Universidade de Aveiro Departamento de Eletrónica, Telecomunicações e Informática
2017

João António Pereira Correia

Sistema Baseado em Técnicas de Compressão para o Reconhecimento de Dígitos Manuscritos

System Based on Compression Techniques for the Recognition of Handwritten Digits

Universidade de Aveiro Departamento de
Eletrónica, Telecomunicações e Informática
2017

João António Pereira Correia

Sistema Baseado em Técnicas de Compressão para o Reconhecimento de Dígitos Manuscritos

System Based on Compression Techniques for the Recognition of Handwritten Digits

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Engenharia Eletrónica e Telecomunicações, realizada sob a orientação científica do Prof. Dr. Armando José Formoso de Pinho, Professor do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro.

agradecimentos / acknowledgements

Agradeço ao Prof. Dr. Armando J. Pinho por os seus conselhos e pela liberdade que me permitiu ao longo deste trabalho.
Agradeço também aos meus pais por me terem dado esta oportunidade e a minha família pelo apoio.

I would like to thank Prof. Dr. Armando J. Pinho for his expert advice and freedom throughout this dissertation.
I would like to thank to my parents for giving me this opportunity and my family for the support.

Palavras Chaves

Reconhecimento de Padrões, Reconhecimento de Dígitos Manuscritos, Algoritmos de Classificação, MNIST, Compressão de Dados, Compressão de Imagem, Modelos de Contexto Finito

Resumo

O reconhecimento de dígitos manuscritos é uma habilidade humana adquirida. Com pouco esforço, um humano pode reconhecer adequadamente em milissegundos uma sequência de dígitos manuscritos. Com o auxílio de um computador, esta tarefa de reconhecimento pode ser facilimente automatizada, melhorando um número significativo de processos. A separação do correio postal, a verificação de cheques bancários e operações que têm como entrada de dados dígitos manuscritos estão incluídas num amplo conjunto de aplicações que podem ser realizadas de forma mais eficaz e automatizada. Nos últimos anos, várias técnicas e métodos foram propostos para automatizar o mecanismo de reconhecimento de dígitos manuscritos. No entanto, para resolver esta desafiante questão de reconhecimento de imagem são utilizadas técnicas complexas e computacionalmente muito exigentes de *machine learning*, como é o caso do *deep learning*. Nesta dissertação é introduzida uma nova solução para o problema do reconhecimento de dígitos manuscritos, usando métricas de similaridade entre imagens de dígitos. As métricas de similaridade são calculadas com base na compressão de dados, nomeadamente pelo uso de Modelos de Contexto Finito.

Keywords Pattern Recognition, Handwritten Digit Recognition, Classification Algorithms, MNIST, Data Compression, Image Compression, Finite Context Model

Abstract

The Recognition of Handwritten Digits is a human-acquired ability. With little effort, a human can properly recognize, in milliseconds, a sequence of handwritten digits. With the help of a computer, the task of handwriting recognition can be easily automated, improving and making a significant number of processes faster. The postal mail sorting, bank check verification and handwritten digit data entry operations are in a wide group of applications that can be performed in a more effective and automated way. In the recent past years, a number of techniques and methods have been proposed to automate the handwritten digit recognition mechanism. However, to solve this challenging question of image recognition, there are used complex and computationally demanding *machine learning* techniques, as it is the case of *deep learning*. In this dissertation is introduced a novel solution to the problem of handwritten digit recognition, using metrics of similarity between digit images. The metrics are computed based on data compression, namely by the use of Finite Context Models.

Contents

List of Figures

List of Tables

Acronyms

AC	Alternating Current
ANN	Artificial Neural Network
ASCII	American Standard Code for Information Interchange
CNN	Convolutional Neural Network
CV	Computer Vision
DC	Direct Current
DCT	Discrete Cosine Transform
DL	Deep Learning
FCM	Finite Context Model
GIF	Graphics Interchange Format
GPU	Graphics Processing Unit
JBIG	Joint Bilevel Image Processing Group
JPEG	Joint Photographic Experts Group
k-NN	k-Nearest Neighbors
LZ77	Lempel-Ziv Coding 1977
LZ78	Lempel-Ziv Coding 1978
LZW	Lempel-Ziv-Welch Coding
ML	Machine Learning
MNIST	Modified National Institute of Standards and Technology
NCCD	Normalized Conditional Compression Distance
NCD	Normalized Compression Distance
NID	Normalized Information Distance
NIST	National Institute of Standards and Technology

NN	Neural Network
NNS	Nearest-Neighbor Scaling
NRC	Normalized Relative Compression
PCA	Principal Component Analysis
PNG	Portable Network Graphics
PR	Pattern Recognition
RBF	Radial Basis Function
ReLU	Rectified Linear Unit
RLE	Run-length Encoding
SD-1	Special Database 1
SD-3	Special Database 3
SVD	Single Value Decomposition
SVM	Support Vector Machine

Chapter 1

Introduction

This chapter starts by describing the handwriting recognition and data compression concepts, followed by an exposition on how the two are interconnected. This produces a better understanding of the concepts used in the implemented work. Afterwards, the motivations, objectives and contributions are presented. Lastly, an outline of the thesis structure is introduced.

1.1 Overview

Pattern Recognition (PR) is a noted research field with a wide and continuous scientific attention and significant growing application areas, which was introduced in the early 1960's [1][2]. It quickly became related to Perception, due to the identical underlying procedures of both fields. In psychology and cognitive sciences, Perception is described as the method of acquiring, interpreting, selecting, and organizing sensory information [3]. Over the years, several definitions have been proposed for PR, being that nowadays is mostly described as the method for acquiring, processing, extracting features and classifying the patterns of the input data, as represented in Figure 1.1 [4]. Therefore, it is assumed that the core of both processes is similar.

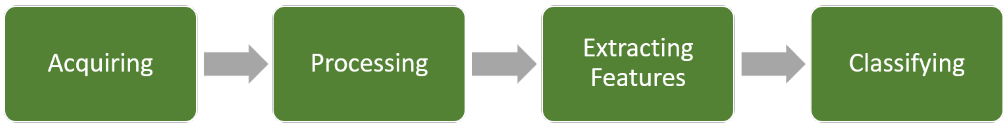


Figure 1.1: Stages of the Pattern Recognition Process.

In a generic PR procedure, a technical device collects the data or sensitive information and assigns it to pre-established classes. The ultimate goal in PR is to design an automatic system capable of executing the respective exigent task. The feature extraction and classification stages are the most relevant processing modules of a functioning PR system, since they are the key points in order to achieve a proper performance. One major difference between these two modules consists on the possibility of being re-applied to different applications/scenarios. While the classification algorithms can be re-applied, the same does not happen with the feature extraction modules, which are unique to every scenario. Figure 1.2 depicts an example of the feature extraction module of the digit 3.

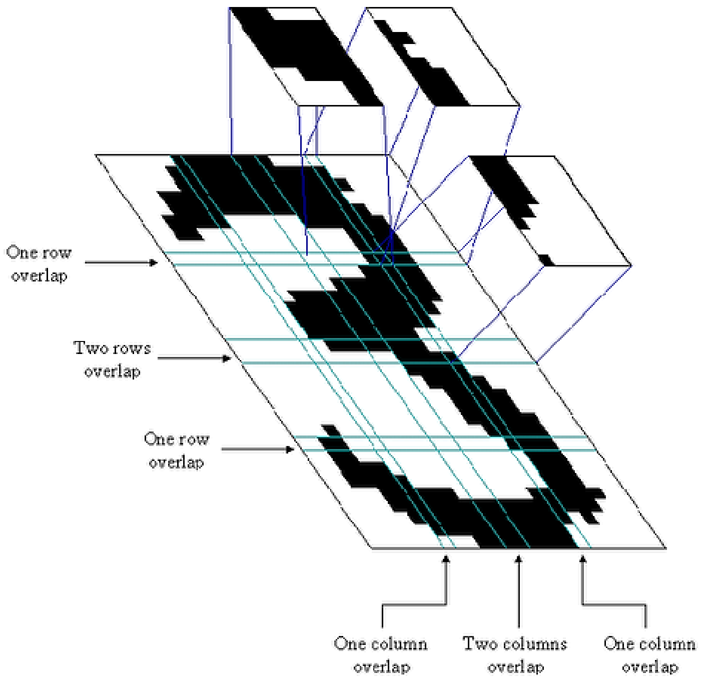


Figure 1.2: Feature Extraction of Digit 3 [5].

An extensive number of classification algorithms were developed in the last two decades with the research on automated recognition. The advances in hardware technology led to the study of new computational and more demanding pattern classification algorithms. The improvements in computer hardware allowed recognition systems to store larger models and more complex prototypes. Consequently, former developed classification algorithms considered too complex in the past, became achievable due to the tremendous evolution in computer hardware [6].

One of the classification algorithms that had an exponential growth with the development in computer hardware is the Artificial Neural Network (ANN) scheme, with applications in several different areas, such as the industrial process and financial control, but also in the PR field [7].

The underlying concept of an ANN scheme is to reproduce the biological nervous system procedure of handling information [8]. This outstanding information processing paradigm is a combination of a multitude of highly interconnected processing components. The basic elements, neurons, work in cooperation between them to solve the proposed problem. Similarly like people, the ANN learns by practice and experience [9]. The learning process in natural systems is described as changing the effectiveness on the synapses connections, affecting the influence of one neuron on another [10]. A similar process occurs in the ANN. Figure 1.3 depicts the structural differences between the biological network and the ANN.

The ANN has the capacity to learn by giving examples without task-specific programming, possessing the ability to resolve tasks automatically. For example, in image recognition, an ANN can learn to classify images that contain giraffes by examining sample pictures that have been previously labeled as “giraffe” or “not giraffe”, using collected analytic data to identify giraffes in other figures [11]. The ANN established their ground while the traditional rule-based programming had difficulty succeeding.

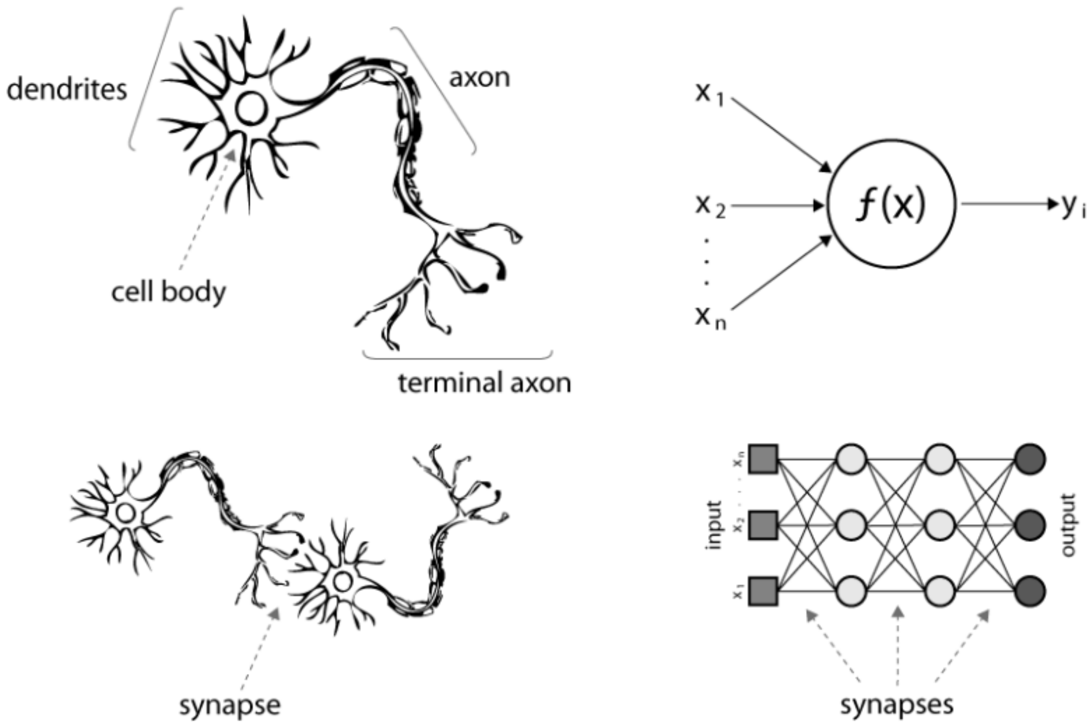


Figure 1.3: Comparison of a Biological Network and an ANN [12].

Although the recent benefits presented about ANN algorithms, these do not offer a trivial solution to all the artificial perception problems. The processing and feature extraction PR stages, used in an ANN, must be also thoroughly delineated. In addition, recent statements on ANN showed some drawbacks of this sophisticated technique. Therefore, the most expressed disadvantage of an ANN is its ”black box” nature, *i.e.*, in the absence of additional effort, it is extremely difficult, if not impracticable, to obtain an insight of the problem when looking into the ANN model [13].

The training performed by an ANN, in order to achieve a suitable architecture, is by itself an expensive computational burden [14]. In an ANN, the learning process requires enormous amounts of corrected labeled data to train. The time needed to train a very complex model can be in the scale of weeks, using hundreds of machines equipped with expensive Graphics Processing Units (GPUs) [15].

The limitations of such architecture are also present in most Machine Learning (ML) techniques. In order to overcome these exigent computational challenges, new PR approaches have been studied. PR schemes based on data compression have shown interesting results, since all the recognition and classification structures rely on compressed reflections of the input data, showing that compression may represent a new paradigm on PR [16].

The science of representing information in a compact form, over the uncompressed or original form, is known as data compression [17]. The human nature has the remarkable ability to compress information. An example of this ability is the act of capturing an image by the light sensitive cells in the retina of an eye, where the captured visual photo's size is the order of a megapixel. The memory and transmission capacities of the brain are limited, making it impossible to deal with a lifetime of megapixel pictures. Consequently, the brain has to select the most important information to be able to understand the surrounding world [18].

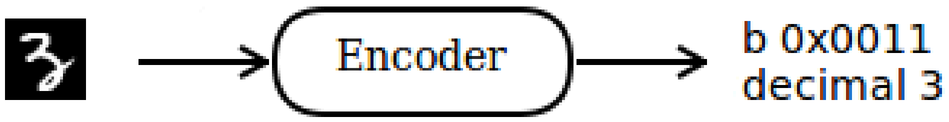


Figure 1.4: Recognition System Based on a Data Compression Encoder.

The task of a recognition system can be seen as a lossy compression system, *i.e.*, the recognition procedure follows a task order for the final purpose of assigning a classification to the input data. Similarly, a lossy compression system tries to compress the input data with the goal of attaining a good degree of compression without losing the perception of the original data. In the recognition of handwritten digits, the aim is to assign a correct classification to a digit image, resembling to the process of a compressor, where the input data is compressed to a binary representation of a digit. Figure 1.4 represents a type of data compression encoder that classifies digits.

1.2 Motivation and Objectives

The advances in technology revealed more efficient ways of doing tasks and the better emerged processes showed profitable results. The technological advances of computers enabled progresses in many areas such as education, agriculture and in the medical area. Nowadays, a student can learn on a worldwide scale without leaving the classroom, where the information is just a click away. In the past, processes in the agricultural field needed a lot of manpower work, although currently the human interaction is almost non-essential, all due to the technological revolution. With the use of high-tech devices, like machines and computers in the medical area, it was possible to make discoveries at an exhaling speed on this important worldwide theme.

The theme of PR was decisive in the advancement of science and technology, allowing to accelerate and automate many processes in different areas, such as voice recognition in engineering, traffic control and license plate recognition in civil administration, medical diagnosis in medicine, stock exchange forecast in economy and the classification of rocks in geology.

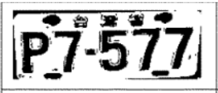
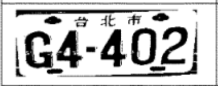
	**P7577**	P7577
	**G4402**	G4402

Figure 1.5: License Plate Recognition Example [19].

In engineering, handwritten digit recognition is a well-known subject, with important and well-founded research, presenting results with high accuracy. With such technology, the post offices are able to scan envelops adequately and sort them by zip code. Banks can quickly and automatically process bank checks. Also, handwritten digit recognition has an enormous potential when applied to manuscripts that contain plenty numerical information.

The importance and applicability of this type of technology is enormous. Currently, to solve the problem of image recognition, there are used complex and computationally very demanding ML techniques, as is the case of Deep Learning (DL). In this dissertation, it is intended to apply innovative ideas, based on data compression models, whose computational efficiency is generally much superior to other existing techniques.

The goal of this dissertation is to design and study a handwritten digit recognition system based on data compression techniques. The basic idea is to use techniques commonly associated with data compression to determine how far two objects are from each other (in the sense of the amount of information needed to transform one object into the other). In this particular case, the objects are images of handwritten digits.

1.3 Contributions

The main contributions of this dissertation include,

- The exploratory research work on a novel PR approach to handwritten digit recognition, using data compression.

- The development of a new handwritten digit recognition system supported by compression concepts.

1.4 Outline

This dissertation starts with an overview of the PR field, providing a brief introduction on the theme. Then, the view of the main idea of the developed work on this dissertation is established, followed by the motivation and objectives. The remaining chapters are organized as follows:

Chapter 2 starts by rendering a brief history of the current numerical system and the benefits that such structure originates on the human civilization. The Modified National Institute of Standards and Technology (MNIST) database is disclosed, starting by presenting the origins and a statistical and informative analysis. The Handwriting Recognition topic is exhibited and a differentiation is made between the Online versus Offline Handwriting Recognition. Finishing the chapter, an overview of the contemporary handwritten digit recognition techniques already tested on the MNIST database is given.

Subsequently, in Chapter 3, two different definitions of information are presented. The main focus of this chapter is to give a presentation of the influential data compression techniques. The link between the Shannon entropy and the Markov models is illustrated. Following, the coding techniques that revolutionized the compression methods are explained. An elucidation on the dictionary based compression is made and, to finish the chapter, the leading image compression methods are briefly explained.

Next, in Chapter 4, the notion of similarity measure and the important measures are introduced. The chapter is exclusively dedicated to the experimental evaluation of the proposed handwritten digit classification method. The proposed method modules and the pre-processing procedures are extensively described. Closing the chapter, the performance results earned from the proposed method are presented and some considerations about the results are made.

The last chapter, Chapter 5, completes this dissertation with the conclusions and future possible research guidelines.

1.5 Notations

The following notation are used in this dissertation: boldface uppercase letters denote matrices, boldface lowercase letters stand for vectors, italic lowercase and uppercase letters denote scalars and lowercase Monospace letters denote objects. The interval notation $[a, b]$ denote the closed interval from a to b, that is the interval including the values a and b themselves. Differently, (a, b) denotes the corresponding open interval, that is the interval that includes the values between a and b, but excluding a and b. Furthermore, $[a, b)$ denotes an interval that excludes b but contains a.

The function $\min(\mathbf{n})$ returns the smallest value in the $\mathbf{n}$ vector. On the other hand the $\max(\mathbf{n})$ function outputs the largest value in $\mathbf{n}$. The $mean(\mathbf{n})$ function returns the mean for the $\mathbf{n}$ vector and the $round(\mathbf{n})$ function rounds each element of $\mathbf{n}$ to the nearest integer. The distances $D_E\ (\mathbf{x}, \mathbf{y})$, $D_{Man}\ (\mathbf{x}, \mathbf{y})$ and $D_{Min}\ (\mathbf{x}, \mathbf{y})$ represent the Euclidean Distance, Manhattan Distance and Minkowski Distance, respectively. The notation $f[n] * g[n]$ denotes a convolution for a 1D signal.

The notation $\mathtt{xy}$ denotes the concatenation of $\mathtt{x}$ to $\mathtt{y}$ and Ji identically denotes a concatenation of J to i. The notation $\mathcal{A}$ and $\mathcal{H}$ denotes a distinct alphabet and a hash table data structure, respectively. The notation $|\mathtt{n}|$ denotes the number of elements in $\mathtt{n}$, if $\mathtt{n}$ is an object. For an alphabet, the notation $|\mathcal{A}|$ represents the size of $\mathcal{A}$.

The notation $H(X)$ denotes the Shannon entropy of a random variable X. $K(\mathtt{x})$ denotes the Kolmogorov complexity of a object, $\mathtt{x}$. $K(\mathtt{x}|\mathtt{y})$ denotes the Conditional Kolmogorov complexity of $\mathtt{x}$ given another object, $\mathtt{y}$. $C(\mathtt{x})$ defines the length of a object, $\mathtt{x}$, after being compressed by a compressor C. $C(\mathtt{x}|\mathtt{y})$ denotes the length of an object, $\mathtt{x}$, after being compressed by a compressor C using also the models of $\mathtt{y}$. Lastly, the $C(\mathtt{x}||\mathtt{y})$ defines the compressed length of $\mathtt{x}$ by the compressor C using only the models of $\mathtt{y}$. $E(\mathtt{x},\mathtt{y})$, $\mathrm{NID}(\mathtt{x},\mathtt{y})$, $\mathrm{NCD}(\mathtt{x},\mathtt{y})$, $\mathrm{NCCD}(\mathtt{x},\mathtt{y})$ and $\mathrm{NRC}(\mathtt{x}||\mathtt{y})$ denote similarity measures between objects.

Chapter 2

Handwritten Digit Recognition

The handwritten digit recognition is a notable classic problem of PR. Assuming that a user submitted an image of a digit via a tablet, a scanner or another digital gadget, an automatic system that uses PR is designed to accurately assign a classification to the unknown digit. Such system has a vast number of applications. Governmental organizations such as postal or tax departments or private financial institutions such as banks are dependent users of this technology. For these institutions, having an automatized system, capable of correctly assigning important information, is fundamental to efficiently provide a resolution of corporations activities.

This chapter presents an overview of the methods used in the MNIST database. A background of the most actual employed numerical digit system is exhibited, followed by the origin of the MNIST database and a statistical observation of its constituents. The chapter ends with an overview on what is the handwriting recognition and the most significant methods applied to the MNIST database are explained.

2.1 A Brief History of Numerical Systems

The word *digit* [1] comes from the *Latin* noun *digitus* which means "finger". Such meaning comes from the ancient human instinct to use fingers to count. The human nature's limit of ten digits determined the common use of the base 10 on numeral systems, *i.e.*, use of decimal digits. The term *decimal* [1] is derived from the *Latin* adjective *decem* that means "ten". Consequently, decimal digits can be explained as a representation of 10 numbers [20].

The numerical system frequently used in today's world was designed many years ago in India. Nowadays, this system is commonly referred as *Arabic numerals*, although this designation is inaccurate, since its cradle and its extensive use is not specifically referred to the Arabs [21]. Another evidence is found on the *Rock Edicts of Ashoka*, which is assigned to the period of 256 B.C., were "Arabic" numerals appeared engraved thousand years before its appearance on Arabic literature [22]. Moreover, the Arabs call the current universal numerical system "*arqam hindiyyah*", which translates to Indian numerals [21].

[1]American Heritage®Dictionary of the English Language, Fifth Edition, 2011.

The diffusion of the well accepted modern numerical system started many years ago. In the 9th century, the Persian mathematician *Al-Khwarizmi* wrote the book "*On the Calculation with Hindu Numerals*" and the Arab mathematician *Al-Kindi* wrote four volumes with the name "*On the Use of the Indian Numerals*". These two literary works played an important role for the propagation of the numerical system to the Middle East and West [21]. The emergence of this system in Europe is undetermined. Nevertheless, the most agreeable statement is that it was introduced several times without a specific date and concrete place.

In Europe, the oldest manuscript containing Indian numerals is referenced to Spain and the year of 976 A.D.. Despite the early introduction, Greek numeral system sustained the popularity for many years between the restrict group of scientists and merchants, who prolonged the use of the Roman system in their books. The French mathematician *Gerbert of Aurillac*, which in 999 A.D. assumed the position of Pope Sylvester II, used the Indian numeral system on a manifold of owned authorship writings in an attempt of introducing the system in Europe, but without succeeding [23].

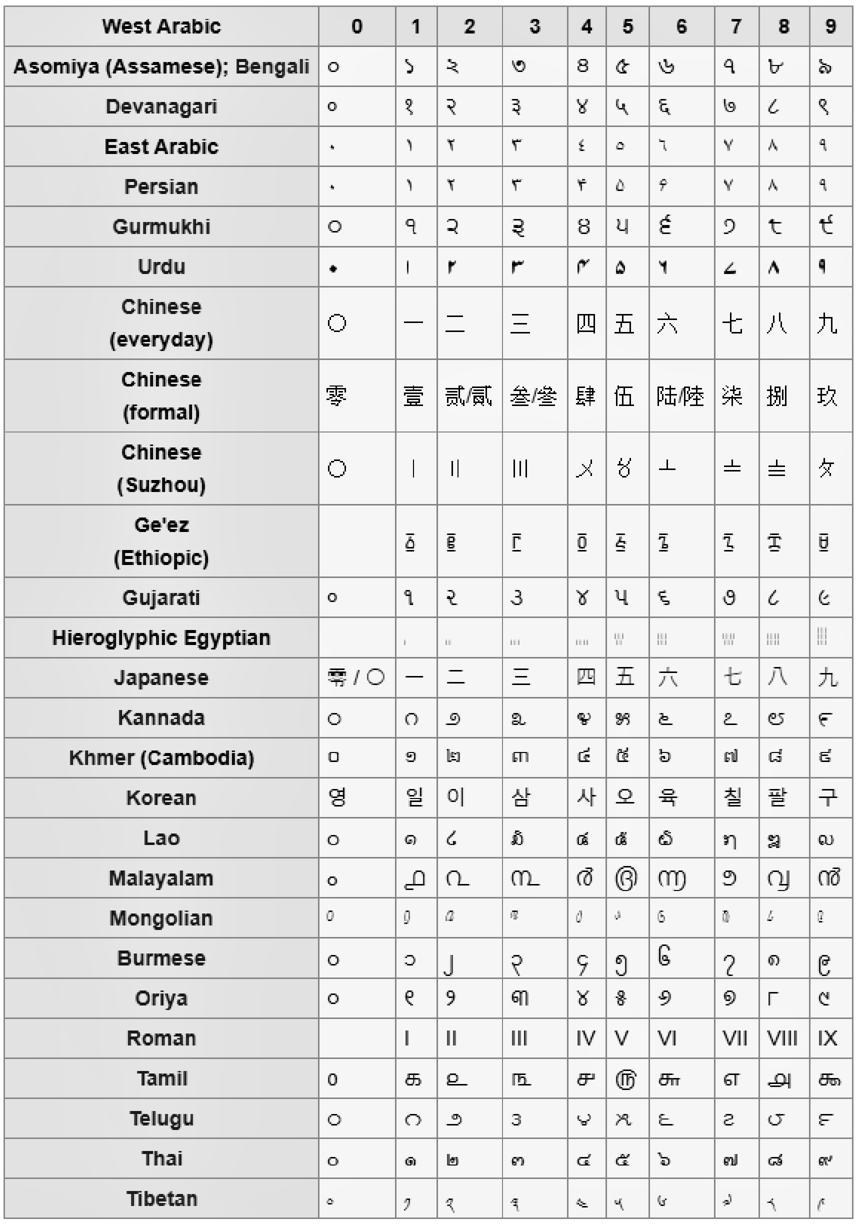
Leonardo de Pisa, or better known as simply *Fibonacci*, was an Italian mathematician, who composed the major work "*Liber Abaci*" or "*The Book of Counting*" in 1202 A.D.. His work produced the flame that ignited the propagation of the Hindu-Arabic numeral system in Europe [24]. Therewith, the European scientists started applying the numeral system to their work. One century later, the German inventor "*Johannes Guttenberg*" introduced the printing press to the world, which is regarded as the most important invention of the second millennium, contributing to the enormous dissemination of the Indian numeral system, and most importantly allowed the diffusion of learning it to the masses [25].

