# INFORMATION-THEORETIC ANALYSIS OF CONVOLUTIONAL AUTOENCODERS

FREDERICO CARVALHO FONTES DO AMARAL

DISSERTAÇÃO DE MESTRADO
EM ENGENHARIA ELÉTRICA

DEPARTAMENTO DE ENGENHARIA ELÉTRICA

FACULDADE DE TECNOLOGIA

UNIVERSIDADE DE BRASÍLIA

Universidade de Brasília
Faculdade de Tecnologia
Departamento de Engenharia Elétrica

# Information-Theoretic Analysis of Convolutional Autoencoders

Frederico Carvalho Fontes do Amaral

## FICHA CATALOGRÁFICA

CARVALHO FONTES DO AMARAL, FREDERICO

Information-Theoretic Analysis of Convolutional Autoencoders. [Brasília/DF] 2022.

xxx, 79p., 210 x 297 mm (ENE/FT/UnB, Mestre, Dissertação de Mestrado, 2022).

Universidade de Brasília, Faculdade de Tecnologia, Departamento de Engenharia Elétrica.

Departamento de Engenharia Elétrica

| | |
|---|---|
| 1. Information Theoretic Learning | 2. Convolutional Autoencoders |
| 3. Information Theory | |
| I. ENE/FT/UnB | II. Título (série) |

## REFERÊNCIA BIBLIOGRÁFICA

CARVALHO FONTES DO AMARAL, FREDERICO (2022). Information-Theoretic Analysis of Convolutional Autoencoders. Dissertação de Mestrado, Publicação PPGEE.DM-792/22 /2022, Departamento de Engenharia Elétrica, Universidade de Brasília, Brasília, DF, 79p.

## CESSÃO DE DIREITOS

Frederico Carvalho Fontes do Amaral

Universidade de Brasília (UnB)

Campus Darcy Ribeiro

Faculdade de Tecnologia - FT

Departamento de Engenharia Elétrica(ENE)

Brasília - DF CEP 70919-970

*To my family, for the unwavering love and support.*

*To my friends, for providing and sharing some of the best moments of my life with me.*

*To Kazuki Takahashi, whose creation was a huge part of my childhood and, to this day, is an important part of my life. Rest in peace, king of games.*

# ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to my master's advisor, professor Dr. Daniel Guerreiro e Silva, for his constant support, dedication and friendship, which were pivotal for the elaboration and conclusion of this work, as well as my personal development as a researcher. To professor Edmar Candeia Gurjão, for his guidance and support since the beginning of my graduation, without whom I wouldn't be where I am today. To the teachers and professors that taught me valuable lessons, and inspired in me the love for learning. Finally, to those who contributed positively to getting me to this point, and for helping me become who I am today.

# ABSTRACT

The use of Information Theory concepts to understand deep neural networks has been extensively explored in recent years. The Information Theoretic Learning framework that resulted from such use has been acknowledged as a potentially important tool to comprehend the learning mechanisms employed during the deep neural networks' training process, for the study of which theoretical and systematic methods of analysis are still lacking. The use of statistical measurements derived from Information Theory such as entropy and mutual information has allowed for a better understanding of how the information flows through the aforementioned networks during their training. It also enabled the creation of systematic methods to design and analyze these networks in a more rigorous manner, which in turn allows the creation of more efficient and robust architectures. This work aims to investigate the extension of a method based on the aforementioned framework for the automatic detection of the bottleneck size of a convolutional autoencoder, whose objective is to find the optimal compression for the images presented to it.

Keywords: Information Theoretic Learning, Convolutional Autoencoders, Information Theory

# RESUMO

O uso de conceitos da Teoria da Informação para entender redes neurais profundas tem sido extensivamente explorado nos últimos anos. O *framework* do Aprendizado Teórico da Informação que resultou desse uso foi reconhecido como uma ferramenta potencialmente importante para compreender os mecanismos de aprendizado empregados durante o processo de treinamento das redes neurais profundas, para o estudo do qual ainda faltam métodos teóricos e sistemáticos de análise. O uso de medidas estatísticas derivadas da Teoria da Informação, tais como entropia e informação mútua, tem permitido um melhor entendimento de como a informação flui através das redes supracitadas durante seu treinamento. Ele também possibilitou a criação de métodos sistemáticos para projetar e analizar essas redes de uma maneira mais rigorosa, o que por sua vez permite a criação de arquiteturas mais eficientes e robustas. Este trabalho visa investigar a extensão de um método baseado no *framework* supracitado para a detecção automática da dimensão do gargalo de um autocodificador convolucional, cujo objetivo é encontrar a compressão óptima para as imagens a ele apresentadas.

Palavras-chave: Aprendizado Teórico da Informação, Autoencoders Convolucionais, Teoria da Informação

# TABLE OF CONTENTS

**Table of contents** i

**List of figures** iii

**List of tables** vi

**Glossary** vii

**Chapter 1 – Introduction** 1

1.1 Motivation . . . 3
1.2 Objectives . . . 4
1.3 Work Organization . . . 5

**Chapter 2 – Theoretical Foundation and Groundwork** 6

2.1 Multilayer Perceptrons . . . 6
2.2 Deep Neural Networks . . . 8
2.3 Convolutional Neural Networks . . . 12
2.4 Autoencoders . . . 16
2.5 Information Theory . . . 18
2.5.1 Rényi's Entropy and Mutual Information . . . 19
2.6 Matrix-Based Estimators for Rényi's Entropy . . . 22
2.6.1 Extension of the Matrix-Based Estimators for CNNs . . . 23
2.7 Related Works: Understanding AEs and CAEs with Information Theoretic Concepts . . . 27
2.8 Method for Automatic Sizing of the Bottleneck Layer Dimension . . . 30

**Chapter 3 – Method and Experiments** 33

3.1 Initial Experiments: MNIST Dataset . . . 33
3.2 Experimental Procedures: Wiki-Art Dataset . . . 35
3.3 Experimental Results . . . 38
3.3.1 CAE_1_4 . . . 39

3.3.2 CAE_1_6 . . . . . 43
3.3.3 CAE_1_7 . . . . . 45
3.3.4 CAE_1_8 . . . . . 46
3.3.5 CAE_1_8_N . . . . . 50
3.3.6 CAE_2_8 . . . . . 53
3.3.7 CAE_2_12 . . . . . 54
3.3.8 CAE_2_12_N . . . . . 56
3.3.9 RGB Images . . . . . 59

