFNN along each dataset. Then, a CNN with a single convolutional layer with the estimated parameters was executed. The architecture for the single CNN was composed of one convolutional layer with ReLU activation function, followed by a max-pooling layer with size 3×3 and stride 2×2 (based on the literature – see Sections 2.2.1 and 5.2.1) (KRIZHEVSKY, 2010; KRIZHEVSKY; SUTSKEVER; HINTON, 2012; JIA *et al.*, 2014). The assessment of this latter parametrization is out of the scope of this thesis. The max-pooling operation was included as it improved the overall CNN performance (KRIZHEVSKY, 2010; KRIZHEVSKY; SUTSKEVER; HINTON, 2012; CIREGAN; MEIER; SCHMIDHUBER, 2012).

After training the CNN architecture with a single convolutional layer, the max-pooling output was used as input to IFNN in order to estimate the parameters for a second convolutional layer. In that sense, the next CNN architecture was composed of two convolutional layers interposed by two max-pooling layers with the same parameters (size 3×3 and stride 2×2). In this situation, the first convolutional layer was set with the parameters estimated considering the training set and the second layer was configured with the parameters estimated using the max-pooling outputs. The same process could be applied as more convolutional layers are included. In those experiments, we limited the number of convolutional layers to three.

In summary, IFNN is used to estimate all parameters of CNN architectures in an iterative way. It is worth to mention that the estimation does not require a large input data sample. In addition, the CNN training execution, that provides the next input for the IFNN, did not require neither the whole dataset nor several iterations, decreasing the time spent on estimations. The final network employed in the selected datasets is composed of three convolutional layers interposed by three max-pooling layers.

Algorithm 2 was executed on the input images, until reaching the stop criterion with a threshold equals to 0.001. The number of rows and columns varied in range $[1, 15]$ for all datasets, an interval also predefined according to previous experiments. At the end, the element-wise average matrix $\bar{F}$ containing the measurements of the fraction of false neighbors was obtained. Finally, a histogram is computed to select the best five convolutional mask sizes (more details in Section 4.3).

After selecting the five best mask sizes, Algorithm 3 was executed on 200 input images

from every training set, producing an average vector $\vec{u}'$ containing the measurements of fraction of false neighbors (more details in Section 4.3). In this case, the number of convolutional units was varied in range $[1,50]$, and the most adequate number of units is given by the index of vector $\vec{u}'$ with the smallest fraction of neighbors. In this algorithm, the range was empirically selected after assessing a greater interval which did not provide any result above 50.

Experiments, performed using the Caffe framework, evaluated the 5 best parameters estimated with IFNN. For every dataset, CNNs ran along $10,000$ iterations in an attempt to select the best convolutional mask sizes. As described, max-pooling was applied right after the convolutional layer, and the last layer was a fully-connected one whose units represented dataset labels and performed the classification using the softmax function (see Section 3.2). At the training stage, the parameters were empirically selected and fixed for all datasets to evaluate the impact of the convolutional mask sizes and number of units on the classification performance. The SGD method was employed to provide convergence, in which learning rate was set as 0.001 (fixed – without reduction); momentum as 0.9; weight decay as 0.004 (regularization term), and the training batch as 100 samples [7]. Those parameters were chosen after assessing other CNN architectures implemented on the Caffe framework.

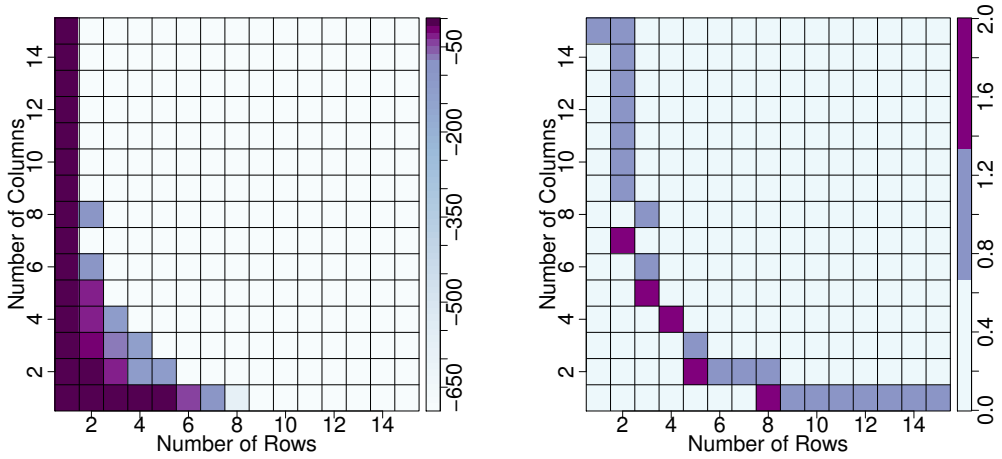
Considering the CNN architecture composed of a single layer, the average matrix $\bar{F}$ and the histogram obtained after the IFNN are illustrated along Figures 17b to 21b. As the estimation is conducted in an iterative way, the error rate for the 5 best convolutional mask sizes, estimated for each additional convolutional layer, is provided. It is worth to mention that just the mask size producing the best result in a previously step was kept to estimate the next, in which the features produced by the max-pooling layers were given as input to IFNN.

Figures 17a and 17b illustrate the results obtained by the IFNN applied on the CMU dataset to estimate the convolutional mask size for the first convolutional layer. Figure 17a shows the average of fraction of false nearest neighbors, while the counting histogram of the best sizes is presented in Figure 17b. According to such figures, CMU requires small convolutional mask sizes to represent image patterns, indicating that small details can provide a good dataset representation.

Table 3 presents the CNN error rates considering the 5 best convolutional mask sizes estimated using IFNN for the three architectures. The influences of the mask dimensions can be analyzed by taking the 5 best sizes for a CNN with a single convolutional layer, that produced an average accuracy of 0.99 while classifying the person on the image; and an average accuracy of 0.95 for the presence of sunglasses. Those results confirm that IFNN estimates good parameters without label information, given IFNN is based on data content.

In case of MNIST, the IFNN results for the first convolutional layer are illustrated in Figures 18a and 18b, in which the average of fraction of false nearest neighbors and the histogram

[7] All source codes used to perform experiments are available at <https://github.com/marthadais/Estimate-Parameters> (FERREIRA *et al.*, 2018).



(a) Average of IFNN measurements. (b) Histogram of the best sizes.

Figure 17 – Results of the Image-based False Nearest Neighbors (IFNN) obtained for the CMU dataset: Figure (a) represents the average of the false nearest neighbors produced by IFNN; and Figure (b) corresponds to the histogram computed from the average of IFNN measurements, being used to estimate the most adequate convolutional mask size. In case of Figure (a), a logarithm function was applied on the average matrix $\bar{F}$ to improve visualization.

of the most adequate mask sizes are shown, respectively. Both figures allow to conclude that MNIST requires a greater convolutional mask size to extract relevant features when compared to CMU. Despite MNIST is less complex than others, image features contain greater patterns to be represented. In addition, the convolutional mask size 1×1 is highlighted by the histogram in some cases, what occurs because row 1 always indicates the column 1 as the best, and vice-versa, increasing the counting on the histogram. However, such mask size is never considered, given it is not adequate for the scenarios under analysis.

Table 4 summarizes the error rates obtained for the 5 best convolutional mask sizes (based on histograms) estimated with the IFNN method for the three convolutional layers. The average accuracy produced by CNN with a single convolutional layer was 0.98, confirming the estimation of an adequate convolutional mask size and number of units for MNIST. Additionally, this result allows us to conclude that simple CNN architectures can be designed using IFNN.

Figures 19a to 19b illustrate the IFNN results obtained with the CIFAR-10 dataset, in which Figure 19a shows the average matrix $\bar{F}$ and, Figure 19b, the counting histogram of the best convolutional mask sizes. Both figures comprise the results for the first convolutional layer, given the IFNN assessed the images from the training set. In this case, small convolutional mask sizes are enough to detect features for most of the images from CIFAR-10, however few images require too large sizes to perform an adequate feature extraction, influencing the estimation of average sizes.

The 5 best convolutional mask sizes were selected based on IFNN, in which the error rates are summarized in Table 5 as convolutional layers were included. The average accuracy provided by the 5 best mask sizes for a single convolutional layer architecture was 0.56, which

Table 3 – Experimental results obtained for the CMU dataset, considering two classification tasks: i) the corresponding person, and ii) the use of sunglasses. Error rates (column ER) were obtained using CNNs architectures, as convolutional layers were included. The mask sizes (column Mask) and the number of convolutional units (column Units) provided by IFNN were used in those experiments. CNN was trained with the SGD along $10,000$ iterations.

CMU dataset								
1°Conv. Layer			2°Conv. Layer			3°Conv. Layer		
Mask	Units	ER	Mask	Units	ER	Mask	Units	ER
8×1	13	1.08%	$\mathbf{9 \times 1}$	**12**	**1.08%**	4×2	20	3.26%
$\mathbf{5 \times 2}$	**13**	**0.54%**	7×2	10	1.62%	6×4	5	2.16%
4×4	10	1.09%	5×3	14	1.08%	2×6	7	2.70%
3×5	11	1.09%	3×4	10	1.62%	6×1	14	3.82%
2×7	9	1.09%	2×7	8	2.16%	$\mathbf{3 \times 2}$	**29**	**1.62%**

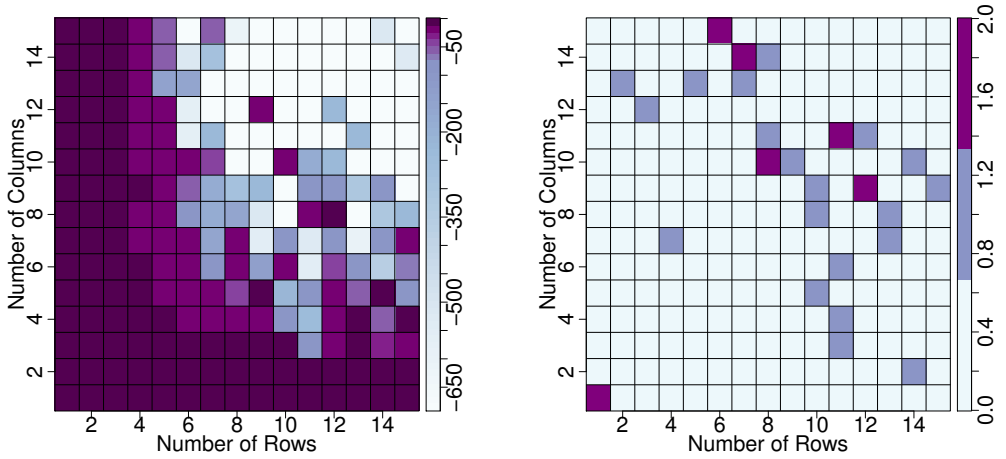
CMU dataset (eyes)								
1°Conv. Layer			2°Conv. Layer			3°Conv. Layer		
Mask	Units	ER	Mask	Units	ER	Mask	Units	ER
8×1	13	6.52%	8×4	8	2.72%	4×3	9	4.86%
5×2	13	4.89%	7×5	7	2.70%	$\mathbf{3 \times 4}$	**8**	**4.34%**
4×4	10	7.60%	6×7	8	3.78%	5×5	6	4.88%
3×5	11	4.35%	$\mathbf{4 \times 9}$	**7**	**2.16%**	2×6	8	4.88%
$\mathbf{2 \times 7}$	**9**	**4.34%**	10×1	12	2.18%	6×1	18	4.34%

may indicate that CIFAR-10 requires more layers and iterations. It is worth to mention that individual images provided significant differences of the best convolutional mask size, leading the IFNN to estimate the most adequate size to represent dataset patterns.

Next, COIL-100 was assessed, whose results are illustrated in Figures 20a and 20b. It required large convolutional mask sizes to extract relevant features in order to perform classification.

The error rates for the 5 best convolutional mask sizes for three convolutional layers are presented in Table 6. The average accuracy obtained was 0.99, while considering a single convolutional and a max-pooling layer, indicating that IFNN supports the estimation of parameters of convolutional layers and the design of simpler CNN architectures, while dealing with object recognition.

Finally, Figures 21a and 21b show the results for the GTSRB dataset while considering the first convolutional layer, in which the average matrix $\bar{F}$ and its histogram are provided. According to both figures, the convolutional mask size estimated is not as small as for CMU nor as large as for MNIST. In addition, GTSRB images present a small variation of the most adequate mask sizes, differently from CIFAR-10.



(a) Average of IFNN measurements. (b) Histogram of the best sizes.

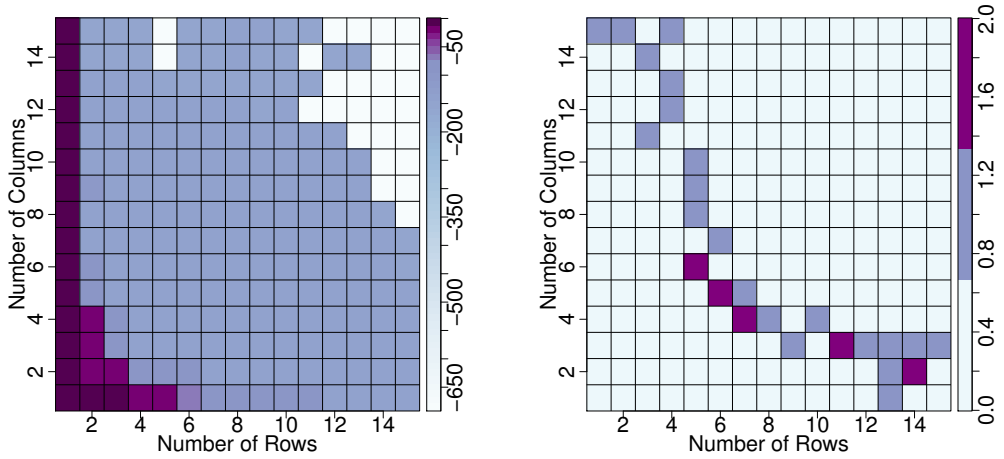
Figure 18 – Results of the Image-based False Nearest Neighbors (IFNN) for the MNIST dataset: Figure (a) represents the average of the false nearest neighbors produced by IFNN; and Figure (b) corresponds to the histogram computed from the average of IFNN measurements, being used to estimate the most adequate convolutional mask size. In case of Figure(a), a logarithm function was applied on the average matrix $\bar{F}$ to improve visualization.

Table 4 – Experimental results obtained for the MNIST dataset. Error rates (column ER) were obtained using CNNs architectures, as convolutional layers were included. The mask sizes (column Mask) and the number of convolutional units (column Units) provided by IFNN were used in those experiments. CNN was trained with SGD along $10,000$ iterations.

MNIST dataset								
1°Conv. Layer			2°Conv. Layer			3°Conv. Layer		
Mask	Units	ER	Mask	Units	ER	Mask	Units	ER
12×9	7	2.18%	8×4	8	2.76%	6×3	5	2.86%
$\mathbf{8 \times 10}$	**9**	**1.85%**	7×5	7	2.10%	5×4	5	3.02%
11×11	7	1.87%	$\mathbf{6 \times 6}$	**7**	**1.96%**	2×5	6	2.92%
7×14	8	2.11%	5×8	7	2.36%	$\mathbf{3 \times 6}$	**6**	**2.84%**
6×15	7	2.13%	10×1	12	2.36%	6×1	5	3.10%

Table 7 summarizes the error rates of the GTSRB dataset, obtained for the 5 best convolutional mask sizes estimated using the IFNN. The average accuracy provided by the CNN with a single convolutional layer was 0.83, confirming that IFNN provides an adequate convolutional mask size and number of units, given a shallow CNN was used.

Based on such experiments, we conclude that a small amount of data (around 200 images) was enough to feed IFNN while assessing the convolutional mask sizes and the number of units. Also, IFNN does not require a full network training process in order to provide a fair estimation for those parameters and, then, to build up the next convolutional layer. Additionally, as IFNN is based on data content, it is independent of the labels to provide adequate parameters, what is confirmed after the CMU classification results that produced lower error rates when two types of



(a) Average of IFNN measurements. (b) Histogram of the best sizes.

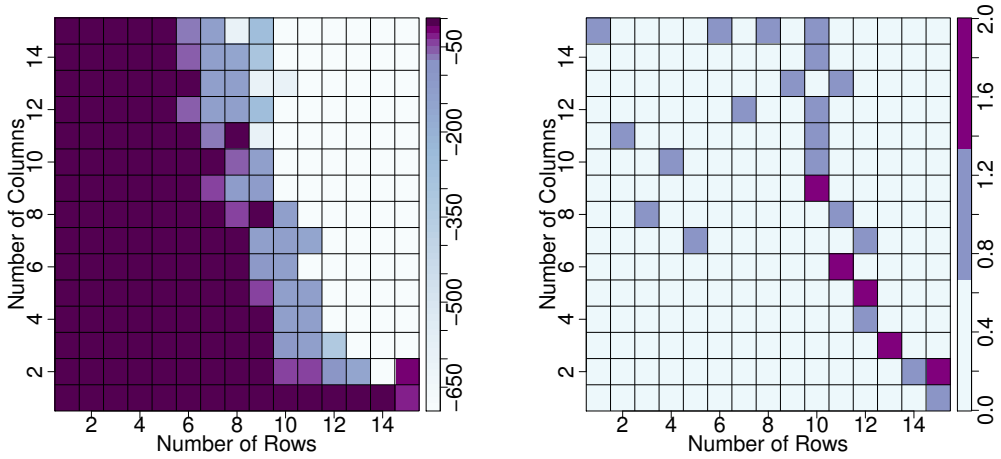
Figure 19 – Results of the Image-based False Nearest Neighbors (IFNN) obtained for the CIFAR-10 dataset: Figure (a) represents the average of the false nearest neighbors produced by IFNN; and Figure (b) corresponds to the histogram computed from the average of IFNN measurements, being used to estimate the most adequate convolutional mask size. In case of Figure(a), a logarithm function was applied on the average matrix $\bar{F}$ to improve visualization.

Table 5 – Experimental results obtained for the CIFAR-10 dataset. Error rates (column ER) were obtained using CNNs architectures, as convolutional layers were included. The mask sizes (column Mask) and the number of convolutional units (column Units) provided by IFNN were used in those experiments. CNN was trained with SGD along 10,000 iterations.

CIFAR-10 dataset								
1ºConv. Layer			2ºConv. Layer			3ºConv. Layer		
Mask	Units	ER	Mask	Units	ER	Mask	Units	ER
14×2	11	49.49%	9×2	11	40.12%	3×3	10	39.50%
11×3	14	44.45%	3×4	14	38.76%	$\mathbf{5 \times 4}$	**9**	**38.10%**
7×4	17	43.39%	2×6	17	40.06%	4×5	6	39.80%
$\mathbf{6 \times 5}$	**23**	**41.82%**	$\mathbf{10 \times 10}$	**23**	**37.38%**	6×6	5	39.60%
5×6	23	42.66%	10×1	23	40.66%	6×1	11	38.36%

labels (person and eyes) were employed.

This analysis indicates that the space reconstruction described in Section 3.2.1 impacts the classification performed by CNN, in which the image region to be convolved is defined by a convolutional mask size that is not necessarily square (the same number of rows and columns) as it is typically used in literature (KRIZHEVSKY; SUTSKEVER; HINTON, 2012; SRIVASTAVA; SALAKHUTDINOV; HINTON, 2013; HE *et al.*, 2016). In fact, each image presents a given variation about the most adequate convolutional mask size, showing that the dimension to reconstruct the phase space varies according to those attributes. In addition, the influences of the convolutional mask sizes on classification accuracies was observed, in which adequate settings can lead to a good classification and support the design of simpler but efficient



(a) Average of IFNN measurements. (b) Histogram of the best sizes.

Figure 20 – Results of the Image-based False Nearest Neighbors (IFNN) obtained for the COIL-100 dataset: Figure (a) represents the average of the false nearest neighbors produced by IFNN; and Figure (b) corresponds to the histogram computed from the average of IFNN measurements, being used to estimate the most adequate convolutional mask size. In case of Figure(a), a logarithm function was applied on the average matrix $\bar{F}$ to improve visualization.