Nowadays, the international standard numeral system appears so clean and perfect that is seems to have simply been acquired, but the reality is that the system had to be thought out and deeply elaborated. In the 18th century, *Pierre Laplace*, one of the most influential mathematicians of all times, expressed a thought about this system, stating the follow:

"*The ingenious method of expressing every possible number using a set of ten symbols (each symbol having a place value and an absolute value) emerged in India. The idea seems so simple nowadays that its significance and profound importance is no longer appreciated. Its simplicity lies in the way it facilitated calculation and placed arithmetic foremost amongst useful inventions. The importance of this invention is more readily appreciated when one considers that it was beyond the two greatest men of Antiquity, Archimedes and Apollonius.*" [22]

The numerical system that we use in our quotidian didn't always had the same visual presentation, but evolved through time. This is, the ability to express and represent numeration was developed separately among different civilizations, resulting in a vast number of numeral representations, as depicted in Figure 2.1 [21]. It is interesting to see that some representations, such as Roman numerals, that were widely used in the past, are still being used in the present [26].

The old Babylonian numeral system has a particular characteristic that has a similarity with our actual time counting system and on the measurement of angles [27]. With a close observation on our surroundings, it is possible to detect different base systems, such as the use of a base 2 or also known as binary system, which is used daily on all digital devices. Additionally programmers sometimes use a base 8 or *octal* and a base 16 or *hexadecimal* representation, for a more compact notation.



West Arabic	0	1	2	3	4	5	6	7	8	9
Asomiya (Assamese); Bengali	০	১	২	৩	৪	৫	৬	৭	৮	৯
Devanagari	०	१	२	३	४	५	६	७	८	९
East Arabic	٠	١	٢	٣	٤	٥	٦	٧	٨	٩
Persian	۰	۱	۲	۳	۴	۵	۶	۷	۸	۹
Gurmukhi	੦	੧	੨	੩	੪	੫	੬	੭	੮	੯
Urdu	۰	۱	۲	۳	۴	۵	۶	۷	۸	۹
Chinese (everyday)	〇	一	二	三	四	五	六	七	八	九
Chinese (formal)	零	壹	贰/貳	叁/參	肆	伍	陆/陸	柒	捌	玖
Chinese (Suzhou)	〇	〡	〢	〣	〤	〥	〦	〧	〨	〩
Ge'ez (Ethiopic)		፩	፪	፫	፬	፭	፮	፯	፰	፱
Gujarati	૦	૧	૨	૩	૪	૫	૬	૭	૮	૯
Hieroglyphic Egyptian		𓏺	𓏻	𓏼	𓏽	𓏾	𓏿	𓐀	𓐁	𓐂
Japanese	零 / 〇	一	二	三	四	五	六	七	八	九
Kannada	೦	೧	೨	೩	೪	೫	೬	೭	೮	೯
Khmer (Cambodia)	០	១	២	៣	៤	៥	៦	៧	៨	៩
Korean	영	일	이	삼	사	오	육	칠	팔	구
Lao	໐	໑	໒	໓	໔	໕	໖	໗	໘	໙
Malayalam	൦	൧	൨	൩	൪	൫	൬	൭	൮	൯
Mongolian	᠐	᠑	᠒	᠓	᠔	᠕	᠖	᠗	᠘	᠙
Burmese	၀	၁	၂	၃	၄	၅	၆	၇	၈	၉
Oriya	୦	୧	୨	୩	୪	୫	୬	୭	୮	୯
Roman		I	II	III	IV	V	VI	VII	VIII	IX
Tamil	0	௧	௨	௩	௪	௫	௬	௭	௮	௯
Telugu	౦	౧	౨	౩	౪	౫	౬	౭	౮	౯
Thai	๐	๑	๒	๓	๔	๕	๖	๗	๘	๙
Tibetan	༠	༡	༢	༣	༤	༥	༦	༧	༨	༩

Figure 2.1: Most Known Graphical Digits Representation [28].

The number invention represents one of the most important discoveries to humankind, as it allowed progress and evolution to happen in such a fast pace. The human ability to perceive and recognize the number system is an important skill. The development of an automatic process that accomplishes this task presents a huge advantage to automate a enormous quantity of processes that have the necessity to identify digits.

2.2 Modified NIST Dataset

The MNIST is a large well known database of handwritten digits between 0 and 9 that is commonly used to train PR systems, although is more familiar in the ML research area [29]. To support the research on ML and PR, a wide number of standard databases were assembled in the last years. Of an enormous bulk of databases, the freely available MNIST benchmark emerged as a standard on testing ML schemes, due to the pre-processed handwritten digit images. These processes applied on the database included segmentation and normalization. The ready-to-use feature allowed researchers to gain the ability to validate recognition techniques, share and compare easily the results [30]. Figure 2.2 illustrates a sample of digits from the MNIST dataset.

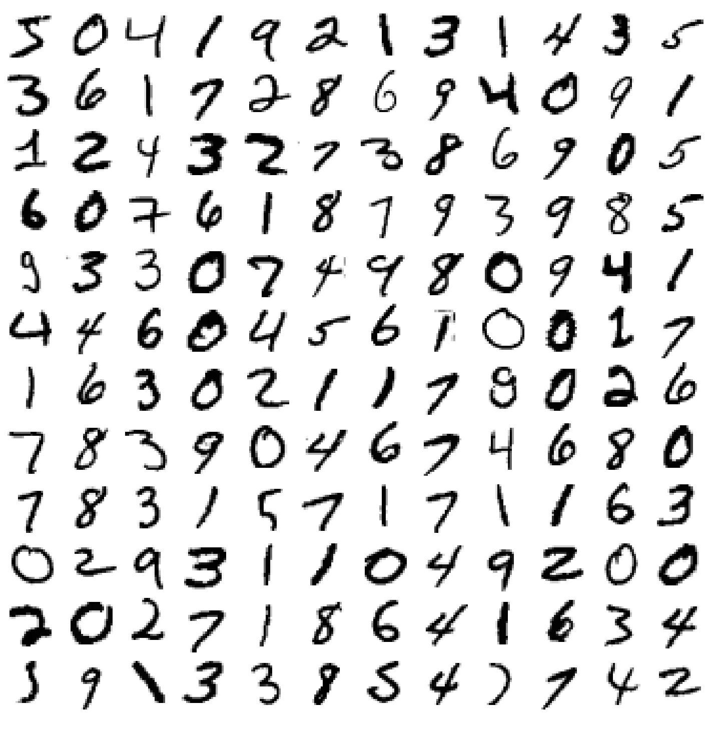


Figure 2.2: The First 144 Digit Images from the MNIST Training Set.

The National Institute of Standards and Technology (NIST) organized, in the Spring of 1992, a competition with the subject of handwriting digit classification, allowing competitors to access a training set of 223,000 samples and a test set of 59,000 samples. In Figure 2.3 it is possible to observe fragments of these sets [31]. The divergent distributions on the test and training set, or more specifically denominated as Special Database 1 (SD-1) and Special Database 3 (SD-3), respectively, revealed that these distributions influenced the results. The SD-3 was assembled from collecting handwriting digits among the United States Census Bureau employees. On the opposite side, the SD-1 was gathered among high-school students, evidence that SD-3 was clearer and easier to recognize than SD-1.

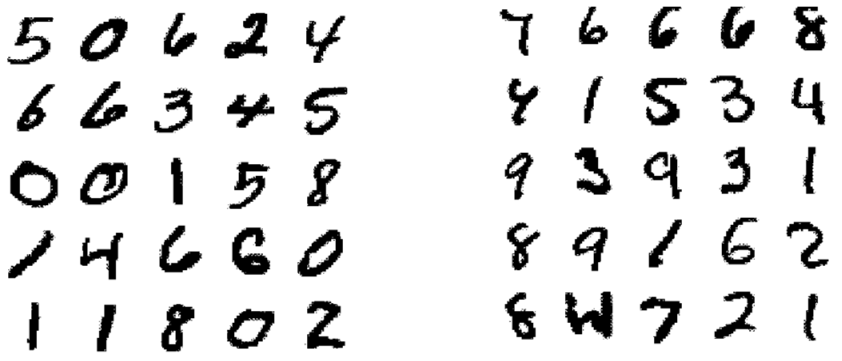


Figure 2.3: Digit Images from NIST Training Set (left) and NIST Test Set (rigth) [32].

The conclusions reached with the classification experiments were that the results must be independent of the selection of the training and test set from the overall complete set of samples. To overcome this faulty detail, the need to constitute a new database arose by mixing both NIST datasets [32].

The MNIST database was assembled by Yann LeCun, Corinna Cortes and Christopher J.C. Burges, combining the SD-3 and SD-1 sets using a 50/50 ratio, and resulting a new training set with 60,000 binary images of digits and 10,000 test samples [32]. Therefore, the new subset created from the NIST databases constitutes a collection of 70,000 binary images of digits written by hand in the range of 0 to 9. The digits frequency on the MNIST set is presented on Table 2.1.

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
Training Set	5923	6742	5958	6131	5842	5421	5918	6265	5851	5949
Test Set	980	1135	1032	1010	982	892	958	1028	974	1009

Table 2.1: The Digit Frequency of the MNIST Training and Test Set.

The digits images that compose the MNIST dataset were centered by center of mass, size-normalized, and stored in binary files sequentially as gray-scale images with 28×28 dimensions, where pixels had an intensity range from 0 to 255. The size of a simple image sample vector is 784. Table 2.2 displays more information on the name of the files and the respective dimensions. Associated with the files that contain the images, the MNIST provides two other files that enclose labels which accurately assign a classification to the images. The MNIST is a fairly simple database for test and develop ML and PR methods on real-world data without spending huge amounts of energy on pre-processing and formatting.

Files	**Files Size in Bytes**
Training Set Images	47040016
Training Set Labels	60008
Test Set Images	7840016
Test Set Labels	10008

Table 2.2: MNIST Dataset Files and Sizes.

A handful of major classification techniques have been used on this training set and test set. In some approaches, the images were deskewed, this is, they were calculated by finding the principal axis of the shape of the digit closest to the vertical and then shifting the lines to rearrange them in the vertical direction. In some other experiments, the training set was increased with artificially distorted versions of the original training samples [32].

2.3 Handwriting Recognition: An Overview

Handwriting is a competence that is particular to individuals, and mastered many years ago with the intent to expand human memory and ease communication. The act of handwriting consists in inserting artificial marks on a surface, with the purpose of achieving the graphical representation of a certain language. Together with the individual, the handwriting skill is aggregated to the ability to recognize and identify shapes and curves of the artificial marks of the language. These competences provided the sharing of knowledge over time, allowing access to the future generations [33].

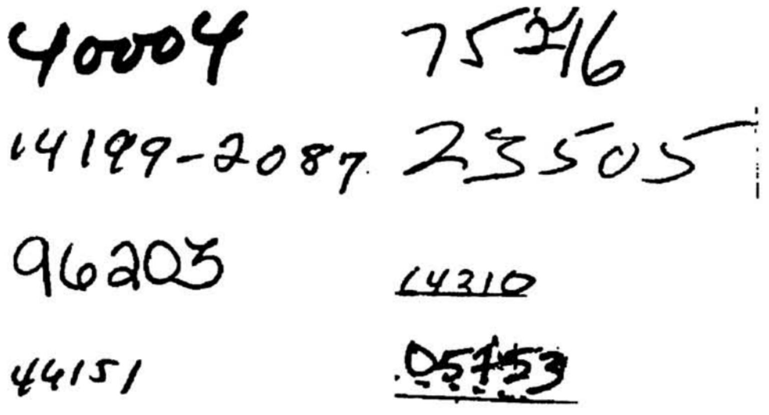


Figure 2.4: Handwritten Example of Original Zip Codes [34].

The technological evolution inflicted an impact on handwriting practice, since the digital devices could provide clean and pronto formatted documents. However, in numerous situations, the symbiosis between pen and paper is much more practical than a keyboard. The

Handwriting Recognition represents a solution to the other side of the spectrum, being the tool that can transform a language represented in its spatial form of graphical marks into symbolic representation. More specifically, a handwritten Latin language can be represented in the form of 16-bit Unicode. Figure 2.4 portrays an example of zip codes showing handwritten digits that can be translated to a typically 8-bit American Standard Code for Information Interchange (ASCII) representation.

2.3.1 Online versus Offline Handwriting Recognition

In *Handwriting Recognition*, the algorithms can be distinguished into two different categories: online and offline handwriting recognition. The divider between these categories can be established by looking to the final use of the application. If the method is used in real time, then is designated an online classification algorithm. Moreover, if the classification objects were previously saved and the class assignment processes can be done at another time, then the system can be characterized as offline handwriting recognition. Nevertheless, offline algorithms might have associated time features connected to the data input. Figure 2.5 depicts examples of the two types of handwritten recognition algorithms.

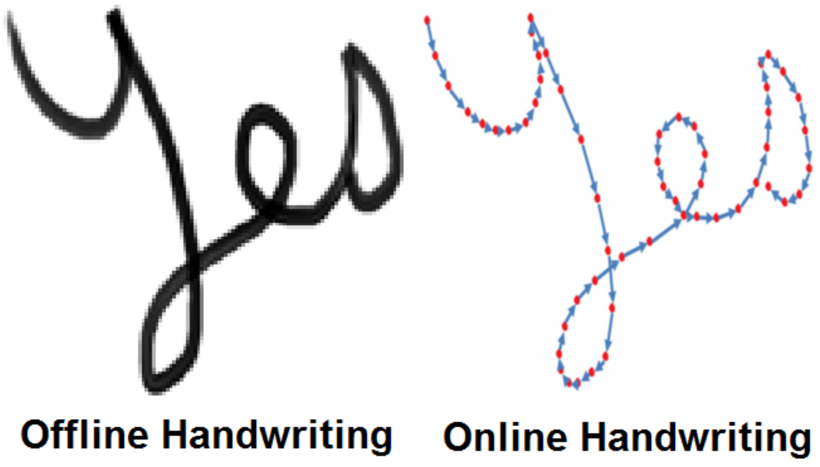


Figure 2.5: Online and Offline Handwriting Recognition [35].

Online handwriting recognition algorithms convert text at the same time as the writing process takes place. Within the process of designing an online recognition system, it is important to keep in mind the speed of recognition, which should be similar to the writing speed of the language. For alphanumeric English, the average writing rate is around 1.5-2.5 characters/s. For more graphically detailed characters such as Chinese symbols, the average speed is positioned at 0.2-2.5 characters/s. [36]. Depending on the characteristics of the type of characters, it is possible to achieve a 5-10 characters/s in English, *e.g.*, a sequence of 1's can be rapidly drafted.

The real time component on the online recognition allows the extraction of features such as stoke pressure, velocity and trajectory. Furthermore, the submitted data has low noise, which translates to an easier classification. The Palm PDA and Google Handwrite are contemporary tools that use online handwriting recognition [37].

Offline handwriting recognition is performed after the writing process, so the algorithm can be applied days, months or years later. Generally, the offline recognition process is stated as a more complex task than the online method [38]. On the offline case, the only available information are images or the scanned data, while in the online context it is possible to extract features from the pen trajectory and from the resulting image. The offline recognition is a challenging task that has been subject to extensive research, combining Computer Vision (CV) with sequence learning. The MNIST is a well-known database used for testing offline recognition systems.

2.4 Classification Methods on the MNIST Dataset

The MNIST database is considered a standard to estimate the relative performance of a new algorithm in the ML field. Several methods and techniques have been evaluated on the MNIST training and test sets. Of the diversified published ML techniques which attempt to classify the MNIST set, it is possible to divide the classifiers in seven general categories:

- **Linear Classifiers**
- **Non-linear Classifiers**
- **k-Nearest Neighbors**
- **Boosted Stumps**
- **Support Vector Machines**
- **Neural Networks**
- **Convolutional Neural Networks**

The ML techniques tested on the MNIST set can also be differentiated in four different categories, where the separation is molded with the type of *feedback* that the learning system may receive:

- **Supervised Learning Algorithms**
- **Semi-supervised Learning Algorithms**
- **Unsupervised Learning Algorithms**
- **Reinforcement Learning Algorithms**

In **supervised learning**, the system receives the input data and the respective labels, being the main objective the ability of the system to compute on its own a generic rule with the competence to assign inputs to outputs [39]. In **unsupervised learning**, the system has the responsibility to learn patterns from the input data without receiving labels, which can represent a challenge. In **reinforcement learning**, the system learns from the granted series of reinforcements, rewards or punishments, giving important information to the system and enabling the learning process on the positive and negative decisions.

In the real world, these transparent distinctions are not always so sharp. The **semi-supervised learning** lays down between the supervised learning and unsupervised learning. The data available to a semi-supervised learning system has a few labeled examples and eventually will have to handle and treat a vast collection of unlabeled examples [40]. In such system, the labels may not represent the true story of the reality.

The semi-supervised learning represents the learning type of the following example: a system that guesses a person's age from a picture. The training labeled information can be extracted by snapping pictures of people and asking their age. Theoretically, this is supervised learning. In reality, some people of the training set can lie about their age, therefore introducing to the system random noise. The unsupervised learning can address the systematic imprecision by involving the set of information available, photos, ambiguous true ages and self-reported ages. Consequently, the noise and lack of labels conjunction create a continuum between supervised and unsupervised learning. In Figure 2.6 are depicted the main learning styles or learning models that an algorithm can adopt.

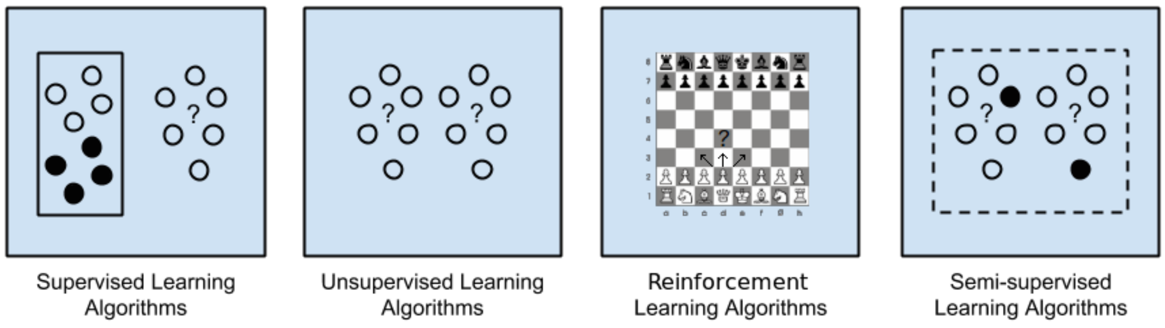

Supervised Learning Algorithms
Unsupervised Learning Algorithms
Reinforcement Learning Algorithms
Semi-supervised Learning Algorithms

Figure 2.6: Forms of Learning in Machine Learning.

2.4.1 Linear Classifier

A **Linear Classifier** is considered one of the most straightforward classifiers, where the classification decision is established based on the value of a linear combination realized on the input data characteristics. The characteristics of the input object or also denominated as feature values, are commonly introduced to the machine in a set termed feature vector [41]. A discriminant function that is a linear combination of the training data $(l_i, \mathbf{x}_i) \in \{-1, +1\} \times \mathbf{R}^n$, where l_i is the label and $\mathbf{x}_i$ is the feature vector, can be written as,

$$g(\mathbf{x}) = \mathbf{w}^t\mathbf{x} + w_0, \tag{2.1}$$

where w_0 is an intercept or better known as bias or threshold weight and $\mathbf{w}$ is the weight vector [42]. For a two class linear classifier based on the precedent equation, the following decision rule applies: if $g(\mathbf{x}) > 0$, then the result is l_1. On the other hand, if $g(\mathbf{x}) < 0$, then l_2 is determined as the classification result. Therefore, if the inner product $\mathbf{w}^t\mathbf{x}$ exceeds the threshold weight w_0, the label l_1 is assigned to $\mathbf{x}$, and alternatively l_2. To the particular $g(\mathbf{x}) = 0$ solution, the system designer can arbitrarily choose between l_1 and l_2.

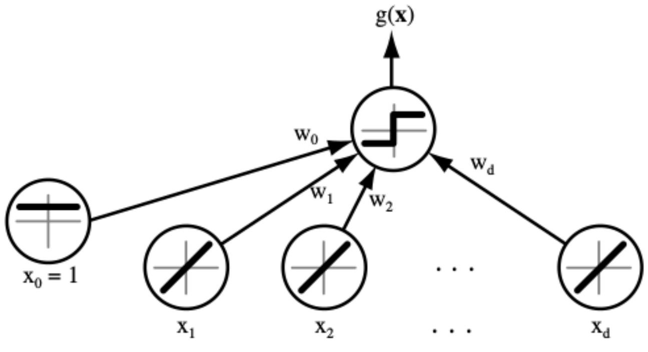


Figure 2.7: Example of a Simple Linear Classifier [42].

A simple example of a linear classifier is depicted in Figure 2.7, with d input units which correspond to the values of the components of the feature vector. The respective feature values of the $\mathbf{x}$ feature vector are multiplied by its corresponding values from the $\mathbf{w}$ weight vector. The output emits $+1$ or -1 if the sums of the products are $\mathbf{w}^t\mathbf{x} + w_0 > 0$ and $\mathbf{w}^t\mathbf{x} + w_0 < 0$, respectively [42].

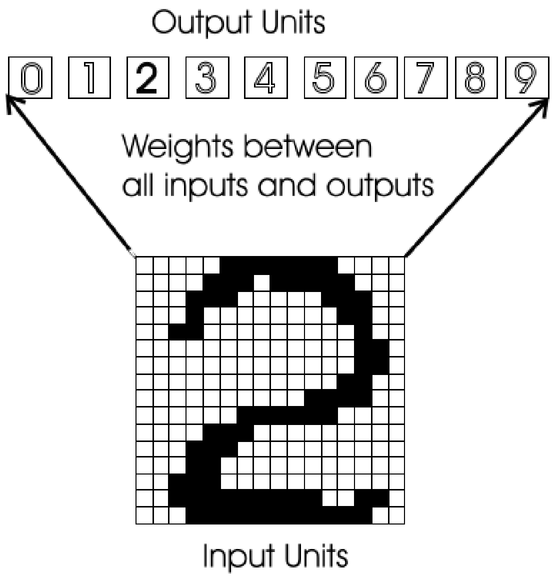


Figure 2.8: Linear Classifier applied to Digit Recognition [32].

On the MNIST data, techniques such as the linear classifier had been evaluated and tested, being one of the methods already evaluated displayed in Figure 2.8. In this classifier, the output units are composed by a weighted sum in which each input pixel value has its own contribution. The output unit with the largest computed sum between the weighted sum and the threshold weight expresses the class of the input digit [32]. For this example, there are $10N$ weights and 10 bias constants, where N denotes the number of input pixels, and therefore the number of system parameters can be expressed by $10N + 10$. The input of Figure 2.8 example is an image with a dimension of 16×16 pixels. Hence, the number of free parameters in Figure 2.8 classifier is 2570.

Classifier	**Pre-processing**	**Error Rate (%)**
Linear Classifier (1-Layer NN)	None	12.0
Linear Classifier (1-Layer NN)	Deskewing	8.4
Pairwise Linear Classifier	Deskewing	7.6

Table 2.3: Linear Classifier Results on the MNIST Test Set [29].

Table 2.3 displays the classifiers names, the pre-processing methods used and the performance results of the linear classifiers applied to the MNIST dataset.

2.4.2 Non-Linear Classifier

The Non-Linear Classifier category consists of methods that perform transformations in the data in order to represent new multi-dimensional spaces and apply afterwards the classification techniques. In general, if a classification technique reforms the input data before applying the classifier, the method may be defined as a non-linear one [43]. The Non-Linear Classifiers assessed in the MNIST database and the corresponding results are displayed in Table 2.4. The two methods tested in the set are the Principal Component Analysis (PCA) with a quadratic classifier and Radial Basis Functions (RBFs).

Classifier	**Pre-processing**	**Error Rate (%)**
1000 RBF + Linear Classifier	None	3.6
40 PCA + Quadratic Classifier	None	3.3

Table 2.4: Non-Linear Classifier Results on the MNIST Test Set [29].

The PCA is a statistical method that extracts important information from possibly correlated variables of the input data, transforming the information in new orthogonal variables called principal components [44]. The PCA based method presented in [29] has a pre-processing stage with the task of computing the projection of the input pattern on the 40 principal components of the training set vectors [32].

The principal components calculation was produced by firstly computing the mean of each input component and then subtracting from the training vectors. Consequently, the covariance matrix is computed from the resulting vectors and diagonalized using Single Value Decomposition (SVD). Lastly, the classification process is handled by a second degree polynomial classifier with the resultant 40 dimensional feature vector as input. A linear classifier with 821 inputs and a module with the capability to compute all the pair of input variables could replace the polynomial classifier.

The RBF is an efficient modern technique used to approximate multivariate functions, and also highly useful in lower dimensional problems. The RBF technique is well known, tested and analyzed, having many positive properties identified [45]. In [32], one of the non-linear methods uses a RBF network. The first layer of the RBF procedure is configured with 1000 Gaussian RBF units with 400 inputs, and a simple linear classifier complements the second layer. The RBF units are aggregated in 10 groups of 100. This way, each group is individually trained using the k-means algorithm in one of the ten classes of the training set. The computation of the second layer weights is performed with the aid of a regularized pseudo-inverse method.

2.4.3 k-Nearest Neighbors

The k-Nearest Neighbors (k-NN) classification is a simple classifier method, where the k-NN model representation is defined by the entire training set. The k-NN algorithm stores the entire dataset, implying that the classifier does not requires training time, so there is no learning stage. To obtain predictions, the k-NN uses the training set directly, and the dataset storing process can be accomplished using complex data structures like k-dimensional trees, allowing searching and matching of new patterns efficiently [32].

In the k-NN algorithm, predictions for a new instance $\mathbf{x}$ can be settled by searching on the full labelled training set for the k more similar instances, termed commonly by neighbors, and setting the output variable based on k neighbors. By using the k-NN as classifier, the output will assign the instance $\mathbf{x}$ to a class [46].

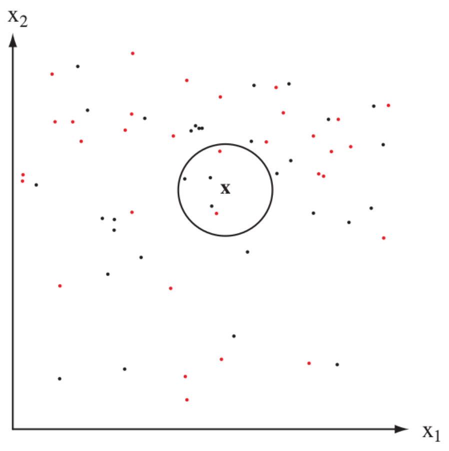


Figure 2.9: k-Nearest Neighbor Classification Example [42].

The value for k can be established with algorithm tuning, by trying different values for k [47]. Figure 2.9 shows a classification example of the k-NN algorithm, for a $k = 5$ case. The test point $\mathbf{x}$ is surrounded by the k training samples, so $\mathbf{x}$ would be labelled as black point because the black point class exhibits the majority voting class.