**Chapter 4 – Conclusions** 62

**References** 64

# LIST OF FIGURES

2.1 Simple perceptron. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 7

2.2 Layer of $k$ perceptrons in parallel. . . . . . . . . . . . . . . . . . . . . . . . 8

2.3 Example of densely connected MLP. . . . . . . . . . . . . . . . . . . . . . . 10

2.4 Structure of a MLP. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 10

2.5 Comparison between the impact of $x_3$ on $\mathbf{z} = (z_1, \ldots, z_5)$ in densely and sparsely connected layers. The hatched circles exemplify the activation of some of the layer' neurons. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 14

2.6 Comparison between the impact of $\mathbf{x} = (x_1, \ldots, x_5)$ on $z_3$ in densely and sparsely connected layers. The hatched circles indicate how a PE in the deeper layer is influenced by the PEs of the shallower layer. . . . . . . . . . . . . . . . . . . 14

2.7 Influence of the shallower layers of the CNNs on the deeper ones. The hatched circles indicate how the activation of the PEs in the shallower layers influence the PE in a deeper layer. . . . . . . . . . . . . . . . . . . . . . . . . . . . 15

2.8 Components os a typical convolutional layer. . . . . . . . . . . . . . . . . . . 15

2.9 Example of a symmetric AE with two hidden layers in the encoder and two in the decoder. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 16

3.1 Images extracted from the “Wiki-Art: Visual Art Encyclopedia” dataset. . . . . 36

3.2 Illustration of the architectures in CAE_1. . . . . . . . . . . . . . . . . . . . . 37

3.3 Illustration of the architectures in CAE_2. . . . . . . . . . . . . . . . . . . . . 37

3.4 Evolution of the CAE_1_4's information quantities' magnitudes during experiment 1. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 40

3.5 Evolution of the CAE_1_4's $\mathbf{RST}_{\%}$ and $\mathbf{WNRI}_{\%}$ during experiment 1. . . . . 41

3.6 Evolution of the CAE_1_4's information quantities' magnitudes during experiment 2. . . . . 42

3.7 Evolution of the CAE_1_4's $\mathbf{RST}_{\%}$ and $\mathbf{WNRI}_{\%}$ during experiment 2. . . . . 43

3.8 Evolution of the CAE_1_6's information quantities' magnitudes during the experiment. . . . . 44

3.9 Evolution of the CAE_1_6's $\mathbf{RST}_{\%}$ and $\mathbf{WNRI}_{\%}$ during the experiment. . . . 44

3.10 Evolution of the CAE_1_7's information quantities' magnitudes during the experiment. . . . . 45

3.11 Evolution of the CAE_1_7's $\mathbf{RST}_{\%}$ and $\mathbf{WNRI}_{\%}$ during the experiment. . . . 46

3.12 Evolution of the CAE_1_8's information quantities' magnitudes during experiment 1. . . . . 47

3.13 Evolution of the CAE_1_8's $\mathbf{RST}_{\%}$ and $\mathbf{WNRI}_{\%}$ during experiment 1. . . . . 48

3.14 Evolution of the CAE_1_8's information quantities' magnitudes during experiment 2. . . . . 48

3.15 Evolution of the CAE_1_8's $\mathbf{RST}_{\%}$ and $\mathbf{WNRI}_{\%}$ during experiment 2. . . . . 49

3.16 Evolution of the CAE_1_8_N's information quantities' magnitudes during experiment 1. . . . . 50

3.17 Evolution of the CAE_1_8_N's $\mathbf{RST}_{\%}$ and $\mathbf{WNRI}_{\%}$ during experiment 1. . . 51

3.18 Evolution of the CAE_1_8_N's information quantities' magnitudes during experiment 2. . . . . 52

3.19 Evolution of the CAE_1_8_N's $\mathbf{RST}_{\%}$ and $\mathbf{WNRI}_{\%}$ during experiment 2. . . 52

3.20 Evolution of the CAE_2_8's information quantities' magnitudes during the experiment. . . . . 53

3.21 Evolution of the CAE_2_8's $\mathbf{RST}_{\%}$ and $\mathbf{WNRI}_{\%}$ during the experiment. . . . 54

3.22 Evolution of the CAE_2_12's information quantities' magnitudes during the experiment. . . . . 55

3.23 Evolution of the CAE_2_12's $\mathbf{RST}_{\%}$ and $\mathbf{WNRI}_{\%}$ during the experiment. . . . 55
3.24 Evolution of the CAE_1_12_N's information quantities' magnitudes during experiment 1. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 56
3.25 Evolution of the CAE_1_12_N's $\mathbf{RST}_{\%}$ and $\mathbf{WNRI}_{\%}$ during experiment 1. . . 57
3.26 Evolution of the CAE_1_12_N's information quantities' magnitudes during experiment 2. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 58
3.27 Evolution of the CAE_1_12_N's $\mathbf{RST}_{\%}$ and $\mathbf{WNRI}_{\%}$ during experiment 2. . . 58
3.28 Evolution of the CAE_1_8's information quantities' magnitudes during experiment 1. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 59
3.29 Evolution of the CAE_1_8's $\mathbf{RST}_{\%}$ and $\mathbf{WNRI}_{\%}$ during experiment 1. . . . . 60
3.30 Evolution of the CAE_1_8's information quantities' magnitudes during experiment 2. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 61
3.31 Evolution of the CAE_1_8's $\mathbf{RST}_{\%}$ and $\mathbf{WNRI}_{\%}$ during experiment 2. . . . . 61

# LIST OF TABLES

3.1 Specifications of the CAEs' architectures . . . . . . . . . . . . . . . . . . . . . . . 38

# GLOSSARY

| | |
|---|---|
| AE | Autoencoder |
| CAE | Convolutional Autoencoder |
| CNN | Convolutional Deep Network |
| DNN | Deep Neural Network |
| FM | Feature Map |
| ITL | Information Theoretic Learning |
| IT | Information Theory |
| KPI | Key Performance Indicator |
| ML | Machine Learning |
| MLP | Multilayer Perceptron |
| MMI | Multivariate Mutual Information |
| MI | Mutual Information |
| PDF | Probability Density Function |
| PE | Processing Element |
| RST | Redundancy-Synergy Trade-Off |
| SAE | Stacked Autoencoder |
| WNRI | Weighted Non-Redundant Information |

CHAPTER 1

# INTRODUCTION

Since the creation of the first PC in the 1970s, it is possible to verify a constant advance in communication technologies and digital devices, as well as the growing diffusion and popularization of their use in several countries. Currently, the use of these technologies by billions of users worldwide produces a massive amount of data daily, hence why several authors and experts claim that we live in the era of *"Big Data"* [1]. The comprehension of the underlying data structures and statistical behavior that constitutes large datasets has attracted the attention of various researchers throughout the years.