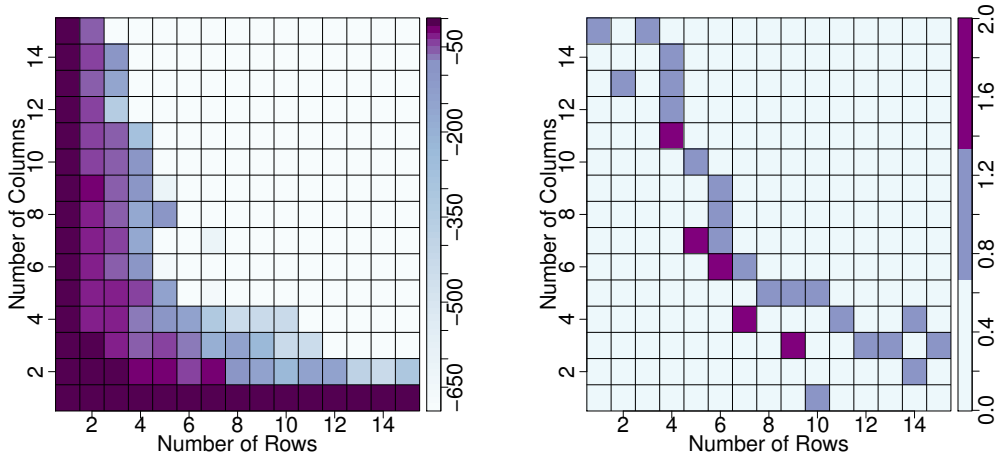
Table 6 – Experimental results obtained for the COIL-100 dataset. Error rates (column ER) were obtained using CNNs architectures, as convolutional layers were included. The mask sizes (column Mask) and the number of convolutional units (column Units) provided by IFNN were used in those experiments. CNN was trained with SGD along 10,000 iterations.

COIL-100 dataset								
1°Conv. Layer			2°Conv. Layer			3°Conv. Layer		
Mask	Units	ER	Mask	Units	ER	Mask	Units	ER
15×2	13	0.72%	6×3	13	0.50%	3×3	8	0.82%
13×3	15	0.58%	5×7	17	0.48%	2×4	11	0.56%
12×5	14	0.54%	$\mathbf{10 \times 1}$	**36**	**0.30%**	$\mathbf{5 \times 1}$	**26**	**0.48%**
$\mathbf{11 \times 6}$	**9**	**0.32%**	9×2	13	0.44%	4×2	14	0.48%
10×9	21	0.38%	7×3	17	0.62%	6×3	8	0.96%

CNN architectures (more information in Section 4.3).

5.4 Literature Comparison

From the evaluation conducted in Section 5.3, the convolutional mask sizes and the correspondent number of units that provided the lowest error rates were selected to perform the CNN training using the SGD method along 50,000 iterations, significantly improving accuracies. The CNN architectures were the same as in Section 5.3, containing consecutive layers of convolution operation with ReLU and max-pooling with size 3×3 and stride of 2×2. The parameters employed at the training stage were also the same as described in Section 5.3, which



(a) Average of IFNN measurements. (b) Histogram of the best sizes.

Figure 21 – Results of the Image-based False Nearest Neighbors (IFNN) obtained for the GTSRB dataset: Figure (a) represents the average of the false nearest neighbors produced by IFNN; and Figure (b) corresponds to the histogram computed from the average of IFNN measurements, being used to estimate the most adequate convolutional mask size. In case of Figure(a), a logarithm function was applied on the average matrix $\bar{F}$ to improve visualization.

Table 7 – Experimental results obtained for the GTSRB dataset. Error rates (column ER) were obtained using CNNs architectures, as convolutional layers were included. The mask sizes (column Mask) and the number of convolutional units (column Units) provided by IFNN were used in those experiments. CNN was trained with SGD along $10,000$ iterations.

GTSRB dataset								
1ºConv. Layer			2ºConv. Layer			3ºConv. Layer		
Mask	Units	ER	Mask	Units	ER	Mask	Units	ER
9×3	14	16.86%	9×1	47	15.26%	4×2	31	15.38%
7×4	22	16.40%	3×5	23	14.96%	$\mathbf{6 \times 4}$	**9**	**14.74%**
$\mathbf{6 \times 6}$	**26**	**16.21%**	4×6	16	15.62%	6×1	21	19.04%
5×7	23	16.83%	5×7	18	16.18%	5×2	24	15.78%
4×11	22	17.14%	$\mathbf{7 \times 2}$	**22**	**14.50%**	3×3	8	16.46%

had already provided good classification results. Table 8 presents the error rates obtained for the five datasets considering more training iterations, showing the results for the three architectures evaluated as convolutional and max-pooling layers were added.

In case of the CMU dataset, the classification based on people identification does not improve neither with a second nor with a third convolutional layer, indicating that a single layer is enough to extract relevant features. On the other hand, the classification based on the eyes (wearing sunglasses) presented some improvements when a second convolutional layer was included. However, the third layer significantly increased the error rate, confirming such a layer is unnecessary and that it leads to overfitting. This very same behavior was observed for the CIFAR-10 dataset, in which the third layer worsened the classification results.

Table 8 – Experimental results obtained for the five datasets, using the convolutional mask sizes (Mask) and the number of convolutional units (Units) provided by the IFNN method (see Section 5.3). Error rates (ER) were computed using the Caffe deep learning framework trained with SGD along 50,000 iterations.

Datasets	1°Conv. Layer			2°Conv. Layer			3°Conv. Layer		
	Mask	Units	ER	Mask	Units	ER	Mask	Units	ER
CMU	$\mathbf{5 \times 2}$	**13**	**0.54%**	9×1	12	1.08%	3×2	29	1.64%
CMU (eyes)	2×7	9	3.80%	$\mathbf{4 \times 9}$	**7**	**1.62%**	3×4	8	6.50%
MNIST	8×10	9	1.96%	6×6	7	1.78%	$\mathbf{3 \times 6}$	**6**	**1.74%**
CIFAR-10	6×5	23	34.00%	$\mathbf{10 \times 10}$	**23**	**28.44%**	5×4	9	30.22%
COIL-100	11×6	9	0.38%	$\mathbf{10 \times 1}$	**36**	**0.28%**	5×1	26	0.28%
GTSRB	6×6	26	13.02%	7×2	22	12.58%	$\mathbf{6 \times 4}$	**9**	**11.78%**

The CIFAR-10 results were significantly improved when a second layer was included, confirming the features require a deeper architecture as well as more training iterations. However, given a third layer increased its error rate, we believe that indicates CIFAR-10 requires different types of layers to lead to a better generalization and classification performance. On the other hand, the additional convolutional layers improved the results obtained for MNIST and GTSRB, confirming they depend on deeper architectures to extract relevant features and reach a better learning convergence.

Finally, COIL-100 presented a good error rate even with a single convolutional layer, which decreased after the addition of a second but it did not change after including a third layer. This behavior confirms that two convolutional layers are enough to provide good results without leading to overfitting. It is worth to mention that COIL-100 is a well-behaved dataset, which was collected in a controlled environment and also preprocessed color equalization. At the same object recognition scenario, CIFAR-10 was collected from Web without any preprocessing stage, which might improve the CNN results as well as for GTSRB.

In this context, the same preprocessing applied on COIL-100 was used on CIFAR-10 and GTSRB, thus normalizing color intensities. Table 9 presents the error rates obtained for the preprocessed versions of CIFAR-10 and GTSRB with the same CNN architectures along 50,000 iterations. For CIFAR-10, the preprocessing stage increased the error rate in 16.49%, showing the color equalization is not adequate for it. However, for GTSRB, the error rate decreased in approximately 10%, confirming the color equalization may help to improve classification results depending on the target task. This was indeed expected because the original GTSRB images contain a bad contrast (too much light or dark images). In addition, the preprocessing stage reduced the network complexity for GTSRB, due to a single-layer CNN produced the lowest error rate, in comparison with the scenario without preprocessing that required 3 convolutional layers.

Table 9 – Experimental results obtained for CIFAR-10 and GTSRB after a preprocessing, using the convolutional mask sizes (Mask) and the number of convolutional units (Units) provided by the IFNN method (see Section 5.3). Error rates (ER) were computed using the Caffe deep learning framework trained with SGD along 50,000 iterations.

Datasets	1°Conv. Layer			2°Conv. Layer			3°Conv. Layer		
	Mask	Units	ER	Mask	Units	ER	Mask	Units	ER
CIFAR-10	6×5	23	35.79%	10×10	23	30.22%	5×4	9	33.13%
GTSRB	6×6	26	10.52%	7×2	22	10.55%	6×4	9	11.68%

In summary, the CNN classification results after IFNN parametrizations are very similar to the ones reported in the literature, confirming that our approach can support the design of simpler but still efficient CNN architectures. Table 10 shows the error rates reported in the literature (approaches are described in Section 5.2.1), and the best ones obtained by IFNN (Table 8) for the five datasets under analysis.

In this context, deep and complex CNNs were employed in the literature to obtain lowest error rates without enough justification for such empirical parametrization, while good accuracies can also be reach with simple CNNs architecture under a good and justified parametrization. CMU experiments produced similar results as found in the literature, confirming that IFNN can support the design of simple CNNs. For such dataset, two different classification tasks were performed: i) the person in the image (person); and ii) the presence of sunglasses (eyes).

Considering the first task (person), Teller and Veloso (1995) reached an error rate of 8%, while a single convolutional layer configured with IFNN produced an error rate of 0.55%. For the second classification task (eyes), a CNN composed of a single convolutional layer configured with IFNN provided 4.32%, which is lower than 5.68% reported by Zhu *et al.* (2016). Those results confirm that IFNN provides adequate estimations for convolutional layers. As IFNN attempts to identify the best linear transformation in order to extract relevant features based on its neighborhood (Section 3.2), being also capable of supporting different classifications tasks.

In case of MNIST, CNN with IFNN support achieved an error rate equals to 1.36% for a single convolutional layer, 1.09% for two convolutional layers, and 1.22% for three convolutional layers. All those results are inferior when compared with 0.23%, which is the lowest (best) error rate reported in the literature (CIREGAN; MEIER; SCHMIDHUBER, 2012). Despite such error rate, we believe our result is still relevant and close to others, such as Hinton and Salakhutdinov (2006) that reached 1.25%, confirming again that IFNN provides an adequate estimation to design simpler CNN architectures. Besides our approach did not outperform the state-of-the-art results, it is worth to remind that other architectures are much deeper and, therefore, more complex than the ones we found using IFNN. For example, the MCDNN used by Ciregan, Meier and Schmidhuber (2012), which is composed of 35 CNNs, and the DBN applied by Hinton and

Table 10 – Summary of error rates for the five dataset considered throughout experiments (sorted by error rates). They were obtained with CNNs parametrized using IFNN and the ones reported in the literature, including authors and their techniques. in an attempt to simplify the notation, just the best result obtained by IFNN is presented (CNN+IFNN), in which the architecture with a single convolutional layer is represent by 1C, two convolutional layers by 2C, and three convolutional layers by 3C. Results involving the image preprocessing stage (PP) were included.

Dataset	Proposal	Error Rate
CMU Face Images	**CNN+IFNN (1C)**	**0.54%**
	Teller and Veloso (1995) (Genetic Program)	8.00%
CMU Face Images (eyes)	**CNN+IFNN (2C)**	**1.62%**
	Zhu *et al.* (2016) (CNN)	5.68%
MNIST	Ciregan, Meier and Schmidhuber (2012) (MCDNN)	0.23%
	Goodfellow *et al.* (2013) (Maxout + Dropout)	0.45%
	LeCun *et al.* (1999) (LeNet-4 + distorted data)	0.70%
	Srivastava *et al.* (2014) (DBN + Dropout)	0.79%
	LeCun *et al.* (1999) (LeNet-5 + distorted data)	0.80%
	LeCun *et al.* (1999) (LeNet-5)	0.95%
	Hinton and Salakhutdinov (2006) (DBN)	1.25%
	CNN+IFNN (3C)	**1.74%**
CIFAR-10	Goodfellow *et al.* (2013) (Maxout + Dropout)	9.38%
	Ciregan, Meier and Schmidhuber (2012) (MCDNN)	11.21%
	Srivastava *et al.* (2014) (CNN + Dropout)	12.61%
	Coates and Ng (2011) (3-CNN + SVM)	18.00%
	Coates and Ng (2011) (1-CNN + SVM)	19.40%
	Krizhevsky (2010) (DBN)	21.10%
	CNN+IFNN (2C)	**28.44%**
	PP+CNN+IFNN (3C)	**33.13%**
COIL-100	**CNN+IFNN (2C)**	**0.28%**
	Hassairi, Ejbali and Zaied (2015) (DCWN)	0.8%
	Yang, Roth and Ahuja (2000) (SNoW)	7.69%
	Zou *et al.* (2012) (NN + PCA)	13%
GTSRB	Ciregan, Meier and Schmidhuber (2012) (MCDNN)	0.54%
	Sermanet and LeCun (2011) (MS-CNN)	0.83%
	PP+CNN+IFNN (1C)	**10.52%**
	CNN+IFNN (3C)	**11.78%**

Salakhutdinov (2006), composed of 5 layers with more than 500 units (see Section 5.2.1).

In the same direction, a Deep Convolutional Wavelet Network (DCWN) was employed by Hassairi, Ejbali and Zaied (2015), reporting an error rate equals to 0.8% for the COIL-100 dataset. The DCWN architecture was built up using a more complex configuration in conjunction with wavelet layers. On the other hand, IFNN estimation provided 0.28% as error rate with a CNN composed of a single convolutional and max-pooling layers, significantly improving the classification results and again confirming IFNN supports the detection of common image patterns in order to provide the network estimation also for object recognition tasks.

The CIFAR-10 dataset also involves the task of object recognition, but it is more complex, since it was not created under a controlled environment such as COIL-100. In addition, it contains colorful backgrounds. The CNN with two convolutional layers configured with IFNN achieved an error rate of 28.44%, while the best error rate reported on literature is 9.38% (GOODFELLOW *et al.*, 2013) in which a CNN composed of 2 layers with $1,200$ units is used. Based on those results, it is possible to observe that CIFAR-10 requires a more complex model to extract more relevant features. On the other hand, our result is still very close to others reported in the literature, such as 21.10% obtained by Krizhevsky (2010), in which a DBN with 2 layers and $1,024$ units each is used.

In the context of GTSRB, the lowest (best) error rate provided on literature is 0.54% by Ciregan, Meier and Schmidhuber (2012), while a 3-convolutional-layer CNN configured with IFNN produced an error rate equals to 11.78%. Despite the unsatisfactory results achieved for CIFAR-10 and GTSRB, our classification performances are still good when considering the usage of simple CNN architectures, indicating that IFNN supports the estimation of the convolutional mask sizes and the number of units per layer.

Finally, those results confirm that a well parametrized shallow CNN architectures can lead to satisfactory classification performances even for different application domains. In summary, according to the experiments, IFNN provided adequate parameters, showing it is possible to reach good accuracies with a simpler CNN architecture, saving processing time and computer resources. In addition, shallow networks provide stronger convergence guarantees, since they present smaller Shattering coefficients, therefore requiring less input examples to ensure learning bounds (see Chapter 6).

5.5 Final Considerations

In this chapter, we reported experimental results to support the assessment of the IFNN method proposed in this PhD thesis. IFNN was designed to support the estimation of the convolutional mask sizes and the number of units per CNN layer. Five datasets were analyzed, including handwritten digits, face detection, and the object recognition task. The state-of-the-art methods from the literature were compared as baseline to our results. Besides some unsatisfactory results, IFNN has confirmed a good parameter estimation for the CNN convolutional layers, independently on the label or the problem domain. That confirms IFNN supports the design of simple and shallow CNNs, producing low as possible error rates, many of them comparable to the results reported in the literature, which were mostly based on deeper and, therefore, more complex networks. Moreover, it is worth to mention that IFNN was also employed to design a CNN architecture to deal with the speech recognition task, bringing relevant improvements to the classification of phonemes as reported in Shulby *et al.* (2017).

CHAPTER

6

ON THE CONVERGENCE OF CONVOLUTIONAL NEURAL NETWORKS

6.1 Initial Considerations

The demonstration of learning convergence of Convolutional Neural Networks (CNN) was formulated based on the Statistical Learning Theory (SLT) (LUXBURG; SCHÖLKOPF, 2011; MELLO; PONTI, 2018), supporting to estimate the minimal sample sizes required to train different architectures. This is specially interesting and relevant in the context of this thesis, given we intend to design simpler CNN architectures with the support of the Image-based False Nearest Neighbors (IFNN) method (FERREIRA *et al.*, 2018). At first, we introduce the main concepts of the SLT in this chapter, then formulate the Shattering coefficient to characterize the bias imposed by architectures/algorithms, i.e. the complexity of the space of admissible functions (MELLO; PONTI; FERREIRA, 2018).

6.2 Statistic Learning Theory

The Statistical Learning Theory (SLT) provides the theoretical foundation for supervised machine learning whose main objective is to induce some classification/regression function $f : \mathscr{X} \rightarrow \mathscr{Y}$, given training pairs $(x_i, y_i) \in (\mathscr{X}, \mathscr{Y})$ are independently and identically sampled from the joint probability distribution $P(\mathscr{X} \times \mathscr{Y})$. On the other hand, the unsupervised learning scenario considers input data without the labeling information, so that they are partitioned (clustering) according to space similarities. For more details on theoretical frameworks for this second branch, please refer to (CARLSSON; MÃŠMOLI, 2010).

In order to proceed with the theoretical guarantees for supervised learning, SLT assumes the following: i) the joint probability distribution $P(\mathscr{X} \times \mathscr{Y})$ has no restriction, differing from other theories (ROSENBLATT, 1956) that require P to belong to a specific family (e.g. Normal,

Poisson, etc.); ii) data may assume nondeterministic labels, i.e. two equal examples $x_i = x_j$ can have different labels due to noise imposed by supervision; iii) examples must be independent from each other, therefore x_i does not influence in the probability of sampling a next element $x_{i+n}, \forall n$; iv) P must be fixed/static, thus it cannot change along time [1]; and, finally, v) P is unknown at the training stage, therefore the only way of estimating it is by accessing samples from such distribution.

In this context, a supervised algorithm must induce some function f that minimizes the classification error for a specific learning task. As such algorithm may produce any classifier in some space $\mathscr{F}$, here referred to as algorithm bias, it is necessary some measure to evaluate their relative performances. A loss function $\ell(\cdot)$ is then used to quantify the classification error for some example. Equation 6.1 defines the squared-error loss function (the most typical), in which x_i corresponds to the example (its attributes), y_i is the expected (correct) label, and $f(x_i)$ is a classifier/regressor that outputs the label information for x_i.

$$\ell(x_i, y_i, f(x_i)) = (y_i - f(x_i))^2 \tag{6.1}$$

In addition, the loss function is used to compute the expected risk $R(f)$ for supervised learning algorithms, as defined in Equation 6.2, considering every possible input in $\mathscr{X}$ and its respective output in $\mathscr{Y}$. Observe this equation requires full knowledge about the joint probability distribution $P(\mathscr{X} \times \mathscr{Y})$, consequently it can only be computed when both sets $\mathscr{X}$ and $\mathscr{Y}$ have a finite number of possibilities. However, this is not the situation found in real-world scenarios, specially when example attributes are defined in terms of the continuity, such as in the real line $\mathbb{R}$.

$$R(f) = \mathbf{E}(\ell(\mathscr{X}, \mathscr{Y}, f(\mathscr{X})) \tag{6.2}$$

Given $R(f)$ is not computable for real-world problems, SLT can somehow rely on the empirical risk $R_{\text{emp}}(f)$ to ensure learning bounds, since this last term only requires some sample $z_n = \{(x_1, y_1), \ldots, (x_n, y_n\}) \in \mathscr{X} \times \mathscr{Y}$, as defined in Equation 6.3.

$$R_{\text{emp}}(f) = \frac{1}{n} \sum_{i=1}^{n} \ell(x_i, y_i, f(x_i)) \tag{6.3}$$

Therefore, the main goal of SLT is to ensure that any function $f : \mathscr{X} \to \mathscr{Y}$ has the empirical risk $R_{\text{emp}}(f)$ as a good estimator of the expected risk $R(f)$, so that one can select the best $f \in \mathscr{F}$ (Equation 6.4) by simply assessing a computable quantity what is referred to as the Empirical Risk Minimization Principle (ERMP) (LUXBURG; SCHÖLKOPF, 2011).

[1] This is a very important assumption to ensure learning, besides several studies disregard it while addressing data samples from time series, data streams and other time-dependent scenarios (MELLO; PONTI, 2018).