The process to determine the most identical features that a new input has in comparison with the k instances in the training set, the k-NN to accomplish this task uses a distance measure [42]. The most prominent distance measure is the *Euclidean distance*, which is computed as the square root of the sum of the squared differences between a new instance $\mathbf{x}$ and an existing instance $\mathbf{y}$ in the k-NN model representation, and can be written as,

$$D_E\left(\mathbf{x},\mathbf{y}\right) = \sqrt{\sum_{i=1}^{k}(\mathbf{x}_i - \mathbf{y}_i)^2}. \tag{2.2}$$

The *Euclidean distance* is a suitable distance measure for real value input variables and similar in type, *e.g.* widths and heights. The *Manhattan distance*, also called *City Block Distance*, is also another widely used distance measure, and can be expressed as,

$$D_{Man}\left(\mathbf{x},\mathbf{y}\right) = \sum_{i=1}^{k}|\mathbf{y}_i - \mathbf{y}_i|. \tag{2.3}$$

The *Manhattan distance* represents a good measure to use in input data that does not share similarities in type, *e.g* age, gender, height, etc. The *Minkowski Distance* is a generalization of the Euclidean and Manhattan distance measures [46].The Minkowski Distance is expressed in the following equation,

$$D_{Min}\left(\mathbf{x},\mathbf{y}\right) = \left(\sum_{i=1}^{k}(|\mathbf{x}_i - \mathbf{y}_i|)^q\right)^{\frac{1}{q}}. \tag{2.4}$$

Additionally to the presented distance measures, more methods are used such as the *Hamming Distance*, *Tanimoto metric*, *Jaccard*, *Mahalanobis* and the *Cosine* distance.

The k-NN represents an important role in classification. The attribution of a label is performed based on the class with the highest frequency from the k-most similar instances. On the recognition operation, each k instance votes for their class. In the end, the class with the majority votes gives a label to the new instance. Whenever choosing the value of k, the number of classes has to be taken into account, and if the number of classes is even, then it is a good idea to select an odd value for k in order to avoid draws. On the inverse situation, if the number of classes is odd, then is a good practice to use an even number for k. Draws can be broken by expanding the k value or by introducing to the voting the next most similar instance of the training set.

The k-NN is a very well studied memory based model, with a large area of applications and a long history of development [49]. Due to its ease to adapt to the problems and its long period of use, the assigned name is different depending on the discipline, for example:

- **Instance-Based Learning**: The k-NN method in this case is referred as instance-based learning or case-based learning due to the prediction process, where the raw training instances are used to produce predictions, so every training instance is a case from the problem domain [49].
- **Lazy Learning**: This category of k-NN procedure does not need any model information. When a prediction is requested, the algorithm generates the prediction in real time, and therefore this approach is often designated as lazy learning [50].
- **Non-Parametric**: The designation of non-parametric to the ML k-NN algorithm comes from the fact that the approach does not make any assumption on the underlying data distribution [51].

Table 2.5 illustrates the name of the classifier, the pre-processing techniques and the error rate of the k-NN methods implemented on the MNIST test set.

Classifier	**Pre-processing**	**Error Rate (%)**
k-NN, Euclidean L2	None	3.09
k-NN, L3	None	2.83
k-NN, Euclidean L2	Deskewing	2.40
k-NN, Euclidean L2	Deskewing, Noise Removal, Blurring	1.80
k-NN, L3	Deskewing, Noise Removal, Blurring	1.73
k-NN, L3	Deskewing, Noise Removal, Blurring, 1 Pixel Shift	1.33
k-NN, L3	Deskewing, Noise Removal, Blurring, 2 Pixel Shift	1.22
k-NN, Tangent Distance	Subsampling to 16×16 Pixels	1.10
k-NN, Shape Context Matching	Shape Context Feature Extraction	0.63
K-NN with Non-Linear Deformation (IDM)	Shiftable Edges	0.54
k-NN with Non-Linear Deformation (P2DHMDM)	Shiftable Edges	0.52

Table 2.5: k-Nearest Neighbors Classifier Results on the MNIST Test Set [29].

2.4.4 Boosted Stumps

The Boosted Stumps or more specifically *Boosted Decision Stumps* are a conjunction of two ML techniques – *Decision Stump* and *Boosting*. A decision stump is a special case of decision tree and known as *one-level decision tree* [52]. Figure 2.10 presents an example of a decision stump, where is possible to identify the decision node that uses only a single attribute and the two possible classification leaves. The parameters of the decision stumps are acquired on the training process with the training samples.

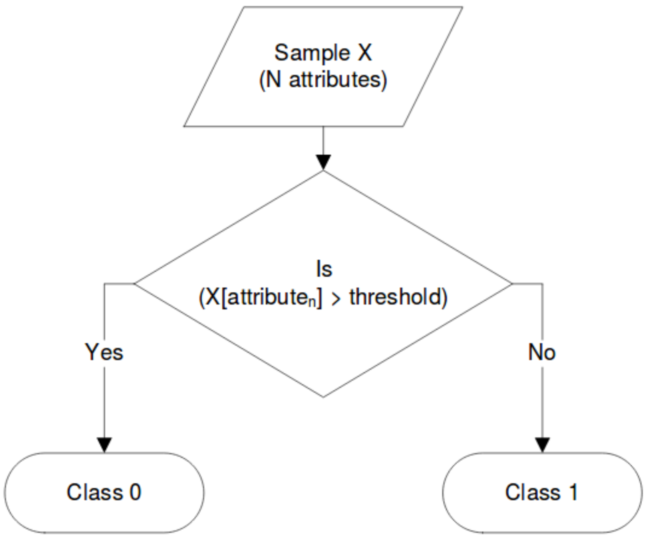


Figure 2.10: Decision Stump [53].

Decision stumps are a member of the supervised learning algorithm class, where the learning algorithm takes as input the attributes, attribute values, sample classification and sample weight. Upon the assimilation of the information, the learning algorithm chooses an attribute and an associated threshold value, which provides the best classification performance. The classification is performed on the evaluation of the inequality equation between the sample's attribute value and the threshold. The result of such inequality equation gives a classification to the test sample.

Decision stumps are used as weak learners in ML ensemble techniques such as Boosting [54]. Boosting are a group of ML algorithms that use weak learners to create strong classifiers, where the method's objective is to improve the accuracy of the learning procedure [55]. On the Boosting algorithm space the most common implementation is the **AdaBoost**, or more explicitly *Adaptive Boosting*. The main idea behind the **AdaBoost** algorithm is to maintain a distribution of weights over the training set. At the beginning, the weights are initiated with the same value. In every round of the learning process, the weights of incorrectly classified examples are increased [56]. Therefore, **AdaBoost** is adaptive due to the successive weak learners that are tweaked in consideration of the instances misclassified by previous classifiers, improving the performance of the used weak learners.

The conjunction of the two techniques shaped the Boosted Stumps, and the results obtained with this approach on the MNIST set are displayed in Table 2.6.

Classifier	**Pre-processing**	**Error Rate (%)**
Boosted Stumps	None	7.70
Boosted Trees (17 Leaves)	None	1.53
Products of Boosted Stumps (3 Terms)	None	1.26
Stumps on Haar Features	Haar Features	1.02
Product of Stumps on Haar Features	Haar Features	0.87

Table 2.6: Boosted Stumps Classifiers Results on the MNIST Test Set [29].

2.4.5 Support Vector Machine

The Support Vector Machines (SVMs) are one of the most popular ML algorithms, due to the relatively simple structure and its capability to achieve a high performance with little tuning of the procedure. The supervised learning of the SVM is based on simple ideas and provides a clear insight on the mechanism of learning from examples [57]. SVM methods construct a maximum margin separator, *i.e.*, the SVM creates a decision boundary with the largest conceivable distance between examples. In addition to creating linear separating hyperplanes, SVMs are also able to embed data in higher dimensional space using the **kernel trick** [58].

Usually, information in its original state is not linearly separable, although into higher dimensional spaces the separation can be simplified. Nevertheless, the SVM technique exposes a remarkably economical way of representing complex surfaces in high dimensional spaces.

In complex surfaces resides a peculiar subset of decision surfaces, the hyperplanes, that are at maximum separation from the convex hulls of the two classes in the high dimensional space of the product terms. Moreover, in image recognition, the k degree polynomial that defines the margin separator can be calculated by firstly computing the scalar product of the input image with a subset of the training samples, the support vectors [59]. The dot product results are then elevated to the k-th power and the acquired coefficient numbers are linearly combined. Table 2.7 shows the performance of SVMs architectures on the MNIST database.

Classifier	Pre-processing	Error Rate (%)
SVM, Gaussian Kernel	None	1.40
SVM deg-4 Polynomial	Deskewing	1.10
Reduced Set SVM deg-5 Polynomial	Deskewing	1.00
Virtual SVM deg-9 Polynomial [distortions]	None	0.80
Virtual SVM, deg-9 Polynomial, 1 Pixel Jittered	None	0.68
Virtual SVM, deg-9 Polynomial, 1 Pixel Jittered	Deskewing	0.68
Virtual SVM, deg-9 Polynomial, 2 Pixel Jittered	Deskewing	0.56

Table 2.7: Support Vector Machine Classifiers Results on the MNIST Test Set [29].

Figure 2.11 depicts an example of a binary classification problem with three candidate decision boundaries. The three candidates are consistent on the separation of the classes and are equally good. Anyhow, the focal point of the SVM is that some samples are more important than others and this additional care can lead to a better organization.

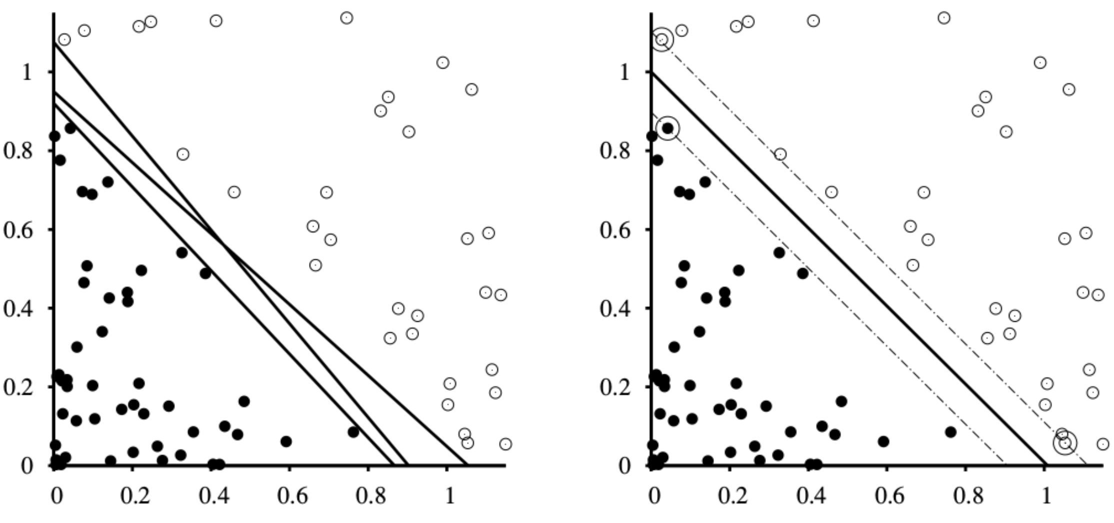


Figure 2.11: Support Vector Machine Classification Example [39].

On the left example, the lower decision boundary comes very close to 5 black examples classifying correctly all samples. However, there are many black examples near the line, which ends up not making sure that a new black sample is on the other side of the line. The SVM method brings up a generalization to the problem by creating a **maximum margin separator**. The maximum margin separator is represent by the heavy line on the right image between the samples and is computed with the assistance of the **support vectors**. The margin is the width of the area bounded by dashed lines created with the assistance of the support vectors, pointed with large circles. The designation of support vectors comes from the evidence that they hold up the separator [39].

2.4.6 Neural Networks

A Neural Network (NN) is a learning computing system inspired by the biological nervous system. The first steps taken in this area date back to the year 1943, where the neurophysiologist Warren McCulloch and the mathematician Walter Pits developed a simple mathematical model of a simple neural network [60]. In Figure 2.12 is depicted the simple model of an artificial neuron.

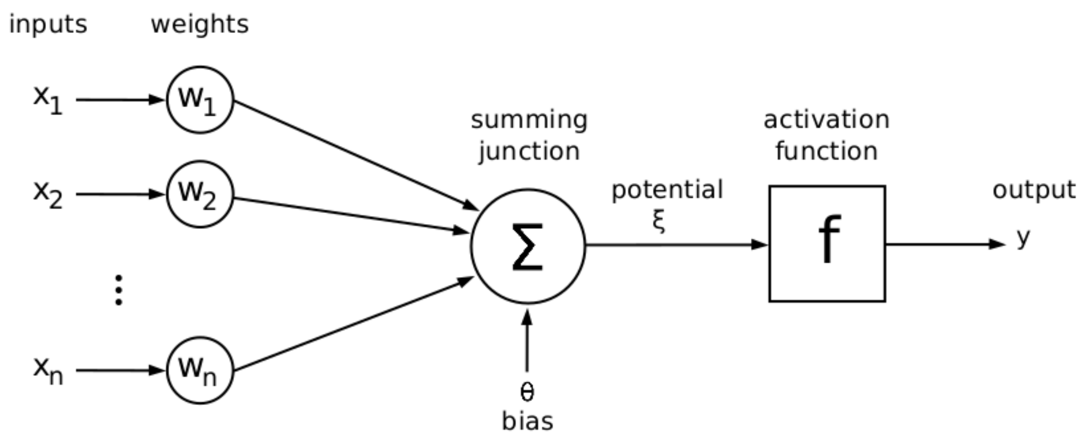


Figure 2.12: Model of Artificial Neuron [61].

NNs are a structure composed of nodes or simple neurons connected by direct links. In Figure 2.14 is displayed an example of a more complex NN. A simple neuron architecture has the following components: $\mathbf{x}_i$ as the inputs, the associated weights $\mathbf{w}_i$, the threshold θ and the output y computed by the activation function $f(\xi)$. The neuron first computes a weighted sum of its inputs as shown in Equation 2.5 and then in Equation 2.6 it applies to the sum an activation function outputting y.

$$\xi = \sum_{i=1}^{n} \mathbf{x}_i \mathbf{w}_i + \theta \tag{2.5}$$

$$y \;=\; f(\xi) \;=\; f\left(\sum_{i=1}^{n} \mathbf{x}_i \mathbf{w}_i + \theta\right) \tag{2.6}$$

The output of each neuron is produced by the node's activation function f. The most simple activation functions are depicted in Figure 2.13. The functions presented in Figure 2.13 are: on the left, the **sign function**, a hard limiting threshold function; in the middle, the linear or **semi-linear function**; in the right, the **sigmoid function** or S-shaped, a smoothly limiting threshold. Another function that is important to refer is the **hyperbolic tangent**.

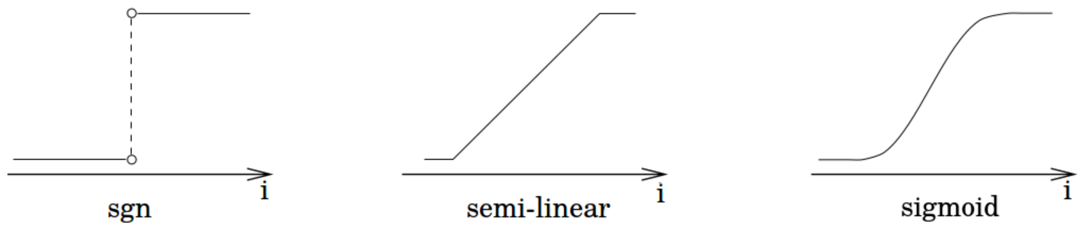


Figure 2.13: Activation Functions Most Commonly used in NN [8].

The **sigmoid function** is a widely used activation function, due to the fact that the function is differentiable at every point, and it can be expressed by,

$$f(\xi) \;=\; \frac{1}{1+e^{-\xi}}. \tag{2.7}$$

The **hyperbolic tangent function**,

$$f(\xi) \;=\; \tanh(\xi) \;=\; \frac{\sinh(\xi)}{\cosh(\xi)} \;=\; \frac{e^{2\xi}-1}{e^{2\xi}+1}, \tag{2.8}$$

is also one of the most popular activation functions in NN architectures due to the fact that inputs with ranges of $(0, 1)$ will produce outputs values between $(-1, 1)$.

The mathematical model of a simple neuron allowed researches like Rosenblatt to develop a hypothetical nervous system called perceptron, a two layered NN model, only with the aid of addition and subtraction operations [62]. The paradigm of connecting several nodes and structuring them in layers with the increase of the computational power contributed to the popularity of the NN algorithms.

The structure of a NN is typically represented by a directed acyclic graph, where the units or the neurons are connected and differentiated in layers. Figure 2.14 shows a three layer NN, with an input, hidden and output layer, where the inputs of the hidden layers are weighted by $\mathbf{w}_{jk}$ and the output layers are weighted by $\mathbf{u}_{ij}$. However, a NN can contain an arbitrary number of layers of neurons. This type of NN is termed as **Multi-Layer Neural Network**.

A NN resembles to a mapping process, where the inputs are delivered to the system through the input layer and lastly mapped to the output layer. The NN are qualified to approximate any function to a certain degree of accuracy [63].

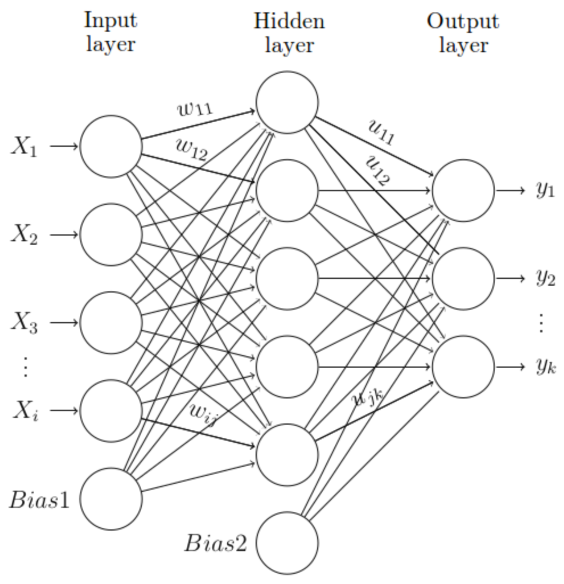


Figure 2.14: Three Layer Neural Network [64].

There are two types of NN topologies: the **Feedforward** and **Feedback**. On the **Feedforward** NN, the data flow is unidirectional. A unit provides information to the following unit and no feedback is sent backwards. On the other side, **Feedback** NN aims to get feedback from the following units [65]. An illustration of the differences between the two types is shown in Figure 2.15.

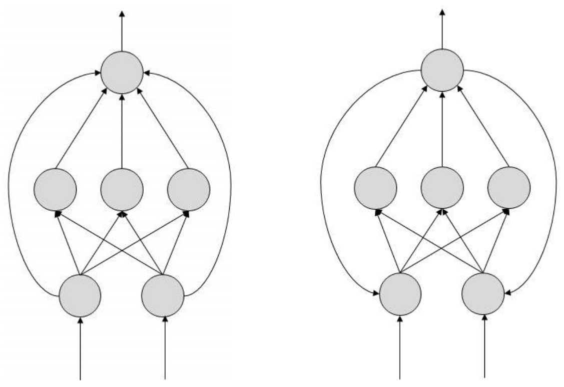


Figure 2.15: Feedforward and Feedback NN.

The results on the MNIST dataset when using NN classifiers are described in Table 2.8. Additionally, the type of the NN that was tested and the associated pre-processing techniques are presented.

Classifier	**Pre-processing**	**Error Rate (%)**
2-Layer NN, 300 HU, MSE	None	4.70
2-Layer NN, 1000 HU	None	4.50
2-Layer NN, 1000 HU, [distortions]	None	3.80
2-Layer NN, 300 HU, MSE, [distortions]	None	3.60
3-Layer NN, 300+100 HU	None	3.05
3-Layer NN, 500+150 HU	None	2.95
3-Layer NN, 300+100 HU [distortions]	None	2.50
3-Layer NN, 500+150 HU [distortions]	None	2.45
2-Layer NN, 300 HU	Deskewing	1.60
2-layer NN, 800 HU, Cross Entropy Loss	None	1.60
3-Layer NN, 500+300 HU, Softmax, Cross Entropy, Weight Decay	None	1.53
2-Layer NN, 800 HU, Cross Entropy [affine distortions]	None	1.10
NN, 784-500-500-2000-30 + Nearest Neighbor, RBM + NCA Training	None	1.00
2-Layer NN, 800 HU, MSE [elastic distortions]	None	0.90
Deep Convex Net, Unsupervised Pre-Training	None	0.83
2-Layer NN, 800 HU, Cross Entropy [elastic distortions]	None	0.70
Committee of 25 NN 784-800-10 [elastic distortions]	Width Normalization, Deslanting	0.39
6-Layer NN 784-2500-2000-1500-1000-500-10 (on GPU) [elastic distortions]	None	0.35

Table 2.8: Neural Networks Results on the MNIST [29].

2.4.7 Convolution Neural Networks

Convolutional Neural Networks (CNNs) are a class of NNs that showed positive results in areas such as image recognition and classification. The CNN is a system inspired on the animal visual cortex, one of the powerful visual processing system in existence [66]. The pioneer work on CNN was performed by Yann LeCun, who developed the LeNet Architecture [32]. The CNN in Figure 2.16 is a representation of the LeNet5 structure used for handwritten digit recognition.

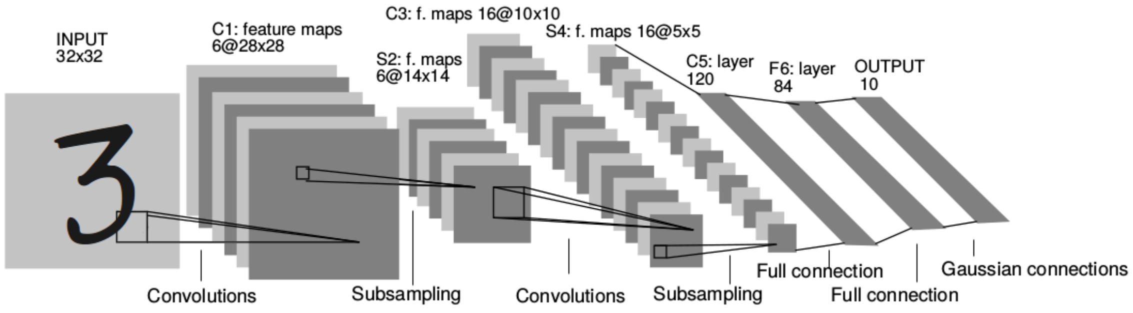


Figure 2.16: Convolutional Neural Network Example for Digit Recognition [57].

The principal building blocks in CNNs are the convolution operation, non linearity operation or more specifically Rectified Linear Units (ReLUs), pooling or subsampling and the classification commonly performed by a fully connected layer. Evidently, the CNN term came from the use of convolution operations on the learning algorithm. The role of convolution in CNNs methods is typically the feature extraction from the input images [67]. The definition of convolution for a $1D$ and $2D$ signal can be respectively described as,

$$f[n] * g[n] = \sum_{k=-\infty}^{\infty} f[k] \cdot g[n-k] = \sum_{k=-\infty}^{\infty} f[n-k] \cdot g[k], \tag{2.9}$$

$$f[m,n] * g[m,n] = \sum_{i=-\infty}^{\infty} \sum_{j=-\infty}^{\infty} f[i,j] \cdot g[m-i,n-j]. \tag{2.10}$$

The convolution operation preserves the spatial relationship between pixels by using small squares of input data to learn image features. In the CNN field, the matrices used in the convolutional process are usually designated as kernel or filter or even feature detector. The filter has the operation of feature extraction from the input image. The convolution computation is done by sliding the filter matrix over the matrix image and computing the dot product as showed in Equation 2.10. The convolution result can be referred as Feature Map or Convolved Feature or Activation Map. In Figure 2.17, it is possible to observe an example of the convolution process applied to a binary image.

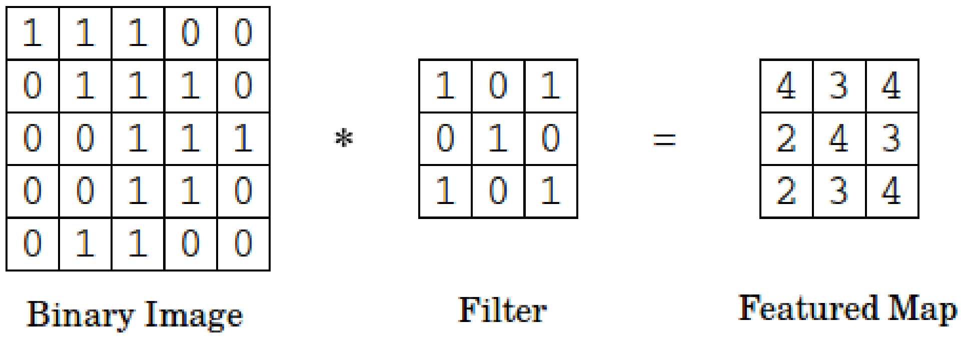


Figure 2.17: An Example of the Convolution Operation.

The filter matrix can have different values. The changing of values produces unequal Feature Maps for the same input image. With adjustment in the values of the filter matrix is possible to perform operations such as Sharpen, Edge Detection and Blur. This aspect implies that distinct filters can detect diverse features from an image such as edges and curves [68]. A CNN assigns the values of the filters during the training process. However, before training the CNN, it is necessary to initialize specific parameters, such as the filter size, architecture of the network, number of filters and others.

The dimensions of the Feature Map are controlled by three hyper-parameters that need attention and need to be carefully assigned before the convolution step [69]. The parameters are:

- **Depth**: The Depth parameter corresponds to the number of filters used in the convolution operation. A depth of two means that in the convolution process two distinct filters are used to extract an equally number of feature maps. The depth of the stacked $2D$ feature matrices is two.

- **Stride**: The Stride aspect symbolizes the number of pixels that the filter matrix slides over the input matrix. The attribution of five to the stride parameter implies that the filters jump five pixels each time they slide.

- **Zero-padding**: Zero-padding is also denominated *wide convolution* due to the allowance of control of the feature maps sizes. In the process of applying the filter matrix, the zero-padding technique allows the appliance of the filter to every element of the input matrix.

The ReLU operation precedes every convolution. As the name implies, the ReLU function is a rectifier activation function and is a non-linear operation that can be expressed as,

$$f(x) = \max(0, x) . \tag{2.11}$$

The ReLU is an element-wise operation. Analysing Equation 2.11, it is obvious that the method replaces the negative values in the activation map by zeros. Most of the real data has non-linear characteristics, so the ReLU is introduced in the CNN with the determination to introduce non-linearity in the architecture [70]. The graphical representation of the ReLU function is depicted in Figure 2.18.

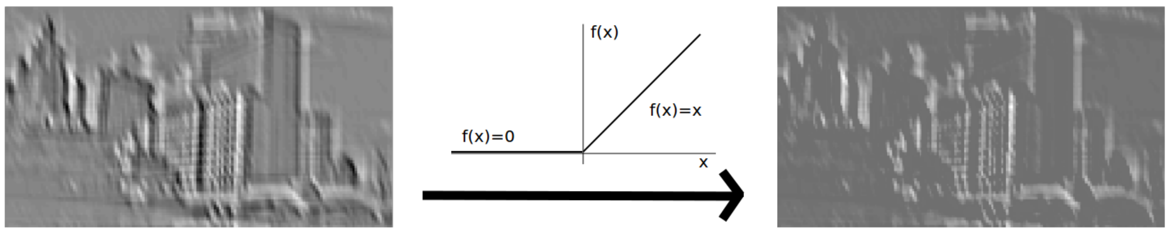


Figure 2.18: Application of the ReLU Ativation Function.