For this reason, the theoretical study of deep neural networks (DNNs), which are data-driven computational structures, has been widely done due to the great successes obtained by their use in numerous practical applications [1]. However, despite this success, the current theoretical comprehension of DNNs and their learning mechanisms requires improvements. In order to tackle this issue, the Information Theoretic Learning (ITL) framework, which employs concepts and statistical measures from Information Theory (IT) within Machine Learning, has gained increasing attention in recent years. This is because it enables the creation of theoretical and systematic methods to analyze DNNs [6].

In particular, the ITL framework has been used to study Autoencoders (AEs), commonly used for data compression and representation learning, with considerable success in recent years [2, 6, 14]. Essentially, data compression aims to create more compact representations of datasets that are usually filled with redundancies and noise. It is a vastly studied topic that is crucial for many real world problems, since it allows the suppression of undesirable elements contained in the data (e.g., redundancies and noise), which in turn enables its better understanding and use [1]. AEs are unsupervised DNN architectures whose purpose is to transform inputs into new representations while causing minimal distortion. In order to do so, they create a compact representation of the input by encoding it into a lower dimension representation (i.e., code) and then decoding it into a reconstructed version of the original input. Hence, an AE consists on two main parts: the encoder, which maps the input to the code stored in the AE's bottleneck layer, and the decoder, that recreates the input using its respective code [4]. The bottleneck size plays an indispensable role for the proper functioning of this DNN, because it determines the NN's capacity of capturing the underlying statistical structure of the input. If the bottleneck's size is too big, the code created by the AE will contain redundant dimensions and the overall size of the NN will increase, alongside the memory and computational costs associated with its operation. If it is too small, the AE will be unable to completely capture the important information present in the input, which in turn may provoke the loss of relevant information. Despite its importance, the systematic evaluation of the bottleneck's size adequacy is still a topic that demands further investigation.

In order to tackle this issue, the authors of [11] proposed a method to automatically estimate the optimal dimension for the bottleneck of a densely connected stacked autoencoder (SAE) during its training. Inspired by the results presented in [6], this method aimed to enable the NN to achieve maximal data compression without compromising the overall performance of the decoder. This work aims to verify if this method can be extended to determine the bottleneck dimension of convolutional autoencoders (CAEs) (i.e., the amount of filters in their bottleneck layers), whose objective is to compress images from a given dataset. Due to structural differences between convolutional neural networks (CNNs) and densely connected ones, we show that the behavior of the information quantities associated with these networks' layers during their training process is considerably distinct. We argue that, differently from what has been verified for the dense SAE studied in [11], the bottleneck's entropy does not work as a key performance indicator (KPI) of the image compressing process inside CNN architectures. We also investigate how new information quantities defined in [7] evolve in different CAE architectures during their training, and provide an initial discussion of how their combined analysis during the learning process could be used as KPIs.

## 1.1 MOTIVATION

As previously stated, DNNs have been extensively used in various practical applications to process massive amounts of data in an unsupervised manner, with considerable success. However, the increasing number of features available from data sources constitutes an obstacle to the extension of this use to a larger number of applications. The continuous advances achieved in the areas of electronics and communications provoked a substantial increase in the number of said features and, consequently, the DNNs' input space dimensionality increased as well [2].

The aforementioned scenario is problematic because, in many practical applications, it is necessary to use probabilistic proprieties of the data. This is due to the fact that, by doing so, not only noise and redundancies in the data can be identified and attenuated, but also statistical structures can be detected and used to more effectively understand its statistical behavior. To this end, multiple mathematical methods, such as the Principal Component Analysis (PCA) and Independent Component Analysis (ICA), are widely accepted and used to process the data in order to reveal its statistical proprieties [1]. However, although they yield positive results when used in applications that involve low-dimensional data, they are not suited for those that involve high-dimensional data. This is mainly due to the fact that most of these methods either are parametric (i.e., require a previous assumption of the data's PDF) or involve the estimation of the data's PDF. Parametric methods are not suited for applications that involve high-dimensional data because its complexity usually makes so that the PDF previously assumed is insufficient to adequately represent the data's statistical behavior. Similarly, the methods that involve the estimation of the data's PDF are also not suited for said applications because the estimation of PDFs in spaces with high dimensionality is a difficult and costly problem from a computational point of view [2].

Although problematic, this scenario is precisely one of the reasons for the great interest in DNNs today. That is because they are able to yield positive results in practical applications that involve high-dimensional, raw data without resorting to feature engineering (i.e., process of using domain knowledge to extract features from raw data) [4]. Despite this fact, however, the aforementioned increase of the data's dimensionality caused by the larger amount of features available from data sources is an obstacle for the use of DNNs in a greater number of real-world applications. That is because it requires increasingly larger, deeper DNNs which, in turn, severely increases their memory and computational costs. Thus, reducing the dimension of the DNNs' input space has become crucial for the extension of their use to a larger amount of practical applications [2, 11]. In order to achieve said reduction, the use of AEs has become increasingly common in practical applications [6]. As stated earlier, these DNNs aim to compress their inputs by creating more compact representations of them. This is done in such a way that these representations retain the highest possible amount of important information (i.e., non-redundant, with the smallest amount of noise as possible) for a given task. However, despite the increase in their use, the determination of the optimal bottleneck layer's dimension of the AEs is still a subject little explored in the literature [11]. This is problematic, since the dimension of this layer directly impacts the quality of the representations produced by the AE and, consequently, its ability to accommodate the natural structure of the data.

In order to solve this problem, the authors of [11] proposed a method for automatically determining the dimension of SAEs. It was inspired by the results presented in [6], where the ITL framework was used to interpret the learning process of AEs. Despite the promising results presented in [11], this work explored a single SAE architecture, and made use of a small number of simple image databases and dataset used in road traffic forecasting applications. These experimental limitations severally reduce the range of the conclusions drawn from the results obtained. Nevertheless, they suggest that it may be feasible to extend the proposed method to more complex DNN architectures, as well as applications that involve more complex, larger datasets.

## 1.2 OBJECTIVES

The present work aims to investigate the possibility of extending the method for automatic sizing of SAEs proposed in [11] to CAEs. In order to do so, the matrix-based estimators for entropy and mutual information proposed in [10], as well as the new information quantities proposed in [7], were used to evaluate the evolution of the information quantities inside the encoder and bottleneck layers of various CAE architectures. These CAEs were designed such that the effects of different parameters (e.g., size of the bottleneck, interpolation function used in the decoder) on the evolution of the quantities could be evaluated in a more precise fashion.