However, ERMP can be inconsistent, depending on the learning algorithm used. For example, considering a learning algorithm that just memorizes the training set and assigns random values to unseen examples, the obtained empirical risk $R_{\text{emp}}(f)$ will be zero, since all examples were memorized. In the meantime, the expected risk $R(f)$ will be large, because the algorithm has a random behavior for unseen examples. In this case, ERMP is inconsistent due to the empirical risk $R_{\text{emp}}(f)$ does not converge to the expected risk $R(f)$ when n is large enough.

$$f_n = \arg\min_{f \in \mathscr{F}} R_{\text{emp}}(f) \tag{6.4}$$

One way to avoid the ERMP inconsistency is by checking if the set of all classifiers in $\mathscr{F}$ reduces the difference ε between $R_{\text{emp}}(f)$ and $R(f)$ while n tends to infinity, as defined in Equation 6.5.

$$|R_{\text{emp}}(f) - R(f)| > \varepsilon \tag{6.5}$$

ERMP is formulated on top of the Law of Large Numbers (LLN) defined in Equation 6.6, which ensures the average value of any independent and identically distributed random variable ξ_i sampled from any (fixed) probability distribution P converges to the expected value as the sample size tends to infinite. The need for some independent and identically distributed random variable comes directly from the LLN, while the static/fixed joint probability distribution is necessary to ensure convergence as samples are collected. The remaining assumptions were defined to support general-purpose learning (VAPNIK, 2000).

In sequence, Equation 6.7 is obtained with the application of LLN on top of the empirical risk $R_{\text{emp}}(f)$, in which the loss function $\ell(\cdot)$ replaces the random variable ξ of Equation 6.6.

$$\frac{1}{n}\sum_{i=1}^{n} \xi_i \rightarrow \mathbf{E}(\xi) \text{ for } n \rightarrow \infty \tag{6.6}$$

$$R_{\text{emp}}(f) = \frac{1}{n}\sum_{i=1}^{n} \ell(x_i, y_i, f(x_i)) \rightarrow \mathbf{E}(l(x, y, f(x))) = R(f) \text{ for } n \rightarrow \infty \tag{6.7}$$

It is widely known that LLN is upper bounded by Chernoff's inequality (Equation 6.8), which is used to provide a measure of how much the empirical mean approximates its expected value (CHERNOFF, 1952; HOEFFDING, 1963; LUXBURG; SCHÖLKOPF, 2011). In the same sense, Vapnik (2000) took advantage of such bound (Equation 6.9) to measure how close the empirical risk $R_{\text{emp}}(f)$ is from the expected risk $R(f)$, allowing to take theoretical learning conclusions.

$$P\left(\left|\frac{1}{n}\sum_{i=1}^{n} \xi_i - \mathbf{E}(\xi)\right| > \varepsilon\right) \leq e^{-2n\varepsilon^2} \tag{6.8}$$

$$P\left(\left|R_{\text{emp}}(f) - R(f)\right| > \varepsilon\right) \leq e^{-2n\varepsilon^2} \tag{6.9}$$

However, LLN only accepts a function f that is independently selected from the training sample, what does not happen in practical learning tasks. In order to solve this issue, Vapnik (2000) considered the worst possible classifier $f \in \mathscr{F}$ provided by some supervised algorithm, therefore obtaining Equation 6.10 in which $\sup_{f \in F}$ corresponds to the worst function in space $\mathscr{F}$ (the supreme measure the greatest divergence between functions $R_{\text{emp}}(f)$ and $R(f)$).

$$P(\sup_{f \in F} |R_{\text{emp}}(f) - R(f)| > \varepsilon) \leq e^{-2n\varepsilon^2} \tag{6.10}$$

In order to demonstrate this solution, consider that all classifiers in $\mathscr{F}$ are similar to the worst classifier, so that:

$$\begin{gathered} P(|R_{\text{emp}}(f_1) - R(f_1)| > \varepsilon) \leq 2e^{-n\varepsilon^2} \\ P(|R_{\text{emp}}(f_2) - R(f_2)| > \varepsilon) \leq 2e^{-n\varepsilon^2} \\ \vdots \\ P(|R_{\text{emp}}(f_m) - R(f_m)| > \varepsilon) \leq 2e^{-n\varepsilon^2}, \end{gathered}$$

in which m is the total number of classifiers. From this system we obtain Equation 6.11 (referred to as the union bound due to the or operators), from which we derive Equation 6.12 to have the sum of all terms in the right side.

$$\begin{gathered} P(|R_{\text{emp}}(f_1) - R(f_1)| > \varepsilon \vee |R_{\text{emp}}(f_2) - R(f_2)| > \varepsilon \vee \cdots \\ \vee |R_{\text{emp}}({}_f m) - R(f_m)| > \varepsilon) \leq \sum_{i=1}^{m} P(|R_{\text{emp}}(f_i) - R(f_i)| > \varepsilon) \end{gathered} \tag{6.11}$$

$$\begin{aligned} \sum_{i=1}^{m} P(|R_{\text{emp}}(f_i) - R(f_i)| > \varepsilon) &\leq \sum_{i=1}^{m} 2e^{-n\varepsilon^2} \\ \sum_{i=1}^{m} P(|R_{\text{emp}}(f_i) - R(f_i)| > \varepsilon) &\leq 2me^{-n\varepsilon^2} \end{aligned} \tag{6.12}$$

Consequently, Equation 6.13 is a natural result of the previous ones, having m as the cardinality of the space of admissible functions $\mathscr{F}$ as the sample size n is increased, i.e., the number of classifiers in the algorithm bias, which is referred to as the Shattering coefficient (or growth function) (VAPNIK; CHERVONENKIS, 2015; MELLO; PONTI, 2018).

$$P(\sup_{f \in \mathscr{F}} |R_{\text{emp}}(f) - R(f)| > \varepsilon) \leq \sum_{i=1}^{m} P(|R_{\text{emp}}(f_i) - R(f_i)| > \varepsilon) \leq 2me^{-n\varepsilon^2} \tag{6.13}$$

From Equation 6.13, Vapnik (2000) formalized the uniform convergence theorem (Equation 6.14), which is a necessary and enough to provide consistency to the ERMP (VAPNIK; CHERVONENKIS, 2015; VAPNIK; CHERVONENKIS, 1974). From that, we conclude that whenever the algorithm bias $\mathscr{F}$ is excessively large, such convergence is more difficult to be satisfied, what is inverse for smaller spaces of functions. In such a context, function space properties were taken into account to ensure the convergence of both risks formalized in Equation 6.14 (VAPNIK; CHERVONENKIS, 2015; VAPNIK; CHERVONENKIS, 1974; LUXBURG; SCHÖLKOPF, 2011). In that sense, the uniform LLN is valid if and only if the number of functions m is finite and polynomial for a given sample size, producing an enough formalization for the Shattering coefficient.

$$P\left(\sup_{f\in\mathscr{F}}\left|R(f)-R_{\text{emp}}(f)\right|>\varepsilon\right)\rightarrow 0 \quad \text{for } n\rightarrow\infty, \forall\varepsilon>0. \tag{6.14}$$

As the expected risk $R(f)$ cannot be computed once it requires all possible examples for a given classification task, the symmetrization lemma was applied (DEVROYE; GYÖRFI; LUGOSI, 1997). Such a lemma employs a ghost sample, i.e. another sample $\{(x'_1,y'_1),\ldots,(x'_n,y'_n)\}$ from the same joint probability distribution $P(\mathscr{X}\times\mathscr{Y})$, to substitute the expected risk by another function is terms of an unknown sample (VAPNIK; CHERVONENKIS, 1974; LUXBURG; SCHÖLKOPF, 2011; VAPNIK, 2000). From that, Vapnik (2000) could use the LLN to represent the convergence of supervised learning algorithms as follows:

$$\begin{aligned}
2P(\sup_{f\in\mathscr{F}}|R_{\text{emp}}(f)-R(f)|-\sup_{f\in F}|R'_{\text{emp}}(f)-R(f)|>\varepsilon) &\leq 2\{2me^{-n\varepsilon^2}\}\\
2P(\sup_{f\in\mathscr{F}}|R_{emp}(f)-R(f)|-|R'_{emp}(f)-R(f)|>\varepsilon) &\leq 2\{2me^{-n\varepsilon^2}\}\\
2P(\sup_{f\in\mathscr{F}}R(f)-R_{\text{emp}}(f)-R(f)+R'_{\text{emp}}(f)>\varepsilon) &\leq 2\{2me^{-n\varepsilon^2}\}\\
P(\sup_{f\in\mathscr{F}}R'_{\text{emp}}(f)-R_{\text{emp}}(f)>\varepsilon/2) &\leq 2me^{-n(\varepsilon/2)^2}\\
P(\sup_{f\in\mathscr{F}}R'_{\text{emp}}(f)-R_{\text{emp}}(f)>\varepsilon/2) &\leq 2me^{-n\varepsilon^2/4},
\end{aligned}$$

in which $R_{\text{emp}}(f)\leq R(f)$ once the empirical risk is biased to the training sample so that it will always produce a lower or equal error measurement. From that, we summarize the union bound from Equation 6.11 as shown in Equation 6.15.

$$P\left(\sup_{f\in\mathscr{F}}\left|R(f)-R_{\text{emp}}(f)\right|>\varepsilon\right)\leq 2me^{-n\varepsilon^2/4} \tag{6.15}$$

From Equation 6.15 and the symmetrization lemma, Vapnik (2000) shows that $\mathscr{F}$ should be finite and restricted to two samples each of with size equals to n, so that the number of different functions should be smaller than 2^{2n}. Thus, the Shattering coefficient $\mathscr{N}(\mathscr{F},2n)$ was

defined to represent the grows in the cardinality of $\mathscr{F}$ given two samples with $2n$ examples in total, what corresponds to the number of functions that produce different classification results for a sample with $2n$ elements, as defined in Equation 6.16 ($z_n = \{(x_1, y_1), \ldots, (x_{2n}, y_{2n})\}$ refers to the examples in those two samples and x_i identifies every single example (set of attributes) from the input space $\mathscr{X}$).

$$\mathscr{N}(\mathscr{F}, 2n) = \max\{|\mathscr{F}_{Z_n}| \mid \{x_1, \ldots, x_{2n}\} \in \mathscr{X}\} \tag{6.16}$$

In practical terms, the Shattering coefficient informs us about the number of all possible (worst case) and different classifications some sample with size $2n$ has. For example, considering a binary classification whose labels are $+1$ and -1, it is possible to segment all those $2n$ inputs in at most 2^{2n} different ways. However, 2^{2n} different classifications is an unlimited bias which makes learning impossible, therefore the supervised algorithm must provide some bias whose corresponding growth function results in less than 2^{2n}.

Considering the union bound and the Chernoff bound, the uniform convergence bound was formulated by Vapnik (2000) as defined in Equation 6.17 (VAPNIK; CHERVONENKIS, 1974). ERMP is consistent if and only if the right side term of Equation 6.17 converges to zero as the sample size tends to infinite, making Equation 6.18 necessary and sufficient to ensure the consistency of ERMP (LUXBURG; SCHÖLKOPF, 2011; VAPNIK, 2000) (all steps of this formulation are provided in (MELLO; PONTI, 2018)).

$$P\left(\sup_{f \in \mathscr{F}} \left|R(f) - R_{\text{emp}}(f)\right| > \varepsilon\right) \leq 2\mathscr{N}(\mathscr{F}, 2n)e^{-n\varepsilon^2/4} \tag{6.17}$$

$$\lim_{n \to \infty} \frac{\log \mathscr{N}(\mathscr{F}, 2n)}{n} = 0 \tag{6.18}$$

Rewritten Equation 6.17, the generalization bound is obtained, which is defined in Equation 6.19, where δ is greater than zero and represents the right side of Equation 6.17, as show in Equation 6.20.

$$R(f) \leq R_{\text{Emp}}(f) + \sqrt{\frac{4}{n}(\log \mathscr{N}(\mathscr{F}, 2n)) - \log \delta} \tag{6.19}$$

$$\delta = 2\mathscr{N}(\mathscr{F}, 2n)e^{-n\varepsilon^2/4} \tag{6.20}$$

Based on the union convergence bound and the generalization bound, Vapnik (2000) formalized the concept of VC (Vapnik-Chervonenkis) dimension, which computes the greatest sample size for which an algorithm obtains 2^n different classifications (remember, this is for a binary problem) (VAPNIK; CHERVONENKIS, 2015; VAPNIK; CHERVONENKIS, 1974).

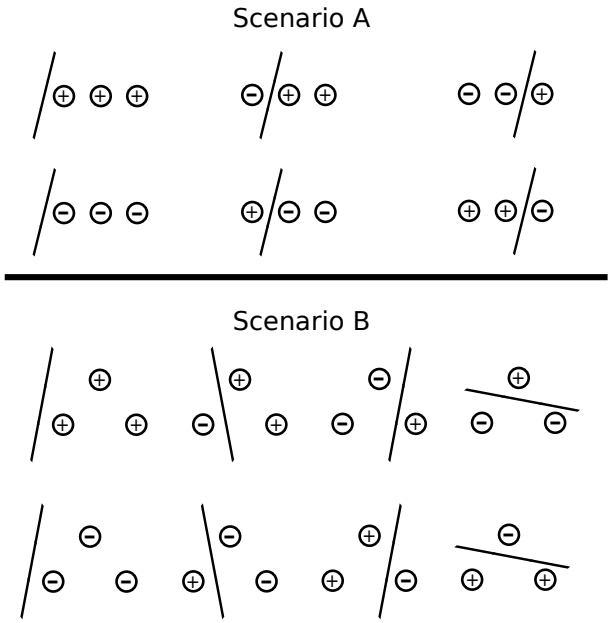


Figure 22 – Illustration of all possible classifications of three points in a 2 dimensional space considering a single hyperplane. In scenario A, points are collinear so that the single hyperplane is not enough to produce all 2^n different classifications, which is $2^3 = 8$. In scenario B, points are not collinear so that the single hyperplane provides all distinct classification possibilities.

In that sense, the VC dimension, shown in Equation 6.21, is seen as a capacity measure to the space of admissible functions $\mathscr{F}$, in which $|\mathscr{F}_{Z_n}|$ is the cardinality of $\mathscr{F}$ while classifying some specific sample $z_n = \{(x_1, y_1), \ldots, (x_n, y_n)\}$, or in other words, the number of functions in $\mathscr{F}$ that are distinguishable provided $\{x_1, \ldots, x_n\}$. If there is no maximum for such equation, the VC dimension is infinite. According to Vapnik (2000), the ERMP is consistent if and only if the VC dimension is finite (VAPNIK; CHERVONENKIS, 2015; VAPNIK; CHERVONENKIS, 1974).

$$VC(\mathscr{F}) = \max\{n \in \mathbb{N} \mid |\mathscr{F}_{z_n}| = 2^n \text{ for some } z_n\} \tag{6.21}$$

Figures 22 and 23 illustrate the Shattering coefficient, which is a polynomial growth function that defines the maximum number of different classifications for some sample, being here defined as n^h for some $h \geq 1$. Figure 22 presents possible classifications performed in some $\mathbb{R}^2$ space by having a single hyperplane, in which the scenario A has three collinear points, and scenario B has three noncollinear points. In case of scenario A, it is not possible separate the space in all possible ways as in scenario B that certainly contains 2^n different functions. From that we conclude that having $n = 3$ in such 2-dimensional space, the VC dimension is at least equals to 3.

Figure 23 considers a complementary scenario with four noncollinear points and a single hyperplane. The four points in such space $\mathbb{R}^2$ cannot be classified in all distinct 2^n possibilities with a single hyperplane, therefore the VC-dimension is equal to 3. Here, it is important to mention that if all distinct classifications are possible, the VC dimension is unlimited and no

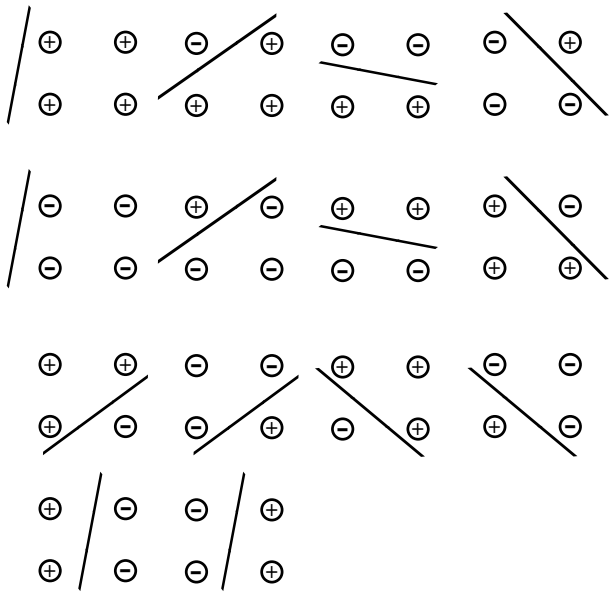


Figure 23 – Illustration of all possible classifications of four noncollinear points in a 2 dimensional space considering a single hyperplane. In this scenario, the hyperplane is not capable of producing all 2^n distinct classifications in form $2^4 = 16$, but 14 instead.

theoretical learning guarantee could be given (that is why the VC dimension is referred to as a capacity measure for the space $\mathscr{F}$).

The VC dimension supports a basic characterization for the growth behavior of the Shattering coefficient $\mathscr{N}(\mathscr{F},n)$ (n simply refers to a single data sample), as defined in Equation 6.22 in which d corresponds to the VC dimension of inputs in some Hilbert space $\mathscr{H}$ (VAPNIK; CHERVONENKIS, 2015; LUXBURG; SCHÖLKOPF, 2011; MELLO; PONTI; FERREIRA, 2018). According to Vapnik and Chervonenkis (2015), such equation is valid if $n > i$, otherwise the 2^n is used [2]. All those concepts have been used to formulate a convergence proof for deep neural networks as discussed in Mello, Ponti and Ferreira (2018), whose complexity is computed based on the Shattering coefficient (VAPNIK, 2000; LUXBURG; SCHÖLKOPF, 2011; MELLO; PONTI, 2018).

$$\mathscr{N}(\mathscr{F},n) \leq \sum_{i=0}^{d} \binom{n}{i} \tag{6.22}$$

6.3 Shattering Coefficient of Convolutional Neural Networks

The learning convergence of Convolutional Neural Networks (CNN) is here discussed and analyzed in terms of the Shattering coefficient. Our main motivation is to assess CNN

[2] For more information about the VC dimension and Sauer–Shelah's Lemma see Mello and Ponti (2018), Sauer (1972) and Vapnik and Chervonenkis (2015).

architectures estimated after the Image-based False Nearest Neighbors (IFNN) method and counterpose them to other solutions discussed in the literature. According to the algebraic formulation provided in Section 3.2.1, the first convolutional layer produces linear hyperplanes on the input space, corresponding to a linear transformation applied on the embedded space obtained from input images. The activation function ReLU is computed after the convolution operation in order to select the most relevant space regions for the next layer, which is typically a subsampling layer used to summarize features and reduce data dimensions.