In Figure 2.18, the ReLU is applied to a Feature Map, where the negative values represent the black pixels and the positive are represented by the white pixels. On the output rectified feature, is possible to detect the vanishing of black pixels. Despite the focus on the ReLU, other non-linear functions such as *tanh* or *sigmoid* can be also used, however, the ReLU shows to operate better in most cases [71].

A key aspect of CNN is the *subsampling layers*, also named *pooling layers*, commonly applied after the ReLUs units of the convolutional layers. The purpose of the Spatial pooling is to reduce the dimensionality of each feature map while retaining the most salient information. There are many types of pooling steps, but among the best known are the **Average Pooling**, **Sum Pooling**, **L2-norm Pooling** and **Max Pooling**. Historically, the Average Pooling was more popular, but recently it fell into disuse when compared to Max Pooling [72].

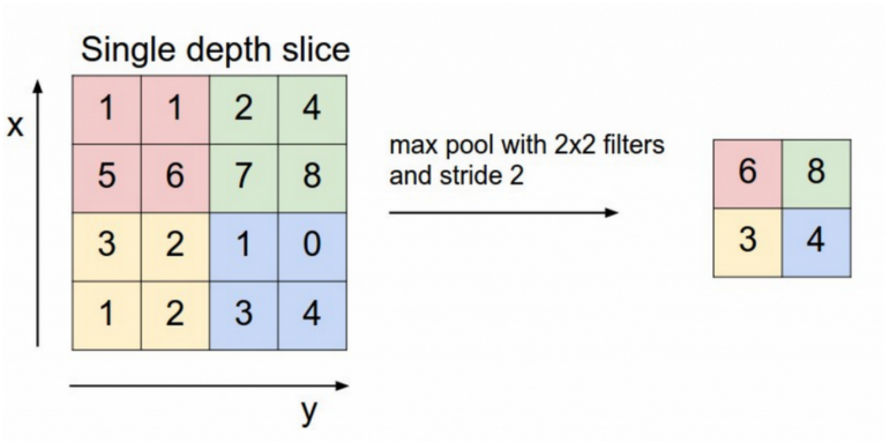


Figure 2.19: Example of Max Pooling [73].

In the Max Pooling process, a spatial neighborhood is attributed and the largest element from the rectified feature map within that neighborhood is taken. Figure 2.19 displays a particular case of Max Pooling, where the dimension of the window is 2×2 and the maximum value from the rectified feature map within that window is mapped into a new matrix. However, there is no need to pool over the complete matrix. The example shows that the pooling operation jumps over a window and as a result is the assignment of two to the stride value. Alternatively to Max Polling, another solution passes by taking the average of the values on the pooling window or even the sum of all elements in that window.

The last crucial structure in a CNN system is the Fully Connected Layer that can consider a classical Multi Layer NN with the particularity of using a softmax activation function on the output layer. The nomenclature *Fully Connected* signifies that every neuron on the antecedent layer is connected to each neuron in the adjacent layer [71]. Figure 2.20 exposes an archetype of a Fully Connected Layer.

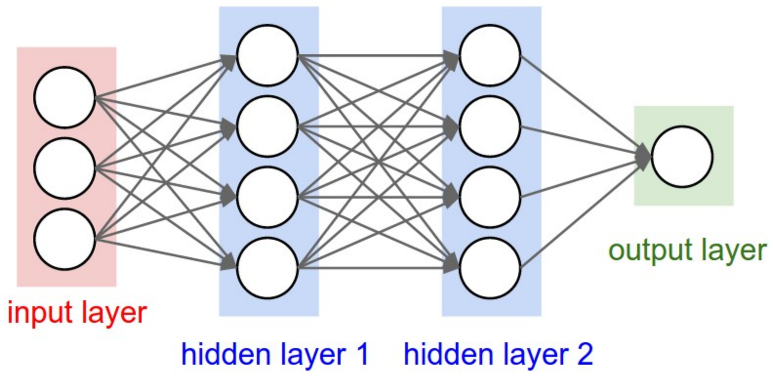


Figure 2.20: Fully Connected Layer [73].

The convolutional and pooling layers output high level features from the input data. Therefore, the Fully Connected Layer receives the features with the objective to classify the input into the classes learned on the training dataset, representing a satisfying way of learning non-linear combinations of the features [57]. In Figure 2.16, the scheme performs handwritten digit classification having ten classes as possible outputs. In the output layer of the Fully Connected Layer, the **Softmax** activation function can be used [67],

$$\sigma\left(\mathbf{x}\right)_j = \frac{e^{\mathbf{x}_j}}{\sum_{k=1}^{K} e^{\mathbf{x}_k}}, \qquad j = 1, \ldots, K. \tag{2.12}$$

The softmax function ensures that a vector of arbitrary real values is compressed, outputting a vector with values between zero and one, being the sum of the vector one.

The training of the complete network can be accomplished by error minimization using backpropagation, evaluating the gradient of the error function [74]. To satisfy the shared-weight constrains, the conventional backpropagation algorithm has to be slightly modified. The use of local receptive fields diminishes the number of weights in the network compared to the number of weights if the network was fully connected. Also, due to the significant number of constraints on the weights, the independent parameters to be learned from the input information is much smaller. The results of the implemented CNNs architectures in

MNIST database are displayed in Table 2.9.

Classifier	**Pre-processing**	**Error Rate (%)**
CNN LeNet-1	Subsampling to 16×16 Pixels	1.70
CNN LeNet-4	None	1.10
CNN LeNet-4 with k-NN Instead of Last Layer	None	1.10
CNN LeNet-4 with Local Learning Instead of Last Layer	None	1.10
CNN LeNet-5	None	0.95
CNN LeNet-5, [huge distortions]	None	0.85
Large CNN, Random Features	None	0.89
TFE + SVMs	None	0.83
CNN LeNet-5, [distortions]	None	0.80
CNN Boosted LeNet-4, [distortions]	None	0.70
Large CNN, Unsupervised Features	None	0.62
CNN, Cross Entropy [affine distortions]	None	0.60
Large CNN, Unsupervised Pre-Training	None	0.60
Unsupervised Sparse Features + SVM	None	0.59
TFE + SVMs [elastic distortions]	None	0.56
TFE + SVMs [affine distortions]	None	0.54
Large CNN, Unsupervised Pre-Training	None	0.53
CNN, Cross Entropy [elastic distortions]	None	0.40
Large CNN, Unsupervised Pre-Training [elastic distortions]	None	0.39
Large/Deep CNN, 1-20-40-60-80-100-120-120-10 [elastic distortions]	None	0.35
Committee of 7 CNN, 1-20-P-40-P-150-10 [elastic distortions]	Width Normalization	0.27 +- 2
Committee of 35 CNN, 1-20-P-40-P-150-10 [elastic distortions]	Width Normalization	0.23
Committee of 5 CNN, 784-800-800-10 [data augmentation][75]	none	0.21

Table 2.9: Convolution Neural Networks Results on the MNIST [29].

Chapter 3

Data Compression

Data compression is the art of converting an input data stream into a smaller data stream, without loosing any information. In the year of 1838, the American painter and inventor Samuel Morse created the Morse Code, an early example of data compression. In the code development, Morse detected that certain letters occurred more often than others, letters such as "e" and "t", which were more common in English. Due to this observation, Morse assigned longer sequences to letters that occur less frequently, such as q $(--.-)$ and j $(.---)$, and for letters that occur more frequently Morse assigned shorter sequences, such as e $(.)$ and t $(-)$.

The development of information theory in the late 1940s propelled the evolution of many techniques of data compression. In the beginning of the computation era, Claude Shannon and Robert Fano created a methodical way of assigning codewords based on the probabilities of blocks, the Shannon-Fano coding. Two years later, Robert Fano's student David Huffman discovered a very similar, yet more efficient, binary coding technique compared to the Shannon-Fano coding. In the late 1970s, with the Internet expansion and the online storage of data files were becoming more frequent, the development of software compression programs turned into a necessity.

This chapter focuses on data compression techniques. It starts by presenting the two major definitions of Information quantity. The Markov model technique is explained, followed by the explanation of the important coding techniques in data compression. After, the relevant dictionary based compressions are presented. Lastly, image compression techniques that are used in everyday life are briefly explained.

3.1 Information Theory

The idea of a quantitative measure of *information* was first put into firm grounds by Claude Elwood Shannon, in his famous paper entitled "A Mathematical Theory of Communication" [76]. Shannon defined a quantity called *self-information*, and years later the Russian mathematician Andrey Kolmogorov inspired by **Shannon's entropy**, introduced a different measurement of information, **Kolmogorov's Complexity** [77].

In 1948, Shannon established a way to measure the amount of information of an event without considering its meaning, by recognizing the relation between the logarithmic function and information, labeled as **self-information**. The self-information associated to the probability p of an event is $-\log_b p$. This shows that if the probability of an event is one, then the event does not have any information. On the contrary, if the probability approaches zero, the associated information progresses to infinity. The unit of information is dependent on the value of b and can be measured in *bits* $b = 2$, *nats* $b = e$ or in *hartleys* $b = 10$ [78].

The definition of entropy by Shannon, is given by the quantity that describes the average amount of information present in a certain experiment. The entropy is represented by

$$H(X) = \sum_{i=1}^{N} P_i \log_b \frac{1}{P_i}, \tag{3.1}$$

or more specifically the first-order entropy of the source, of a given random experiment X in the N possible set of events and P_i is the probability of each event. Shannon demonstrated that if an information source produces events with probabilities P_i, where X is a partition of N, then, on average, it is not possible to use less bits than H to represent a sequence of those events.

The lacuna on the classical information theory is that Shannon's entropy measures how much information is required to distinguish X between the values in N, and does not measure how much information is intrinsically in X. The gap left by Shannon theory concerning the information in individual objects motivated Kolmogorov to propose a different measurement, the Kolmogorov Complexity. Kolmogorov described his definition as [77]:

"*Our definition of the quantity of information has the advantage that it refers to individual objects and not to objects treated as members of a set of objects with a probability distribution given on it. The probabilistic definition can be convincingly applied to the information contained, for example, in a stream of congratulatory telegrams. But it would not be clear how to apply it, for example, to an estimate of the quantity of information contained in a novel or in the translation of a novel into another language relative to the original. I think that the new definition is capable of introducing in similar applications of the theory at least clarity of principle.*"

One aspect that Shannon's measure does not handle is individual objects. A string like $77777\cdots77777$ or $zzzzz\cdots zzzzz$ can easily be represented by short programs in most languages, meaning that both examples have a small amount of information, regardless of their length.

The Kolmogorov complexity $K(\mathtt{x})$ of a finite object $\mathtt{x}$ is defined as the length of the shortest effective description of $\mathtt{x}$ using any universal language, such as programming languages or a Turing Machine [79]. With antecedent knowledge of the encoding algorithm, $\mathtt{x}$ can be fully recovered from its definition. However, the Kolmogorov complexity is uncomputable for an object [80], so it needs to be approximated by other computable measures. The Kolmogorov complexity is defined as

$$K_{\mathcal{U}}(\mathtt{x}) \;=\; \min_{p} \left\{ |p| : \mathcal{U}(p) = \mathtt{x} \right\}, \tag{3.2}$$

where $\mathcal{U}$ is a universal Turing Machine and $|p|$ represents the length in bits of p, which is the program written in an arbitrary language [81]. The selection of $\mathcal{U}$ is irrelevant, since the program size between different language implementations has a variation between them only by up some constants.

An important definition in the area of compression-based similarity is the conditional Kolmogorov complexity $K\left(\mathtt{x}|\mathtt{y}\right)$, where $\mathtt{x}$ and $\mathtt{y}$ are binary objects [82]. The conditional complexity measures the quantity of information that is in $\mathtt{x}$ and not in $\mathtt{y}$, *i.e.*, the conditional Kolmogorov complexity of $\mathtt{x}$ relatively to $\mathtt{y}$ is defined as the length of the smallest program that can compute $\mathtt{x}$ if $\mathtt{y}$ was introduced as an auxiliary input to the computation [83]. The conditional complexity $K\left(\mathtt{x}|\mathtt{y}\right)$ can be defined as

$$K_{\mathcal{U}}\left(\mathtt{x}|\mathtt{y}\right) \;=\; \min_{p}\left\{|p| : \mathcal{U}(p,\mathtt{y}) = \mathtt{x}\right\}. \tag{3.3}$$

The conditional complexity can expose how much information two objects share or, on the opposite side, how much information between them is different. A particular case is when $\mathtt{y}$ is an empty object, so the conditional complexity is equal to $K(\mathtt{x}|\mathtt{y}) = K(\mathtt{x})$, the Kolmogorov complexity. The Kolmogorov complexity provides useful notions about information, despite the not computable aspect. Using an upper-bound, it is possible to approximate the Kolmogorov complexity, which can be expressed as

$$K(\mathtt{x}) \;\leq\; C(\mathtt{x}) + |D_C|, \tag{3.4}$$

where $C(\mathtt{x})$ is the length of $\mathtt{x}$ after being compressed by a compressor C and $|D_C|$ is the length of a program adequate to decompress the data compressed by C.

3.2 Markov Models

The Markov Models are one of the most popular ways of representing dependence in the data. In lossless data compression, it is used a specific type called **Discrete Time Markov Chain** or **Finite Context Model (FCM)** [78]. A $k-$order Markov model is characterized by

$$P(x_n|x_{n-1},\cdots,x_{n-k}) \;=\; P(x_n|x_{n-1},\cdots,x_{n-k},\cdots), \tag{3.5}$$

where $x_1, x_2, \cdots, x_n$ is the sequence of symbols produced by the source until the instant n. The sequence $x_{n-1},\cdots,x_{n-k}$ is called the state or context of the process. For the particular case of a first-order Markov model, the expression is reduced to

$$P(x_n|x_{n-1}) \;=\; P(x_n|x_{n-1},x_{n-2},\cdots). \tag{3.6}$$

The entropy of a process with M states S_i is naturally the average value of the entropy of each state,

$$H \;=\; \sum_{i=1}^{M} P(S_i)H(S_i), \tag{3.7}$$

where P_i is the probability of occurrence of the state S_i and $H(S_i)$ denotes the entropy of S_i.

A Markov model of a binary image, that has two states white pixel S_w and black pixel S_b, generates four possible transitions: $S_w \rightarrow S_w$, $S_w \rightarrow S_b$, $S_b \rightarrow S_w$, $S_b \rightarrow S_b$. The state transition diagram of this first-order model is depicted in Figure 3.1.

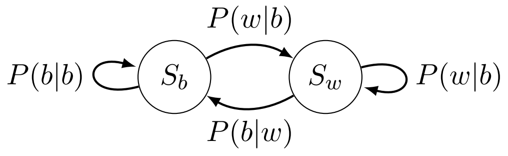


Figure 3.1: Diagram of a Two States Markov Model.

The estimate entropy for the black pixel state S_b can be defined as

$$H(S_b) \;=\; P(w|b)\frac{1}{\log_2 P(w|b)} + P(b|b)\frac{1}{\log_2 P(b|b)}. \tag{3.8}$$

For the white pixel state S_w, the estimate entropy can be expressed as

$$H(S_w) \;=\; P(b|w)\frac{1}{\log_2 P(b|w)} + P(w|w)\frac{1}{\log_2 P(w|w)}. \tag{3.9}$$

Considering the first-order Markov model for the binary image, the estimate for the entropy of the source is

$$H \;=\; P(S_b)H(S_b) + P(S_w)H(S_w). \tag{3.10}$$

Markov models are distinctively useful in text compression, due to the general heavily influence that the preceding letters have in the next letter in a word. In fact, the use of Markov models for written English appeared in the original work of Shannon [76]. Shannon used a second-order model for English text consisting of the 26 letters plus space, obtaining an estimate of an entropy of 3.1 bits/letter. Using words instead of letters, he got an estimate of 2.4 bits/letter.

The number of states of the Markov process or different contexts is given by $|\mathcal{A}|^k$, where $|\mathcal{A}|$ is the size of the alphabet and k is the order of the model. The states of the Markov process grow exponentially with the order of the model. For a two-symbol alphabet, a FCM of depth 5 originates $2^5 = 32$ states or different contexts. If the depth size is 32, then the number of contexts jumps to $2^{32} = 4294967296$.

x_{n-3}	x_{n-2}	x_{n-1}	S_b	S_w
b	b	b	0.96	0.04
b	b	w	0.67	0.33
b	w	b	0.15	0.85
b	w	w	0.45	0.55
w	b	b	0.66	0.34
w	b	w	0.10	0.90
w	w	b	0.39	0.61
w	w	w	0.22	0.78

Table 3.1: Static Markov Model of Depth 3.

The process of estimating probabilities for a model with context depth of 3 is defined in Table 3.1. The probability of having a "w" after the sequence "bww" is 0.55. The FCM can be trained online, *i.e.*, as the sequence is processed. In that case, a table collects counters that account for the number of times that each symbol occurs in each context. The probability of having a "b" after the sequence "wbw" can be estimated according to $\frac{105}{105+12} \approx 0.90$.

x_{n-3}	x_{n-2}	x_{n-1}	S_b	S_w
b	b	b	99	34
b	b	w	97	21
b	w	b	116	98
b	w	w	97	76
w	b	b	114	32
w	b	w	105	12
w	w	b	110	34
w	w	w	97	12

Table 3.2: Dynamic Markov Model of Depth 3.

3.3 Compression Techniques

The term *compression*[1] is defined as "the process by which data is compressed into a form that minimizes the space required to store or transmit it". By using a compression algorithm, it is possible to achieve reduction in data size. The unfold of recovering data from the compressed data is a decompression algorithm. In Figure 3.2 is depicted an example of data compression. Data compression can be obtained by reducing the **statistical redundancy** or by diminishing the **perceptual redundancy** [78].

A message has statistical redundancy if it is possible to convert into another message with less bits, providing that the original message can be reconstructed. The compression process can also be feasible by eliminating information that has low relevance, in this case the message has perceptual redundancy. The concept of perceptual redundancy is inspired on the fact that the human senses are not perfect, such as audition and vision. The perceptual

[1]American Heritage® Dictionary of the English Language, Fifth Edition, 2011.

redundancy method discards information, making impossible to recover the original message.

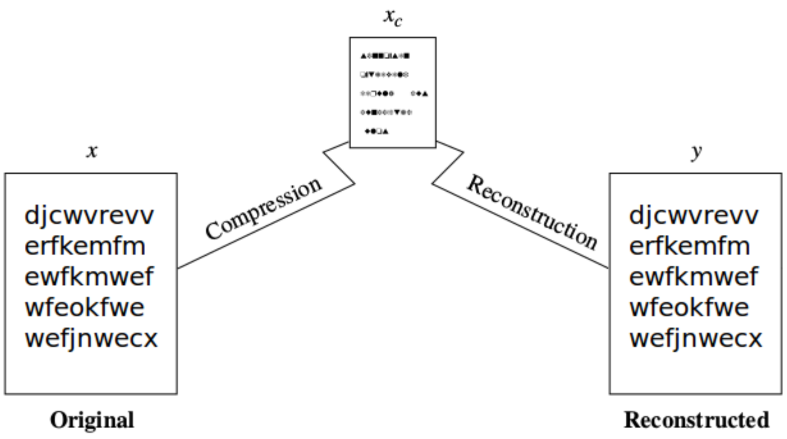


Figure 3.2: Compression and Decompression [78].

The data compression techniques can be classified in three categories:

- **Lossless Compression:** it is based only on the reduction of the statistical redundancy. Generally the lossless compression is used in applications that do not permit difference between the original and reconstructed data. Usually attains small compression rates.
- **Lossy Compression:** it is based both on the reduction of the statistical and perceptual redundancies. It is used in many applications were the loss of information is tolerable. Large compression rates can be obtained.
- **Near-lossless Compression:** it is based on the reduction of the statistical redundancy and on a controlled reduction of the perceptual redundancy [84]. Attains intermediate compression rates.

On text compression, it is not acceptable to have loss of information. An example can be observed in the compression of the following phrase:

Be Happy, as you never ?een.

In fact, this message could be:

Be Happy, as you never been.

But it could also be:

Be Happy, as you never keen.
Be Happy, as you never seen.
. . .

Therefore, the algorithms for text compression should only be based on the reduction of the statistical redundancy.

3.3.1 Huffman Coding

The Huffman coding technique was published in 1952 by David Huffman. The Huffman codes are variable length codes which have been shown that the codes are optimal prefix codes, *i.e.*, they minimize the average length of the encoded messages, among all possible variable length codes [85]. The prefix code characteristic means that none of the codewords has a prefix made of a shorter codeword. For example, if the binary sequence "10" is assigned, then the binary sequence "101" cannot represent any element. Therefore, this property ensures unique and instantaneous decoding.

Symbol	Frequency	Probability
a	35	0.35
b	19	0.19
c	16	0.16
d	16	0.16
e	9	0.09
f	5	0.05

Table 3.3: Symbol Frequencies and Probabilities.

Consequently, for constructing the Huffman codes, the symbols with the smallest probabilities are combined first. The new node gets the sum of the probabilities of the two combined nodes. This procedure is repeated until having a single node with a probability equal to one. Finally, the codewords are obtained by associating 0's and 1's to the branches of the tree. Figure 3.3 represents the Huffman Tree created based on the frequency of Table 3.3's elements.

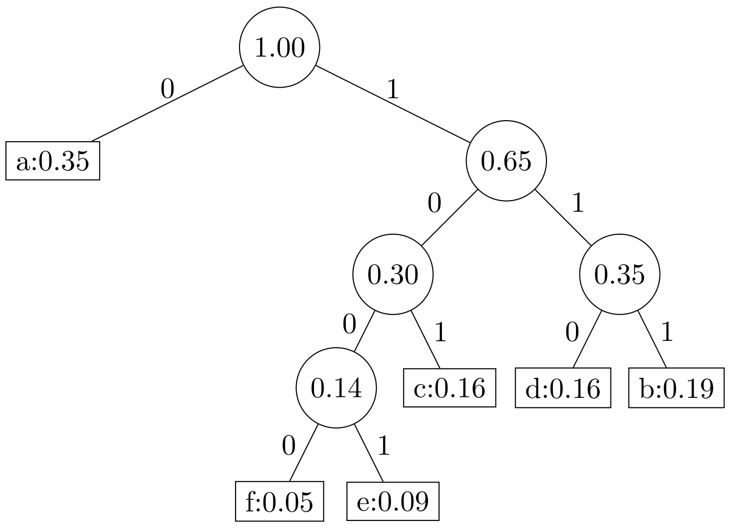


Figure 3.3: Huffman Tree Example.

The decoding process is straightforward. Suppose that the decoder received the binary sequence "0111101100010011011110". Then, the symbol sequence is:

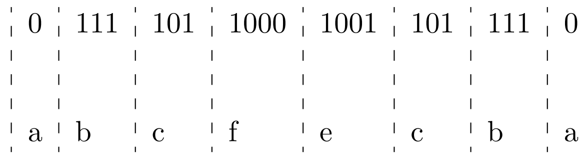


Figure 3.4: Huffman Decoding Process.

The Huffman procedure for building the codes is based on two observations regarding optimum prefix codes. In an optimum code, symbols that occur more frequently should have shorter codewords than symbols that occur less frequently. In an optimum code, the two symbols that occur less frequently should have the same length. The Huffman procedure contains the additional restriction that the codewords of the two lowest probability symbols differ only in the last bit.

3.3.2 Arithmetic Coding

The Huffman codes are optimal prefix codes, however, they require to estimate the probabilities associated with the symbols of the alphabet, performing inefficiently relatively to the entropy. **Arithmetic coding** represents more than one symbol in the message by a single codeword, encoding a message by a number in the $[0, 1)$ interval [86]. The variable length codes have the disadvantage of being affected by the skewness of the probability distribution, notwithstanding arithmetic coding is not affected by this particular issue. Besides, arithmetic coding has the advantage of allowing a clear separation between the *coding process* and *source modelling* [87].

Symbol	Probability	Interval
space	0.1	[0, 0.1)
a	0.25	[0.1, 0.35)
b	0.15	[0.35, 0.5)
e	0.3	[0.5, 0.8)
r	0.2	[0.8, 1)

Table 3.4: Arithmetic Coding Intervals.

The $[0, 1)$ interval is partitioned according to the probability distribution of the symbols. Table 3.4 represents an example of the intervals partitioning. Nonetheless, the symbols need to be in the same order in the decoder. Consider the message "*barb*" to be encoded. In the start of the procedure [88], the coding interval is $[0, 1)$. The first symbol to encode is "*b*", hence, according to the probability on Table 3.4, the assigned interval to it is $[0.35, 0.5)$. After encoding the "*b*" symbol, the coding interval will be $[0.35, 0.5)$.

The next symbol, "a", is associated with the $[0.1, 0.35)$ interval. Then, the computation of the **lower limit** of the new coding interval will be

$$0.35 + 0.1(0.5 - 0.35) = 0.365, \tag{3.11}$$

and the **upper limit**

$$0.35 + 0.35(0.5 - 0.35) = 0.4025. \tag{3.12}$$

The next symbol, "r", is associated with the $[0.8, 1)$ interval, transforming the existent coding interval $[0.365, 0.4025)$ into the new interval range,

$$0.365 + 0.8(0.4025 - 0.365) = 0.395, \tag{3.13}$$

and the **upper limit**

$$0.365 + 1(0.4025 - 0.365) = 0.4025. \tag{3.14}$$

The next symbol in the message is "b" and the associated interval is $[0.35, 0.5)$. The adjustment in the interval is

$$0.395 + 0.35(0.4025 - 0.395) = 0.397625, \tag{3.15}$$

and the **upper limit**

$$0.395 + 0.5(0.4025 - 0.395) = 0.39875. \tag{3.16}$$

Completed the encoding process of the message "$barb$", the final interval is $[0.397625, 0.39875)$. Figure 3.5 depicts the encoding process, showing the evolution of the intervals. The word could be encoded in from of interval or by a value from the interval and the length of the sequence, for example $(0.398, 4)$.

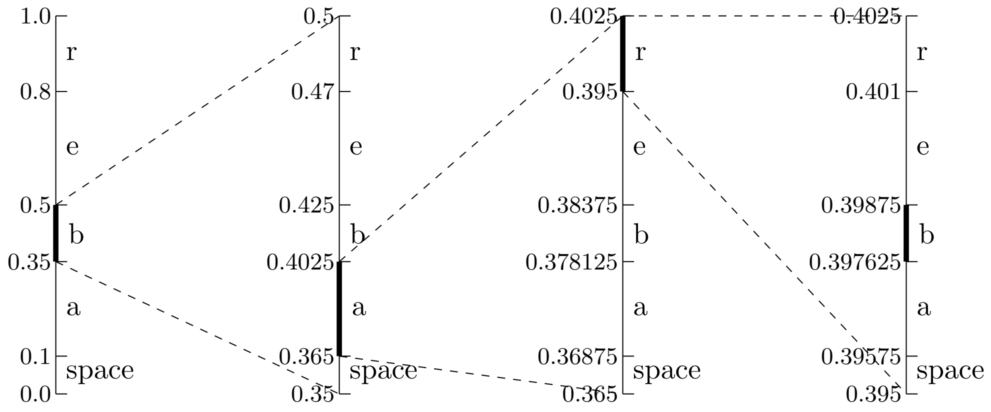


Figure 3.5: Arithmetic Encoding of the Word "$barb$".