This was done in order to identify if the information quantities estimated could be used as Key Performance Indicators (KPIs) for the different CAEs' performance and, if so, which ones.

## 1.3 WORK ORGANIZATION

This work is organized as follows. Chapter 2 briefly addresses the basic concepts and ideas behind the neural networks used in the experiments, as well as exposes the information estimators used in the experiments. Chapter 3 details the experimental procedures, exposes the results obtained and discusses them, explaining why the entropy of the code cannot be used as the sole KPI for CAEs, and suggests how the new information quantities proposed in [10] could be better suited for this purpose. Finally, Chapter 4 concludes this work and highlights topics that can be addressed in future studies.

CHAPTER 2

# THEORETICAL FOUNDATION AND GROUNDWORK

In this chapter, we will address the basic concepts behind the perceptrons, the most basic structure inside a DNN. Next, we will explain how they can be organized to form the multilayer perceptrons, which are some of the most basic types of neural network. After that, we will address the basic ideas behind DNNs, CNNs and AEs, which are the central objects of study of the present work. Then, we will briefly discuss some of the key concepts of the IT and the ideas behind the use of its informational quantities to the study of DNNs, which resulted in the creation of the ITL framework. Finally, we will address a method for automatic sizing of a type of AE that is based on the aforementioned framework.

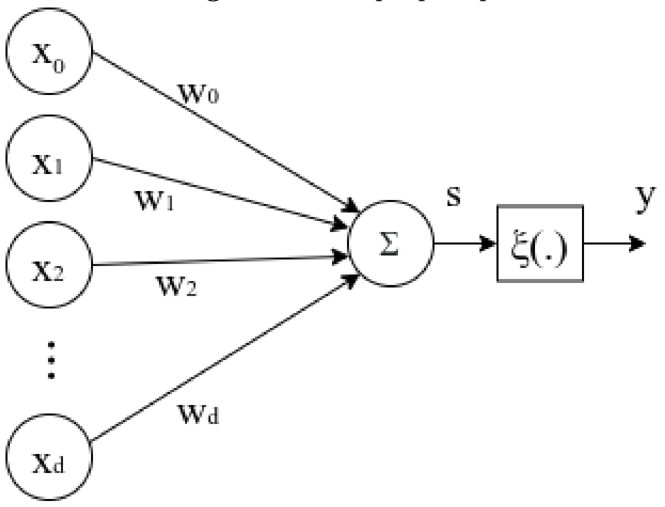
## 2.1 MULTILAYER PERCEPTRONS

As mentioned previously, the field of ML has gained increasing interest within the scientific community in the recent decades and is rapidly expanding. Essentially, it consists of programming computers to make them capable of automatically learning from the data provided to them and making decisions without being explicitly programmed to do so [1]. In particular, the computational structures called artificial neural networks or simply neural networks (NNs) are objects of great interest among researchers, governments and corporations nowadays.

The most basic component of a NN is the neuron or perceptron, depicted in Figure 2.1, which are processing elements (PEs) whose inputs come from the environment or may be the outputs of other perceptrons. The inputs are numbers that can be viewed as the entries of an array $\mathbf{x} = (x_0,x_1,x_2,\dots,x_d) \in \mathbb{R}^d$, where $d$ is the input's dimensionality and $x_0 = 1$, the bias unit. To each entry of $\mathbf{x}$ is assigned a connection weight that can also be viewed as the entries of an array $\mathbf{w} = (w_0,w_1,w_2,\dots,w_d) \in \mathbb{R}^d$, where $w_0$ is the weight assigned to the bias unit.

The perceptron works as follows. First, each component of $\mathbf{x}$ is multiplied by its respective

**Figure 2.1.** Simple perceptron.



**Source:** Own authorship.

weight in **w** and the results are then summed, as described in expression (2.1).

$$s = \mathbf{w}^T.\mathbf{x} = \sum_{i=1}^{d} w_i.x_i + w_0. \tag{2.1}$$

Then, the nonlinear funtion $\xi(.)$ is applied to the result $s$, resulting in the perceptron's output $y$, which is also a scalar. The nonlinear function $\xi(.)$ is the perceptron's activation function, and it determines if the PE will be activated (i.e., $y \neq 0$) or not (i.e., $y = 0$). Although multiple functions can be used as activation functions (e.g., sigmoid, hyperbolic tangent), the Rectified Linear Unit (ReLU) is the most widely used nowadays, being defined as

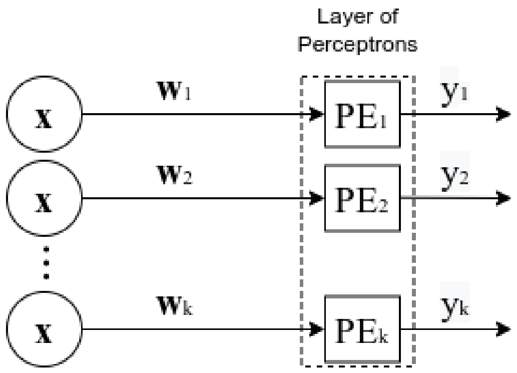
$$\text{ReLU}(s) = \max(0,s) = \begin{cases} s, s \geq 0; \\ 0, s < 0. \end{cases} \tag{2.2}$$

As shown in expression (2.2), the PE will activate if $s > 0$, and will not otherwise. The bias unity's goal is to facilitate or hinder the PE's activation. If $w_0 > 0$, the PE will activate even if $\sum_{i=1}^{d} w_i.x_i$ in expression (2.1) is negative, provided that it's absolute value is smaller than $w_0$. On the other hand, if $w_0 < 0$, the PE will not activate unless the value of $\sum_{i=1}^{d} w_i.x_i$ in (2.1) is larger than the absolute value of $w_0$. Therefore, in the case of a perceptron that has the ReLU as its activation function,

$$y = \begin{cases} s, s \geq 0; \\ 0, s < 0. \end{cases} \tag{2.3}$$

If we have $k$ PEs in parallel, as depicted in Figure 2.2, we can view their individual outputs as the entries of an array $\mathbf{y} = (y_1, y_2, \ldots, y_k)$, where $y_j = \xi(s_j)$, and $s_j = \mathbf{w}_j^T.\mathbf{x}$, $1 \leq j \leq k$.