In sequence, another convolutional layer is applied on the output space provided by the previous subsampling layer, producing further linear hyperplanes that create new half-spaces to perform ReLU and subsampling over again. All this process leads to space divisions that are directly related to the Shattering coefficient after the SLT (LUXBURG; SCHÖLKOPF, 2011; SCHÖLKOPF; SMOLA, 2002). Such a theoretical framework is responsible for ensuring learning on supervised scenarios (Section 6.2), having Equation 6.23 as a necessary condition to guarantee the ERMP consistency, in which the Shattering coefficient $\mathcal{N}(\mathscr{F},n)$ represents the maximum number of distinct classification functions as the sample size increases (always considering the worst possible data space organization (SAUER, 1972; MELLO; PONTI, 2018)).

$$\lim_{n\to\infty} \frac{\log \mathcal{N}(\mathscr{F},n)}{n} = 0 \tag{6.23}$$

The Shattering coefficient of a single unit in a convolutional CNN layer can be approximated through Equation 6.24 considering any d-dimensional Hilbert space $\mathscr{H}$ (VAPNIK; CHERVONENKIS, 2015; MELLO; PONTI; FERREIRA, 2018), in which d is the dimension of the input space and n is the sample size, i.e., the number of instances in the training set (see Section 6.2). Observe that, according to the algebraic formulation in Section 3.2.1, d refers to the mask size $p \times q$ of a convolutional layer (which is represented by $m \times n$ in Section 3.2.1), since it provides the embedding dimension to produce the input space for such layer, therefore leading to Equation 6.25.

$$\mathcal{N}(\mathscr{F},n) = 2\sum_{i=0}^{d} \binom{n}{i} \tag{6.24}$$

$$\mathcal{N}(\mathscr{F},n) = 2\sum_{i=0}^{p\times q} \binom{n}{i} \tag{6.25}$$

The Shattering coefficient of a k-unit convolutional layer is defined in Equation 6.26, having k as the number of hyperplanes (units) of a single convolutional layer. Such power of binomials corresponds to all possible combinations of the different hyperplanes in the same

d-dimensional Hilbert space $\mathscr{H}$ whose sample size is n.

$$\mathscr{N}(\mathscr{F},n) = 2\left(\sum_{i=0}^{p\times q}\binom{n}{i}^{k}\right) \tag{6.26}$$

Next, the Shattering coefficient of consecutive convolutional layers is formalized in terms of the product among the shattering terms of individual layers, as defined in Equation 6.27 in which L is the total number of convolutional layers, k_l corresponds to the number of units at layer l, and $p \times q_l$ represents the convolutional mask size at layer l and responsible for setting the input data dimension d (MELLO; FERREIRA; PONTI, 2017; MELLO; PONTI; FERREIRA, 2018).

$$\mathscr{N}(\mathscr{F},n) = \prod_{l=1}^{L}\left(2\sum_{i=0}^{p\times q_l}\binom{n}{i}^{k_l}\right) \tag{6.27}$$

According to this formulation, it is simple to observe that CNN architectures having less units and layers, also here referred as shallow architectures, converge faster in terms of Chernoff's and Hoeffding's bounds (DEVROYE; GYÖRFI; LUGOSI, 1997; VAPNIK, 2000; LUXBURG; SCHÖLKOPF, 2011). Equation 6.28 defines the necessary sample size to ensure learning convergence, in which $R(f)$ represents the expected risk or the expected value of error given some loss function in range $[0,1]$ and some Joint Probability Distribution $P(X \times Y)$ ($x \in X$ corresponds to an example and $y \in Y$ as its corresponding class); R_{emp} is the empirical risk or the average error produced by some loss function also in range $[0,1]$; Threshold $0 < \varepsilon < 1$ indicates an acceptable divergence limit between risks; $\mathscr{F}$ is the set of all admissible functions provided by a learning algorithm (a.k.a. algorithm bias); and, finally, n is the sample size.

$$P(\sup_{f\in\mathscr{F}}|R(f) - R_{\text{emp}}| > \varepsilon) \leq 2\mathscr{N}(\mathscr{F},n)e^{-n\varepsilon^2/4} \tag{6.28}$$

From this background, it is possible to analyze the convergence of a CNN architecture in terms of its convolutional units and number of training examples. Equation 6.29 defines the generalization probability, in which $P(\sup_{f\in\mathscr{F}}|R(f) - R_{\text{emp}}| > \varepsilon)$ corresponds to the probability that generalization does not occur for some supervised learning algorithm. As demonstrated by Vapnik (2000), generalization is ensured when $\mathscr{N}(\mathscr{F},n)e^{-n\varepsilon^2/4}$ converges to zero as n tends to infinite.

$$P(\sup_{f\in\mathscr{F}}|R(f) - R_{\text{emp}}| > \varepsilon) \leq 2\prod_{l=1}^{L}\left(2\sum_{i=0}^{p\times q_l}\binom{n}{i}^{k_l}\right)e^{-n\varepsilon^2/4} \tag{6.29}$$

As more units and layers are included in a CNN architecture, more examples are required to ensure learning to make the right-side term of Equation 6.29 approach zero. The main term responsible for the convergence behavior is the Shattering coefficient, which might be dominated

by the other Chernoff terms only after some enough sample size is provided thus leading to learning convergence. Based on such formulation (VAPNIK, 2000; MELLO; PONTI, 2018), it is possible to conclude that large and deep CNN architectures require greater sample sizes to ensure learning bounds, and, consequently, to produce good classification results.

6.3.1 Computing Shattering Coefficients

The theoretical formalization discussed in Section 6.3 confirms the need of designing simpler but still enough CNN architectures. In this context, the Shattering coefficient indicates the complexity of the algorithm bias, which permits to measure the complexity of CNN architectures obtained after IFNN as others proposed in the literature. In addition, Equation 6.29 can be used to analyze the convergence of those networks, making evident the minimal number of examples necessary to ensure learning guarantees according to the SLT.

As discussed in Section 5.4 (Table 8), three CNN architectures were designed using IFNN on top of each dataset. Equation 6.27 is then used to formulate the Shattering coefficient of those CNNs. Table 11 summarizes those growth functions (remember CMU is classified considering two different learning tasks, consequently two different architectures are listed).

Finally, we compare the Shattering coefficients of CNNs designed with IFNN against other architectures from the literature, from which we conclude IFNN helps designing simpler and considerably efficient CNN architectures. Table 12 lists the growth functions of our best architectures, i.e. with the lowest error rates, for the five datasets under analysis (for more information, see Section 5.4) as well as for the most prominent architectures from the literature (listed in Table 8 [3]). It is worth to mention that three of the literature architectures were suppressed of this analysis, given they were not clearly described in the corresponding seminal papers (HASSAIRI; EJBALI; ZAIED, 2015; ZOU *et al.*, 2012; SERMANET; LECUN, 2011). In addition, the Shattering coefficient was only computed for the convolutional layers, being therefore a lower-bound approximation, once those architectures may also be composed of fully-connected layer or Support Vector Machines (SVM).

From Equation 6.23, we estimate the minimal number of examples to ensure learning convergence for CNN architectures, according to the SLT. In practice, Equation 6.30 provides the enough convergence condition to supervised learning so that we decided to apply it, considering $\gamma = 0.05$ as upper bound or enough approximation to zero [4].

$$\frac{\log \mathscr{N}(\mathscr{F}, n)}{n} \leq \gamma \tag{6.30}$$

Table 13 lists the minimal sample sizes to ensure learning for the architectures described in Table 12, including their respective error rates. Remembering that the shattering coefficient of

[3] Each architecture from the literature is described in Section 5.2.1.

[4] Zero is only obtained in the limit of such a function, i.e., when n goes to infinity.

Table 11 – Shattering coefficients of CNN architectures designed after applying IFNN on all datasets under analyses – for more information, see Section 5.4 (Table 8).

Datasets	Conv. Layers	$\mathcal{N}(\mathcal{F},n)$
CMU	**1**	$\mathbf{2\left(\sum_{i=0}^{25}\binom{n}{i}^{13}\right)}$
	2	$4\left(\sum_{i=0}^{25}\binom{n}{i}^{13}\right)\times\left(\sum_{i=0}^{9}\binom{n}{i}^{12}\right)$
	3	$8\left(\sum_{i=0}^{25}\binom{n}{i}^{13}\right)\times\left(\sum_{i=0}^{9}\binom{n}{i}^{12}\right)\times\left(\sum_{i=0}^{6}\binom{n}{i}^{29}\right)$
CMU (eyes)	1	$2\left(\sum_{i=0}^{14}\binom{n}{i}^{9}\right)$
	2	$\mathbf{4\left(\sum_{i=0}^{14}\binom{n}{i}^{9}\right)\times\left(\sum_{i=0}^{36}\binom{n}{i}^{7}\right)}$
	3	$8\left(\sum_{i=0}^{14}\binom{n}{i}^{9}\right)\times\left(\sum_{i=0}^{36}\binom{n}{i}^{7}\right)\times\left(\sum_{i=0}^{12}\binom{n}{i}^{8}\right)$
MNIST	1	$2\left(\sum_{i=0}^{80}\binom{n}{i}^{9}\right)$
	2	$4\left(\sum_{i=0}^{80}\binom{n}{i}^{9}\right)\times\left(\sum_{i=0}^{36}\binom{n}{i}^{7}\right)$
	3	$\mathbf{8\left(\sum_{i=0}^{80}\binom{n}{i}^{9}\right)\times\left(\sum_{i=0}^{36}\binom{n}{i}^{7}\right)\times\left(\sum_{i=0}^{18}\binom{n}{i}^{6}\right)}$
CIFAR-10	1	$2\left(\sum_{i=0}^{30}\binom{n}{i}^{23}\right)$
	2	$\mathbf{4\left(\sum_{i=0}^{30}\binom{n}{i}^{23}\right)\times\left(\sum_{i=0}^{100}\binom{n}{i}^{23}\right)}$
	3	$8\left(\sum_{i=0}^{30}\binom{n}{i}^{23}\right)\times\left(\sum_{i=0}^{100}\binom{n}{i}^{23}\right)\times\left(\sum_{i=0}^{20}\binom{n}{i}^{9}\right)$
COIL-100	1	$2\left(\sum_{i=0}^{117}\binom{n}{i}^{40}\right)$
	2	$\mathbf{4\left(\sum_{i=0}^{117}\binom{n}{i}^{40}\right)\times\left(\sum_{i=0}^{20}\binom{n}{i}^{11}\right)}$
	3	$8\left(\sum_{i=0}^{117}\binom{n}{i}^{40}\right)\times\left(\sum_{i=0}^{20}\binom{n}{i}^{11}\right)\times\left(\sum_{i=0}^{20}\binom{n}{i}^{6}\right)$
GTSRB	1	$2\left(\sum_{i=0}^{36}\binom{n}{i}^{26}\right)$
	2	$4\left(\sum_{i=0}^{36}\binom{n}{i}^{26}\right)\times\left(\sum_{i=0}^{14}\binom{n}{i}^{22}\right)$
	3	$\mathbf{8\left(\sum_{i=0}^{36}\binom{n}{i}^{26}\right)\times\left(\sum_{i=0}^{14}\binom{n}{i}^{22}\right)\times\left(\sum_{i=0}^{24}\binom{n}{i}^{9}\right)}$

those architectures was estimated considering only convolutional layers, discarding other layers with adaptable parameters, such as the fully-connected ones. In fact, most of the architectures from the literature include fully-connected layers to improve results, what increases the complexity of the architecture bias and, consequently, the sample sizes required to ensure learning. In our experiments, just one fully-connected layer was added in order to proceed with the classification.

After obtaining the results listed in this section, we conclude that CNN architectures designed with the aid of IFNN are indeed simpler than the others proposed in literature, that simplicity is directly related to a smaller space of admissible functions and, consequently, to less variance (we suggest the study of the Bias-Variance Dilemma for a better understanding (MELLO; PONTI, 2018)). In spite of that, the number of examples available in training sets were not

Table 12 – Shattering coefficients of the CNN architectures listed in Table 10 (see Section 5.4). Some architectures were suppressed from this analysis, once they were not sufficiently detailed in their seminal papers.

Dataset - Proposal	$\mathscr{N}(\mathscr{F},n)$
CMU - CNN+IFNN (1C)	$2\left(\sum_{i=0}^{25}\binom{n}{i}^{13}\right)$
CMU (eyes) - CNN+IFNN (2C)	$4\left(\sum_{i=0}^{14}\binom{n}{i}^{9}\right)\times\left(\sum_{i=0}^{36}\binom{n}{i}^{7}\right)$
MNIST - CNN+IFNN (3C)	$8\left(\sum_{i=0}^{80}\binom{n}{i}^{9}\right)\times\left(\sum_{i=0}^{36}\binom{n}{i}^{7}\right)\times\left(\sum_{i=0}^{18}\binom{n}{i}^{6}\right)$
CIFAR-10 - CNN+IFNN (2C)	$4\left(\sum_{i=0}^{30}\binom{n}{i}^{23}\right)\times\left(\sum_{i=0}^{100}\binom{n}{i}^{23}\right)$
COIL-100 - CNN+IFNN (2C)	$4\left(\sum_{i=0}^{117}\binom{n}{i}^{40}\right)\times\left(\sum_{i=0}^{20}\binom{n}{i}^{11}\right)$
GTSRB - CNN+IFNN (3C)	$8\left(\sum_{i=0}^{36}\binom{n}{i}^{26}\right)\times\left(\sum_{i=0}^{14}\binom{n}{i}^{22}\right)\times\left(\sum_{i=0}^{24}\binom{n}{i}^{9}\right)$
CMU (eyes) - Zhu *et al.* (2016) (CNN)	$16\left(\sum_{i=0}^{4}\binom{n}{i}^{96}\right)\times\left(\sum_{i=0}^{4}\binom{n}{i}^{192}\right)\times\left(\sum_{i=0}^{4}\binom{n}{i}^{256}\right)\times\left(\sum_{i=0}^{4}\binom{n}{i}^{384}\right)$
MNIST - LeCun *et al.* (1999) (LeNet-4)	$8\left(\sum_{i=0}^{4}\binom{n}{i}^{6}\right)\times\left(\sum_{i=0}^{4}\binom{n}{i}^{16}\right)\times\left(\sum_{i=0}^{4}\binom{n}{i}^{120}\right)$
MNIST - LeCun *et al.* (1999) (LeNet-5)	$8\left(\sum_{i=0}^{25}\binom{n}{i}^{6}\right)\times\left(\sum_{i=0}^{25}\binom{n}{i}^{16}\right)\times\left(\sum_{i=0}^{25}\binom{n}{i}^{120}\right)$
MNIST/CIFAR-10 - Goodfellow *et al.* (2013) (Maxout + Dropout)	$8\left(\sum_{i=0}^{9}\binom{n}{i}^{64}\right)^{3}$
CIFAR-10 - Srivastava *et al.* (2014) (CNN + Dropout)	$16\left(\sum_{i=0}^{25}\binom{n}{i}^{96}\right)\times\left(\sum_{i=0}^{25}\binom{n}{i}^{128}\right)\times\left(\sum_{i=0}^{25}\binom{n}{i}^{256}\right)$
CIFAR-10 - Coates and Ng (2011) (1-CNN + SVM)	$2\left(\sum_{i=0}^{36}\binom{n}{i}^{1{,}600}\right)$
CIFAR-10 - Coates and Ng (2011) (3-CNN + SVM)	$16\left(\sum_{i=0}^{36}\binom{n}{i}^{1{,}600}\right)\times\left(\sum_{i=0}^{4}\binom{n}{i}^{3{,}200}\right)^{2}$
MNIST - Ciregan, Meier and Schmidhuber (2012) (MCDNN)	$140\left(\sum_{i=0}^{16}\binom{n}{i}^{20}\right)\times\left(\sum_{i=0}^{25}\binom{n}{i}^{40}\right)$
CIFAR-10 - Ciregan, Meier and Schmidhuber (2012) (MCDNN)	$400\left(\sum_{i=0}^{49}\binom{n}{i}^{100}\right)\times\left(\sum_{i=0}^{16}\binom{n}{i}^{150}\right)\times\left(\sum_{i=0}^{26}\binom{n}{i}^{250}\right)$
GTSRB - Ciregan, Meier and Schmidhuber (2012) (MCDNN)	$256\left(\sum_{i=0}^{9}\binom{n}{i}^{300}\right)^{2}\times\left(\sum_{i=0}^{4}\binom{n}{i}^{300}\right)^{2}$

Table 13 – Minimal sample sizes to ensure learning for the CNN architectures listed in Table 12. Values are sorted according to the complexity of Shattering coefficients (architectures requiring smaller samples are considered less complexes). Moreover, we provide the size of training sets and their corresponding error rates (computed using Tables 10 and 2 – see Section 5.4).

Datasets (Sample Size)	Proposal	Sample Size n	Error Rate
CMU (640)	**CNN+IFNN (1C)**	$\mathbf{5.5997 \times 10^4}$	**0.54%**
CMU (eyes) (640)	**CNN+IFNN (2C)**	$\mathbf{6.5986 \times 10^4}$	**1.62%**
	Zhu *et al.* (2016)	9.64013×10^5	5.68%
MNIST (70,000)	LeCun *et al.* (1999)	1.24271×10^5	0.70%
	CNN+IFNN (3C)	$\mathbf{1.96251 \times 10^5}$	**1.74%**
	Ciregan, Meier and Schmidhuber (2012)	2.71761×10^5	0.23%
	Goodfellow *et al.* (2013)	3.96364×10^5	0.45%
	LeCun *et al.* (1999)	8.00404×10^5	0.80%
CIFAR-10 (60,000)	Goodfellow *et al.* (2013)	3.96364×10^5	9.38%
	CNN+IFNN (2C)	$\mathbf{5.93310 \times 10^5}$	**28.44%**
	Ciregan, Meier and Schmidhuber (2012)	2.1719×10^6	11.21%
	Srivastava *et al.* (2014)	3.02457×10^6	12.61%
	Coates and Ng (2011)	1.60501×10^7	19.40%
	Coates and Ng (2011)	2.48665×10^7	18.00%
COIL-100 (7,200)	**CNN+IFNN (2C)**	$\mathbf{9.88738 \times 10^5}$	**0.28%**
GTSRB (51,739)	**PP+CNN+IFNN (3C)**	$\mathbf{2.93546 \times 10^5}$	**10.52%**
	Ciregan, Meier and Schmidhuber (2012)	2.75703×10^6	0.54%

enough to ensure learning convergence of those architectures, confirming they either require more data or a different theoretical background to provide learning guarantees.

It is worth to mention that one might consider other options to ensure learning, such as by increasing training set sizes through the usage of Data Augmentation strategies, or by reducing even more the complexity of CNN architectures while verifying whether they still perform satisfactorily, i.e. generalize, on input data.

In summary, simpler CNNs require smaller training set sizes, leading to faster convergences and simultaneously reducing the probability of overfitting as discussed in (MELLO; PONTI, 2018). Shattering coefficients confirm IFNN supports the design of such type of CNN architectures, producing simpler architectures while providing adequate parameters to perform efficient classifications in most scenarios.

6.4 Final Considerations

The main concepts of the SLT are briefly described in this chapter, providing the conditions for which learning convergence is ensured for supervised learning tasks. Next, Shattering coefficients were used to assess the complexity of CNN architectures, following our algebraic formulation described in Section 3.2.1. CNN architectures designed with the aid of IFNN were

confirmed to be the simpler learning models in comparison to others proposed in the literature. From that, we decided to assess the minimal sample sizes to ensure learning convergence, again confirming that CNN after IFNN provides smaller spaces of admissible functions so that they are less prone to overfitting and provide stronger learning guarantees.

Furthermore, we conclude datasets are not large enough to ensure learning for the evaluated architectures, therefore one should take advantage of other strategies to improve learning guarantees such as Data Augmentation or the design of even simpler CNNs. In spite of those controversial conclusions, we could still confirm IFNN supports the design of simpler but efficient CNN architectures in most scenarios.

CHAPTER

7

CONCLUSIONS

Deep learning (DL) has confirmed relevant performance results while modeling large amounts of data from several application domains, having the Convolutional Neural Network (CNN) as its state-of-art algorithm (COATES *et al.*, 2011; COX; PINTO, 2011; NAKASHIKA *et al.*, 2012; BERGSTRA; BENGIO, 2012; SERMANET *et al.*, 2014; ZEILER; FERGUS, 2014; CIREGAN; MEIER; SCHMIDHUBER, 2012). CNN architectures are typically built up using several units organized along consecutive layers, resulting in accurate but yet complex, empirical and deep models. As several researchers claim, by increasing an architecture size and thus its inherent complexity, the error is reduced regardless the classification task under analysis (KRIZHEVSKY; SUTSKEVER; HINTON, 2012; SRIVASTAVA *et al.*, 2014; LECUN; BENGIO; HINTON, 2015; SZEGEDY *et al.*, 2015; COGSWELL *et al.*, 2016; GOODFELLOW; BENGIO; COURVILLE, 2016).

In spite of the success of CNNs, they still do not rely on strong theoretical foundations to ensure learning bounds nor count on formal methodologies to support the design of their architectures, what is typically performed using trial-and-error strategies (see Section 5.2.1). In order to exemplify, most researchers have proposed approaches to assess CNN architectures using evolutionary computing, from which they select some setting after training all candidate solutions. Of course this is a very resource demanding process, being also highly dependent on the candidate hyperparameters (e.g. training rate, momentum, etc.) (GARRO; VÁZQUEZ, 2015; YOUNG *et al.*, 2015; ALBELWI; MAHMOOD, 2017). Such an empirical process in conjunction with the lack of theoretical understanding about the CNN learning motivated this PhD thesis, which introduces a new method to design CNN architectures taking advantage of Dynamical Systems concepts.

In this context, we decided to implement the forward process of the CNN algorithm from scratch, allowing to understand its steps and complement our knowledge. In addition, we implemented a Multilayer Perceptron (MLP) with Dropout to study the impact of regularization

terms during learning [1]. Those implementations supported the development of the algebraic formulation detailed in Section 3.2.1, which comprises the first contribution of this thesis. Such formulation demonstrates that CNNs apply linear transformations at every convolutional layer right after some space embedding, i.e. the same Dynamical System concept used to deal with time-dependent data. In that sense, we decided to design the Image-based False Nearest Neighbors (IFNN), a tool based on the False Nearest Neighbors (FNN) (KENNEL; BROWN; ABARBANEL, 1992), to embed images in adequate input spaces so they provide as much information as possible when processed by CNNs (see Section 4.2).

IFNN was initially designed to estimate convolutional mask sizes for single-layer CNNs, which are responsible for embedding input images into new spaces before applying the linear transformations. Motivated by the space reconstructions of time-dependent data as provided by FNN, IFNN detects data recurrences on local regions of images in order to design CNN architectures (see Section 4.3). In addition, IFNN was applied to estimate the number of convolutional units per CNN layer, being consequently responsible for the codomain dimensionality after linear transformations (as supported by our algebraic formulation in Section 4.3.1).