Succeeding the iteration of the encoding process it is possible to show that if the current coding interval is $[L^n, U^n)$, and the next symbol to encode has the $[x, y)$ associated interval, the lower limit and upper limit are

$$L^{n+1} = L^n + x(U^n - L^n) \tag{3.17}$$

and

$$U^{n+1} = L^n + y(U^n - L^n). \tag{3.18}$$

The decoding process is also straightforward. The decoder receives the pair of values (z, l), representing the starting value and the length of the encoded message, respectively. Considering the pair of values $z = 0.32$ and $l = 2$, the decoding procedure starts with the decoding of the first symbol, taking in consideration that $z \in [0.1, 0.35)$. Thus, the first symbol of the message is "*a*". The next phase is to eliminate the contribution of the symbol from the encoded message. The elimination process can be achievable by using Equation 3.19, obtaining $z = 0.88 \in [0.8, 1)$, retrieving the second symbol "*r*". Due to $l = 2$, the process stops decoding the word "*ar*". The general equation to re-normalize the values can be expressed as

$$z_{n+1} = \left(\frac{z_n - L^n}{U^n - L^n} \right), \tag{3.19}$$

where z_{n+1} is the value of the next symbol, z_n the current value and $[L^n, U^n)$ the interval of the current symbol value.

3.3.3 Run-length Coding

The Run-length Encoding (RLE) is probably the simplest lossless compression scheme that takes advantage of the context. The basic idea of this method is to identify identical characters and replace them with a single occurrence along with a count [89], such as

'MMMMGGGGGGEEEEEEE' = M4G6E7.

The RLE is an appropriate technique to compress any type of data despite of its information composition. It is important to not forget that the compression ratio produced by the RLE method is affected by the data's content. The RLE procedure is easy to implement and quick to execute, typifying a good mechanism to compress generic files. Nevertheless, most RLE algorithms have poor performance compared to the high compression ratios of the more advanced compression methods.

3.3.4 Dictionary based Compression

In many circumstances, the information source produces recurring patterns. One possibility to handle the recurring patterns is to keep a dictionary with these patterns. When one of those patterns occurs, it is encoded using a reference to the dictionary. If the new occurrence does not appear in the dictionary, it can be encoded using some other compression method. Therefore, the main idea on the dictionary based compression is to divide the patterns into two broad classes, the **frequent** and the **infrequent** [78].

The statistical compression methods have high dependence on the statistical model of the source, then, in order to obtain good compression results it is needed to have a good data model. On another hand, the dictionary based compression methods do not adopt statistical models nor variable-size codes. Instead, the method elects blocks of symbols and encodes them using a token referring to a dictionary.

The management of the dictionary that holds sequences of symbols can be either **static** or **dynamic/adaptive**. In practical applications, dictionary based compressors expose, for most type of files, good compression results. Consequently, this type of encoders are very popular. Due to the good compression performances on images and audio data as well in text, the dictionary based compression methods are appropriated for a good universal purpose encoder.

LZ77

The Lempel-Ziv Coding 1977 (LZ77) is a lossless data compression algorithm that relies on the separation of data in two parts: data already encoded and data that is still to be encoded [90]. In the LZ77 procedure, the data goes first through a look-ahead buffer and then over a search buffer or the dictionary. The LZ77 algorithm searches, in the buffer, the largest sequence of symbols that can be found in the look-ahead buffer. The larger the sequences that are found, the higher is the coding efficiency. The algorithm expects that repeating sequences occur close to each other.

Dictionary										Look-ahead buffer				Codeword
10	9	8	7	6	5	4	3	2	1					
										x	y	x	x	(0,0,x)
									x	y	x	x	y	(0,0y)
								x	**y**	x	x	y	x	(2,1,x)
						x	**y**	**x**	**x**	y	x	z	y	(3,2,z)
			x	**y**	**x**	**x**	**y**	**x**	**z**	y	x	z	x	...

Figure 3.6: LZ77 Encoding Procedure.

The codeword that the LZ77 algorithm generates in each coding step is composed by three components:

- A **pointer**, that indicates the position of the repeating sequence in the dictionary.
- The **length** of the sequence.
- The **first symbol** in the look-ahead buffer that follows the matching sequence.

There are some problems associated with the LZ77 compression algorithm. The benefits of increasing the size of the dictionary or the the size of the look-ahead buffer are the increasing of the compression efficiency and the probability of finding larger sequences, respectively.

However, in both cases, the processing time also rises and more bits are needed to represent the pointers to the dictionary and the length of the sequences. Generally, the three component codewords are inefficient.

LZ78

The Lempel-Ziv Coding 1978 (LZ78) method does not use any sliding window, look-ahead buffer, or search buffer [91]. The LZ78 uses a distinct approach from that of LZ77, and that is, the data is partitioned into strings. Each string is built of the largest matching string found so far, plus the first non-matching character. The new string is represented using the index of the matching string in the dictionary, followed by the code of the unmatched symbol. Finally, the new string is added to the dictionary. Figure 3.7 exposes an example of the LZ78 procedure.

Dictionary:	x	xx	y	xxx	z	xxy	xxz	k
Index:	1	2	3	4	5	6	7	8
Codeword:	(0,x)	(1,x)	(0,y)	(2,x)	(0,z)	(2,y)	(2,z)	(0,k)

Figure 3.7: LZ78 Coding Message.

Practically, the dictionary can not grow without any limits, as the size of the codewords is related to the size of the dictionary, since the dictionary entries have associated indexes. For example, for a size n dictionary, the pointers have to be stored with $\log_2 n$ bits. The amount of memory for storing the dictionary in the encoder and the decoder might be too large. Therefore, in practice, it is necessary to implement a mechanism for limiting the growth of the dictionary.

Some of those mechanisms could “freeze” the dictionary when it reaches certain predefined size. When the dictionary reaches the predefined size, it is deleted and starts growing again. When the dictionary is full, some entries are deleted, for example, those how are not used for a longer time.

LZW

In 1984, Terry Welch proposed solutions to the problems associated to LZ78 in a published paper [92]. The LZ78 variant proposed received the name of Lempel-Ziv-Welch Coding (LZW). The LZW method starts by initializing the dictionary with all the symbols in the alphabet. With this, the symbols seen for the first time are encoded more efficiently. The number of components of the codewords is reduced to one, the index of the dictionary.

The LZW procedure begins by inserting all the symbols of the alphabet in the dictionary. The LZW’s encoder reads the symbols one by one and collects them into a string J. The concatenation of a new symbol i is performed as long as the new string Ji is in the dictionary. For the case that the search for the string Ji fails, the index of J in the dictionary is sent, Ji is inserted in the dictionary, and J is initialized with symbol i.

3.4 Image Compression

In modern times the digital image has a great importance in the general life of a human being. In the everyday life, digital images have a wide range of applications, such as, representing some type of complex data in an understandable and accessible manner or even converting moments into permanent memories. Therefore, images are very important, but storing the raw image information in general would cost a lot.

Image compression addresses the problem of the spacious feature of images, displaying an important topic in the image field. Image compression can be lossless or lossy. Usually, image compression is obtained by throwing away information that is not so relevant to the human visual system. However, for some applications, it is necessary to ensure that the original data can be reconstructed from the encoded data or, at least, to known the upper bound of the maximum error [93].

3.4.1 GIF

The Graphics Interchange Format (GIF) image format was designed by Compuserve Information Services in 1987. The first version of GIF is known as GIF 87a [94]. Essentially, GIF is not an image compression technique, but rather a graphics file format that uses a variant of LZW to compress the graphics data. The compressed graphics file format GIF is a very popular format in the World Wide Web due to its portability and the animation support.

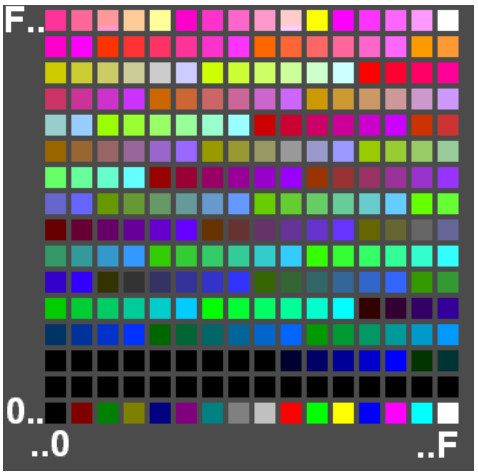


Figure 3.8: GIF 256 Colour Palette Example.

The format supports up to 8 bits per pixel for each image, allowing a single image to reference its own palette of up to 256 different colors chosen from the 24-bit RGB color space. It allows a separate palette of up to 256 colors for each frame [95]. These palette limitations makes GIF less suitable for reproducing color photographs and other images with color gradients, but it is well-suited for simpler images such as graphics or logos with solid areas of color.

The GIF format is not an efficient image compressor, however is continually used nowadays by web browsers. The inefficiency of the GIF format is deduced of the reason that the compression method that the GIF employs is one-dimensional, while an image is two dimensional. The image format scans the image row by row, so it discovers pixel correlations within a row, but not between rows.

3.4.2 PNG

In the 1990s, the PNG development group created the Portable Network Graphics (PNG) image file format, an improved and non-patented replacement for GIF. The aim of the PNG development was to design a flexible sophisticated graphics file format that had the ability to support many different types of images and was simple to transmit over the Internet [96]. The modern PNG version is supported by many web browsers and image viewers on various platforms.

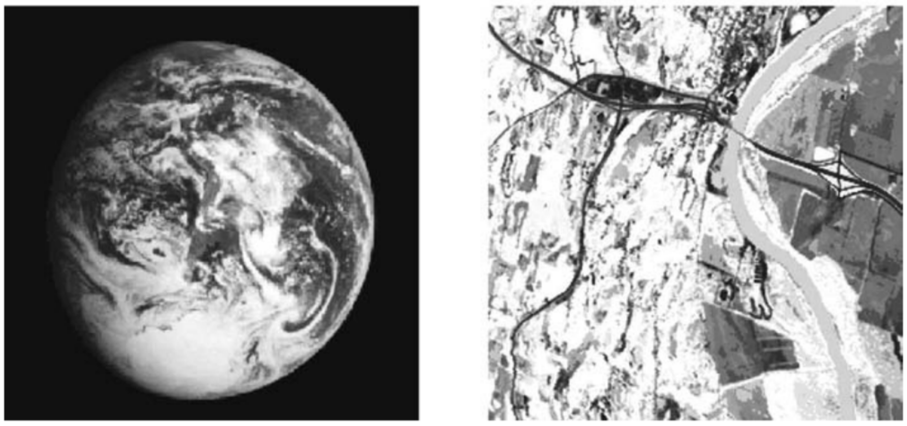


Figure 3.9: Earth and Omaha PNG Example Images [89].

In October of 1996, the PNG development group concluded the first version of the PNG format. The main implemented features on the first version were the support of images with 1, 2, 4, 8 and 16 bitplanes, sophisticated colour matching and the implementation of an alpha channel in order to provide a transparency feature with very fine control. Additionally, there was the possibility of extensibility, this is, new types of meta-information could be added to an image file without creating incompatibility with existing applications.

Image	Original Size	PNG	GIF	Arithmetic Coding of Pixel Values
Earth	65.536	26.995	34.276	38.248
Omaha	65.536	50.185	61.580	56.061

Table 3.5: Comparison of PNG with GIF and Arithmetic Coding in Bytes [89].

Contrary to GIF, the compression algorithm used in the PNG image format is based on the LZ77 method. A performance comparison between PNG and GIF of the images from Figure 3.9 is shown in Table 3.5. The PNG compression is lossless and is executed in two steps. In the first phase is applied a delta filtering or simply defined as filtering. The filtering process replaces the pixel with the difference between the pixel and its predicted value. This process is similar to the prediction in the JPEG lossless mode.

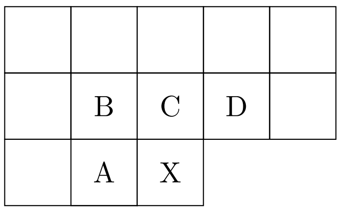


Figure 3.10: PNGs Prediction Structure.

There are five different ways of computing the predicted value and PNG allows the use of a different filters for each row. The method number 0 or *None* is simply no filtering, the number 1 or *Sub* uses the pixel from the row above C as the estimate. The *Up* or number 2 uses the pixel to the left of A. The number 3 or *Average* as the name indicate uses the average of the pixel above and the pixel to the left, $mean(\mathrm{A}, \mathrm{C})$. The final approach is termed *Paeth* and is a bit more complex. The estimation is based on the following expression: $\mathrm{A} + \mathrm{C} - \mathrm{B}$. The pixel that is the closest to the initial estimate is taken as the estimate [89].

The second step on the PNG compression is the application of the Deflate algorithm to encode the differences. The Deflate is a lossless data compression algorithm that uses a combination of the LZ77 algorithm and Huffman coding [97]. Natural images do not have extensive repetition of sequences of pixel values. However, pixel values in the image that are spatially close normally have values that are similar. The PNG standard makes use of this structure by estimating the value of a pixel based on its causal neighbors and subtracting this estimate from the pixel.

3.4.3 JBIG

The Joint Bilevel Image Processing Group (JBIG) method is a context-based lossless compression process for bi-level images. The JBIG is similar to PNG in the way that it uses a local context of pixels to code the current pixel. However, instead of computing prediction values, JBIG uses conditional probabilities directly. A relevant feature of JBIG is the progressive compression, allowing to send an image as a set of layers with increased resolution [98].

The term *progressive compression* indicates that the image is stored in individual layers in the compressed stream. Upon the decoder process, the image displayed is imprecise and rough, but over time the decoder will output improved versions of it. The JBIG algorithm can be applied to grayscale images by separating the bitplanes and compressing each individually as if it was a bi-level image. The use of Gray codes in this process improves the compression efficiency [99].

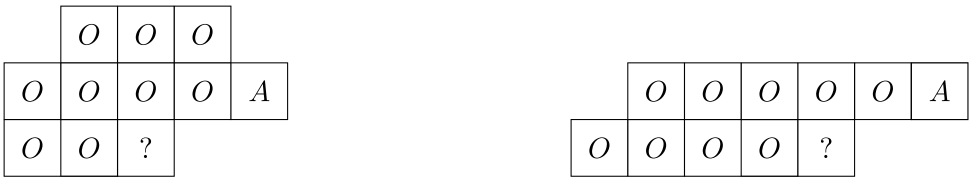


Figure 3.11: JBIG Contexts Three Line (left) and Two Line (right) Templates.

The foundation of the JBIG is on adaptive FCMs followed by arithmetic coding. The JBIG is an example of a Markov model used for source modeling. The context model used in JBIG relies on 1024 contexts when operating in sequential mode or on low resolution layers of the progressive mode, or 4096 contexts when encoding high resolution layers [100]. For each pixel, JBIG examines a template made of the 10 neighboring pixels, marked as "O" and "A". Based on the value of these pixels, the respective statistical model that will be used to encode the current pixel, marked as "?", is chosen.

In Figure 3.11 are depicted two examples of template contexts used in JBIG for the sequential mode and for the low resolution mode. The encoder decides whether to use the three line or the two line template and indicates this choice in the bitstream. The pixel labeled as "A" in the template is called adaptive pixel. The encoder is allowed to use as "A" a pixel outside the template and it uses two parameters in each layer to indicate the position of "A" in that layer.

In 1999, a new version, named JBIG2, has been published, introducing additional functionalities to the JBIG, such as two modes of progressive compression, lossy compression, multipage document compression and differentiated compression methods for different regions of the image [101].

3.4.4 JPEG

Joint Photographic Experts Group (JPEG) is a lossy compression scheme for colour and gray scale images [102]. It works on full 24-bit colour, and was designed to be used with photographic material and naturalistic artwork. However, it is not the ideal format for line-drawings, textual images or other images with large areas of solid colour or a very limited number of distinct colours. The lossless techniques, such as PNG, work better for such type of images.

JPEG is designed so that the loss factor can be tuned by the user to trade-off image size and image quality, and is designed so that the loss has the least effect on human perception [103]. However, it does have some anomalies when the compression ratio gets high, such as odd effects across the boundaries of 8×8 blocks. For high compression ratios, other techniques such as wavelet compression appear to give more satisfactory results. An overview of the JPEG compression process is portrayed in Figure 3.12.

The inputs of JPEG are three colour channels of 8-bits per-pixel, each representing Red, Blue and Green. These are the colours used by hardware to generate images. The first step of JPEG compression, which is optional, is to convert the RGB plane into YC_BC_R colour planes that are designed to better represent human perception. The YC_BC_R colour spaces separate the chrominance component C_BC_R from the luminance component Y. The human eye is more sensitive to greens, which are represented mainly by the Y component. For this reason, it is common to sub-sample the chrominance components C_BC_R, producing a reduction in the data rate.

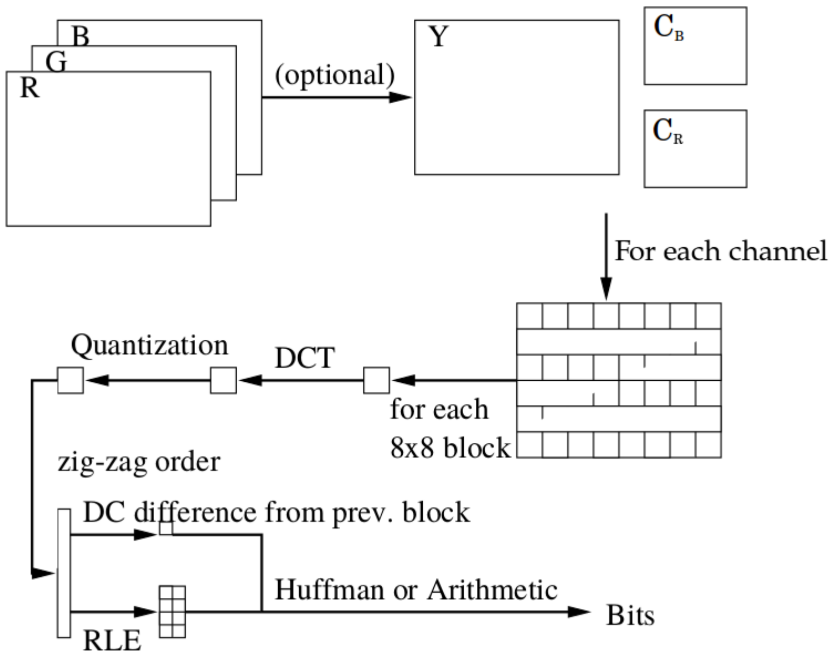


Figure 3.12: JPEG Compression Structure.

The JPEG standard allows all 256 values in an 8 bits per component representation [104]. In this case, considering R,G and B $\in \{0, \cdots, 255\}$, the YC_BC_R conversion is

$$Y = (77/256)R + (150/256)G + (29/256)B, \tag{3.20}$$

$$C_B = (44/256)R - (87/256)G + (131/256)B + 128, \tag{3.21}$$

$$C_R = (131/256)R - (110/256)G - (21/256)B + 128. \tag{3.22}$$

On the other hand, the transposed conversion to RGB is

$$R = Y + 1.371(C_R - 128), \tag{3.23}$$

$$G = Y - 0.698(C_R - 128) - 0.336(C_B 128), \tag{3.24}$$

$$B = Y + 1.732(C_B - 128). \tag{3.25}$$

The JPEG keeps all Y pixels for the intensity and typically it reduces the samples of the chrominance colour channels by a factor of 2 in each dimension. Therefore, the type of chrominance sub-sampling used in JPEG is the $4:2:0$ [105]. This is the first lossy component of JPEG and gives a compression factor of 2. The next stage of the JPEG algorithm is the partition of each colour channel into 8×8 blocks. Then each of the three blocks are coded separately.

The first step in coding a block is to apply the Discrete Cosine Transform (DCT) across both dimensions, which returns an 8×8 block of 8-bit frequency terms. The following step applied to the blocks is the use of uniform scalar quantization on each frequency term. This quantization is controllable based on the user parameters and is the main source of information loss in JPEG compression. Since the human eye is more perceptive to certain frequency components than to others, JPEG allows the quantization scaling factor to be different for each frequency component.

$$\mathbf{Q} = \begin{bmatrix} 16 & 11 & 10 & 16 & 24 & 40 & 51 & 61 \\ 12 & 12 & 14 & 19 & 26 & 58 & 60 & 55 \\ 14 & 13 & 16 & 24 & 40 & 57 & 69 & 56 \\ 14 & 17 & 22 & 29 & 51 & 87 & 80 & 62 \\ 18 & 22 & 37 & 56 & 68 & 109 & 103 & 77 \\ 24 & 35 & 55 & 64 & 81 & 104 & 113 & 92 \\ 49 & 64 & 78 & 87 & 103 & 121 & 120 & 101 \\ 72 & 92 & 95 & 98 & 112 & 100 & 103 & 99 \end{bmatrix} \tag{3.26}$$

The quantization matrix (3.26) controls the compression ratio. However, the encoder can use different quantization tables and store them in the head of the compressed file. The selection of the matrix values are based on studies of human perception. The matrix with the largest components in the lower right corner represents the highest frequency components. Then, the 8×8 quantization matrix $\mathbf{Q}_{i,j}$ is used to element-wise divide the 8×8 unquantized DCT matrix $\mathbf{I}_{i,j}$, followed by a rounding to the nearest integer [106]. The resulting quantized DCT coefficients

$$\mathbf{O}_{i,j} \;=\; round\left(\frac{\mathbf{I}_{i,j}}{\mathbf{Q}_{i,j}}\right), \quad for\ i = 0, \cdots 7,\ and\ j = 0, \cdots 7, \tag{3.27}$$

will often drop most of the terms in the lower left corner to zero.

JPEG then compresses the Direct Current (DC) component separately from the other components. Particularly, it uses a difference coding by subtracting the value given by the DC component of the previous block from the DC component of the current block. Then it is used Huffman or arithmetic coding to encode the differences. The stimulus for this method is that the DC components are often similar from block-to-block so using a difference coding it is possible to achieve better compression ratio.

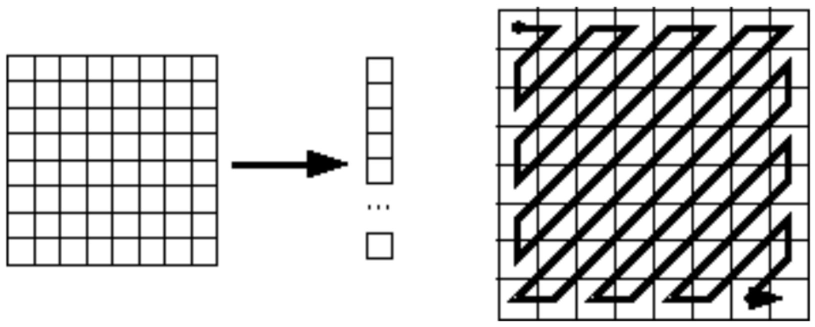


Figure 3.13: JPEG Zig Zag Illustration [93].

The Alternating Current (AC) components are first converted into a linear order by traversing the frequency table in a zig zag order. The zig zag process is exemplified in Figure 3.13. The interest for this order is that it keeps frequencies of approximately equal length close to each other in the linear-order. In particular, most of the zeros will appear as one large contiguous block at the end of the order. A form of run-length coding is used to compress the linear-order. It is coded as a sequence of $(skip, value)$ pairs, where skip is the number of zeros before a value, and value is the value.

Chapter 4

Handwritten Digit Recognition using Finite Context Models

Handwritten Digit Recognition is a long-established classification problem. The use of compression based techniques for Handwritten Digit Recognition is a novel concept. However, in the last decades, the use of data compression models has been positively used in data mining and machine learning problems [107] [108]. The techniques used are usually supported by means of a formalization in terms of the information content of a string or of the information distance between strings. These concepts are heavily influenced by solid foundations on the algorithmic entropy, known as Kolmogorov complexity. Despite its non-computability, new measures depend on approximations provided by data compression algorithms [83].

This chapter is dedicated to the experimental evaluation of the proposed handwritten digit classification method. The chapter starts with a brief introduction of Similarity Measures. Then, the proposed method is presented and explained. The techniques applied to the digit images are exposed and a concise description of the technique is unveiled. Lastly, the performance results earned from the proposed method are presented and some considerations about the results are made.

4.1 Similarity Measures

A similarity measure is used to compute how identical two separate individual objects are. The use of concepts such as Kolmogorov Complexity, $K(\mathtt{x})$, and Kolmogorov Conditional Complexity, $K(\mathtt{x}|\mathtt{y})$, have been used to support the similarity measures in recent years. The first work about similarity measures started with the **Information Distance** [109]. The information distance is delineated as the length of the smallest procedure that generates $\mathtt{x}$ from $\mathtt{y}$ and $\mathtt{y}$ from $\mathtt{x}$. The information distance can be defined as

$$E(\mathtt{x},\mathtt{y}) \;=\; \max\{K(\mathtt{y}|\mathtt{x}), K(\mathtt{x}|\mathtt{y})\}\,. \tag{4.1}$$

The equation presented is not normalized, and may not be a good description for a similarity measure [107]. In the case of two very long strings that differ only in a few positions, it would give the same as the distance between two short strings that differ by the same amount.

4.1.1 Normalized Information Distance

The similarity measure, Normalized Information Distance (NID), is an improvement on the information distance, although it depends on the uncomputable Kolmogorov's complexity $K(\mathtt{x})$. However, it is still a useful measure, and many compression-based alternatives have been proposed to approximate its results [83]. The NID can be expressed as

$$\mathrm{NID}(\mathtt{x},\mathtt{y}) \;=\; \frac{\max\{K(\mathtt{x}|\mathtt{y}),K(\mathtt{y}|\mathtt{x})\}}{\max\{K(\mathtt{x}),K(\mathtt{y})\}}. \tag{4.2}$$

With some knowledge on Kolmogorov's complexity, some results can be extracted from the expression above. If two identical objects are tested, $\mathtt{x} \equiv \mathtt{y}$, then $K(\mathtt{x}|\mathtt{y}) = K(\mathtt{y}|\mathtt{x}) = 0$ and the result of the measure is $\mathrm{NID}(\mathtt{x},\mathtt{y}) = 0$. In the counter example, where $\mathtt{x}$ and $\mathtt{y}$ are completely unrelated, $K(\mathtt{x}|\mathtt{y}) = K(\mathtt{x})$ and $K(\mathtt{y}|\mathtt{x}) = K(\mathtt{y})$, so the distance between them is 1. The use of compressors to approximate NID should return identical results. Such concept achieved appealing results in some PR tasks [107].

4.1.2 Normalized Compression Distance

The Normalized Compression Distance (NCD) was developed as an approximation to the NID measure, endeavoring to give an approximation of Kolmogorov Complexity [110]. The application of a compressor C to compute the distance between to objects $\mathtt{x}$ and $\mathtt{y}$, can be expressed as

$$\mathrm{NCD}(\mathtt{x},\mathtt{y}) \;=\; \frac{C(\mathtt{xy}) - \min\{C(\mathtt{x}),C(\mathtt{y})\}}{\max\{C(\mathtt{x}),C(\mathtt{y})\}}, \tag{4.3}$$

where $\mathtt{xy}$ is the concatenation of $\mathtt{x}$ and $\mathtt{y}$, $C(\mathtt{xy})$, $C(\mathtt{x})$ and $C(\mathtt{y})$ are the compression size of $\mathtt{xy}$, $\mathtt{x}$ and $\mathtt{y}$, respectively. The employment of an optimal compressor C, considering that the information in $\mathtt{x}$ is contained in $\mathtt{y}$, results in $C(\mathtt{y}) \geq C(\mathtt{x})$ and $C(\mathtt{xy}) = C(\mathtt{y})$, ending up with a distance result of 0. For the reversed side, if the $\mathtt{x}$ and $\mathtt{y}$ are entirely unrelated, then $C(\mathtt{xy}) = C(\mathtt{x}) + C(\mathtt{y})$, for example the $\min\{C(\mathtt{x}),C(\mathtt{y})\} = C(\mathtt{x})$ and the inverse $\max\{C(\mathtt{x}),C(\mathtt{y})\} = C(\mathtt{y})$, returning the NCD distance as 1. The NCD has been effectively employed in applications on diverse data types as a parameter free approach [111] [112].