**Figure 2.2.** Layer of $k$ perceptrons in parallel.



**Source:** Own authorship.

By viewing the values $s_i$ as entries of an array $\mathbf{s} = (s_1, s_2, \ldots, s_k)$, we have

$$\mathbf{s} = \mathbf{W}.\mathbf{x}, \tag{2.4}$$

where $\mathbf{W}$ is a matrix whose lines are the vectors $\mathbf{w}_i$, $1 \leq i \leq k$. The matrix $\mathbf{W}$ in expression (2.4) defines a linear transformation from $\mathbb{R}^d$ to $\mathbb{R}^k$. Although the perceptron by itself has several computational limitations [1], by connecting layers of PEs such as the one depicted in 2.2, these limitations are mostly overcome. These structures, called multilayer perceptrons (MLPs), are among the most well-known types of NNs. Essentially a nonparametric estimator, the MLP is a fully connected class of NNs that consists of at least three layers of PEs: an input layer, a hidden layer and an output layer. An example of this structure can be obtained from the one depicted in Figure 2.2 by using the outputs $y_1, \ldots, y_k$ as inputs to another layer of perceptrons. This would become the MLP's output layer, while the layer of perceptrons depicted in this figure would become its hidden layer. Despite its apparent simplicity, this basic MLP has been proved to being able to learn any nonlinear function of the input [1].

## 2.2 DEEP NEURAL NETWORKS

The DNNs are basically non-linear computational structures built using MLPs with multiple hidden layers as illustrated in Figure 2.3. This is done in order to increase the NN's capacity of creating more complex, abstract representations from its raw inputs [1, 2, 4]. At a first glance, this widely accepted fact seems to somewhat contradict the Universal Approximation Theorem, which states that an MLP with a single hidden layer can learn any nonlinear function of its input. However, the theorem neither specifies nor bounds the amount of hidden units necessary to enable this learning: it only guaranties the solution's existence [1].

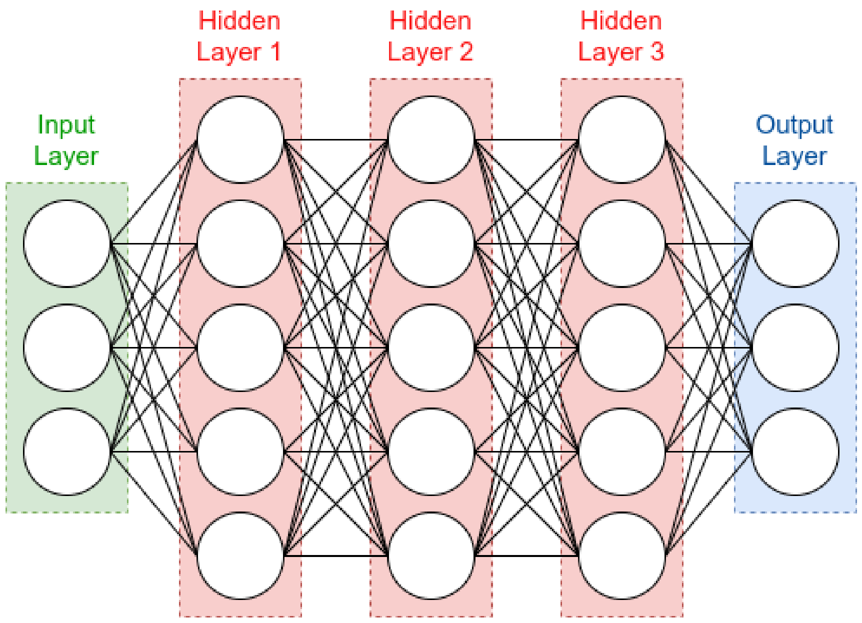
More often than not, in practical applications, it is unfeasible to determine the exact number of PEs inside the single hidden layer of said MLP necessary to make it able of learning the desired nonlinear function of its inputs. By using MLPs with multiple hidden layers, this limitation ceases to exist. Not only these MLPs are also universal approximators, but they also do not require the determination of said exact number of PEs. The abstract representations of the inputs created in their hidden layers enable them to more rapidly determine the nonlinear function of interest. For this reason, in practical applications, instead of using MLPs with one single hidden layer and search for the exact number of hidden PEs necessary for the application, MLPs with multiple hidden layers are used.

The DNNs are deeply associated with the concept of Deep Learning, which consists on the design of computational methods whose goal is to build the most adequate representation of the data for a given task, based on a concept hierarchization (i.e. from the most basic features to the increasingly more abstract characteristics), without being explicitly programmed to do so.

Essentially, a method reasoned on Deep learning aims to build complex representations from the data in terms of simpler ones through the data fed to it. Thus, during the training of a DNN, the idea is that its deeper layers acquire the ability of creating complex representations of the input data using the simpler ones created by the layers that precede them. Through this process, the DNNs automatically learn the appropriate representation of the data for a given task.

DNNs are usually trained through the classical stochastic gradient descent (SGD) in association with the backpropagation method, commonly done through the use of mini-batches of examples to be processed at each training step [1]. These are small batches of examples randomly picked from the dataset that are fed to the NN, so that the gradient is calculated on each one of them and a single update is performed with their average. The SGD is performed

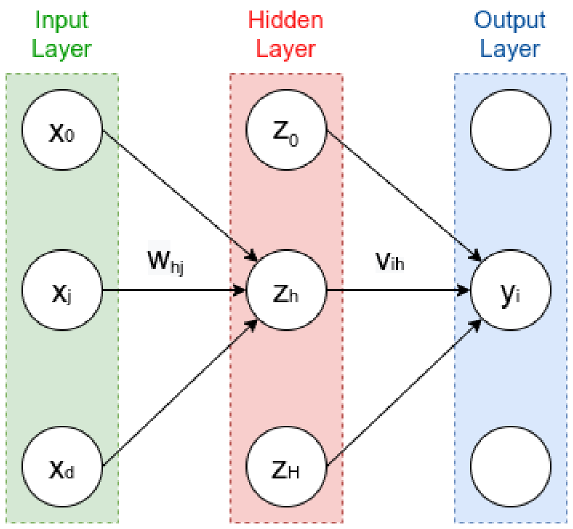
**Figure 2.3.** Example of densely connected MLP.



**Source:** Own authorship.

as follows. Let's consider the MLP depicted in Figure 2.4.