According to FNN, when the fraction of false nearest neighbors does not vary significantly, one concludes there is some stable codomain to represent data recurrences. This assumption is also taken by IFNN to identify the best phase space to reconstruct input images and, thus, represent their recurrences and patterns. Moreover, IFNN does not require the label information to provide a good estimation for the parameters of convolutional layers, given it relies on how pixel values form a given input space. This advantage makes possible to apply IFNN in unsupervised scenarios in an attempt to obtain good data representations. The only assumption is that the input set provided to IFNN must represent the main dataset patterns, thus some subset of stratified images would be enough to support an adequate architecture estimation (see Section 4.3).

Experiments were performed to assess the IFNN method on five different datasets, whose results confirm that the CNNs designed after IFNN produce similar accuracies to other complex DL architectures from the literature (Section 5.4), supporting our conclusion that IFNN provides a fair parameter estimation for specific tasks. Additional experiments confirm a correlation of convolutional mask sizes and their corresponding accuracies, as discussed in Section 4.3, indicating that the CNN design after IFNN provides enough relevant features to perform a proper classifications.

In addition, the CNN architectures built up with the aid of IFNN are simpler than the ones usually employed in the literature (see Section 6.3.1), showing that shallow architectures can also be enough to classify input data under different scenarios. From the results discussed in Sections 5.4 and 6.3.1, we conclude that IFNN supports the design of shallow but still efficient CNN architectures that are faster to train and provide greater learning guarantees. Despite the

[1] Source codes are available at <https://github.com/marthadais/studyForwardCNN.git>

good results of our approach, the number of units estimated per layer seems to be insufficient to model more complex datasets. However, for the most scenarios under analysis, IFNN adequately estimated the number of units per layer once the results did not vary significantly as the number of units increased.

In summary, our two main contributions, to mention the IFNN method and the algebraic formulation, support the design of CNN architectures according to a framework instead of a trial-and-error strategy, as published in Ferreira *et al.* (2018). The source codes comprise a complementary contribution to future studies, which are available at <https://github.com/marthadais/Estimate-Parameters>. It is worth to mention that the IFNN method improved the classification of phonemes in the speech recognition task, as reported in Shulby *et al.* (2017). In such a circumstance, IFNN allowed us to build up simpler and enough architectures while compared to deep and complex models from the literature, which did not have any design justification. Lastly, studies using SLT concepts were presented in Mello, Ferreira and Ponti (2017) and Mello, Ponti and Ferreira (2018) in order to provide some theoretical learning guarantees for CNNs, leading to theoretical formulations in acoustic modeling as reported in Shulby *et al.* (2019).

7.1 Future Work

Considering that the IFNN estimation does not provide yet the most adequate number of units per convolutional layer, an extension of this work must still be developed. From this conclusion, we decided to perform some additional experiments as discussed in Appendix A, which brings up an initial study on the CNN generalization capacity according to the number of units and layers using the CIFAR-10 dataset. Nevertheless, such set of experiments is not enough yet to properly decide on the number of layers, their operation types and their sequences, thus leading to an open problem. Image aspects, such as color and size, and preprocessing also need to be investigate in order to improve CNN results, as indicated by experiments.

Furthermore, IFNN can be extended to estimate an adequate pooling mask size to compute the image reduction; and it can also be adapted to support an adequate CNN architecture for video analysis, which comprises time-dependent data. Techniques to estimate adequate time delays from the Dynamical System area can also be studied to support the estimate of convolutional and pooling strides, leading to most adequate CNN architectures from datasets. Finally, we have been still studying the adequate estimation of CNN complexities, which would lead us to compute better approximations for the Shattering coefficient, so that theoretical learning guarantees could be provided while analyzing CNN models (MELLO; FERREIRA; PONTI, 2017; MELLO; PONTI; FERREIRA, 2018).

BIBLIOGRAPHY

ABADI, M.; AGARWAL, A.; BARHAM, P.; BREVDO, E.; CHEN, Z.; CITRO, C.; CORRADO, G. S.; DAVIS, A.; DEAN, J.; DEVIN, M.; GHEMAWAT, S.; GOODFELLOW, I.; HARP, A.; IRVING, G.; ISARD, M.; JIA, Y.; JOZEFOWICZ, R.; KAISER, L.; KUDLUR, M.; LEVENBERG, J.; MANÉ, D.; MONGA, R.; MOORE, S.; MURRAY, D.; OLAH, C.; SCHUSTER, M.; SHLENS, J.; STEINER, B.; SUTSKEVER, I.; TALWAR, K.; TUCKER, P.; VANHOUCKE, V.; VASUDEVAN, V.; VIÉGAS, F.; VINYALS, O.; WARDEN, P.; WATTENBERG, M.; WICKE, M.; YU, Y.; ZHENG, X. **TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems**. 2015. Software available from tensorflow.org. Available: <http://tensorflow.org/>. Citations on pages 36, 58, and 59.

ABDEL-HAMID, O.; MOHAMED, A.-R.; JIANG, H.; DENG, L.; PENN, G.; YU, D. Convolutional neural networks for speech recognition. **IEEE/ACM Transactions on audio, speech, and language processing**, IEEE, v. 22, n. 10, p. 1533–1545, 2014. Citation on page 44.

ADLER, A.; GUARDO, R. A neural network image reconstruction technique for electrical impedance tomography. **Medical Imaging, IEEE Transactions on**, IEEE, v. 13, n. 4, p. 594–600, 1994. Citation on page 25.

AL-RFOU, R.; ALAIN, G.; ALMAHAIRI, A.; ANGERMUELLER, C.; BAHDANAU, D.; BALLAS, N.; BASTIEN, F.; BAYER, J.; BELIKOV, A.; BELOPOLSKY, A.; BENGIO, Y.; BERGERON, A.; BERGSTRA, J.; BISSON, V.; SNYDER, J. B.; BOUCHARD, N.; BOULANGER-LEWANDOWSKI, N.; BOUTHILLIER, X.; BRÉBISSON, A.; BREULEUX, O.; CARRIER, P.-L.; CHO, K.; CHOROWSKI, J.; CHRISTIANO, P.; COOIJMANS, T.; CÔTÉ, M.-A.; CÔTÉ, M.; COURVILLE, A.; DAUPHIN, Y. N.; DELALLEAU, O.; DEMOUTH, J.; DESJARDINS, G.; DIELEMAN, S.; DINH, L.; DUCOFFE, M.; DUMOULIN, V.; KAHOU, S. E.; ERHAN, D.; FAN, Z.; FIRAT, O.; GERMAIN, M.; GLOROT, X.; GOODFELLOW, I.; GRAHAM, M.; GULCEHRE, C.; HAMEL, P.; HARLOUCHET, I.; HENG, J.-P.; HIDASI, B.; HONARI, S.; JAIN, A.; JEAN, S.; JIA, K.; KOROBOV, M.; KULKARNI, V.; LAMB, A.; LAMBLIN, P.; LARSEN, E.; LAURENT, C.; LEE, S.; LEFRANCOIS, S.; LEMIEUX, S.; LÉONARD, N.; LIN, Z.; LIVEZEY, J. A.; LORENZ, C.; LOWIN, J.; MA, Q.; MANZAGOL, P.-A.; MASTROPIETRO, O.; MCGIBBON, R. T.; MEMISEVIC, R.; MERRIËNBOER, B. van; MICHALSKI, V.; MIRZA, M.; ORLANDI, A.; PAL, C.; PASCANU, R.; PEZESHKI, M.; RAFFEL, C.; RENSHAW, D.; ROCKLIN, M.; ROMERO, A.; ROTH, M.; SADOWSKI, P.; SALVATIER, J.; SAVARD, F.; SCHLÜTER, J.; SCHULMAN, J.; SCHWARTZ, G.; SERBAN, I. V.; SERDYUK, D.; SHABANIAN, S.; SIMON, E.; SPIECKERMANN, S.; SUBRAMANYAM, S. R.; SYGNOWSKI, J.; TANGUAY, J.; TULDER, G.; TURIAN, J.; URBAN, S.; VINCENT, P.; VISIN, F.; VRIES, H.; WARDE-FARLEY, D.; WEBB, D. J.; WILLSON, M.; XU, K.; XUE, L.; YAO, L.; ZHANG, S.; ZHANG, Y. Theano: A Python framework for fast computation of mathematical expressions. **arXiv e-prints**, abs/1605.02688, 2016. Citations on pages 36, 58, and 59.

ALBELWI, S.; MAHMOOD, A. A framework for designing the architectures of deep convolutional neural networks. **Entropy**, Multidisciplinary Digital Publishing Institute, v. 19, n. 6, p. 242, 2017. Citations on pages 28, 32, 33, 40, 70, and 107.

ALLIGOOD, K. T.; SAUER, T. D.; YORKE, J. A. **Chaos**. [S.l.]: Springer, 1997. Citations on pages 29, 32, 61, 62, 65, and 66.

BARBOUNIS, T. G.; THEOCHARIS, J. B.; ALEXIADIS, M. C.; DOKOPOULOS, P. S. Long-term wind speed and power forecasting using local recurrent neural network models. **IEEE Transactions on Energy Conversion**, IEEE, v. 21, n. 1, p. 273–284, 2006. Citation on page 35.

BEN-DAVID, S.; BLITZER, J.; CRAMMER, K.; KULESZA, A.; PEREIRA, F.; VAUGHAN, J. W. A theory of learning from different domains. **Machine learning**, Springer, v. 79, n. 1-2, p. 151–175, 2010. Citation on page 28.

BENGIO, Y.; LAMBLIN, P.; POPOVICI, D.; LAROCHELLE, H. Greedy layer-wise training of deep networks. In: **Advances in neural information processing systems**. [S.l.: s.n.], 2007. p. 153–160. Citation on page 35.

BERGSTRA, J.; BENGIO, Y. Random search for hyper-parameter optimization. **The Journal of Machine Learning Research**, JMLR. org, v. 13, n. 1, p. 281–305, 2012. Citations on pages 27 and 107.

BHATT, H. S.; BHARADWAJ, S.; SINGH, R.; VATSA, M. Memetically optimized mcwld for matching sketches with digital face images. **Information Forensics and Security, IEEE Transactions on**, IEEE, v. 7, n. 5, p. 1522–1535, 2012. Citations on pages 25 and 33.

BOGDANOV, D.; WACK, N.; GÓMEZ, E.; GULATI, S.; HERRERA, P.; MAYOR, O.; ROMA, G.; SALAMON, J.; ZAPATA, J.; SERRA, X. Essentia: an open-source library for sound and music analysis. In: ACM. **Proceedings of the 21st ACM international conference on Multimedia**. [S.l.], 2013. p. 855–858. Citation on page 28.

BOSER, B. E.; GUYON, I. M.; VAPNIK, V. N. A training algorithm for optimal margin classifiers. In: ACM. **Proceedings of the fifth annual workshop on Computational learning theory**. [S.l.], 1992. p. 144–152. Citation on page 34.

BOTTOU, L. Stochastic gradient descent tricks. In: **Neural networks: Tricks of the trade**. [S.l.]: Springer, 2012. p. 421–436. Citation on page 53.

BRIN, S.; PAGE, L. Reprint of: The anatomy of a large-scale hypertextual web search engine. **Computer networks**, Elsevier, v. 56, n. 18, p. 3825–3833, 2012. Citation on page 25.

BROSCH, T.; TAM, R. Efficient training of convolutional deep belief networks in the frequency domain for application to high-resolution 2d and 3d images. **Neural computation**, MIT Press, v. 27, n. 1, p. 211–227, 2015. Citation on page 35.

BROUSSARD, R. P.; ROGERS, S. K.; OXLEY, M. E.; TARR, G. L. Physiologically motivated image fusion for object detection using a pulse coupled neural network. **Neural Networks, IEEE Transactions on**, IEEE, v. 10, n. 3, p. 554–563, 1999. Citation on page 25.

BURGESS, J. Occam's razor and scientific method. **The Philosophy of Mathematics Today. Clarendon Press, Oxford**, p. 195–214, 1998. Citation on page 30.

CARLSSON, G.; MÃŠMOLI, F. Characterization, stability and convergence of hierarchical clustering methods. **Journal of machine learning research**, v. 11, n. Apr, p. 1425–1470, 2010. Citation on page 91.

CHEN, X.; GUNTUBOYINA, A.; ZHANG, Y. On bayes risk lower bounds. **Journal of Machine Learning Research**, v. 17, p. 1–58, 2016. Citations on pages 32 and 40.

CHENG, K.-S.; LIN, J.-S.; MAO, C.-W. The application of competitive hopfield neural network to medical image segmentation. **Medical Imaging, IEEE Transactions on**, IEEE, v. 15, n. 4, p. 560–567, 1996. Citation on page 25.

CHERNOFF, H. A measure of asymptotic efficiency for tests of a hypothesis based on the sum of observations. **Ann. Math. Statist.**, The Institute of Mathematical Statistics, v. 23, n. 4, p. 493–507, 12 1952. Available: <https://doi.org/10.1214/aoms/1177729330>. Citation on page 93.

CHO, K. H.; RAIKO, T.; ILIN, A. Gaussian-bernoulli deep boltzmann machine. In: IEEE. **Neural Networks (IJCNN), The 2013 International Joint Conference on**. [S.l.], 2013. p. 1–7. Citations on pages 34 and 58.

CHOI, K.; FAZEKAS, G.; SANDLER, M.; CHO, K. Transfer learning for music classification and regression tasks. **The 18th International Society for Music Information Retrieval Conference (ISMIR)**, 2017. Citation on page 28.

CIREGAN, D.; MEIER, U.; SCHMIDHUBER, J. Multi-column deep neural networks for image classification. In: IEEE. **Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on**. [S.l.], 2012. p. 3642–3649. Citations on pages 27, 33, 41, 43, 47, 74, 75, 76, 79, 88, 89, 90, 103, 104, and 107.

COATES, A.; CARPENTER, B.; CASE, C.; SATHEESH, S.; SURESH, B.; WANG, T.; WU, D. J.; NG, A. Y. Text detection and character recognition in scene images with unsupervised feature learning. In: IEEE. **Document Analysis and Recognition (ICDAR), 2011 International Conference on**. [S.l.], 2011. p. 440–445. Citations on pages 27, 33, and 107.

COATES, A.; NG, A. Y. Selecting receptive fields in deep networks. In: **Advances in Neural Information Processing Systems**. [S.l.: s.n.], 2011. p. 2528–2536. Citations on pages 39, 77, 89, 103, and 104.

COGSWELL, M.; AHMED, F.; GIRSHICK, R. B.; ZITNICK, L.; BATRA, D. Reducing overfitting in deep networks by decorrelating representations. **International Conference on Learning Representations (ICLR)**, 2016. Citations on pages 26, 28, 30, 40, 41, 76, and 107.

COLLOBERT, R.; BENGIO, S.; MARITHOZ, J. **Torch: A Modular Machine Learning Software Library**. [S.l.]: Idiap Research Institute, 2002. Citations on pages 36, 58, and 59.

CORTES, C.; VAPNIK, V. Support-vector networks. **Machine learning**, Springer, v. 20, n. 3, p. 273–297, 1995. Citations on pages 26 and 34.

COVER, T. M.; THOMAS, J. A. **Elements of information theory**. [S.l.]: John Wiley & Sons, 2012. Citation on page 62.

COVINGTON, P.; ADAMS, J.; SARGIN, E. Deep neural networks for youtube recommendations. In: ACM. **Proceedings of the 10th ACM Conference on Recommender Systems**. [S.l.], 2016. p. 191–198. Citation on page 25.

COX, D.; PINTO, N. Beyond simple features: A large-scale feature search approach to unconstrained face recognition. In: IEEE. **Automatic Face & Gesture Recognition and Workshops (FG 2011), 2011 IEEE International Conference on**. [S.l.], 2011. p. 8–15. Citations on pages 27 and 107.

CYBENKO, G. Approximation by superpositions of a sigmoidal function. **Mathematics of Control, Signals, and Systems (MCSS)**, v. 2, n. 4, p. 303–314, 1989. Citation on page 34.

DENG, J.; DONG, W.; SOCHER, R.; LI, L.-J.; LI, K.; FEI-FEI, L. Imagenet: A large-scale hierarchical image database. In: IEEE. **Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on**. [S.l.], 2009. p. 248–255. Citation on page 37.

DENG, L.; YU, D. Deep learning: methods and applications. **Foundations and Trends in Signal Processing**, Now Publishers Inc., v. 7, n. 3–4, p. 197–387, 2014. Citations on pages 35 and 43.

DEVROYE, L.; GYÖRFI, L.; LUGOSI, G. **A probabilistic theory of pattern recognition**. [S.l.]: Springer Science & Business Media, 1997. Citations on pages 95, 100, and 127.

DOZAT, T. Incorporating nesterov momentum into adam. 2016. Citation on page 57.

DU, Y.; WANG, W.; WANG, L. Hierarchical recurrent neural network for skeleton based action recognition. In: **Proceedings of the IEEE conference on computer vision and pattern recognition**. [S.l.: s.n.], 2015. p. 1110–1118. Citation on page 35.

DUCHI, J.; HAZAN, E.; SINGER, Y. Adaptive subgradient methods for online learning and stochastic optimization. **Journal of Machine Learning Research**, v. 12, n. Jul, p. 2121–2159, 2011. Citation on page 55.

ELKAHKY, A. M.; SONG, Y.; HE, X. A multi-view deep learning approach for cross domain user modeling in recommendation systems. In: INTERNATIONAL WORLD WIDE WEB CONFERENCES STEERING COMMITTEE. **Proceedings of the 24th International Conference on World Wide Web**. [S.l.], 2015. p. 278–288. Citation on page 25.

ELLIS, D. P.; POLINER, G. E. Identifying 'cover songs' with chroma features and dynamic programming beat tracking. In: IEEE. **Acoustics, Speech and Signal Processing, 2007. ICASSP 2007. IEEE International Conference on**. [S.l.], 2007. v. 4, p. IV–1429. Citation on page 28.

FERREIRA, M. D. **Identificação de covers a partir de grandes bases de dados de músicas**. Master's Thesis (Master's Thesis) — Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, 2014. Dissertação de Mestrado em Ciências de Computação e Matemática Computacional. Citations on pages 28 and 29.

FERREIRA, M. D.; CORRÊA, D. C.; NONATO, L. G.; MELLO, R. F. de. Designing architectures of convolutional neural networks to solve practical problems. **Expert Systems with Applications**, Elsevier, v. 94, p. 205–217, 2018. Citations on pages 25, 26, 28, 29, 30, 39, 45, 65, 71, 80, 91, and 109.

FUNAHASHI, K.-I.; NAKAMURA, Y. Approximation of dynamical systems by continuous time recurrent neural networks. **Neural networks**, Elsevier, v. 6, n. 6, p. 801–806, 1993. Citation on page 35.

GAO, Q. **User Modeling and Personalization in the Microblogging Sphere.** [S.l.]: Delft University of Technology, Netherlands, 2013. Citation on page 25.

GARRO, B. A.; VÁZQUEZ, R. A. Designing artificial neural networks using particle swarm optimization algorithms. **Computational intelligence and neuroscience**, Hindawi Publishing Corp., v. 2015, p. 61, 2015. Citations on pages 26, 27, 28, 32, 40, 70, and 107.

GIRSHICK, R.; DONAHUE, J.; DARRELL, T.; MALIK, J. Rich feature hierarchies for accurate object detection and semantic segmentation. In: **Proceedings of the IEEE conference on computer vision and pattern recognition**. [S.l.: s.n.], 2014. p. 580–587. Citations on pages 36 and 59.

GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A. **Deep Learning**. [S.l.]: MIT Press, 2016. <http://www.deeplearningbook.org>. Citations on pages 26, 27, 29, 33, 35, 36, 39, 44, 45, 51, 52, 59, and 107.

GOODFELLOW, I.; WARDE-FARLEY, D.; MIRZA, M.; COURVILLE, A.; BENGIO, Y. Maxout networks. In: **Proceedings of the 30th International Conference on Machine Learning**. Atlanta, Georgia, USA: PMLR, 2013. (Proceedings of Machine Learning Research, 3), p. 1319–1327. Citations on pages 39, 43, 74, 76, 77, 89, 90, 103, and 104.

GREGOR, K.; DANIHELKA, I.; GRAVES, A.; REZENDE, D.; WIERSTRA, D. Draw: A recurrent neural network for image generation. In: BACH, F.; BLEI, D. (Ed.). **Proceedings of the 32nd International Conference on Machine Learning**. Lille, France: PMLR, 2015. (Proceedings of Machine Learning Research, v. 37), p. 1462–1471. Citation on page 35.

GROTH, D.; HARTMANN, S.; KLIE, S.; SELBIG, J. Principal components analysis. In: **Computational Toxicology**. [S.l.]: Springer, 2013. p. 527–547. Citation on page 57.