4.1.3 Normalized Conditional Compression Distance

In 2014, the Normalized Conditional Compression Distance (NCCD) was proposed. This similarity measure computes the distance between two strings of characters [113] [114]. The NCCD is a notation similar to the NID where K changes to C and can be defined as

$$\mathrm{NCCD}(\mathtt{x},\mathtt{y}) \;=\; \frac{\max\{C(\mathtt{x}|\mathtt{y}),C(\mathtt{y}|\mathtt{x})\}}{\max\{C(\mathtt{x}),C(\mathtt{y})\}}. \tag{4.4}$$

In contrast with the previous introduced measures, the NCCD requires a compressor capable of conditional compression, *i.e.*, compressing an object using the models built from another objects. The conditional compression, $C(\mathtt{x}|\mathtt{y})$, encodes $\mathtt{x}$ using models built from both $\mathtt{y}$ and $\mathtt{x}$. The term $C(\mathtt{x}|\mathtt{y})$ represents the number of bits required to represent $\mathtt{x}$ given the information enclosed in both $\mathtt{x}$ and $\mathtt{y}$.

4.1.4 Normalized Relative Compression

In 2016, the Normalized Relative Compression (NRC) similarity measure was published. The investigation produced a less computationally demanding method than the other compression-based measures [115]. The NRC method is restricted to compressors capable of exclusive conditional compression, and for that the NRC can be expressed as

$$\mathrm{NRC}(\mathtt{x}||\mathtt{y}) = \frac{C(\mathtt{x}||\mathtt{y})}{|\mathtt{x}|}, \tag{4.5}$$

where $C(\mathtt{x}||\mathtt{y})$ is an exclusive conditional compression, *i.e.*, x is compressed using exclusively models assembled from `y`. The use of conditional compression means that the information in `x` is not used to compress it, so if x and y are completely independent, $C(\mathtt{x}||\mathtt{y}) = |\mathtt{x}|$, $\mathrm{NRC}(\mathtt{x}||\mathtt{y}) = 1$. That is, if `y` does not contain any information suitable to `x`, `x` will not be compressed at all. Therefore, the NRC is equal to 0 when both objects are identical and 1 if they are completely different. The NRC measure has been successfully used in important data mining problems [116] [117].

4.2 Proposed Method

The proposed method is based on the NRC similarity measure, with the use of a FCM compression method, as the compressor capable of performing conditional compression. The components that constitute the proposed classification method are described with more detail later in this section. In Figure 4.1 is depicted the general plan of architecture of the proposed handwritten digit classification system.

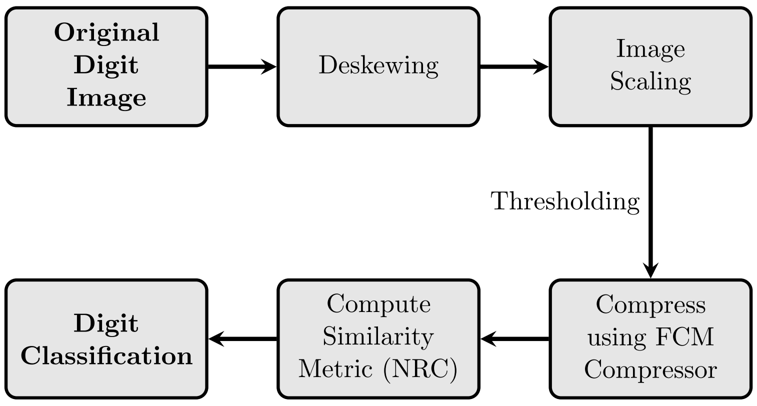


Figure 4.1: Architecture of the Handwritten Digit Classification Proposed Method.

The architecture proposed has as input the raw data of the handwritten digit image. Then the input data is deskewed and downscaled which is explained later in this chapter. The employment of a simple threshold function on the resulting image of the precedent methods creates a binary image. The threshold makes possible to attain an alphabet reduction, by introducing a lossy compression component.

The resultant binary image is then compressed by the specific designed FCM compressor, using exclusively models constructed from a training set. Following the process chain, the compression size is used by the NRC measure to compute the similarity between the models and allowing to give a classification to the handwritten digit image.

The method proposed in this dissertation was constructed around the MNIST database, with the intention of reaching the smallest error in the classification of the test set of the database. The following subsections define the pre-processing techniques used in the proposed method: deskewing and Nearest-Neighbor Scaling (NNS).

4.2.1 Deskewing

The writing process is influenced by the surrounding environment. For example, the writing can be shaped with angles on the printing surface, which generate skewed symbols. The human eye can easily find similarities between images that are transformations of each other. However, the computer has difficulties on this task. The **deskewing** process addresses this problem. Deskewing is the act of aligning an image that has been scanned or written crookedly, an image that is at an angle too pronounced in one direction, or one that is misaligned [118]. Figure 4.1 shows an example of deskewing applied to the MNIST dataset.

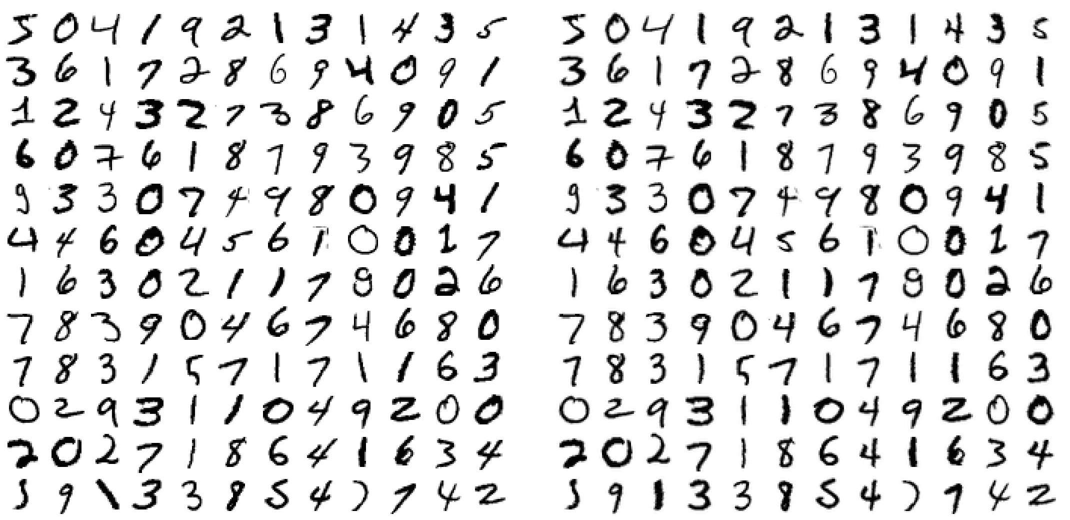


Figure 4.2: The MNIST Training Set Digit Images Skewed (left) and Deskewed (right).

The deskewing method is modeled as an affine transformation, this is, is assumed that the skewed image version is created from the following process: $\mathbf{I}' = \mathbf{MI} + \mathbf{b}$, where the current image $\mathbf{I}'$ is a skewed version from the perfectly aligned image $\mathbf{I}$. The process starts by calculating the center of mass of the image, determining how much is required to offset the image. Followed by the computation of the covariance matrix of the pixel intensities, making it possible to use this matrix to approximate the skew of the angle [119].

4.2.2 Nearest-Neighbour Scaling

The NNS is presented in the literature as the simplest and fastest implementation of an image scaling algorithm. The NNS is an appropriate algorithm were the speed of scaling an image is an important factor. The premise in image scaling is that the reference image is used to produce a new scaled image based on the original. The composed image can have the same dimensions or increased or decreased dimensions, which only depend on the scaling ratio. The NNS algorithm samples and rounds the nearest pixel of the original image to originate the new scaled image[120].

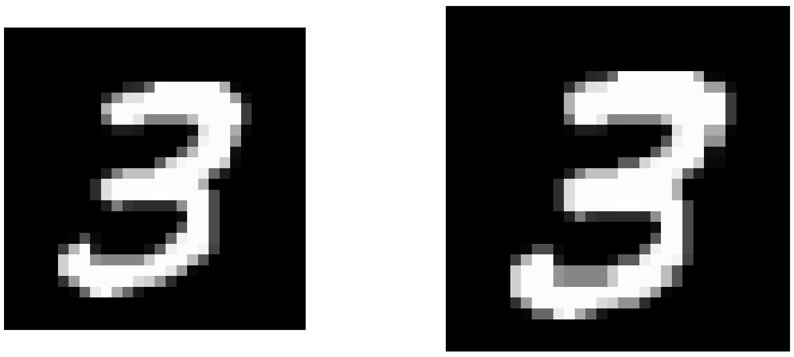


Figure 4.3: NNS Up Scaling Example.

The scaling algorithm has the task of finding appropriate positions to put empty spaces inside the original image and to fill all those spaces with colours [121]. For the nearest neighbor technique, the empty spaces are replaced with the nearest neighboring pixel. Figure 4.3 shows an example of an up scaling using the NNS on a 28×28 to 32×32 image. The NNS is used on the proposed method to shrink images. Figure 4.4 depicts an example of a 28×28 sample of the MNIST dataset transformed to a new scaled 16×16 image. The procedure makes a reduction of pixels and has to find the right pixels to throw away. This process means losing irrecoverable information. The NNS is also one of the lossy components of the system.

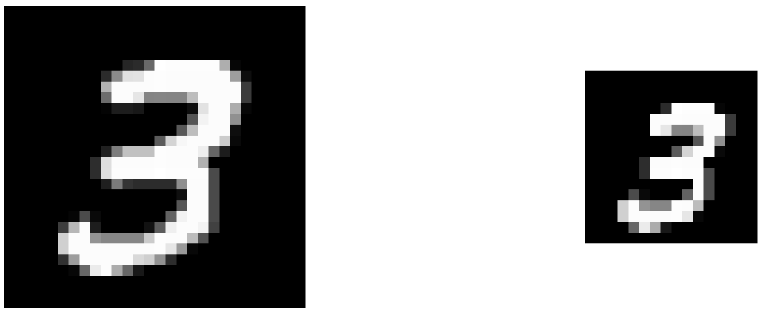


Figure 4.4: NNS Down Scaling Example.

The NNS appears to be the simplest scaling algorithm, although there are more complex variations of scaling algorithms such as sinc, bilinear, spline, bicubic and many others. The major difference between these referred approaches and the NNS algorithm is that they use interpolation of neighboring pixels, accomplishing smoother images. More complex scaling solutions may use adaptive algorithms, where the algorithm can apply different levels of interpolation on different areas of an image.

4.2.3 Finite Context Model Compressor

The proposed digit image encoder is based on finite context modeling, similar in theory to the used by JBIG. The main drawback of the FCM approach is that usually the order of the model makes that the required memory resources grow exponentially. For this reason, it is unpractical to apply them to large alphabets. However, it is possible to ease the memory stress through the use of sophisticated data structures such as hash tables. The implemented method uses a hash table to handle the context modelling and also the size of the alphabet is reduced.

A FCM attribute probability estimates to the symbols of the alphabet $\mathcal{A} = \{s_1, s_2, \cdots, s_{|\mathcal{A}|}\}$, where $|\mathcal{A}|$ expresses the size of the alphabet, having in attention the next outcome of the information source [122]. The alphabet in the FCM used for this work is $\mathcal{A} = \{0, 1\}$. For the probabilities computation, it is taken in account a conditioning context computed over a finite and fixed number $k > 0$ of past outcomes $x_{n-k+1\cdots n} = x_{n-k+1} \cdots x_{n-1}x_n$, an order-$k$ FCM. With this, it is easy to say that the number of conditioning states of the model is $|\mathcal{A}|^k$ [123].

The notion of recent past in the case of multi-dimensional data, and particularly in the case of images, is referred as spatial proximity. This means that $x_{n-k+1\cdots n}$ may refer to the set of the k spatially closest samples, and not necessarily to the k most recently processed samples. Nevertheless, causality is always preserved. The probability estimates, $P(X_{n+1} = s|x_{n-k+1\cdots n}), \forall s \in \mathcal{A}$, are calculated using symbol counts that were accumulated on the training images previous processed.

The estimation of the probabilities can be attained by using the expression

$$P(s|x_{n-k+1\cdots n}) = \frac{d(s|x_{n-k+1\cdots n}) + \alpha}{d(x_{n-k+1\cdots n}) + \alpha|\mathcal{A}|}, \tag{4.6}$$

where $|\mathcal{A}|$ is the alphabet size that describes the objects, $d(s|x_{n-k+1\cdots n})$ represents the number of times that the symbol $s \in \mathcal{A}$ was found in the training set, having $x_{n-k+1\cdots n}$ as the conditioning context. Lastly, the $d(x_{n-k+1\cdots n})$ can be described as the total number of events that has occurred within context $x_{n-k+1\cdots n}$,

$$d(x_{n-k+1\cdots n}) = \sum_{a \in \mathcal{A}} d(a|x_{n-k+1\cdots n}). \tag{4.7}$$

The parameter α in the Equation 4.6 addresses the problem of the zero frequency, and also allows the estimator to balance between the maximum likelihood estimator and an uniform distribution. It is possible to notice that when the total number of events, n, is large, the estimator behaves like a maximum likelihood estimator. Another note that can be pointed is that for $\alpha = 1$ the estimator is the well-known Laplace estimator [124].

After processing the first n symbols of x, the total number of bits generated by an order-k FCM is given by

$$-\sum_{i=1}^{n} \log_2 P(x_i | x_{i-k \cdots i-1}), \tag{4.8}$$

where $P(x_i | x_{i-k \cdots i-1})$ is the estimate probability computed with the auxiliary of the counters performed on the training data.

4.2.4 Algorithm Description

The proposed handwritten digit classification system can be divided in two main structures: the offline FCM training, Table 4.1, and the FCM encoder, Table 4.2. The FCM training procedure uses the MNIST training set to build models from the data. The inputs of the training procedure are the training images matrix **I** containing the full MNIST training set, the digit's training image labels vector **d**, the established $2D$ context vector **c** and the threshold constant t.

Algorithm: The Offline Finite Context Model Training

Inputs: I, d, c, t

```
1: $\mathscr{H} = \{\}$
2: for $i = 1$ to images do
3:     for $j = 1$ to columns do
4:         for $k = 1$ to rows do
5:             $\mathbf{T}_{j,k} = \begin{cases} 0, & \text{if } \mathbf{I}_{i_{j,k}} < t \\ 1, & \text{otherwise} \end{cases}$
6:             $\mathbf{v} = getContext(\mathbf{T}_{j,k}, \mathbf{c})$
7:             $\mathscr{H}_{\mathbf{d}_i}(\mathbf{v}|\mathbf{T}_{j,k}) = \mathscr{H}_{\mathbf{d}_i}(\mathbf{v}|\mathbf{T}_{j,k}) + 1$
8:         end for
9:     end for
10: end for
```

Table 4.1: The Offline Finite Context Model Training.

The method starts by initializing the data structure that will collect counters that account for the number of times that each symbol occurs in each context. In the proposed method, the data structure chosen is a hash table $\mathscr{H}$. The images are processed one by one and the pixels are read in raster scan. On the fly, the images pixel are modified by the simple thresholding method in the two main modules. The threshold function can be expressed as

$$\mathbf{T}_{j,k} = \begin{cases} 0, & \text{if } \mathbf{I}_{i_{j,k}} < t \\ 1, & \text{otherwise} \end{cases}, \tag{4.9}$$

where $\mathbf{I}_i$ is the input image intensity matrix, t is a limiting intensity constant and $\mathbf{T}$ is the resulting image. The first impression is that the loss of information might be too severe, but the obtained results show that the information loss is relatively irrelevant.

The function *getContext* extracts the context for every pixel. If the context of a pixel is outside of the image, the attributed value is zero. This proceeding is identical to an image padding technique, were the image is increased in size and the padded areas are filled with black pixels. The symbol to encode $\mathbf{T}_{j,k}$, and the respective context $\mathbf{v}$ extracted with the assistance of the *getContext* function, are introduced in the hash table according to the corresponding label. If the instance exists, the counter is updated by one, otherwise the counter is initialized as one.

Context			Symbol	
x_{n-3}	x_{n-2}	x_{n-1}	0	1
0	0	0	3694711	141202
0	0	1	7975	135869
0	1	0	7801	432
0	1	1	28604	109482
1	0	0	141199	2642
1	0	1	258	2217
1	1	0	136043	2043
1	1	1	109482	286744

Table 4.2: Conditional Frequency Table of Digit 3 for the MNIST Training Set.

In the training module, the MNIST training set is converted in ten conditional frequency tables, one for each class of the MNIST set. Table 4.2 shows a filled table with counters that model the digit 3 images of the MNIST training set. The template context used to collect the data presented on Table 4.2 can be observed in Figure 4.5, where the context relates to the three previous processed pixels.

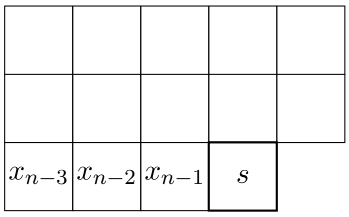


Figure 4.5: Context Template used in the MNIST Training Set.

After processing all the images in the MNIST training set and constructing the models, the MNIST test set is then classified by the proposed encoder. The classifier module, Table 4.3, can be rapidly described as a conditional compressor, in which the images of the MNIST test set are compressed by ten models previously constructed in the training module. The inputs of the classifier module are the hash table $\mathscr{H}$ that contains each digit probabilistic model, and the image matrix **I** that accommodates the MNIST test set images. The **c**, t, α inputs are the same as the training module and it is demanded that the content of these variables are identical for the two modules.

Algorithm: The Proposed Finite Context Model Encoder

Inputs: $\mathscr{H}$, $\mathbf{I}$, $\mathbf{c}$, t, α

```
1: $\mathbf{g} = \{\}$
2: for $i = 1$ to images do
3:     $\mathbf{b} = \{\}$
4:     for $j = 1$ to columns do
5:         for $k = 1$ to rows do
6:             $\mathbf{T}_{j,k} = \begin{cases} 0, & \text{if } \mathbf{I}_{i_{j,k}} < t \\ 1, & \text{otherwise} \end{cases}$
7:             $\mathbf{v} = getContext(\mathbf{T}_{j,k}, \mathbf{c})$
8:             for $n = 0$ to 9 do
9:                 $P_n(\mathbf{v}|\mathbf{T}_{j,k}) = \frac{\mathscr{H}_n(\mathbf{v}|\mathbf{T}_{j,k})+\alpha}{\mathscr{H}_n(\mathbf{T}_{j,k})+\alpha|\mathcal{A}|}$
10:                $\mathbf{b}_n = \mathbf{b}_n + \frac{1}{\log_2 P_n(\mathbf{v}|\mathbf{T}_{j,k})}$
11:            end for
12:        end for
13:    end for
14:    $\mathbf{g}_i = \min\left\{\frac{\mathbf{b}_n}{|\mathbf{I}_i|}\right\}$
15: end for
```

Table 4.3: The Proposed Finite Context Model Encoder.

Analogous to the training module, the input images in the FCM encoder go through a threshold function during the pixels processing. The value of each pixel is read in raster scan and its context is then computed based on the input template context. With the input context template and the current processing pixel, that in this stage belongs to the $\mathcal{A} = \{0, 1\}$, the function *getContext* retrieves the information context. Then the returned data from the function is searched in each digit models $\mathscr{H}_n$ as the number of times that the context occurred on each digit of the training set.

The counters $\mathscr{H}_n(v|\mathbf{T}_{j,k})$ and $\mathscr{H}_n(\mathbf{T}_{j,k})$ are used to compute the estimate probability $P_n(v|\mathbf{T}_{j,k})$ for each digit model. This estimate is useful to retrieve the Shannon self-information associated to the $v|\mathbf{T}_{j,k}$ event. The **b** vector accumulates the self-information for every pixel on the digit using the models conceived in the training module. To compute the compress size of an image from the MNIST test set, the self-information for every pixel in the image is accumulated in the **b** vector. At the end of processing the image, the vector **b** retains ten values corresponding to the compressed size for each of the ten models.

The digit classification is based on the NRC measure and in order to accomplish the measure, each value from the vector **b** is divided by the image size $|\mathbf{I}_i|$. However, the $|\mathbf{I}_i|$ term is irrelevant to the measure since all the images in the database have the same dimensions, hence the image size contribution can be discarded.

The model that uses less bits to represent the image is going to give the classification to the image. This means, that the n index of the vector that has the $\min\{\text{NRC}\} = \min\{\mathbf{b}_n/|\mathbf{I}_i|\} = \min\{\mathbf{b}_n\}$ will classify the digit image. In the exclusive case of appearing two minimum values in the b vector, then the lowest index is attributed as classification to the image.

4.3 Performance Results

In this section, the performance of the proposed digit classifying structure is evaluated. The important factors that influence the accuracy of the proposed method are the context **c**, the threshold constant t and the value of α. To assess the influence of the size context and the structure of the context on the accuracy results, three context templates were studied. The three templates can be divided in horizontal, vertical and zig zag. The difference between them is in the grow, *i.e.*, when the size of the context is increased, the context template is different.

x_{n-48}	x_{n-47}	x_{n-46}	x_{n-45}	x_{n-44}	x_{n-43}	x_{n-42}
x_{n-41}	x_{n-40}	x_{n-39}	x_{n-38}	x_{n-37}	x_{n-36}	x_{n-35}
x_{n-34}	x_{n-33}	x_{n-32}	x_{n-31}	x_{n-30}	x_{n-29}	x_{n-28}
x_{n-27}	x_{n-26}	x_{n-25}	x_{n-24}	x_{n-23}	x_{n-22}	x_{n-21}
x_{n-20}	x_{n-19}	x_{n-18}	x_{n-17}	x_{n-16}	x_{n-15}	x_{n-14}
x_{n-13}	x_{n-12}	x_{n-11}	x_{n-10}	x_{n-9}	x_{n-8}	x_{n-7}
x_{n-6}	x_{n-5}	x_{n-4}	x_{n-3}	x_{n-2}	x_{n-1}	s

x_{n-48}	x_{n-41}	x_{n-34}	x_{n-27}	x_{n-20}	x_{n-13}	x_{n-6}
x_{n-47}	x_{n-40}	x_{n-33}	x_{n-26}	x_{n-19}	x_{n-12}	x_{n-5}
x_{n-46}	x_{n-39}	x_{n-32}	x_{n-25}	x_{n-18}	x_{n-11}	x_{n-4}
x_{n-45}	x_{n-38}	x_{n-31}	x_{n-24}	x_{n-17}	x_{n-10}	x_{n-3}
x_{n-44}	x_{n-37}	x_{n-30}	x_{n-23}	x_{n-16}	x_{n-9}	x_{n-2}
x_{n-43}	x_{n-36}	x_{n-29}	x_{n-22}	x_{n-15}	x_{n-8}	x_{n-1}
x_{n-42}	x_{n-35}	x_{n-28}	x_{n-21}	x_{n-14}	x_{n-7}	s

x_{n-48}	x_{n-47}	x_{n-43}	x_{n-42}	x_{n-34}	x_{n-33}	x_{n-21}
x_{n-46}	x_{n-44}	x_{n-41}	x_{n-35}	x_{n-32}	x_{n-22}	x_{n-20}
x_{n-45}	x_{n-40}	x_{n-36}	x_{n-31}	x_{n-23}	x_{n-19}	x_{n-10}
x_{n-39}	x_{n-37}	x_{n-30}	x_{n-24}	x_{n-18}	x_{n-11}	x_{n-9}
x_{n-38}	x_{n-29}	x_{n-25}	x_{n-17}	x_{n-12}	x_{n-8}	x_{n-3}
x_{n-28}	x_{n-26}	x_{n-16}	x_{n-13}	x_{n-7}	x_{n-4}	x_{n-2}
x_{n-27}	x_{n-15}	x_{n-14}	x_{n-6}	x_{n-5}	x_{n-1}	s

Figure 4.6: Horizontal (left), Vertical (middle), Zig Zag (right) Contexts.

The horizontal context grows in horizontal lines that go from right to left. The vertical also grows in a line manner but in this case in vertical lines that go from bottom to top. The zig zag context has a similarity with the JPEG zig zag encode method, yet the context grows in a backwards manner from the JPEG zig zag method. The performance results obtained with the three contexts have the threshold constant t and the value of α, fixed as $t = 128$ and $\alpha = 1$.

The horizontal context results are presented in Figure 4.7. The context was tested on the original MNIST database as the subsampling 16×16, 20×20, 24×24 versions. The subsampled versions were obtained using the NNS algorithm.

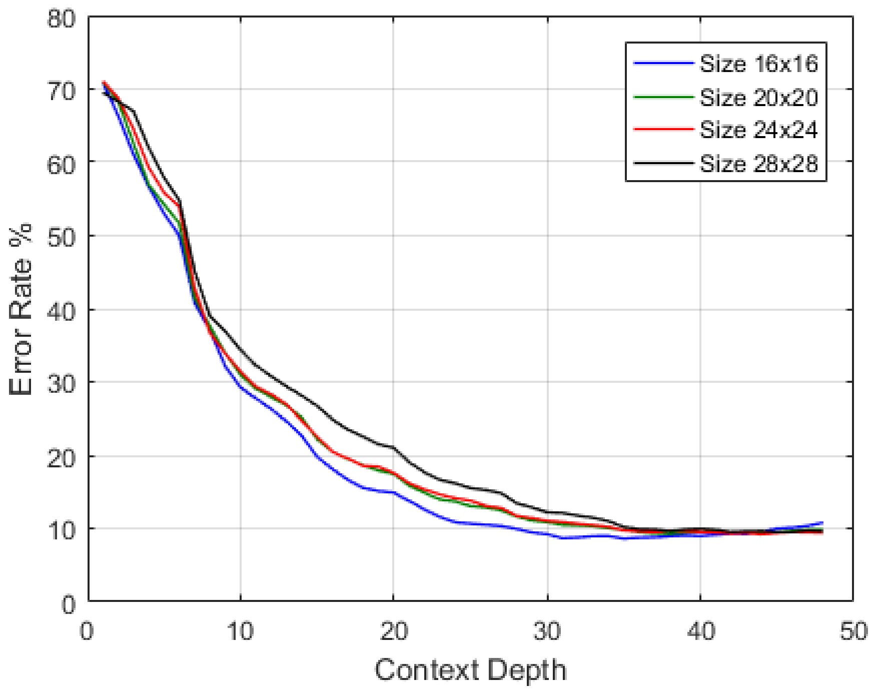


Figure 4.7: Horizontal Context Results on the MNIST Database.