**Figure 2.4.** Structure of a MLP.



**Source:** Own authorship.

In Figure 2.4, $x_j$, $0 \leq j \leq d$, are the NN's inputs. $z_h$, $0 \leq h \leq H$, are the hidden units and $H$, the dimensionality of this hidden space. $x_0 = 1$ and $z_0 = 1$ are the bias' unities of the input and the hidden layer, respectively. $y_i$, $0 \leq i \leq k$ are the output units and $k$, the dimensionality of the output space. Finally, $w_{hj}$ and $v_{ih}$ are weights in the first and second layers, respectively. Initially, an error (cost) function $E$ (e.g., MSE) is selected and random weights are assigned to the perceptrons' connections. If $E$ is a differentiable of the vector of weights $\mathbf{w_h} = [w_{h1}, \ldots, w_{hj} \ldots, w_{hd}]$, its gradient vector is

$$\nabla_{\mathbf{w_h}} E = \left[\frac{\partial E}{\partial w_{h0}}, \ldots, \frac{\partial E}{\partial w_{hj}}, \ldots, \frac{\partial E}{\partial w_{hd}}\right]. \tag{2.5}$$

At each step of the procedure, the entries of $\mathbf{w_h}$ are updated in the opposite direction of the gradient (i.e., the direction of its maximum decrease)

$$w_{hj} = w_{hj} + \Delta w_{hj},\ \Delta w_{hj} = -\eta \frac{\partial E}{\partial w_{hj}}, \tag{2.6}$$

where $\eta$ is the learning factor, and determines how much the value of $w_{hj}$ will change. By using the chain rule to compute the gradient for the first layer's weights $w_{hj}$, we obtain

$$\frac{\partial E}{\partial w_{hj}} = \frac{\partial E}{\partial y_i} \frac{\partial y_i}{\partial z_h} \frac{\partial z_h}{\partial w_{hj}}. \tag{2.7}$$

The name "backpropagation" comes from the fact that, as suggested by expression (2.7), it is as if the error propagates from the output back to the inputs [1]. This training procedure is widely adopted due to its various advantages, among which we can highlight the fact that the backpropagation is fast, simple and easy to program, and that the use of mini-batches makes gradient estimate robust to noise and outliers. Therefore, the learning process of a DNN essentially consists on fitting the weights of its neurons through the minimization of a given cost function by using its gradient.

Due to the use of the SGD, the DNNs training procedure and results are heavily reliant on the choice for the neurons' non-linearity. As mentioned in Section 2.1, the ReLU function, defined in expression (2.2), is the most commonly used nowadays. This is due to the fact that it is very close to a linear function, which means that it preserves many properties that make linear models easy to optimize via gradient-based methods. Also, due to it's nature, the derivatives through a ReLU function remain large whenever the unit is active, which means that the gradient direction calculation remains useful throughout the network's layers. Due do this fact, it does not suffer from the vanishing gradient problem when used in applications that involve stochastic SGD. The main disadvantage of the ReLU function is that its derivative is zero for negative arguments, which means that there is no learning if the weighted sum of the neuron's inputs becomes negative [4].

## 2.3 CONVOLUTIONAL NEURAL NETWORKS

CNNs are a widely used type of DNN that are specialized for processing data that has a known grid-like topology (e.g., images). They are characterized by their use of convolution in place of general matrix multiplication in at least one of their layers [4]. The convolution between two continuous functions is defined as

$$z(t) = (x * k)(t) = \int_{-\infty}^{\infty} x(a)k(t-a)\ da, \tag{2.8}$$

where $x : \mathbb{R} \to \mathbb{R}$ is the input, $k : \mathbb{R} \to \mathbb{R}$ is the filter (or kernel) and $z : \mathbb{R} \to \mathbb{R}$ is the feature map (FM). The convolution between two discrete functions is defined as

$$z[n] = (x * k)[n] = \sum_{a=-\infty}^{\infty} x[a]k[n-a], \tag{2.9}$$

where $x : \mathbb{Z} \to \mathbb{R}$, $k : \mathbb{Z} \to \mathbb{R}$ and $z : \mathbb{Z} \to \mathbb{R}$. In ML applications, the NNs' inputs normally are tensors of parameters, which are, in the general case, arrays of numbers arranged on a regular grid with a variable number of axes [4]. It is noteworthy to point that, although in the ML literature it has become increasingly common to refer to tensors as multidimensional arrays, that is not their actual mathematical definition. A tensor is formally defined as an algebraic object that describes a multilinear relationship between sets of algebraic objects related to a vector space, being widely used in the study of manifolds [5].

We usually assume that the input and kernel functions are zero everywhere but in the finite set of points where their values are stored, which means that the infinity summation in expression (2.9) is implemented over a finite number of array elements in practice. This is often done more than one axis at a time. If the input $M$ is a two-dimensional image and the kernel $K$ is also two-dimensional, their convolution is defined as

$$Z[i,j] = (M * K)[i,j] = \sum_{m}\sum_{n} M[m,n]K[i-m,j-n], \tag{2.10}$$

where $Z \in \mathbb{R}^{r\times s}$, $M \in \mathbb{R}^{m\times n}$ and $\in \mathbb{R}^{l\times k}$, $l \leq r \leq m$, $k \leq s \leq n$. In practice, the convolution operation in expression (2.10) is performed as follows. Let's consider expression (2.11), where the matrix $M$ represents some 2D data (e.g., an image) and $K$ is the convolution kernel.

$$M = \begin{bmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \end{bmatrix}, K = \begin{bmatrix} w & x \\ y & z \end{bmatrix}, m_1 = \begin{bmatrix} a & b \\ e & f \end{bmatrix}, m_2 = \begin{bmatrix} b & c \\ f & g \end{bmatrix}, \ldots, m_6 = \begin{bmatrix} g & h \\ k & l \end{bmatrix} \tag{2.11}$$

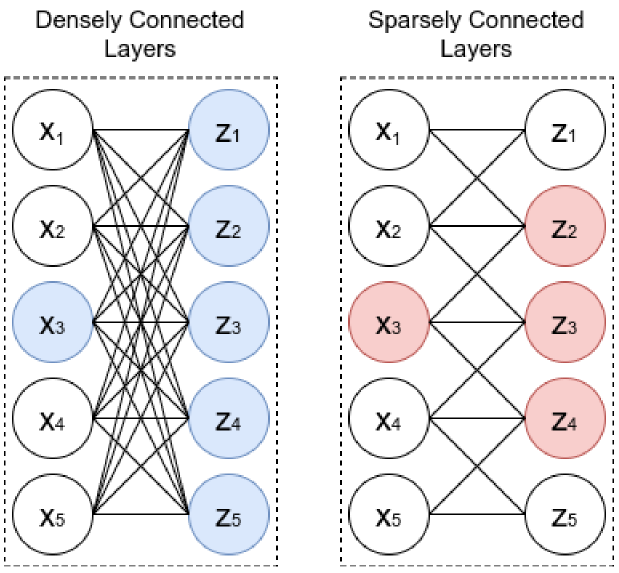
First, $K$ performs the product between itself and $m_1$, which is built from the portion of $M$ highlighted in red, resulting in $o_1$. Then, $K$ performs the product between itself and the portion of $m_2$, resulting in $o_2$, and so forth until $o_6$ is obtained, as shown in expression (2.12), where $*$ denotes the matrix multiplication operation.