HAJILOO, R.; SALARIEH, H.; ALASTY, A. Chaos control in delayed phase space constructed by the takens embedding theory. **Communications in Nonlinear Science and Numerical Simulation**, Elsevier, v. 54, p. 453–465, 2018. Citation on page 29.

HANNEKE, S. The optimal sample complexity of pac learning. **Journal of Machine Learning Research**, v. 17, n. 38, p. 1–15, 2016. Citations on pages 32 and 40.

HASSAIRI, S.; EJBALI, R.; ZAIED, M. Supervised image classification using deep convolutional wavelets network. In: IEEE. **Tools with Artificial Intelligence (ICTAI), 2015 IEEE 27th International Conference on**. [S.l.], 2015. p. 265–271. Citations on pages 74, 77, 89, and 101.

HAYKIN, S. **Neural networks: a comprehensive foundation**. [S.l.]: Prentice Hall PTR, 1994. Citation on page 50.

HE, K.; ZHANG, X.; REN, S.; SUN, J. Deep residual learning for image recognition. In: **Proceedings of the IEEE conference on computer vision and pattern recognition**. [S.l.: s.n.], 2016. p. 770–778. Citations on pages 26, 28, 30, 33, 34, 36, 38, 39, 41, 59, 70, and 84.

HECHT-NIELSEN, R. Theory of the backpropagation neural network. In: IEEE. **International Joint Conference on Neural Networks (IJCNN)**. [S.l.], 1989. p. 593–605. Citation on page 43.

HECHT, R.; HEROLD, H.; MEINEL, G.; BUCHROITHNER, M. Automatic derivation of urban structure types from topographic maps by means of image analysis and machine learning. In: **26th International cartographic conference–Proceedings: International cartographic association.** [S.l.: s.n.], 2013. v. 17, p. 14. Citation on page 25.

HÉNON, M. A two-dimensional mapping with a strange attractor. In: **The Theory of Chaotic Attractors**. [S.l.]: Springer, 1976. p. 94–102. Citation on page 63.

HERSHEY, S.; CHAUDHURI, S.; ELLIS, D. P. W.; GEMMEKE, J. F.; JANSEN, A.; MOORE, R. C.; PLAKAL, M.; PLATT, D.; SAUROUS, R. A.; SEYBOLD, B.; SLANEY, M.; WEISS, R. J.; WILSON, K. Cnn architectures for large-scale audio classification. In: **2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)**. [S.l.: s.n.], 2017. p. 131–135. ISSN 2379-190X. Citation on page 43.

HINTON, G. E.; SALAKHUTDINOV, R. R. Reducing the dimensionality of data with neural networks. **Science**, American Association for the Advancement of Science, v. 313, n. 5786, p. 504–507, 2006. Citations on pages 35, 74, 76, 78, 88, and 89.

HOEFFDING, W. Probability inequalities for sums of bounded random variables. **Journal of the American statistical association**, Taylor & Francis Group, v. 58, n. 301, p. 13–30, 1963. Citation on page 93.

HOGG, R. V.; CRAIG, A. T. **Introduction to mathematical statistics**. [S.l.]: Upper Saddle River, New Jersey: Prentice Hall, 1995. Citation on page 58.

HOUBEN, S.; STALLKAMP, J.; SALMEN, J.; SCHLIPSING, M.; IGEL, C. Detection of traffic signs in real-world images: The German Traffic Sign Detection Benchmark. In: **International Joint Conference on Neural Networks**. [S.l.: s.n.], 2013. Citation on page 74.

HRASKO, R.; PACHECO, A. G. C.; KROHLING, R. A. Time series prediction using restricted boltzmann machines and backpropagation. **Procedia Computer Science**, Elsevier, v. 55, p. 990–999, 2015. Citation on page 35.

HUANG, F. J.; LECUN, Y. Large-scale learning with svm and convolutional for generic object categorization. In: IEEE. **Computer Vision and Pattern Recognition, 2006 IEEE Computer Society Conference on**. [S.l.], 2006. v. 1, p. 284–291. Citations on pages 43 and 48.

HUBEL, D. H.; WIESEL, T. N. Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. **The Journal of physiology**, Wiley Online Library, v. 160, n. 1, p. 106–154, 1962. Citation on page 44.

HUH, M.; AGRAWAL, P.; EFROS, A. A. What makes imagenet good for transfer learning? **arXiv preprint arXiv:1608.08614**, 2016. Citation on page 28.

IDE, H.; KURITA, T. Improvement of learning for CNN with ReLU activation by sparse regularization. In: IEEE. **Neural Networks (IJCNN), 2017 International Joint Conference on**. [S.l.], 2017. p. 2684–2691. Citation on page 47.

IHAKA, R.; GENTLEMAN, R. R: a language for data analysis and graphics. **Journal of computational and graphical statistics**, Taylor & Francis, v. 5, n. 3, p. 299–314, 1996. Citation on page 71.

ISAACSON, E.; KELLER, H. B. **Analysis of numerical methods**. [S.l.]: Courier Corporation, 2012. Citation on page 54.

JARRETT, K.; KAVUKCUOGLU, K.; RANZATO, M.; LECUN, Y. What is the best multi-stage architecture for object recognition? In: IEEE. **Computer Vision, 2009 IEEE 12th International Conference on**. [S.l.], 2009. p. 2146–2153. Citation on page 48.

JIA, Y.; SHELHAMER, E.; DONAHUE, J.; KARAYEV, S.; LONG, J.; GIRSHICK, R.; GUADARRAMA, S.; DARRELL, T. Caffe: Convolutional architecture for fast feature embedding. In: ACM. **Proceedings of the ACM International Conference on Multimedia**. [S.l.], 2014. p. 675–678. Citations on pages 36, 58, 59, and 79.

JING, Y.; LIU, Y.; TSAI, D. **Retrieval of similar images to a query image**. [S.l.]: Google Patents, 2017. US Patent 9,589,208. Citation on page 25.

KANTZ, H.; SCHREIBER, T. **Nonlinear time series analysis**. [S.l.]: Cambridge university press, 2004. Citations on pages 51, 61, 62, and 66.

KARPATHY, A.; TODERICI, G.; SHETTY, S.; LEUNG, T.; SUKTHANKAR, R.; FEI-FEI, L. Large-scale video classification with convolutional neural networks. In: **Proceedings of the IEEE conference on Computer Vision and Pattern Recognition**. [S.l.: s.n.], 2014. p. 1725–1732. Citations on pages 36 and 43.

KENNEL, M. B.; BROWN, R.; ABARBANEL, H. D. Determining embedding dimension for phase-space reconstruction using a geometrical construction. **Physical review A**, APS, v. 45, n. 6, p. 3403, 1992. Citations on pages 29, 30, 32, 61, 62, 64, 66, and 108.

KINGMA, D.; BA, J. Adam: A method for stochastic optimization. **arXiv preprint arXiv:1412.6980**, 2014. Citations on pages 56 and 57.

KOSKELA, T.; LEHTOKANGAS, M.; SAARINEN, J.; KASKI, K. Time series prediction with multilayer perceptron, fir and elman neural networks. In: INNS PRESS SAN DIEGO, USA. **Proceedings of the World Congress on Neural Networks**. [S.l.], 1996. p. 491–496. Citation on page 34.

KOVALEV, V.; KALINOVSKY, A.; KOVALEV, S. **Deep Learning with Theano, Torch, Caffe, Tensorflow, and Deeplearning4J: Which One is the Best in Speed and Accuracy?** [S.l.]: Minsk: Publishing Center of BSU, 2016. Citations on pages 28 and 36.

KRIZHEVSKY, A. Convolutional deep belief networks on cifar-10. 2010. Citations on pages 34, 35, 76, 78, 79, 89, and 90.

KRIZHEVSKY, A.; HINTON, G. **Learning multiple layers of features from tiny images**. [S.l.], 2009. Citation on page 74.

KRIZHEVSKY, A.; SUTSKEVER, I.; HINTON, G. E. Imagenet classification with deep convolutional neural networks. In: **Advances in neural information processing systems**. [S.l.: s.n.], 2012. p. 1097–1105. Citations on pages 25, 26, 27, 28, 30, 33, 34, 37, 38, 39, 40, 41, 43, 47, 48, 52, 57, 58, 70, 75, 76, 79, 84, and 107.

KROGH, A.; HERTZ, J. A. A simple weight decay can improve generalization. In: **Advances in neural information processing systems**. [S.l.: s.n.], 1992. p. 950–957. Citation on page 54.

LAROCHELLE, H.; BENGIO, Y.; LOURADOUR, J.; LAMBLIN, P. Exploring strategies for training deep neural networks. **Journal of Machine Learning Research**, v. 10, n. Jan, p. 1–40, 2009. Citation on page 35.

LAUER, F.; SUEN, C. Y.; BLOCH, G. A trainable feature extractor for handwritten digit recognition. **Pattern Recognition**, Elsevier, v. 40, n. 6, p. 1816–1824, 2007. Citations on pages 35 and 36.

LAWRENCE, S.; GILES, C. L.; TSOI, A. C.; BACK, A. D. Face recognition: A convolutional neural-network approach. **Neural Networks, IEEE Transactions on**, IEEE, v. 8, n. 1, p. 98–113, 1997. Citation on page 25.

LECUN, Y.; BENGIO, Y. Convolutional networks for images, speech, and time series. **The handbook of brain theory and neural networks**, v. 3361, n. 10, 1995. Citations on pages 35, 36, 37, and 44.

LECUN, Y.; BENGIO, Y.; HINTON, G. Deep learning. **Nature**, Nature Publishing Group, v. 521, n. 7553, p. 436–444, 2015. Citations on pages 25, 26, 27, 33, 34, 35, 39, 44, 47, 48, 51, and 107.

LECUN, Y.; BOTTOU, L.; BENGIO, Y.; HAFFNER, P. Gradient-based learning applied to document recognition. **Proceedings of the IEEE**, IEEE, v. 86, n. 11, p. 2278–2324, 1998. Citations on pages 26, 29, 36, 37, 47, 48, 49, 68, and 74.

LECUN, Y.; HAFFNER, P.; BOTTOU, L.; BENGIO, Y. Object recognition with gradient-based learning. In: **Shape, Contour and Grouping in Computer Vision**. [S.l.]: Springer Berlin Heidelberg, 1999. p. 319–345. ISBN 978-3-540-46805-9. Citations on pages 26, 30, 33, 34, 35, 37, 43, 74, 75, 76, 89, 103, and 104.

LI, Q.; CAI, W.; WANG, X.; ZHOU, Y.; FENG, D. D.; CHEN, M. Medical image classification with convolutional neural network. In: IEEE. **Control Automation Robotics & Vision (ICARCV), 2014 13th International Conference on**. [S.l.], 2014. p. 844–848. Citation on page 47.

LIN, C.-T.; HUANG, C.-H.; CHEN, S.-A. CNN-based hybrid-order texture segregation as early vision processing and its implementation on CNN-UM. **IEEE Transactions on Circuits and Systems I: Regular Papers**, IEEE, v. 54, n. 10, p. 2277–2287, 2007. Citation on page 34.

LIU, Y.; LI, Z. Study on texture feature extraction from forensic images with watermark. In: IEEE. **Industrial Electronics and Applications (ICIEA), 2014 IEEE 9th Conference on**. [S.l.], 2014. p. 1471–1475. Citation on page 25.

LU, Y.; JIANG, H.; LIU, W. Classification of EEG signal by STFT-CNN framework: Identification of right-/left-hand motor imagination in bci systems. In: **Proceedings of the 2017 The 7th International Conference on Computer Engineering and Networks. 22-23 July, 2017 Shanghai, China (CENet2017)**. [S.l.: s.n.], 2017. Citation on page 47.

LUKOŠEVIČIUS, M.; JAEGER, H. Reservoir computing approaches to recurrent neural network training. **Computer Science Review**, Elsevier, v. 3, n. 3, p. 127–149, 2009. Citations on pages 34 and 35.

LUXBURG, U. V.; SCHÖLKOPF, B. Statistical learning theory: Models, concepts, and results. In: **Handbook of the History of Logic**. [S.l.]: Elsevier, 2011. v. 10, p. 651–706. Citations on pages 26, 34, 53, 91, 92, 93, 95, 96, 98, 99, 100, and 127.

MA, H.-G.; HAN, C.-Z. Selection of embedding dimension and delay time in phase space reconstruction. **Frontiers of Electrical and Electronic Engineering in China**, Springer, v. 1, n. 1, p. 111–114, 2006. Citation on page 29.

MALLAT, S. **A wavelet tour of signal processing**. [S.l.]: Elsevier, 1999. Citation on page 129.

____. Understanding deep convolutional networks. **Phil. Trans. R. Soc. A**, The Royal Society, v. 374, n. 2065, p. 20150203, 2016. Citations on pages 26, 28, 32, 33, 34, 40, 41, and 129.

MELLO, R. F.; FERREIRA, M. D.; PONTI, M. A. Providing theoretical learning guarantees to Deep Learning Networks. **ArXiv e-prints**, Nov. 2017. Citations on pages 26, 28, 33, 41, 58, 100, and 109.

MELLO, R. F.; PONTI, M. A. **Machine Learning: A Practical Approach on the Statistical Learning Theory**. [S.l.]: Springer International Publishing, 2018. Citations on pages 26, 30, 34, 40, 41, 53, 57, 58, 72, 91, 92, 94, 96, 98, 99, 101, 102, 104, and 128.

MELLO, R. F.; PONTI, M. A.; FERREIRA, C. H. G. Computing the Shattering Coefficient of Supervised Learning Algorithms. **ArXiv e-prints**, May 2018. Citations on pages 26, 28, 33, 41, 58, 91, 98, 99, 100, and 109.

MINSKY, M. L.; PAPERT, S. A. **Perceptrons - Expanded Edition: An Introduction to Computational Geometry**. [S.l.]: MIT press Boston, MA:, 1987. Citations on pages 34 and 43.

MOHAMED, A.-r.; DAHL, G. E.; HINTON, G. Acoustic modeling using deep belief networks. **IEEE Transactions on Audio, Speech, and Language Processing**, IEEE, v. 20, n. 1, p. 14–22, 2012. Citation on page 35.

NAKASHIKA, T.; GARCIA, C.; TAKIGUCHI, T.; LYON, I. D. Local-feature-map integration using convolutional neural networks for music genre classification. In: **International Speech Communication Association (INTERSPEECH)**. [S.l.: s.n.], 2012. Citations on pages 27 and 107.

NELDER, J. A.; MEAD, R. A simplex method for function minimization. **The computer journal**, Oxford University Press, v. 7, n. 4, p. 308–313, 1965. Citation on page 40.

NENE, S.; NAYAR, S.; MURASE, H. **Columbia Object Image Library (coil 100) 1996**. [S.l.], 1996. Citation on page 74.

NESTEROV, Y. A method of solving a convex programming problem with convergence rate o (1/k2). In: . [S.l.: s.n.], 1983. v. 27, p. 372–376. Citation on page 54.

NG, J. Y.-H.; HAUSKNECHT, M.; VIJAYANARASIMHAN, S.; VINYALS, O.; MONGA, R.; TODERICI, G. Beyond short snippets: Deep networks for video classification. In: **Proceedings of the IEEE conference on computer vision and pattern recognition**. [S.l.: s.n.], 2015. p. 4694–4702. Citation on page 47.

NIU, X.-X.; SUEN, C. Y. A novel hybrid CNN–SVM classifier for recognizing handwritten digits. **Pattern Recognition**, Elsevier, v. 45, n. 4, p. 1318–1325, 2012. Citation on page 50.

NOWLAN, S. J.; HINTON, G. E. Simplifying neural networks by soft weight-sharing. **Neural computation**, MIT Press, v. 4, n. 4, p. 473–493, 1992. Citation on page 54.

ONO, J. H. P.; CORRÊA, D. C.; FERREIRA, M. D.; MELLO, R. F.; NONATO, L. G. Similarity graph: Visual exploration of song collections. In: **Electronic Proceedings of Sibgrapi 2015 (28th Conference on Graphics, Patterns and Images). 6th Workshop on Visual Analytics, Information Visualization and Scientific Visualization**. [S.l.: s.n.], 2015. Citation on page 28.

OQUAB, M.; BOTTOU, L.; LAPTEV, I.; SIVIC, J. Learning and transferring mid-level image representations using convolutional neural networks. In: IEEE. **Computer Vision and Pattern Recognition (CVPR), 2014 IEEE Conference on**. [S.l.], 2014. p. 1717–1724. Citations on pages 35 and 36.

ORHAN, U.; HEKIM, M.; OZER, M. EEG signals classification using the k-means clustering and a multilayer perceptron neural network model. **Expert Systems with Applications**, Elsevier, v. 38, n. 10, p. 13475–13481, 2011. Citation on page 34.

OSADCHY, M.; CUN, Y. L.; MILLER, M. L. Synergistic face detection and pose estimation with energy-based models. **The Journal of Machine Learning Research**, JMLR. org, v. 8, p. 1197–1215, 2007. Citation on page 36.

OSMALSKY, J.; EMBRECHTS, J.-J.; FOSTER, P.; DIXON, S. Combining features for cover song identification. In: **16th International Society for Music Information Retrieval Conference**. [S.l.: s.n.], 2015. Citation on page 28.

PAGLIOSA, L. C.; MELLO, R. F. Applying a kernel function on time-dependent data to provide supervised-learning guarantees. **Expert Systems with Applications**, Elsevier, v. 71, p. 216–229, 2017. Citations on pages 30, 62, 64, 65, and 66.

PAPYAN, V.; ROMANO, Y.; ELAD, M. Convolutional neural networks analyzed via convolutional sparse coding. **Journal of Machine Learning Research**, v. 18, n. 83, p. 1–52, 2017. Citations on pages 32, 40, and 41.

PAZZANI, M.; BILLSUS, D. Learning and revising user profiles: The identification of interesting web sites. **Machine learning**, Springer, v. 27, n. 3, p. 313–331, 1997. Citation on page 25.

PEARSON, K. LIII. on lines and planes of closest fit to systems of points in space. **The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science**, Taylor & Francis, v. 2, n. 11, p. 559–572, 1901. Citation on page 57.

PEREZ, L.; WANG, J. The effectiveness of data augmentation in image classification using deep learning. **arXiv preprint arXiv:1712.04621**, 2017. Citations on pages 26, 28, 33, 38, 41, 57, and 58.

POULTNEY, C.; CHOPRA, S.; CUN, Y. L. Efficient learning of sparse representations with an energy-based model. In: **Advances in neural information processing systems**. [S.l.: s.n.], 2006. p. 1137–1144. Citation on page 48.

RANZATO, M. A.; HUANG, F. J.; BOUREAU, Y.-L.; LECUN, Y. Unsupervised learning of invariant feature hierarchies with applications to object recognition. In: IEEE. **Computer Vision and Pattern Recognition, 2007. CVPR'07. IEEE Conference on**. [S.l.], 2007. p. 1–8. Citations on pages 35 and 48.

RASOULI, K.; HSIEH, W. W.; CANNON, A. J. Daily streamflow forecasting by machine learning methods with weather and climate inputs. **Journal of Hydrology**, Elsevier, v. 414, p. 284–293, 2012. Citation on page 25.

RHODES, C.; MORARI, M. The false nearest neighbors algorithm: An overview. **Computers & Chemical Engineering**, Elsevier, v. 21, p. S1149–S1154, 1997. Citation on page 62.

RIOS, R. A.; MELLO, R. F. Improving time series modeling by decomposing and analyzing stochastic and deterministic influences. **Signal Processing**, Elsevier, v. 93, n. 11, p. 3001–3013, 2013. Citations on pages 62, 64, and 66.

ROBBINS, H.; MONRO, S. A stochastic approximation method. **The annals of mathematical statistics**, JSTOR, p. 400–407, 1951. Citation on page 54.

ROOHULLAH, K.; JAAFAR, J. Semantic query expansion using knowledge based for images search and retrieval. **International Journal of Computer Science & Emerging Technologies**, IJSET, v. 2, n. 1, p. 1–5, 2011. Citation on page 25.

ROSENBLATT, F. The perceptron: A probabilistic model for information storage and organization in the brain. **Psychological review**, American Psychological Association, v. 65, n. 6, p. 386, 1958. Citation on page 34.

ROSENBLATT, M. A central limit theorem and a strong mixing condition. **Proceedings of the National Academy of Sciences**, National Acad Sciences, v. 42, n. 1, p. 43–47, 1956. Citation on page 91.

ROWEIS, S. T.; SAUL, L. K. Nonlinear dimensionality reduction by locally linear embedding. **Science**, American Association for the Advancement of Science, v. 290, n. 5500, p. 2323–2326, 2000. Citation on page 73.

ROWLEY, H.; BALUJA, S.; KANADE, T. Neural network-based face detection. **Pattern Analysis and Machine Intelligence, IEEE Transactions on**, IEEE, v. 20, n. 1, p. 23–38, 1998. Citation on page 25.