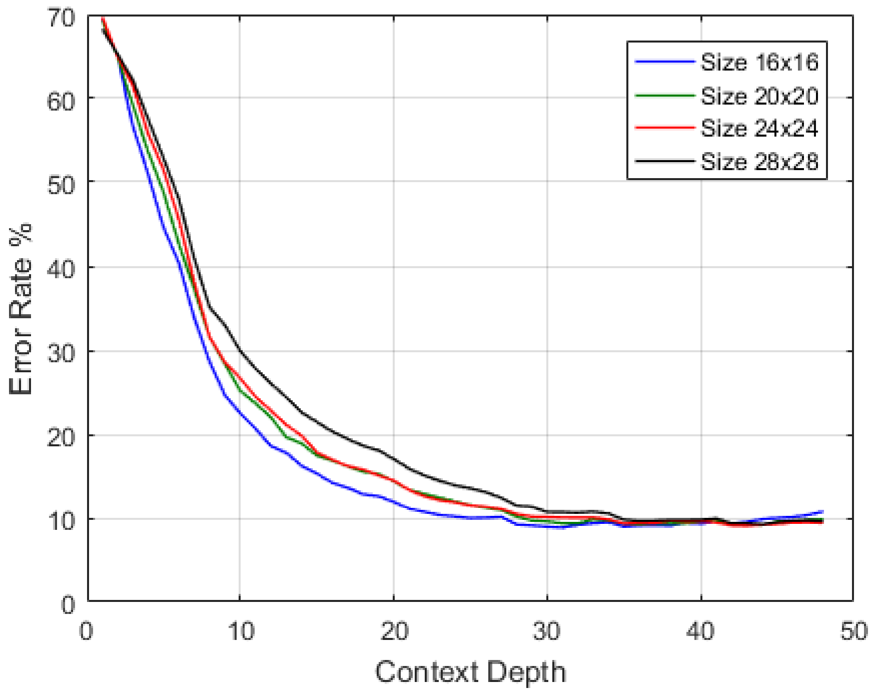


Figure 4.8: Vertical Context Results on the MNIST Database.

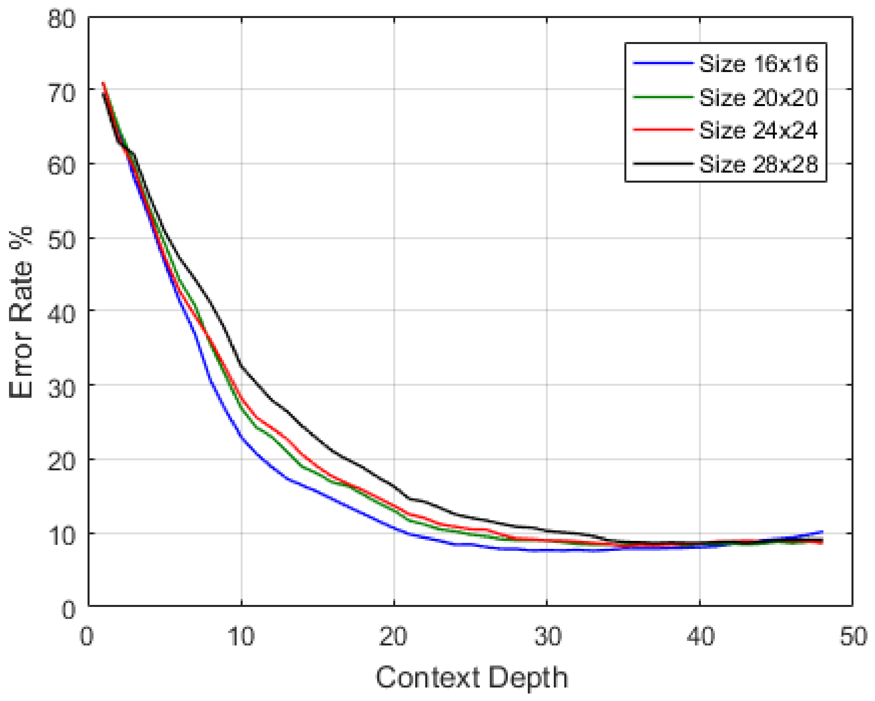


Figure 4.9: Zig Zag Context Results on the MNIST Database.

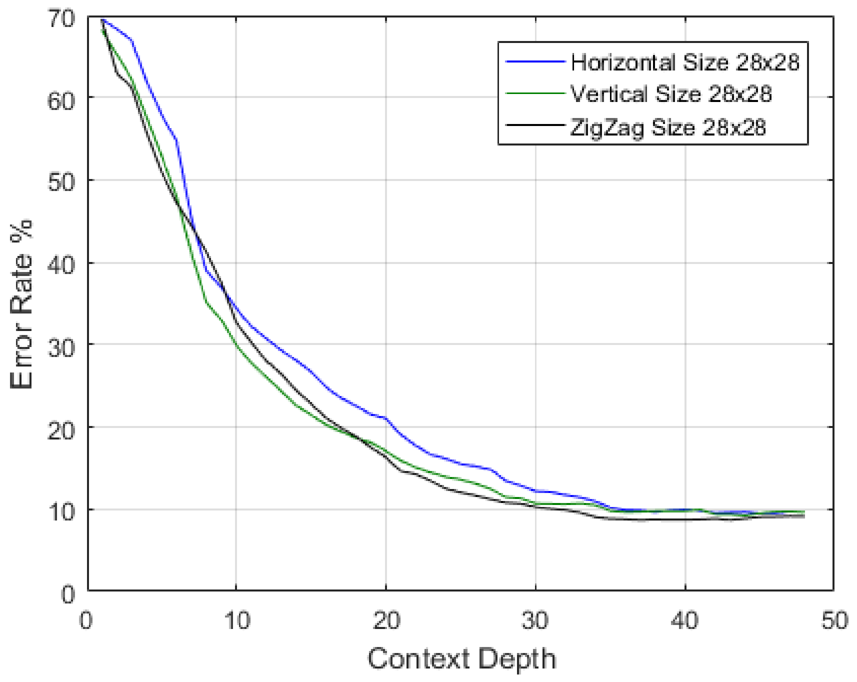


Figure 4.10: Horizontal vs. Vertical vs. Zig Zag Contexts on the Original MNIST Set.

The results on the MNIST database for the original dimensions and the decreased dimensions are depicted in Figure 4.7 for the horizontal context, Figure 4.8 for the vertical context and in Figure 4.9 for the zig zag context. For the three cases, independently of the size of the images for the context depth of 1, the value of the error rate is around 70%. As the context depth increases the error rate decreases. For all cases, when the depth reaches the value of 35, the error rate stays below the 15%.

The choice of the context template is an important fragment in the classification process. Looking to the same context depth for different context examples used to measure the system performance, the results are different. For the MNIST 28 × 28 thresholded images, Figure 4.10 demonstrates the context importance, showing that for the same context depth the error rate is different. The zig zag and the vertical show approximated results while the horizontal presents worse results.

Another pre-processing method used in the MNIST dataset is the deskewing procedure. This method is used to assess if the FCM mechanism is influenced by the skew in the digits. Again, the three templates were applied to the four version of the MNIST set, now deskewed with the deskwing process described earlier. The results on the horizontal context are presented in Figure 4.11, the vertical results in Figure 4.12 and the zig zag in the Figure 4.13. In the deskwed digit images, the error rate easily reached a value below 10% for a context depth bigger than 30.

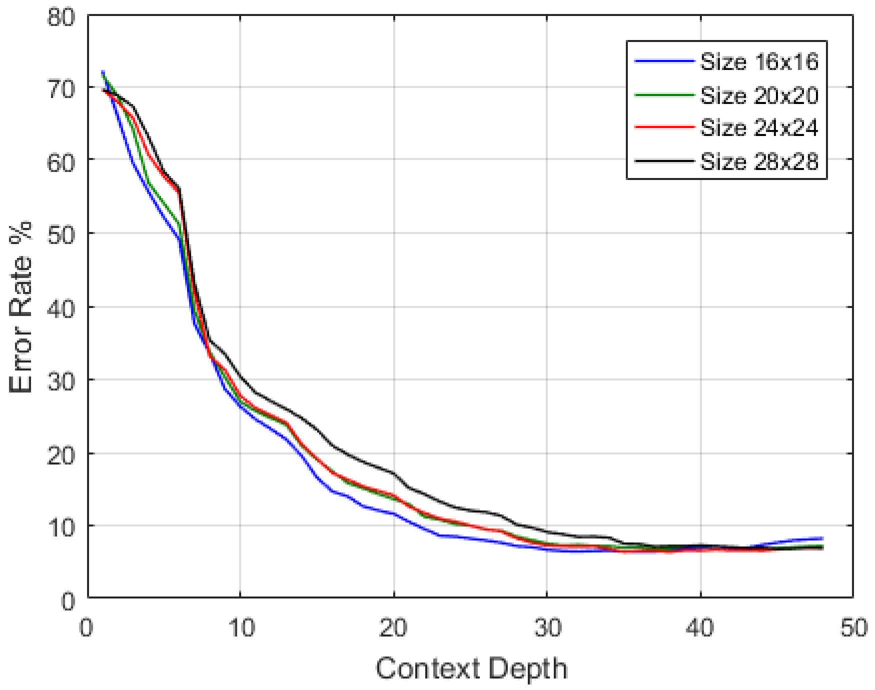


Figure 4.11: Horizontal Context Results on the MNIST Deskwed Database.

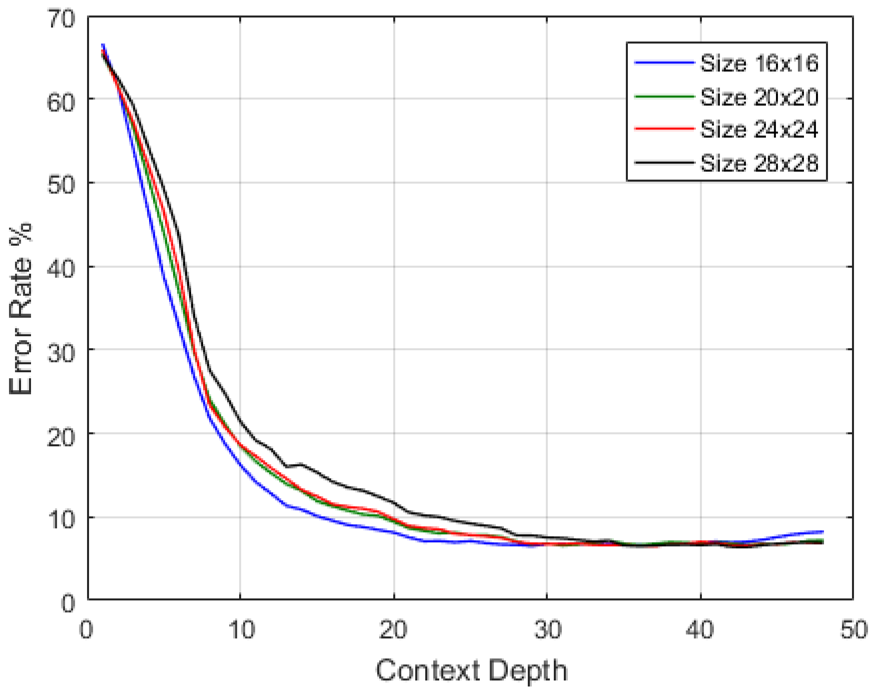


Figure 4.12: Vertical Context Results on the MNIST Deskwed Database.

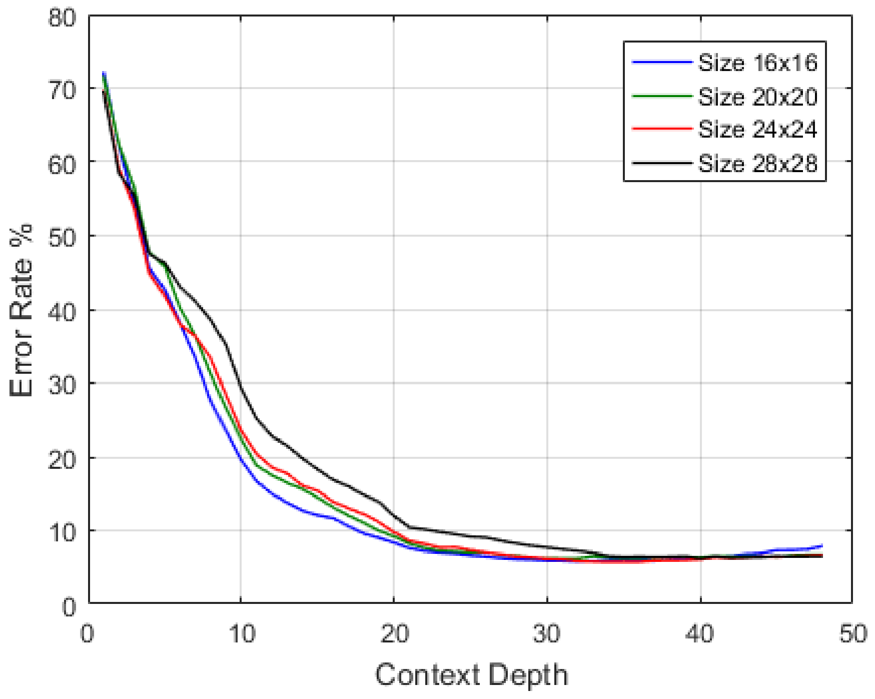


Figure 4.13: Zig Zag Context Results on the MNIST Deskwed Database.

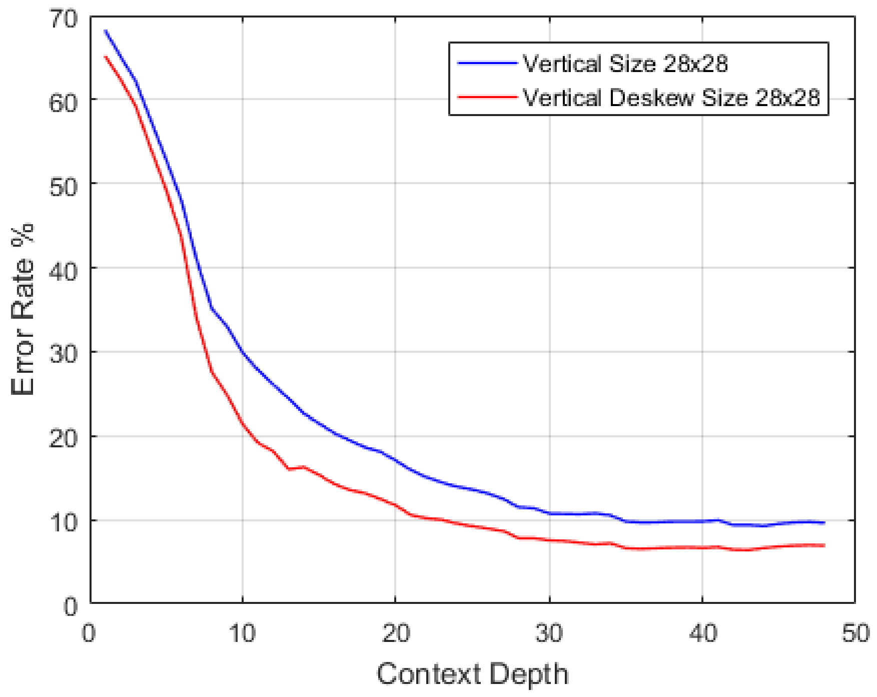


Figure 4.14: Vertical Context Applied to the Original and Deskewed MNIST Dataset.

Deskewing the MNIST dataset improves the results for the contexts, which is shown in Figure 4.14, where is possible to observe that the error rate is smaller for the deskewed MNIST than for the original MNIST database. Generally, the presented FCM approach can achieve better results if the digit images are pre-deskewed before the image introduction in the FCM architecture.

The other parameters that influence the accuracy results are the α and the threshold constant t. From experience, the α values that yielded better results were in $[0, 2]$ interval. To give some insight in the behavior of the α in the error rate, Figure 4.15 demonstrates the evolution of the error rate in the downscaled 16×16 MNIST for a zig zag context of depth 33 with a threshold constant $t = 128$. The results show that the best results are achieved in the $[1, 3]$ range, introducing the notion that in this case values smaller than 1/2 do not decrease the error rate, as presented in some works using FCMs [115].

The last parameter that has an impact on the error rate is the threshold constant t. Figure 4.16 shows the error rate evolution for the downscaled 16×16 MNIST for a zig zag context of depth 33 with an $\alpha = 1$. The results show that for a value smaller than $\alpha < 64$, the error rate has a slower growth. On the other hand, to higher values, the growth of the error rate is more accentuated. The influence of the threshold constant is dependent on the size of the digit image, and the variations of the error rate may be more accentuated depending on the context template.

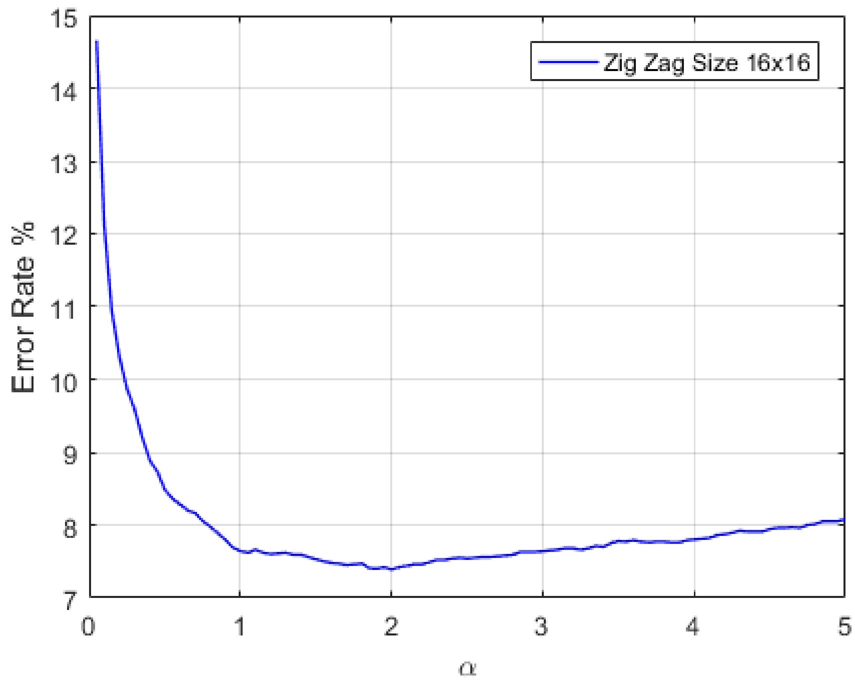


Figure 4.15: The α Influence on the Accuracy on the 16×16 MNIST Database.

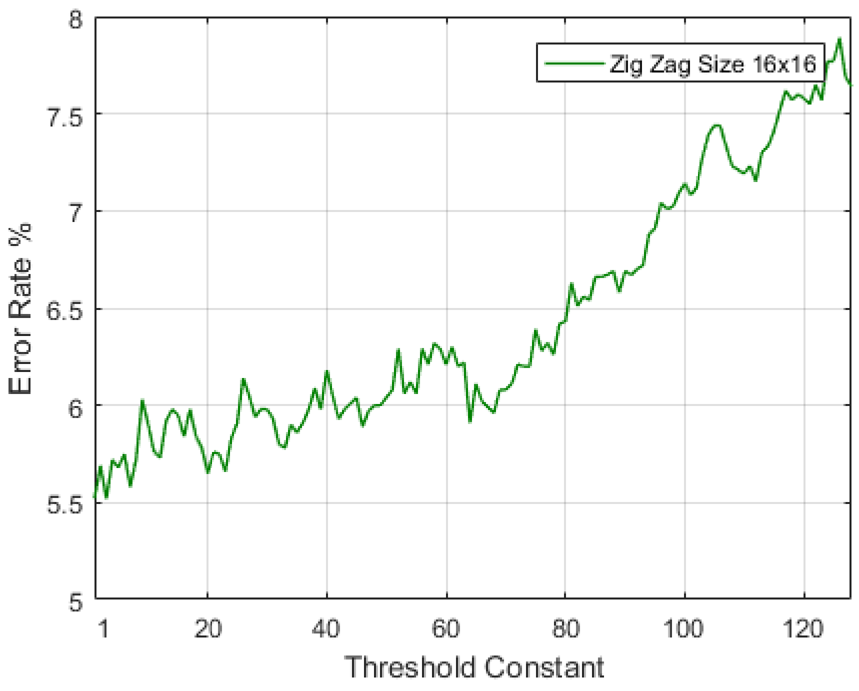


Figure 4.16: The t Constant Influence on the Accuracy on the 16×16 MNIST Database.

After introducing the characteristics of the proposed architecture and the effects of the input parameters, the final proposed method is exhibited and the error rate accomplished is revealed. The proposed method has as input parameters, a threshold constant $t = 49$, $\alpha = 0.5$ and a variation of the zig zag context. The context used has the particularity of having the symbol to encode included in the context. The template is depicted in Figure 4.17.

x_{n-33}	x_{n-31}	x_{n-30}	x_{n-29}		x_{n-17}	x_{n-16}
x_{n-32}		x_{n-27}	x_{n-28}	x_{n-18}		x_{n-15}
	x_{n-26}		x_{n-20}	x_{n-19}	x_{n-14}	
x_{n-25}		x_{n-21}		x_{n-13}		x_{n-5}
x_{n-24}	x_{n-22}		x_{n-12}		x_{n-6}	x_{n-4}
x_{n-23}		x_{n-11}		x_{n-7}		x_{n-3}
	x_{n-10}	x_{n-9}	x_{n-8}		x_{n-2}	x_{n-1} s

Figure 4.17: Proposed Context.

The MNIST training and test input images are pre-deskwed and subsampled to 16×16 digit images, before the training and encoding module. The performance of the proposed method on the MNIST test set can be observed in the confusion matrix in Table 4.4. The digit that has the best classification is the digit 0, and the one with the worst classification results is the digit 9. The digit order of best to worse classification is: $0, 2, 4, 1, 3, 5, 6, 8, 7, 9$. The results can be verified in Table 4.5.

The proposed architecture displays the major misclassification for the digit 7, where 18 digit images were misclassified as 2. The following digit to obtain a middling misclassification is the digit 9, where the method imprecisely classified twelve 9's as the digit 8. The less confusing digit was digit 1, for only one digit 6 and two digits 9 were misinterpreted as digit 1. The digit to be confused more times was the digit 2, confused 50 times as other digits.

Digit	0	1	2	3	4	5	6	7	8	9
0	**973**	0	2	0	1	1	2	1	0	0
1	2	**1111**	3	2	8	0	4	3	2	0
2	1	0	**1012**	4	1	1	1	8	4	0
3	0	0	10	**984**	0	8	0	3	5	0
4	1	0	2	0	**962**	0	2	2	5	8
5	2	0	3	10	0	**869**	2	1	4	1
6	7	1	1	0	4	11	**933**	0	1	0
7	1	0	18	4	3	1	0	**993**	3	5
8	6	0	3	4	5	10	1	2	**942**	1
9	5	2	9	5	8	8	0	6	12	**954**

Table 4.4: Confusion Matrix for the MNIST Database.

Digit	0	1	2	3	4	5	6	7	8	9
Error Rate %	0.714	2.115	1.938	2.574	2.037	2.578	2.609	3.405	3.285	5.451

Table 4.5: Individual Results for the MNIST Test Set.

The error rate of the overall proposed method is 2.67%. However, if the input images are solely downscaled and not deskewed, the error rate achieved is higher, 3.80%. If the proposed method is applied to the MNIST without pre-processing techniques the result obtained is 9.17%. Table 4.6 depicts a comparison of the proposed method with the most simple methods in the seven different MNIST algorithm categories introduced earlier. It is possible to observe that the proposed method can achieve better results that a simple linear classifier, a boosted stump and a simple 2-layer NN with 300 hidden units.

Scheme	Pre-processing	Overall Error Rate (%)
Linear Classifier (1-layer NN)	none	12.0
Proposed Method	**none**	**9.17**
Boosted Stumps	none	7.70
Proposed Method	**Deskewing**	**5.66**
2-layer NN, 300 HU, MSE	none	4.70
Proposed Method	**Subsampling to 16×16 pixels**	**3.80**
1000 RBF + Linear Classifier	none	3.60
k-NN, Euclidean L2	none	3.09
Proposed Method	**Subsampling to 16×16 pixels, deskewing**	**2.67**
CNN LeNet-1	Subsampling to 16×16 pixels	1.70
SVM, Gaussian Kernel	none	1.40
Human [32]	–	0.20

Table 4.6: Comparison of the Most Simple Methods with the Proposed Method.

With the use of deskwed digit images, the error rate decreases and the method achieves a better result than a Non-Linear Classifier 1000 RBF and a k-NN using the euclidean distance between 2 neighbors. However, this novel classification method with the presented configuration does not achieve error rates close to a CNN such as the LeNet-1, or a SVM algorithm. The new method introduced in this dissertation demonstrates that can easily reach an error rate below the 10%.

The two modules from the proposed method that achieve an error rate of 2.67%, take around 12.5s to train and 14.5s to encode the MNIST training and test sets, respectively. On the classification of one deskwed 16×16 digit image, the encoder module takes a mean time of 1.45ms to give a label to a single image. It should be noted that the results were obtained using a single processor core. The proposed method upon build each digit model in the training module, allocates 212MB of RAM memory to load the ten models.

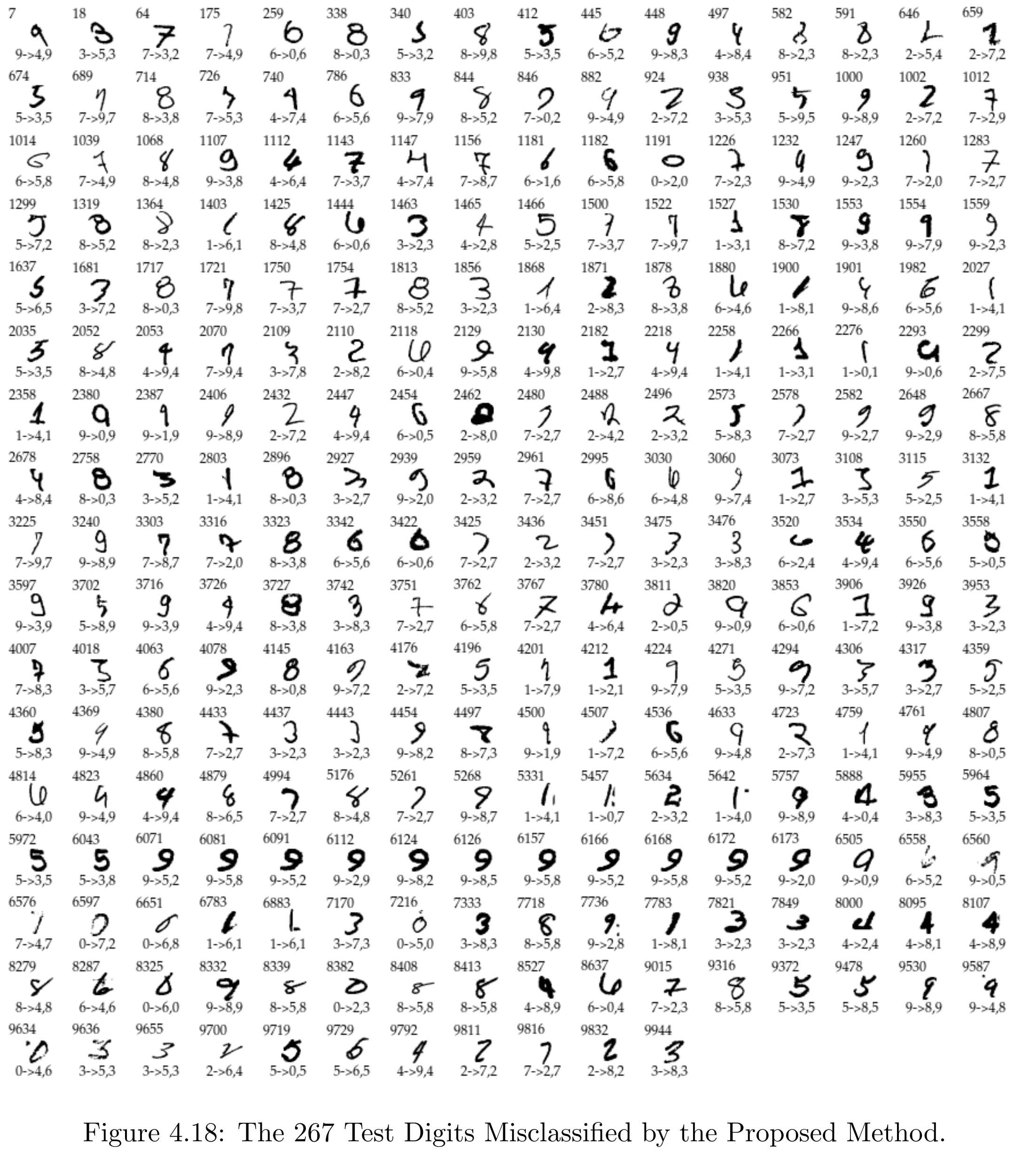


Figure 4.18: The 267 Test Digits Misclassified by the Proposed Method.