$$\begin{gathered} K * m_1 = o_1 = aw + bx + ey + fz;\ K * m_2 = o_2 = bw + cx + fy + gz; \\ K * m_3 = o_3 = cw + dx + gy + hz;\ K * m_4 = o_4 = ew + fx + iy + jz; \\ K * m_5 = o_5 = fw + gx + jy + kz;\ K * m_6 = o_6 = gw + hx + ky + lz. \end{gathered} \tag{2.12}$$

As $K$ slides along $M$, the convolution operation generates a FM, which in turn contributes to the input of the next layer. Therefore, the discrete convolution in expression (2.10) can be viewed as multiplication by a matrix. Differently from traditional DNNs, whose layers use the matrix multiplication described in expression (2.4), the CNNs have sparse interactions, which is accomplished by making $K$ (the dimension of their convolutional layers's kernels) smaller than $M$ in expression (2.10). By doing this, features such as edges in $M$ can be detected by kernels that can be thousands of times smaller than it. Therefore, CNNs require the storage of a smaller number of parameters when compared to densely connected DNNs, as well as require fewer operations to compute their outputs. This makes so that the memory and computational costs of the former are considerably smaller than those of the latter [4].

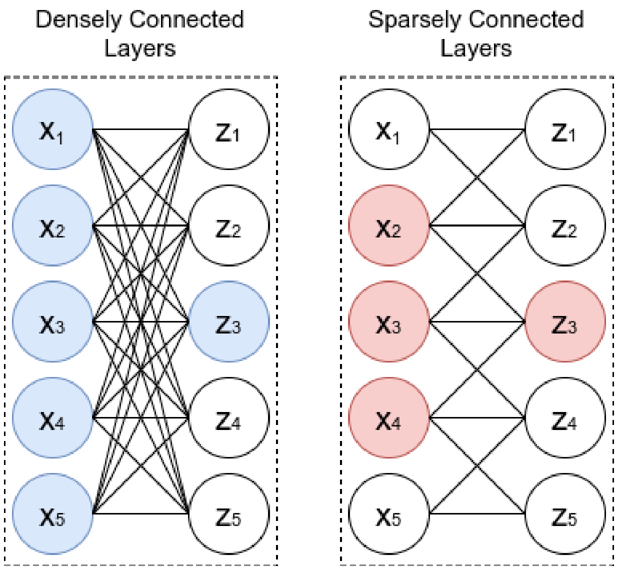
Let's consider Figures 2.5 and 2.6, where $\mathbf{z} = (z_1, \ldots, z_5)$ is formed by a matrix multiplication in the densely connected case and by a convolution with a kernel of width 3 in the sparsely connected one.

**Figure 2.5.** Comparison between the impact of $x_3$ on $\mathbf{z} = (z_1, \ldots, z_5)$ in densely and sparsely connected layers. The hatched circles exemplify the activation of some of the layer' neurons.



**Source:** Own authorship.

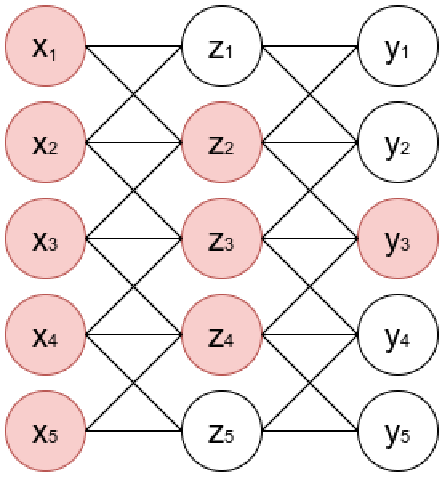
**Figure 2.6.** Comparison between the impact of $\mathbf{x} = (x_1, \ldots, x_5)$ on $z_3$ in densely and sparsely connected layers. The hatched circles indicate how a PE in the deeper layer is influenced by the PEs of the shallower layer.



**Source:** Own authorship.

In Figure 2.5, it can be seen that, in the case of the densely connected layers, the value of $x_3$ affects the values all the entries of $\mathbf{z}$. On the other hand, this Figure also shows that, in the case of the sparsely connected layers, the value of $x_3$ only affects the values of $z_2$, $z_3$ and $z_4$. In turn, Figure 2.6 shows that, in the case of the densely connected layers, the value of $z_3$ is affected by all the entries of $\mathbf{x}$. However, in the case of the sparsely connected layers, the value of $z_3$ is affected only by the values of $x_2$, $x_3$ and $x_4$. This makes the CNN able to capture local information from data (e.g., that associated with specific parts of an image, such as edges and shapes). However, it does not mean that $\mathbf{x}$ affects only a small portion of the PEs of the next layers PEs. As seen in Figure 2.7, the deeper a given layer is, the larger will be the amount of entries of $\mathbf{x}$ that will affect its PEs' values [4].

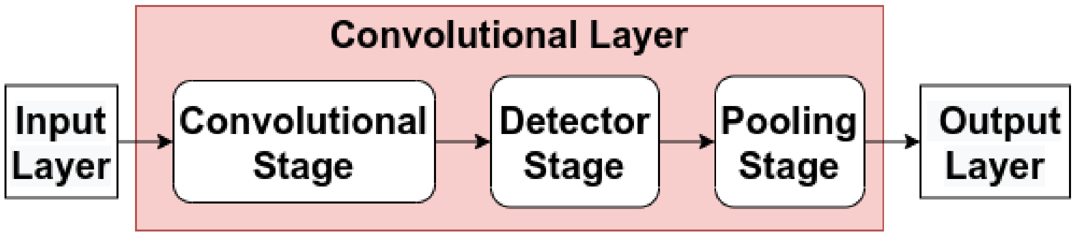
**Figure 2.7.** Influence of the shallower layers of the CNNs on the deeper ones. The hatched circles indicate how the activation of the PEs in the shallower layers influence the PE in a deeper layer.