RUDER, S. An overview of gradient descent optimization algorithms. **arXiv preprint arXiv:1609.04747**, 2016. Citations on pages 54, 55, 56, and 57.

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**, v. 115, n. 3, p. 211–252, 2015. ISSN 1573-1405. Citations on pages 36 and 59.

SAK, H.; SENIOR, A.; BEAUFAYS, F. Long short-term memory recurrent neural network architectures for large scale acoustic modeling. In: **Fifteenth Annual Conference of the International Speech Communication Association**. [S.l.: s.n.], 2014. Citations on pages 34 and 35.

SAK, H.; SENIOR, A.; RAO, K.; BEAUFAYS, F. Fast and accurate recurrent neural network acoustic models for speech recognition. **arXiv preprint arXiv:1507.06947**, 2015. Citation on page 35.

SALAKHUTDINOV, R.; HINTON, G. Deep boltzmann machines. In: **Artificial Intelligence and Statistics**. [S.l.: s.n.], 2009. p. 448–455. Citation on page 35.

SAUER, N. On the density of families of sets. **Journal of Combinatorial Theory, Series A**, v. 13, n. 1, p. 145 – 147, 1972. ISSN 0097-3165. Available: <http://www.sciencedirect.com/science/article/pii/0097316572900192>. Citations on pages 98 and 99.

SCHERER, D.; MÜLLER, A.; BEHNKE, S. Evaluation of pooling operations in convolutional architectures for object recognition. In: **Artificial Neural Networks–ICANN 2010**. [S.l.]: Springer, 2010. p. 92–101. Citation on page 35.

SCHERER, D.; SCHULZ, H.; BEHNKE, S. Accelerating large-scale convolutional neural networks with parallel graphics multiprocessors. In: **Artificial Neural Networks–ICANN 2010**. [S.l.]: Springer, 2010. p. 82–91. Citation on page 35.

SCHLUTER, J.; BOCK, S. Improved musical onset detection with convolutional neural networks. In: IEEE. **Acoustics, Speech and Signal Processing (ICASSP), 2014 IEEE International Conference on**. [S.l.], 2014. p. 6979–6983. Citations on pages 25, 36, 43, 44, and 47.

SCHÖLKOPF, B.; SMOLA, A. J. **Learning with kernels: support vector machines, regularization, optimization, and beyond**. [S.l.]: MIT press, 2002. Citations on pages 26, 30, 50, and 99.

SERMANET, P.; EIGEN, D.; ZHANG, X.; MATHIEU, M.; FERGUS, R.; LECUN, Y. Overfeat: Integrated recognition, localization and detection using convolutional networks. In: **International Conference on Learning Representations (ICLR2014), CBLS, April 2014**. [S.l.: s.n.], 2014. Citations on pages 27, 33, and 107.

SERMANET, P.; LECUN, Y. Traffic sign recognition with multi-scale convolutional networks. In: IEEE. **Neural Networks (IJCNN), The 2011 International Joint Conference on**. [S.l.], 2011. p. 2809–2813. Citations on pages 43, 74, 76, 77, 78, 89, and 101.

SERRÀ, J.; ZANIN, M.; ANDRZEJAK, R. G. Cover song retrieval by cross recurrence quantification and unsupervised set detection. **MIREX 2009**, Music Information Retrieval Evaluation eXchange, 2009. Citation on page 28.

SEVERYN, A.; MOSCHITTI, A. Twitter sentiment analysis with deep convolutional neural networks. In: ACM. **Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval**. [S.l.], 2015. p. 959–962. Citations on pages 43 and 48.

SHANG, W.; SOHN, K.; ALMEIDA, D.; LEE, H. Understanding and improving convolutional neural networks via concatenated rectified linear units. In: **International Conference on Machine Learning**. [S.l.: s.n.], 2016. p. 2217–2225. Citation on page 58.

SHULBY, C. D.; FERREIRA, M. D.; MELLO, R. F.; ALUISIO, S. M. Acoustic modeling using a shallow CNN-HTSVM architecture. **2017 Brazilian Conference on Intelligent Systems**, p. 85–90, 2017. Citations on pages 90 and 109.

SHULBY, C. D.; FERREIRA, M. D.; MELLO, R. F. de; ALUISIO, S. M. Theoretical learning guarantees applied to acoustic modeling. **Journal of the Brazilian Computer Society**, v. 25, Jan 2019. ISSN 1678-4804. Available: <https://doi.org/10.1186/s13173-018-0081-3>. Citation on page 109.

SIMONYAN, K.; ZISSERMAN, A. Very deep convolutional networks for large-scale image recognition. **arXiv preprint arXiv:1409.1556**, 2014. Citations on pages 30, 34, 36, 38, and 59.

SIVAKUMAR, B.; JAYAWARDENA, A.; FERNANDO, T. River flow forecasting: use of phase-space reconstruction and artificial neural networks approaches. **Journal of hydrology**, Elsevier, v. 265, n. 1-4, p. 225–245, 2002. Citation on page 29.

SMOLA, A. J.; SCHÖLKOPF, B. **Learning with kernels**. [S.l.]: Citeseer, 1998. Citation on page 53.

SMOLENSKY, P. **Information processing in dynamical systems: Foundations of harmony theory**. [S.l.], 1986. Citation on page 35.

SOCIETY, I. N. N. **The International Joint Conference on Neural Networks (IJCNN)**. [S.l.], 1989 (accessed September 27, 2018). Available: <https://www.ijcnn.org/>. Citations on pages 31 and 74.

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**, JMLR. org, v. 15, n. 1, p. 1929–1958, 2014. Citations on pages 26, 28, 30, 33, 38, 39, 40, 41, 58, 74, 75, 76, 77, 78, 89, 103, 104, and 107.

SRIVASTAVA, N.; SALAKHUTDINOV, R.; HINTON, G. Modeling documents with a deep boltzmann machine. In: **Proceedings of the Twenty-Ninth Conference on Uncertainty in Artificial Intelligence**. [S.l.]: AUAI Press, 2013. (UAI'13), p. 616–624. Citations on pages 34 and 84.

STALLKAMP, J.; SCHLIPSING, M.; SALMEN, J.; IGEL, C. Man vs. computer: Benchmarking machine learning algorithms for traffic sign recognition. **Neural networks**, Elsevier, v. 32, p. 323–332, 2012. Citation on page 74.

STRANG, G. **Linear Algebra and Its Applications**. [S.l.]: Brooks Cole, 1988. Citation on page 70.

SZEGEDY, C.; LIU, W.; JIA, Y.; SERMANET, P.; REED, S.; ANGUELOV, D.; ERHAN, D.; VANHOUCKE, V.; RABINOVICH, A. Going deeper with convolutions. In: **Proceedings of the IEEE conference on computer vision and pattern recognition**. [S.l.: s.n.], 2015. p. 1–9. Citations on pages 26, 27, 28, 30, 33, 38, 39, 40, 70, 76, 78, and 107.

TAIGMAN, Y.; YANG, M.; RANZATO, M.; WOLF, L. Deepface: Closing the gap to human-level performance in face verification. In: **Proceedings of the IEEE conference on computer vision and pattern recognition**. [S.l.: s.n.], 2014. p. 1701–1708. Citation on page 49.

TAKENS, F. Detecting strange attractors in turbulence. **Lecture notes in mathematics**, Springer, v. 898, n. 1, p. 366–381, 1981. Citations on pages 29, 32, 51, 61, 62, 65, and 73.

TEAM, D. D. **Deeplearning4j: Open-source distributed deep learning for the JVM, Apache Software Foundation License 2.0.** 2017. Available: <http://deeplearning4j.org>. Citations on pages 36, 58, and 59.

TELLER, A.; VELOSO, M. Algorithm evolution for face recognition: what makes a picture difficult. In: IEEE. **Evolutionary Computation, 1995., IEEE International Conference on**. [S.l.], 1995. v. 2, p. 608–613. Citations on pages 78, 88, and 89.

TIELEMAN, T.; HINTON, G. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. **COURSERA: Neural networks for machine learning**, v. 4, n. 2, p. 26–31, 2012. Citation on page 55.

VAPNIK, V. **The nature of statistical learning theory**. [S.l.]: Springer science & business media, 2000. Citations on pages 26, 30, 34, 72, 93, 94, 95, 96, 97, 98, 100, and 101.

VAPNIK, V. N.; CHERVONENKIS, A. Y. Theory of pattern recognition. Nauka, 1974. Citations on pages 95, 96, and 97.

____. On the uniform convergence of relative frequencies of events to their probabilities. In: **Measures of complexity**. [S.l.]: Springer, 2015. p. 11–30. Citations on pages 30, 40, 53, 58, 94, 95, 96, 97, 98, and 99.

VITÁNYI, P.; LI, M. Minimum description length induction, bayesianism, and kolmogorov complexity. In: CITESEER. **IEEE Transactions on Information Theory**. [S.l.], 1996. Citation on page 30.

WERBOS, P. J. Generalization of backpropagation with application to a recurrent gas market model. **Neural Networks**, Elsevier, v. 1, n. 4, p. 339–356, 1988. Citations on pages 34 and 43.

WESTON, J.; RATLE, F.; MOBAHI, H.; COLLOBERT, R. Deep learning via semi-supervised embedding. In: ____. **Neural Networks: Tricks of the Trade: Second Edition**. [S.l.]: Springer Berlin Heidelberg, 2012. p. 639–655. ISBN 978-3-642-35289-8. Citation on page 74.

YAN, H.; JIANG, Y.; ZHENG, J.; PENG, C.; LI, Q. A multilayer perceptron-based medical decision support system for heart disease diagnosis. **Expert Systems with Applications**, v. 30, n. 2, p. 272 – 281, 2006. ISSN 0957-4174. Citation on page 34.

YANG, M.-H.; ROTH, D.; AHUJA, N. Learning to recognize 3d objects with snow. In: SPRINGER. **European Conference on Computer Vision**. [S.l.], 2000. p. 439–454. Citations on pages 78 and 89.

YANG, S.; LUO, P.; LOY, C.-C.; TANG, X. From facial parts responses to face detection: A deep learning approach. In: **Proceedings of the IEEE International Conference on Computer Vision**. [S.l.: s.n.], 2015. p. 3676–3684. Citation on page 28.

YOUNG, S. R.; ROSE, D. C.; KARNOWSKI, T. P.; LIM, S.-H.; PATTON, R. M. Optimizing deep learning hyper-parameters through an evolutionary algorithm. In: ACM. **Proceedings of the Workshop on Machine Learning in High-Performance Computing Environments**. [S.l.], 2015. p. 4. Citations on pages 28, 32, 40, 70, and 107.

ZEILER, M. D. Adadelta: an adaptive learning rate method. **arXiv preprint arXiv:1212.5701**, 2012. Citation on page 56.

ZEILER, M. D.; FERGUS, R. Visualizing and understanding convolutional networks. In: SPRINGER. **European conference on computer vision**. [S.l.], 2014. p. 818–833. Citations on pages 27, 33, 37, 38, and 107.

ZHU, C.; ZHENG, Y.; LUU, K.; LE, T. H. N.; BHAGAVATULA, C.; SAVVIDES, M. Weakly supervised facial analysis with dense hyper-column features. In: **Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops**. [S.l.: s.n.], 2016. p. 25–33. Citations on pages 43, 78, 88, 89, 103, and 104.

ZOU, W.; ZHU, S.; YU, K.; NG, A. Y. Deep learning of invariant features via simulated fixations in video. In: **Advances in neural information processing systems**. [S.l.: s.n.], 2012. p. 3212–3220. Citations on pages 78, 89, and 101.

Appendix

APPENDIX

A

STUDYING THE GENERALIZATION OF CONVOLUTIONAL NEURAL NETWORKS

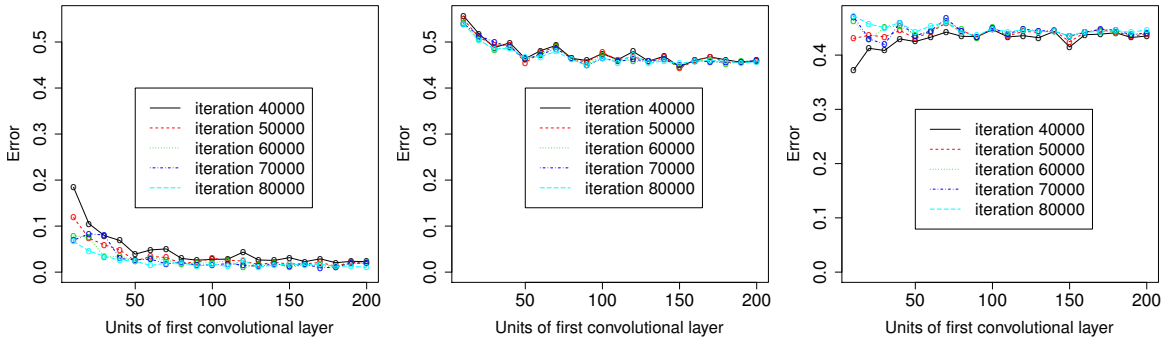
This appendix presents additional experiments on the CIFAR-10 dataset, complementing Section 5.4. Such experiments were necessary due to issues observed in this challenging object recognition dataset composed of 32×32 tiny colored images with complex structures. Additional experiments were performed to analyze the generalization of Convolutional Neural Networks (CNN) while the number of CNN units per convolutional layer was increased. In such situation, the CNN was executed with a learning rate equals to 0.001, a momentum of 0.9, weight decay as 0.004, the convolutional mask size was set to 6×5, and subsampling was configured with 3×3 along $80,000$ iterations.

According to Statistical Learning Theory (SLT), the generalization is given by the difference between the expected and the empirical risks. As the expected risk requires all possible samples to be calculated, which are infinite for most real-world scenarios, it can be represented by another empirical risk computed on a ghost sample given the Symmetrization Lemma (LUXBURG; SCHÖLKOPF, 2011; DEVROYE; GYÖRFI; LUGOSI, 1997). In that scenario, two different independent samples are considered, but having the same sizes. One might associate the first sample to the training set and the second, or the ghost sample, to the test set, what is indeed a simpler manner of interpreting the whole formulation after the SLT. In our scenarios, training and test sets were build up with $10,000$ different and independent images to obtain a fair divergence between them which expresses the generalization itself (for more details please refer to Section 6.2).

The learning generalization capacities of four CNN architectures under two subsampling strategies were analyzed while incrementing the number of convolutional units, having mask sizes parametrized by IFNN. In this scenario, the IFNN mask-size estimation performed at the first convolutional layer (6×5) was also employed on the additional ones, while analyzing the CIFAR-10 training set (see Section 5.3). Error rates and the difference among empirical risks (an

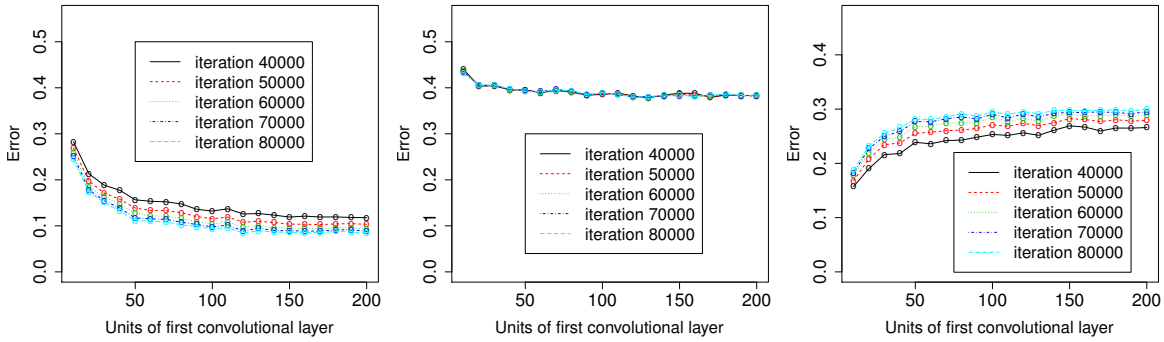
estimation of the generalization capacity (MELLO; PONTI, 2018)) are illustrated from Figure 24 to 36 in terms of each CNN setting. The y-axis of those figures corresponds to the error rate and the x-axis is associated with the number of convolutional units at the first layer. Each curve represents a snapshot under different numbers of iterations: $40,000$, $50,000$, $60,000$, $70,000$, and $80,000$.

The first architecture was composed of a single convolutional layer without subsampling whose results are presented in Figure 24, increasing the number of units up to 200 with a step of 10. Figures 25 and 26 illustrate the results for the same architecture but adding a subsampling layer. Figure 25 is associated to a network architecture with a max-pooling layer right after the convolutional one, while Figure 26 is resultant from an average-pooling layer.



(a) Error of the training set. (b) Error of the test set. (c) Error divergences.

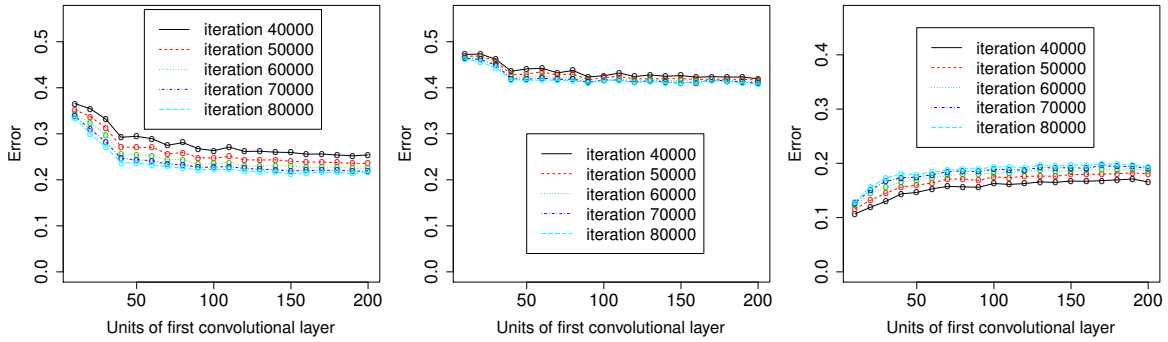
Figure 24 – Estimating the generalization capacity of a single-convolutional-layer CNN by measuring the divergence among empirical risks while varying the number of units in range $[10, 200]$ with steps of 10.



(a) Error of the training set. (b) Error of the test set. (c) Error divergences.

Figure 25 – Estimating the generalization capacity of a single-convolutional-layer CNN composed with max-pooling subsampling in order to measure the divergence among empirical risks while varying the number of convolutional units in range $[10, 200]$ with steps of 10.

The architecture without the subsampling layer produced a soon effect of overfitting which just after $40,000$ iterations achieved empirical risk divergences greater than 0.4 (40%),



(a) Error of the training set. (b) Error of the training set. (c) Error divergences.

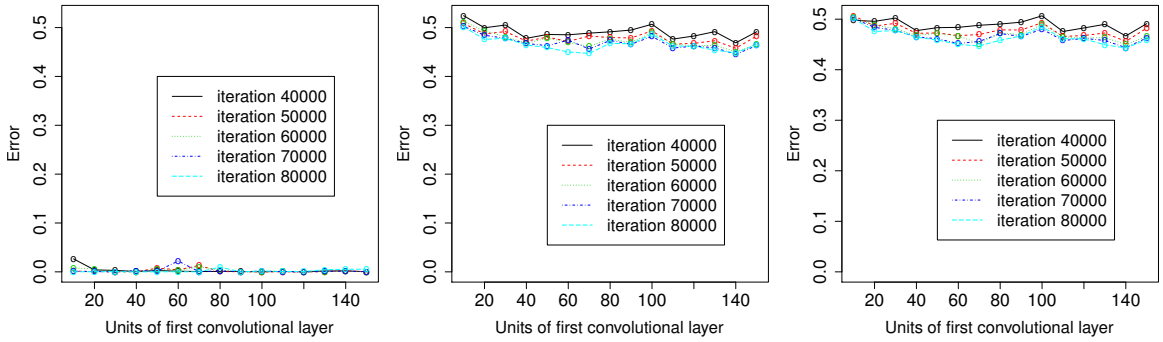
Figure 26 – Estimating the generalization capacity of a single-convolutional-layer CNN composed with average-pooling subsampling in order to measure the divergence among empirical risks while varying the number of convolutional units in range $[10, 200]$ with steps of 10.

meaning generalization was jeopardized. In case of architectures under subsampling, max and average-pooling layers improved the learning generalization for CIFAR-10, thus avoiding overfitting. The CNN with average-pooling (Figure 26) produced a better generalization, being around 0.15 for most of the units assessed. In addition, we expected the presence of overfitting while the number of units was incremented, resultant from error divergences between training and test sets. However, errors were somehow stable, thus indicating the need of more units to reach such perturbation threshold or even more layers.