The Figure 4.18 shows the 267 misclassified digits, where the index of the digit is represented in the left top of the digit. Below to left of the digit image, the correct answer is displayed as well the answer of the proposed method. The second most probable answer is also presented. To a better understanding, the first digit image has index 7, the label of the image is 9 and the proposed method classified it as a 4, although the second most probable answer is the digit 9. If the second most probable answer was the given label for this misclassified set, the results would have been 161 well classified and 106 misclassified.

All the experiments were performed using C++11 on an Intel® Core™ i3-2310M CPU @ 2.10GHz $\times$ 4, with 4GB of RAM.

Chapter 5

Conclusion and Future Work

FCM scheme variations had been successfully used in some ML problems, such as Authorship Attribution [115] [125], Biometric Identification Systems [116] and Facial Recognition [126]. The work done in this dissertation attempts to initialize the research on compression based techniques to give a solution to one of the most popular PR problem, handwritten digit recognition. This chapter gives an end to this dissertation, providing an openly overlook on the work developed.

5.1 Conclusion

This dissertation started by presenting an overview on the PR field, particularly on the handwritten digit recognition theme. The dataset used in this dissertation is the MNIST dataset, so its main classification categories were presented and analyzed. The objective of this dissertation was to developed a handwritten digit recognition system based on compression techniques and therefore the contemporary data compression techniques were analyzed and revealed.

The proposed method uses a compression based classifier, where through the use of data compressors it is possible to make an approximation of the uncomputable Kolmogorov Complexity and get the information shared between two objects. The proposed classification method was developed on the Markov models foundation, in special the FCM scheme. The classification method uses the FCM procedure to model the training data, endowing the contribution to compute a similarity measure between the digit classes. The best results obtained with a causal context were 3.80% for the dataset subsampled to images of 16×16 dimensions. With the equivalent input parameters, but using the digit images deskewed, the proposed method obtained the best error rate, 2.67%. The conclusion collected from the work made around the proposed method was that it does not need a feature extraction phase. The architecture is fairly straightforward and with low complexity. The parameters used on this scheme are reduced, as it only needs three parameters, the threshold constant, the α value used on the estimator and the structure of the context used to model the data. In contrast to the ML techniques applied to the MNIST database, the proposed method is comparably simple and easy to implement. The proposed method, with a context depth of 40 or more, can easily achieve an error rate of 10% or less. The time needed for the proposed method to assign a classification to a digit image is on the same order as more complex machine learning techniques. The work done in this dissertation demonstrated that a classification based compression method can achieve good and robust results on the MNIST dataset.

5.2 Future Work

Towards the end of this dissertation, new ideas have emerged in order to complement and expand this work. The following suggestions are presented for future research:

- Instead of using a thresholding function to reduce the alphabet size, another approach that would present interesting results is the use of quantization techniques.
- As presented in this dissertation, the NNS algorithm is the simplest scaling method and for this in some cases the subsampled image obtained does not display a good perception. The error rate could improve using more complex scaling algorithms.
- The contexts used were all causal, therefore a stimulating point of investigation would be to use non-causal contexts.
- Another compelling factor would be to use the proposed method on other handwriting digits databases.
- The use of distortions such as affine or elastic distortions are another interesting point of research.
- The method proposed is a process that is easily parallelizable, thus using a multicore programming core approach could increase the overall speed of the process.

Bibliography

[1] G. S. Sebestyen, *Decision Processes in Pattern Recognition*, Macmillan, New York, 1962.

[2] N. J. Nilsson, *Learning Machines-Foundations of Trainable Pattern-Classifying Systems*, McGraw-Hill, New York, 1965.

[3] Daniel L. Schacter, Daniel T. Gilbert, Daniel M. Wegner, *Psychology (2nd Edition)*, Worth Publishers, New York, 2011.

[4] Jie Liu, Jigui Sun, Shengsheng Wang, "Pattern Recognition: An overview, *IJCSNS International Journal of Computer Science and Network Security*, vol. 6, no. 6, 2006.

[5] Deep Learning vs Machine Learning vs Pattern Recognition [Available online at] `http://www.computervisionblog.com/2015/03/deep-learning-vs-machine-learning-vs.html` [Accessed] 21/08/2017.

[6] Berkant Savas, *Master Thesis: Analyses and Tests of Handwritten Digit Recognition Algorithms*, Linköping University, Linköping, 2003.

[7] E. Vonk, L.P.J. Veelenturf, L.C. Jain, "Neural Networks: Implementations and Applications", *IEEE Aerospace and Electronic Systems Magazine*, no. 7, pp. 11–16, 1996.

[8] P. Smagt, B. Kröse, *An Introduction to Neural Networks (8th Edition)*, The University of Amsterdam, Amsterdam, 1996.

[9] Geoffrey E. Hinton, "How neural networks learn from experience", *Scientific American*, vol. 267, no. 3, pp.144–151, 1992.

[10] Larry R. Squire, *Memory and brain*, Oxford University Press, New York, 1987.

[11] Yousif R. Asfour, Gail A. Carpenter, Stephen Grossberg, Gregory Lesher, "Fusion ARTMAP: A neural network architecture for multichannel data fusion and classification", *Proceedings of the World Congress on Neural Networks, Portland*, vol. 2, pp. 210–215, July 1993.

[12] Vinícius Gonçalves Maltarollo, Káthia Maria Honório, Albérico Borges Ferreira da Silva, "Applications of artificial neural networks in chemical problems", *Artificial neural networks-architectures and applications*, pp. 203–223, INTECH, 2013.

[13] Daniel J. Sargent, "Comparison of artificial neural networks with other statistical approaches: results from medical data sets", *Cancer*, vol. 91, no. 8, pp. 1636–1642, 2001.

[14] Jack V. Tu, “Advantages and Disadvantages of Using Artificial Neural Networks versus Logistic Regression for Predicting Medical Outcomes”, *Journal of Clinical Epidemiology*, vol. 49, pp. 1225–1231, 1996.

[15] Patricia Jimeno-Sáez, Javier Senent-Aparicio, Julio Pérez-Sánchez, David Pulido-velazquez, José Cecilia, “Estimation of Instantaneous Peak Flow Using Machine-Learning Models and Empirical Formula in Peninsular Spain”, *Water*, vol. 9, pp. 347, 2017.

[16] Gunther Heidemann, Helge Ritter, “Data compression - a generic principle of pattern recognition?”, *Computer Vision and Computer Graphics - Theory and Application*, pp. 202–212, 2009.

[17] Ida Mengyi Pu, *Fundamental Data Compression*, Elsevier, Britain, 2006.

[18] Eric T. Carlson, Russell J. Rasquinha, Kechen Zhang, Charles E. Connor, “A Sparse Object Coding Scheme in Area V4”, *Current Biology*, vol. 21, no. 4, pp. 288–293, 2011.

[19] S. L. Chang, L. S. Chen, Y. C. Chung, S. W. Chen, “Automatic license plate recognition”, *IEEE Transactions on Intelligent Transportation Systems*, vol. 5, no. 1, pp. 42–53, 2004.

[20] G. Donald Allen, *The Origins of Mathematics*, Texas A&M University, Texas, 2014.

[21] Igor Ushakov, *In the Beginning Was the Number*, Lulu, San Diego, 2012.

[22] Will Durant, *Our Oriental Heritage: The Story of Civilization*, Simon and Schuster, New York, 2011.

[23] Thomas Hockey, Virginia Trimble, Thomas R. Williams, Katherine Bracher, Richard Jarrell, Jordan D. Marché, F. Jamil Ragep, *Biographical Encyclopedia of Astronomers*, Springer, New York, 2007.

[24] David Eugene Smith, Louis Charles Karpinski, *The Hindu-Arabic Numerals*, Dover Publications, New York, 2004.

[25] Elizabeth L. Eisenstein, *The Printing Press as an Agent of Change.* , Cambridge University Press, Cambridge, 1890.

[26] Georges Ifrah, *The Universal History of Numbers: From Prehistory to the Invention of the Computer*, Harville Press, London, 1998.

[27] Pieter Hendrik van Cittert, *Astrolabes: A Critical Description of the Astrolabes, Noctilabes, and Quadrants in the Care of the Utrecht University Museum*, E. J. Brill, Leiden, 1954.

[28] Numerical digit [Available online at] `https://en.wikipedia.org/wiki/Numerical_digit` [Accessed] 5/08/2017.

[29] The MNIST Database of Handwritten Digits [Available online at] `http://yann.lecun.com/exdb/mnist/` [Accessed] 7/08/2017.

[30] Li Deng, “The MNIST Database of Handwritten Digit Images for Machine Learning Research, *Signal Processing Magazine*, vol. 29, no. 9, pp. 141–142, 2012.

[31] P. Simard, Y. LeCun, J. Denker, B. Victorri, "Transformation invariance in pattern recognition - tangent distance and tangent propagation", *International Journal of Imaging System and Technology*, vol. 11, no. 2, pp. 181–194, 2001.

[32] Yann LeCun, L. D. Jackel, Léon Bottou, Corinna Cortes, John S. Denker, Harris Drucker, Isabelle Guyon, Urs A. Müller, Eduard Säckinger, Patrice Simard, Vladimir Vapnik, "Learning Algorithms For Classification: A Comparison On Handwritten Digit Recognition", *Neural Networks: The Statistical Mechanics Perspective*, pp. 261–279, 1995.

[33] Réjean Plamondon, Sargur N. Srihari, "On-line and off-line handwriting recognition: a comprehensive survey", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 22, pp. 63–84, 2000.

[34] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, L. D. Jackel, "Handwritten digit recognition with a back-propagation network", *Advances in Neural Information Processing Systems*, vol. 2, pp. 396–404, 1990.

[35] Neural Networks in the Wild: Handwriting Recognition [Available online at] `https://pt.slideshare.net/guard0g/neural-networks-in-the-wild-handwriting-recognition` [Accessed] 10/09/2017.

[36] Charles C. Tappert, Ching Y. Suen, Toru Wakahara, "The State of Art in On-Line Handwriting Recognition ", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 12, no. 8, pp. 787–808, 1990.

[37] Arti Shivram, Bilan Zhu, Srirangaraj Setlur, "Segmentation Based Online Word Recognition: A Conditional Random Field Driven Beam Search Strategy", *Proceedings of 12th International Conference on Document Analysis and Recognition (ICDAR)*, pp. 852–856, 2013.

[38] Alex Graves, Jürgen Schmidhuber. "Offline handwriting recognition with multidimensional recurrent neural networks" *Advances in Neural Information Processing Systems*, pp. 545–552, 2009.

[39] Stuart J. Russell, Peter Norvig, *Artificial Intelligence: A Modern Approach (3rd Edition)*, Prentice Hall Press, New Jersey, 2009.

[40] Claude Sammut, Geoffrey I. Webb, *Encyclopedia of Machine Learning*, Springer US, 2011.

[41] Guo-Xun Yuan, Chia-Hua Ho, Chih-Jen Lin, "Recent advances of large-scale linear classification", *Proceedings of the IEEE*, vol. 100, no. 9, pp. 2584–2603, 2012.

[42] Richard O. Duda, Peter E. Hart, David G. Stork, *Pattern Classification (2nd Edition)*, Wiley-Interscience, 2000.

[43] Klaus-Robert Müller, Charles W. Anderson, Gary E. Birch, "Linear and Non-linear Methods for BrainComputer Interfaces", *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, vol. 11, no. 2, pp. 165–169, 2003.

[44] Hervé Abdi, Lynne J. Williams, "Principal component analysis", *Wiley Interdisciplinary Reviews: Computational Statistics*, vol. 2, no. 4, pp. 433–459, 2010.

[45] Martin D. Buhmann, “Radial basis functions, *Acta numerica*, vol. 9, pp. 1–38, 2000.

[46] Thomas Cover, Peter Hart, “Nearest neighbor pattern classification” *IEEE Transactions on Information Theory*, vol. 13, no. 1, pp. 21–27, 1967.

[47] Cheng-Lin Liu, Kazuki Nakashima, Hiroshi Sako, Hiromichi Fujisawa, “Handwritten digit recognition: benchmarking of state-of-the-art techniques”, *Pattern Recognition*, vol. 36, no. 10, pp. 2271–2285, 2003.

[48] Evelyn Fix, J.L. Hodges, “Discriminatory analysis, nonparametric discrimination: Consistency properties (Technical Report 4)”, *USAF School of Aviation Medicine*, Randolph Field, Texas, 1951.

[49] David W. Aha, Dennis Kibler, Marc K. Albert, “Instance-based learning algorithms.” *Machine Learning*, vol. 6, no. 1, pp. 37–66, 1991.

[50] Dietrich Wettschereck, David W. Aha, Takao Mohri, “A review and empirical evaluation of feature weighting methods for a class of lazy learning algorithms”, *Lazy Learning Springer Netherlands*, pp. 273–314, 1997.

[51] F. J. Cortijo, N. Pérez De La Blanca, R. Molina, J. Abad, “On the Combination of Nonparametric Nearest Neighbor Classification and Contextual Correction”, *Pattern Recognition and Image Analysis Proccedings of the VI Spanish Symposium on Pattern Recognition and Image Analysis*, pp. 503–510, Córdoba, Spain, 1995.

[52] Wayne Iba, Pat Langley, “Induction of one-level decision trees”, *Proceedings of the Ninth International Conference on Machine Learning*, pp. 233–240, 1992.

[53] Kannan Umadevi Venkataraju, *Master Thesis: Automatic Markup of Neural Cell Membranes using Boosted Decision Stumps*, The University of Utah, Utah, 2009.

[54] Balázs Kégl, Róbert Busa-Fekete, “Boosting products of base classifiers” *Proceedings of the 26th Annual International Conference on Machine Learning*, vol. 26, pp. 497–504, Montreal, Canada, 2009.

[55] Yoav Freund, Robert E. Schapire, “Experiments with a new boosting algorithm.”, *Machine Learning: Proceedings of the Thirteenth International Conference*, vol. 96, pp. 148–156, 1996.

[56] Yoav Freund, Robert E. Schapire, “A Short Introduction to Boosting”, *Journal of Japanese Society For Artificial Intelligence*, vol. 14, no. 5, pp. 771–780, 1999.

[57] Yann LeCun, Léon Bottou, Yoshua Bengio, Patrick Haffner, “Gradient-based Learning Applied to Document Recognition”, *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[58] Dennis Decoste, Bernhard Schölkopf, “Training invariant support vector machines”, *Machine Learning*, vol. 46, no. 1, pp. 161–190, 2002.

[59] Bernhard E. Boser, Isabelle M. Guyon, Vladimir N. Vapnik, “A training algorithm for optimal margin classifiers”, *Proceedings of the Fifth Annual Workshop on Computational Learning Theory*, pp. 144–152, 1992.

[60] W. Mccuuoch, W. Pitts, “A logical calculus of the ideas immanent in nervous activity”, *Bulletin of Mathematical Biology*, vol. 52, pp. 99115, 1990.

[61] Ján Vojt, *Master Thesis: Deep neural networks and their implementation*, Charles University in Prague, Prague, 2016.

[62] F. Rosenblatt, “The perceptron: A probabilistic model for information storage and organization in the brain”, *Psychological Review*, vol. 65, pp. 386–408, 1958.

[63] K. Hornik, M. Stinchcombe, H. White, “Multilayer feedforward networks are universal approximators”, *Neural Networks*, vol. 2, no. 5, pp. 359–366, 1989.

[64] Matej Uličný, *Master Thesis: Methods for Increasing Robustness of Deep Convolutional Neural Networks*, Halmstad University, Halmstad, 2015.

[65] Yen-Ming Chiang, Li-Chiu Chang, Fi-John Chang, “Comparison of static-feedforward and dynamic-feedback neural networks for rainfall-runoff modeling”, *Journal of Hydrology*, vol. 290, no. 3, pp. 297–311, 2004.

[66] D. H. Hubel, T. N. Wiesel, “Receptive fields, binocular interaction and functional architecture in the cat’s visual cortex”, *Journal of Physiology*, vol. 160, no. 1, pp. 106–154, 1962.

[67] Christopher M. Bishop, *Pattern Recognition and Machine Learning*, Springer, Singapore, 2006.

[68] Frank Y. Shih, *Image Processing and Mathematical Morphology: Fundamentals and Applications*, CRC Press, 2009.

[69] Karen Simonyan, Andrew Zisserman,“Very deep convolutional networks for large-scale image recognition”, *Proc. International Conference on Learning Representations*, pp. 1409–1556, 2014.

[70] Xavier Glorot, Antoine Bordes, Yoshua Bengio, “Deep sparse rectifier neural networks”, *In Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, pp. 315–323, 2011.

[71] Alex Krizhevsky, Ilya Sutskever, Geoffrey E. Hinton, “ImageNet Classification with Deep Convolutional Neural Networks”, *Advances in Neural Information Processing Systems*, pp. 1097–1105, 2012.

[72] Dominik Scherer, Andreas Müller, Sven Behnke, “Evaluation of Pooling Operations in Convolutional Architectures for Object Recognition”, *20th International Conference on Artificial Neural Networks*, pp. 92–101, 2010.

[73] Convolutional Neural Networks (CNNs / ConvNets) [Available online at] `http://cs231n.github.io/convolutional-networks/` [Accessed] 20/09/2017.

[74] Yann LeCun, Yoshua Bengio, “Convolutional networks for images, speech, and time series”, *The Handbook of Brain Theory and Neural Networks*, pp. 255–258, 1995.

[75] L. Wan, M. Zeiler, S. Zhang, Y. LeCun, R. Fergus, "Regularization of Neural Networks using DropConnect", *International Conference on Machine Learning*, pp. 1058–1066, 2013.

[76] Claude Elwood Shannon, "A Mathematical Theory of Communication", *The Bell System Technical Journal*, vol. 27, pp. 379–423 and 623–65, 1948.

[77] Andrei N. Kolmogorov, "Combinatorial foundations of information theory and the calculus of probabilities", *Russian Mathematical Surveys*, vol. 38, no. 4, pp. 29–40, 1983.

[78] Khalid Sayood, *Introduction to data compression, Third Edition*, Elsevier, San Francisco, 2006.

[79] Peter Grünwald, Paul M. B. Vitányi, "Shannon Information and Kolmogorov Complexity", *IEEE Trans. Information Theory*, 2004.

[80] Moses Charikar, Eric Lehman, Ding Liu, Risna Panigrahy, "Approximating the smallest grammar: Kolmogorov complexity in natural models", *Proceedings of the Thiry-fourth Annual ACM Symposium on Theory of Coputing*, pp. 792–801, 2002.

[81] A. Gammerman, V. Vovk, "Kolmogorov Complexity: Sources, Theory and Applications", *Computer Journal*, vol. 42, no. 4, pp. 252–255, 1999.

[82] Andrej A. Muchnik, "Conditional complexity and codes", *Theoretical Computer Science*, vol. 271, no. 1–2, pp. 97–109, 2002.

[83] Ming Li, Xin Chen, Xin Li, Bin Ma, Paul M.B. Vitányi, "The similarity metric", *IEEE Transactions on Information Theory*, vol. 50, no. 12, pp. 3250–3264, 2004.

[84] Keshi Chen, Tenkasi V. Ramabadran, "Near-lossless compression of medical images through entropy-coded DPCM", *IEEE Transactions on Medical Imaging*, vol. 13, no. 3, pp. 538–548, 1994.

[85] David Huffman, "A Method for the Construction of Minimum-Redundancy Codes", *Proceedings of the IRE*, vol. 40, no. 9, pp. 1098–1101, 1952.

[86] Norman Abramson, *Information Theory and Coding*, McGraw-Hill, New York, 1963.

[87] David J. C. MacKay, *Information Theory, Inference and Learning Algorithms*, Cambridge University Press, Cambridge, 2003.

[88] H. Witten, R. M. Neal, J. G. Cleary, "Arithmetic coding for data compression", *Communications of the ACM*, vol. 30, no. 6, pp. 520–540, 1987.

[89] D. Salomon, *Data compression - The complete reference. 4th Edition*, Springer, London, 2007.

[90] J. Ziv, A. Lempel, "A universal algorithm for sequential data compression", *IEEE Trans. on Information Theory*, vol. 23, pp. 337-343, 1977.

[91] J. Ziv, A. Lempel, "Compression of individual sequences via variable-rate coding", *IEEE Trans. on Information Theory*, vol. 24, pp. 530–536, 1978.

[92] Terry Welch, “A Technique for High-Performance Data Compression”, *Computer*, vol. 17, no. 6, pp. 8–19, 1984.

[93] David Taubman, Michael Marcellin, *JPEG2000 Image Compression Fundamentals, Standards and Practice: Image Compression Fundamentals, Standards and Practice*, Springer Science and Business Media, New York, 2012.

[94] Borko Furht, *Encyclopedia of Multimedia*, Springer US, New York, 2008.

[95] Richard Koman, *GIF Animation Studio: Animating Your Web Site, Volume 1*, Songline Studios and O’Reilly and Associates, Sebastopol, California, 1996.

[96] Greg Roelofs, *PNG: The Definitive Guide*, O’Reilly Media, USA, 1999.

[97] David Salomon, *A Concise Introduction to Data Compression*, Springer Science and Business Media, London, 2007.

[98] H. Hampel, R. B. Arps, C. Chamzas, D. Dellert, D. L. Duttweiler, T. Endoh, W. Equitz, F. Ono, R. Pasco, I. Sebestyen, C. J. Starkey, S. J. Urban, Y. Yamazaki, T. Yoshida, “Technical features of the JBIG standard for progressive bi-level image compression” *Signal Processing: Image Communication*, vol. 4, no. 2, pp.103–111, 1992.

[99] M. Abdat, M. G. Bellanger, “Combining Gray coding and JBIG for lossless image compression”, *In Proc. of the IEEE Int. Conf. on Image Processing*, vol. 3, pp. 851–855, 1994.

[100] Guy Blelloch, *Introduction to Data Compression*, Computer Science Department, Carnegie Mellon University, 2002.

[101] ISO/IEC, *JBIG2 bi-level image compression standard*, International Standard ISO/IEC 14492 and ITU-T Recommendation T.88, 2000.

[102] William B. Pennebaker, Joan L. Mitchell, *JPEG: Still Image Data Compression Standard*, Kluwer Academic Publishers, Norwell, USA, 1992.

[103] Savitri Bevinakoppa, *Still Image Compression on Parallel Computer Architectures*, Springer US, New York, 1999.

[104] Eric Hamilton, *JPEG File interchange format*, C-Cube Microsystems, 1992.

[105] Douglas A. Kerr, “Chrominance subsampling in digital images”, *The Pumpkin*, vol. 1, 2005.

[106] Gregory K. Wallace, “The JPEG still picture compression standard”, *IEEE Transactions on Consumer Electronics*, vol. 38, no. 1, 1992.

[107] Iker Gondra, Douglas R. Heisterkamp, “Content-based image retrieval with the normalized information distance”, *Computer Vision and Image Understanding*, vol. 111, no. 2, pp. 219–228, 2008.

[108] Armando. J. Pinho, Paulo. J. S. G. Ferreira, “Image similarity using the normalized compression distance based on finite context models”, *18th IEEE International Conference on Image Processing*, pp. 1993–1996, 2011.

[109] C. H. Bennett, P. Gács, M. Li, P. M. Vitányi, W. H. Zurek, “Information Distance”, *IEEE Transactions on information theory*, vol. 44, no. 4, pp. 1407–1423, 1998.

[110] Rudi Cilibrasi, Paul M. B. Vitányi, “Clustering by compression”, *IEEE Transactions on Information Theory*, vol. 51, no. 4, pp. 1523–1545, 2005.

[111] Daniele Cerra, Mihai Datcu, “A fast compression-based similarity measure with applications to content-based image retrieval”, *Journal of Visual Communication and Image Representation*, vol. 23, no. 2, pp. 293–302, 2012.

[112] Armando J. Pinho, Paulo. J. S. G. Ferreira, “Finding unknown repeated patterns in images, *European Signal Processing Conference*, pp. 584–588, 2011.

[113] Diogo Pratas, Armando J. Pinho, “A Conditional Compression Distance that Unveils Insights of the Genomic Evolution”, *Proceedings of the Data Compression Conference*, pp. 421–421, 2014.

[114] Armando J. Pinho, Diogo Pratas, Paulo. J. S. G. Ferreira, “A New Compressor for Measuring Distances among Images”, *Image Analysis and Recognition: 11th International Conference, ICIAR 2014*, pp. 30–37, 2014.

[115] Armando J. Pinho, Diogo Pratas, Paulo J. S. G. Ferreira, “Authorship attribution using relative compression”, *Proceedings of the Data Compression Conference*, pp. 329–338, 2016.

[116] J. M. Carvalho, S. Brás, J. Ferreira, S. C. Soares, A. J. Pinho, “Impact of the Acquisition Time on ECG Compression-Based Biometric Identification Systems”, *Iberian Conference on Pattern Recognition and Image Analysis*, pp. 169–176, 2017.

[117] J. M. Carvalho, S. Brás, D. Pratas, J. Ferreira, S. C. Soares, A. J. Pinho, “Extended-Alphabet Finite-Context Models”, *arXiv preprint arXiv:1709.07346*, 2017.

[118] Stefano Ferilli, *Automatic Digital Document Processing and Management: Problems, Algorithms and Techniques*, Springer Science and Business Media, London, 2011.

[119] Deskewing [Available online at] `https://fsix.github.io/mnist/Deskewing.html` [Accessed] 15/09/2017.

[120] Alain Horé, François Deschênes, Djemel Ziou, “A simple scaling algorithm based on areas pixels”, *Image Analysis and Recognition: 5th International Conference, ICIAR*, pp. 53–64, 2008.

[121] Nearest Neighbor Image Scaling [Available online at] `http://tech-algorithm.com/articles/nearest-neighbor-image-scaling/` [Accessed] 18/07/2017.

[122] Timothy C. Bell, John G. Cleary, Ian H. Witten, *Text Compression*, Prentice Hall Englewood Cliffs, New Jersey, 1990.

[123] Armando J. Pinho, Paulo J. S. G. Ferreira, António J. R. Neves, Carlos A. C. Bastos, “On the representability of complete genomes by multiple competing finite-context (Markov) models”, *PLoS ONE*, vol. 6, no. 6, pp. 1–7, 2011.

[124] D. De Cao, R. Basili, *Probability Estimation (Corso di Web Mining e Retrieval)*, Facoltá di Ingegneria, University of Rome Tor Vergata, 2009.

[125] Filipe Teixeira, Armando J. Pinho, "Boosting Compression-based Classifiers for Authorship Attribution ", *Proceedings of the 22nd Portuguese Conference on Pattern Recognition, RecPad 2016*, pp. 57–58, 2016.

[126] António Neves, Marco Henriques, Armando Pinho, "Facial recognition based on image compression", *22nd Portuguese Conference on Pattern Recognition, RECPAD*, vol. 1, pp. 43-44, 2016.