Next, architectures with two convolutional layers were evaluated in order to observe changes in the network complexity. According to Mallat (2016), an adequate number of units in a hidden layer is associated to the number of units at a previous layer but multiplied by a power of 2, once the CNN multilayer structure is similar to the multiresolution filtering designed using the Wavelet transform (MALLAT, 1999). Considering this assumption, we defined the number of units at the second convolutional layer to be equal to the number of units at the first but multiplied by 2.

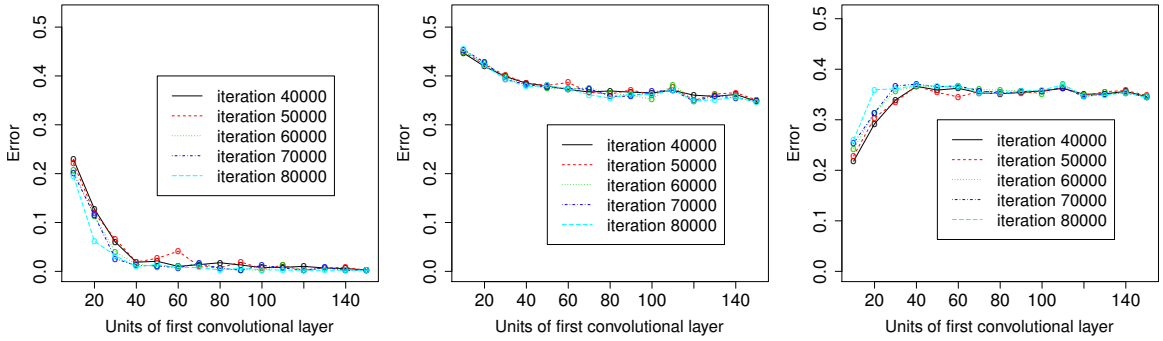
Figure 27 presents the error rate obtained for an architecture composed of two convolutional layers without subsampling, in which the number of units at the first convolutional layer increases up to 150 in steps of 10. Figures 28 and 29 illustrate the results for the same network configuration, but Figure 28 considers an architecture interposed by max-pooling layers, while Figure 29 takes into account the average subsampling.

The CNN architecture with two convolutional layers without subsampling presents a similar behavior to that of a single convolutional layer also with no subsampling. In both scenarios, overfitting is evident in few iterations what is confirmed by the empirical risk divergence of 0.5. Such result allows us to conclude that CNN architectures with no subsampling are more prone to overfitting, i.e., they tend to be unable to generalize, consequently they do not infer adequate learning models for CIFAR-10.



(a) Error of the training set. (b) Error of the test set. (c) Error divergences.

Figure 27 – Estimating the generalization capacity of a 2-convolutional-layer CNN without subsampling in order to measure the divergence among empirical risks while varying the number of convolutional units at the first layer in range $[10, 150]$, in steps of 10, and the following layer with twice as many units.

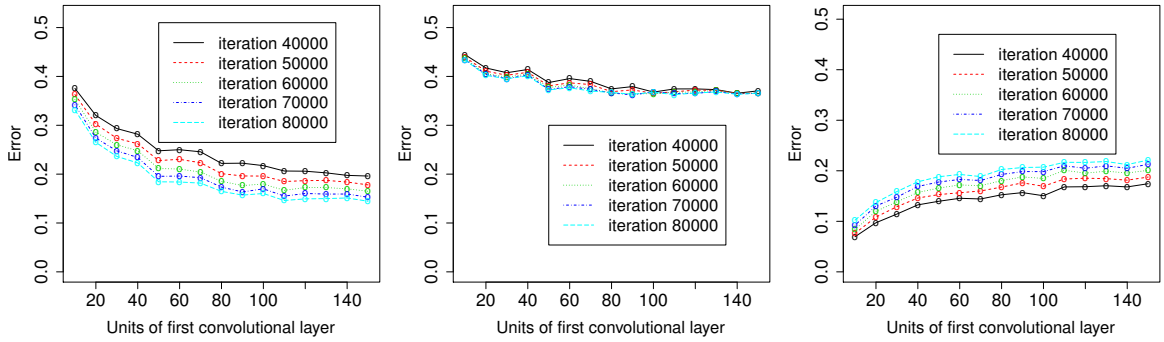


(a) Error of the training set. (b) Error of the test set. (c) Error divergences.

Figure 28 – Estimating the generalization capacity of a 2-convolutional-layer CNN with max-pooling subsampling in order to measure the divergence among empirical risks while varying the number of convolutional units at the first layer in range $[10, 150]$, in steps of 10, and the following layer with twice as many units.

In case of CNN with two convolutional layers interposed by max-pooling layers, the overfitting occurs when the number of units is greater than 30, thus providing a threshold for the units as formulated in Section 3.2.1. In addition, this threshold confirms our conclusions drawn in Section 4.3.1 from which the number of units should be equal or smaller than the size of convolution masks ($5 \times 6 = 30$) in order to provide the necessary space transformation. The best result for CIFAR-10 with two convolutional layers was obtained in conjunction with the average-pooling subsampling, given the divergence among empirical risks is smaller than the others.

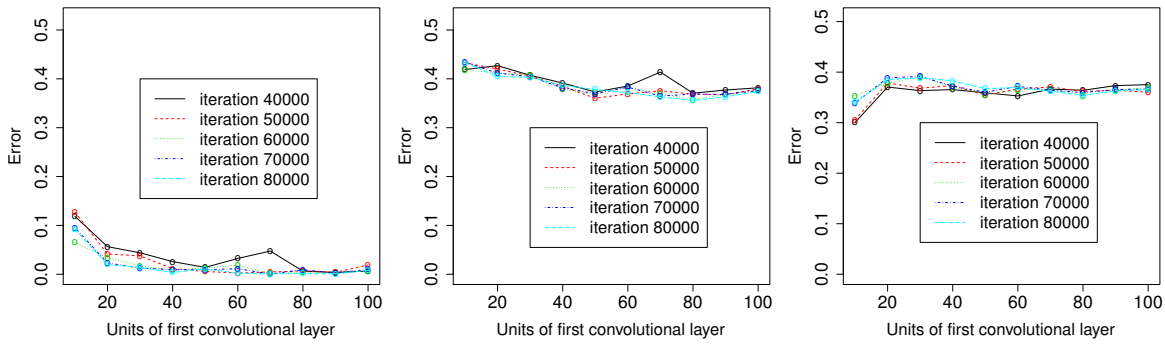
Based on these results, we decided to include some data overlapping along subsampling layers when two convolutional layers were considered. This approach aims to evaluate the



(a) Error of the training set. (b) Error of the test set. (c) Error divergences.

Figure 29 – Estimating the generalization capacity of a 2-convolutional-layer CNN with average-pooling subsampling in order to measure the divergence among empirical risks while varying the number of convolutional units at the first layer in range $[10, 150]$, in steps of 10, and the following layer with twice as many units.

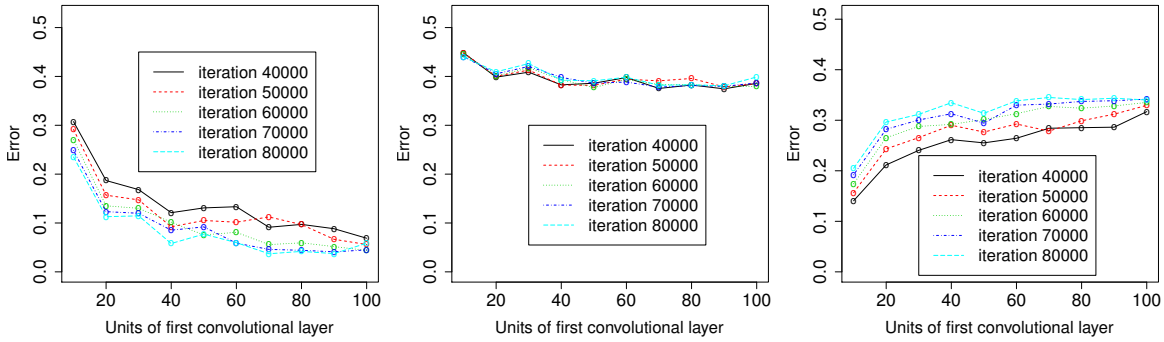
training behavior in the presence of additional features, increasing the amount of information manipulated by the network layers. In such scenario, our architectures with 3×3-subsampling layers and 2×2 stride produced the results illustrated in Figures 30 and 31, in which the number of units at the first convolutional layer increases up to 100 in steps of 10. Figure 30 illustrates the results using max-pooling layers and Figure 31 using average-pooling layers. In both cases, the subsampling overlapping did not improve the generalization capacity, also leading to overfitting. Such behavior indicates that the network requires features at lower dimensionality, confirming the subsampling masks need to be properly estimated, something out of the scope of this thesis.



(a) Error of the training set. (b) Error of the test set. (c) Error divergences.

Figure 30 – Estimating the generalization capacity of a 2-convolutional-layer CNN with an overlapping of 2×2 in max-pooling subsampling in order to measure the divergence among empirical risks while varying the number of convolutional units at the first layer in range $[10, 100]$, in steps of 10, and the following layer with twice as many units.

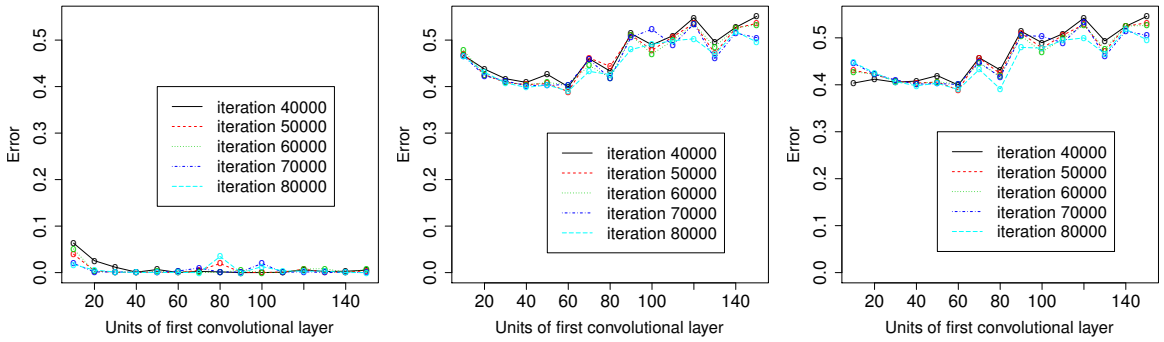
Next, an architecture with three convolutional layers was evaluated, in which the number of units at the third layer doubles the units at the second one, which contains twice as many



(a) Error of the training set. (b) Error of the test set. (c) Error divergences.

Figure 31 – Estimating the generalization capacity of a 2-convolutional-layer CNN with an overlapping of 2×2 in average-pooling subsampling in order to measure the divergence among empirical risks while varying the number of convolutional units at the first layer in range $[10, 100]$, in steps of 10, and the following layer with twice as many units.

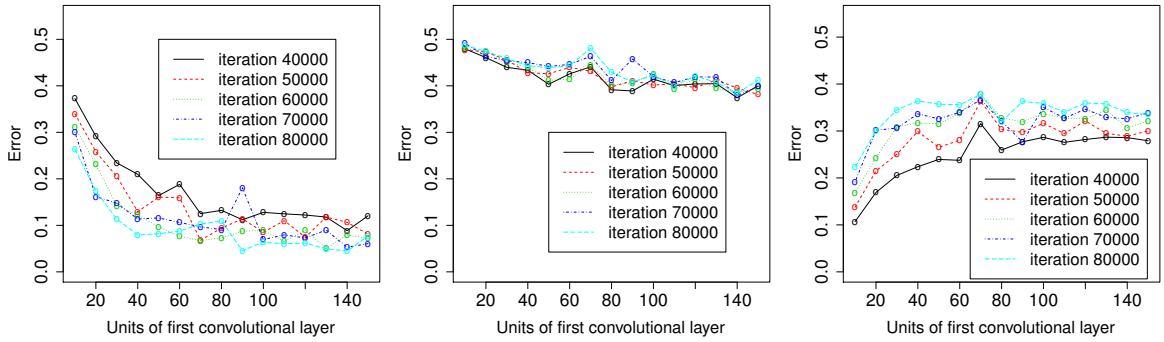
units as the first layer. In this scenario, we did not assess architectures without subsampling due to the overfitting obtained with previous similar architectures. Figures 32 and 33 present the results obtained for an architecture composed of three convolutional layers interposed with subsampling layers, in which Figure 32 corresponds to max-pooling and Figure 33 to average-pooling subsampling. In both figures, the number of units at the first convolutional layer increases up to 150, in steps of 10.



(a) Error of the training set. (b) Error of the test set. (c) Error divergences.

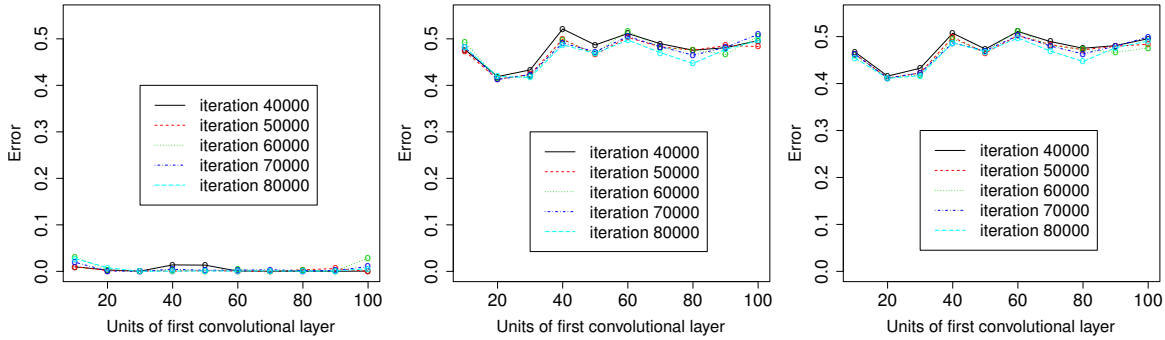
Figure 32 – Estimating the generalization capacity of a 3-convolutional-layer CNN with max-pooling subsampling in order to measure the divergence among empirical risks while varying the number of convolutional units at the first layer in range $[10, 150]$, in steps of 10, and the following layers with twice as many units as the previous one.

In addition, CNNs with three convolutional layers were also evaluated using a 2×2 stride at the subsampling layers. Figure 34 presents the results considering max-pooling and Figure 31 using average-pooling layers. In both circumstances, the number of units at the first convolutional layer increases up to 100 in steps of 10.



(a) Error of the training set. (b) Error of the test set. (c) Error divergences.

Figure 33 – Estimating the generalization capacity of a 3-convolutional-layer CNN with average-pooling subsampling in order to measure the divergence among empirical risks while varying the number of convolutional units at the first layer in range $[10, 150]$, in steps of 10, and the following layers with twice as many units as the previous one.

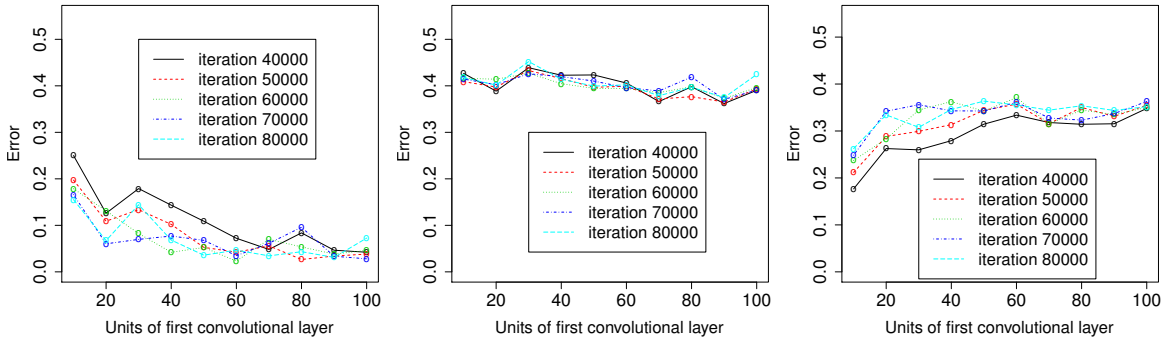


(a) Error of the training set. (b) Error of the test set. (c) Error divergences.

Figure 34 – Estimating the generalization capacity of a 3-convolutional-layer CNN with an overlapping of 2×2 in max-pooling subsampling in order to measure the divergence among empirical risks while varying the number of convolutional units at the first layer in range $[10, 100]$, in steps of 10. Every following layer employs twice as many units as its precedent.

Considering the results obtained for CNNs with three convolutional layers, the error rates obtained during tests do not show a significant difference while compared to previous performances. However, the generalization capacity estimated for those architectures gave us some insights. Both CNN architectures composed of max-pooling subsampling with and without overlapping present a poor generalization and overfitting performances, as expected according to previous results. In addition, the overlapping while using max-pooling accelerated the occurrence of overfitting, which happened with few units.

On the other hand, both architectures composed of average-pooling layers with and without overlapping avoided overfitting and provided good generalization indices. However, overlapping has still jeopardized the generalization capacity, confirming such a network requires



(a) Error of the training set. (b) Error of the test set. (c) Error divergences.

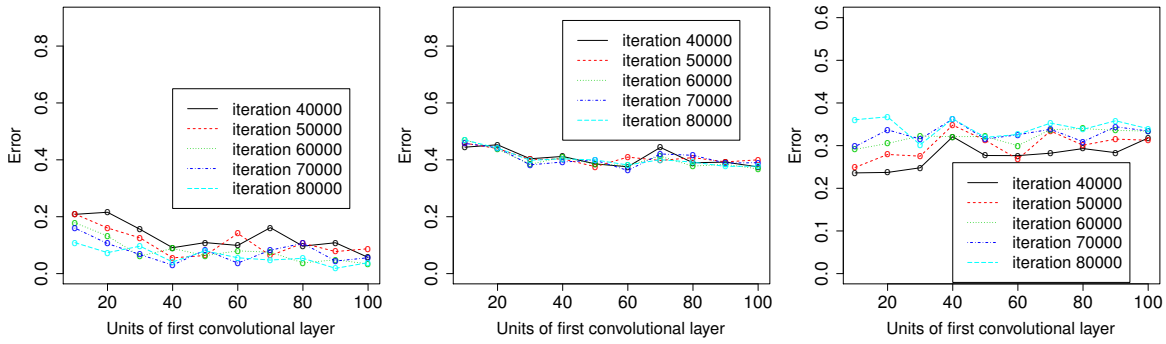
Figure 35 – Estimating the generalization capacity of a 3-convolutional-layer CNN with an overlapping of 2×2 in average-pooling subsampling in order to measure the divergence among empirical risks while varying the number of convolutional units at the first layer in range $[10, 100]$, in steps of 10. Every following layer employs twice as many units as its precedent.

features at lower dimensionality to improve classification. Results confirm that average-pooling is the most adequate subsampling operation to tackle CIFAR-10.

Lastly, one experiment was performed using four convolutional layers interposed with 3×3 average-pooling layers (layers double units as previously seen) with a stride of 2×2, as reported in Figure 36. The number of units in the first layer was increased up to 100, in steps of 10. It is worth to mention that we decided to employ overlapping once the setting with four consecutive subsampling layers has excessively reduced the dimensionality of features and also because such operation did not significantly jeopardize results.

In this scenario, architectures with four convolutional layers present a good generalization capacity, however results were very similar to the ones obtained with three convolutional layers in conjunction with average-pooling, suggesting CIFAR-10 may require other operations, such as layer types, or different parameters to improve the classification performance.

In summary, we observed that CIFAR-10 requires more units than estimated by IFNN and most probably more layers. Even so, in the most architecture settings, generalization is fairly stable after 60 units at the first convolutional layer, confirming the possibility of estimating such number but using a different strategy. CIFAR-10 also provided better convergence guarantees with the use of the average-pooling subsampling (it somehow avoided overfitting). Such aspect shows that CNNs still require additional studies and analysis on subsampling components, such as the operation itself, the mask and stride sizes, once they all impact generalization. Finally, the overlapping operation applied on the subsampling layer did not significantly jeopardize the overall results, given it kept and passed along more information for next layers, thus considering a slow reduction in the dimensionality of features.



(a) Error of the training set. (b) Error of the test set. (c) Error divergences.

Figure 36 – Estimating the generalization capacity of a 4-convolutional-layer CNN with an overlapping of 2×2 in average-pooling subsampling in order to measure the divergence among empirical risks while varying the number of convolutional units at the first layer in range $[10, 100]$, in steps of 10. Every following layer employs twice as many units as its precedent.


ICMC
USP
SÃO CARLOS