**Source:** Own authorship.

A typical convolutional layer of a CNN is depicted in Figure 2.8. The convolutions between its inputs and kernels is performed in the convolutional stage. Then, in the detector stage, the convolutions' results are fed to a nonlinear activation function (e.g., ReLU). The nonlinear function's activations will compose the FMs, which will then be fed to a pooling function in the pooling stage. This function is commonly used to reduce the dimension of the FMs by replacing their elements with a summary statistic of its neighbors. For example, the max pooling function extracts the maximum output over a rectangular region of a FM [4].

**Figure 2.8.** Components os a typical convolutional layer.



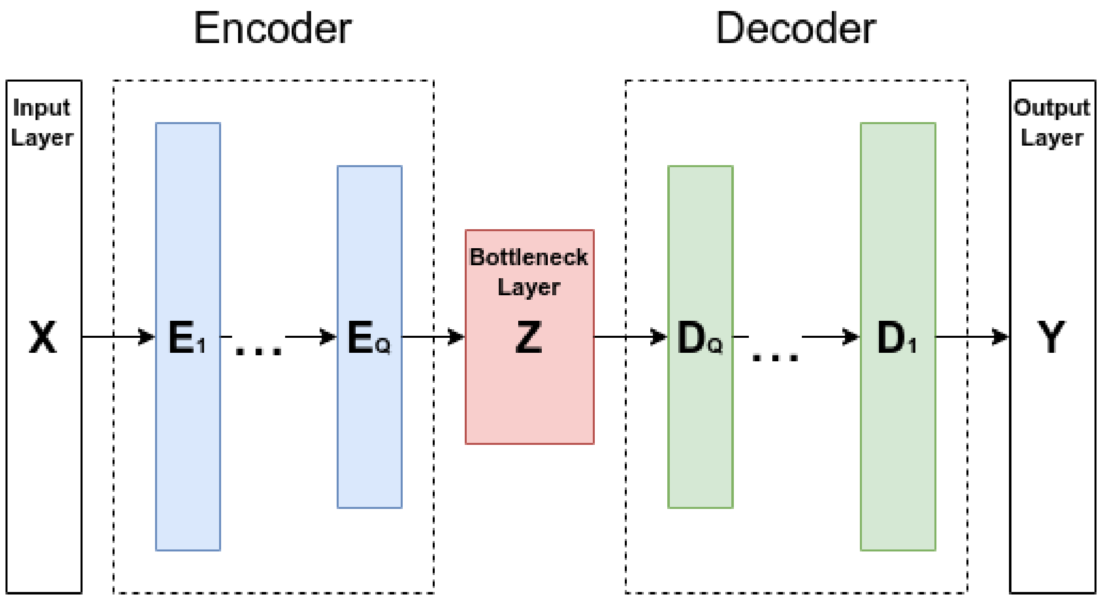
**Source:** Own authorship.

The presence of the pooling stage inside the convolutional layers makes their representations invariant to small translations of the input (i.e., small translations in the input will not modify the output of the pooling operation). This makes the CNNs specially useful in applications that involve determining if a given feature is present in the data or not, rather than knowing exactly where it is [4].

## 2.4 AUTOENCODERS

A well-known type of DNN architecture, the AE is commonly employed in the context of dimensionality reduction and representation learning, having great importance in the area of data compression. AEs are unsupervised DNNs whose purpose is to copy its input to its output while causing minimal distortion. In order to do so, they create a compact representation of the input by encoding it into a lower dimension representation called code and then decoding it into a reconstructed version of the original input [1].

**Figure 2.9.** Example of a symmetric AE with two hidden layers in the encoder and two in the decoder.



**Source:** Own authorship.

Figure 2.9 depicts a AE with $Q$ hidden layers in the encoder $(E_1, \dots, E_Q)$, $Q$ in the decoder $(D_1, \dots, D_2)$ and the bottleneck layer $Z$. If the AE is such as its encoder and decoder have the same amount of hidden layers, and correspondent layers have the same amount of neurons (e.g., $E_1$ have the same amount of PEs as $D_1$), it is called a stacked AE (SAE). If the AE has at least one convolutional layer, it is called a convolutional AE (CAE).

In general, AEs consists on two parts: the encoder, which maps the input to the code stored in the DNN's bottleneck layer, and the decoder, that recreates the input using its respective code [4]. The encoder can be viewed as

$$\mathbf{z} = f(\mathbf{x}), \tag{2.13}$$

which is a function that maps the AE's input $x$ to the code $z$ stored inside the bottleneck layer. In turn, the dencoder can be viewed as

$$\mathbf{y} = g(\mathbf{z}) \tag{2.14}$$

which is a function that creates a reconstruction $y$ of the AE's input $x$ using the code $z$ stored inside the bottleneck layer. If the dimension of $z$ is smaller than that of $x$, the AE is called undercomplete. These AEs are of particular importance because they are forced to capture the most important features of the training data in order to reduce the dimension of $x$ [4]. Therefore, the goal of their training is to minimize the cost function

$$L\{\mathbf{x}, g[f(\mathbf{x})]\}, \tag{2.15}$$

where the cost function $L$ (e.g., MSE) penalizes $g[f(\mathbf{x})]$ for being dissimilar from $\mathbf{x}$. Due to this characteristics, AEs are widely used in applications that require the suppression of undesirable elements contained in data (e.g., redundancies and noise), which in turn enables its better understanding and use [1].

Expressions (2.13) and (2.14) show that the bottleneck size ($size(Z)$) plays an indispensable role for the proper functioning of this type of DNN. If $size(Z)$ is too small, the AE will be unable to completely capture the important information present in the input. This means that the AE will not be able to properly minimize the cost function (2.15) due to the loss of relevant information provoked by its structural limitations. If $size(Z)$ is too big, the AE's code will contain redundant dimensions. This means that $f(.)$ will be unable to suppress the maximum amount of noise and redundancy in $x$ during the creation of $z$. Despite its importance, there is a lack of methods to evaluate the adequacy of $size(Z)$ [11].

## 2.5 INFORMATION THEORY

Originally, Information Theory (IT) was conceptualized and formalized in Shannon's work with the aim of dealing with the problem of optimally transmitting messages through noisy channels [2]. To this end, he defined statistical quantities in order to measure the amount of information contained in these messages, as well as the amount of information that a message can contain about others. By modeling these messages as random variables (RVs), we can use their probability distribution functions (PDFs) to describe their statistical behavior and quantify their informational content. Thus, for any PDF, we define a quantity called entropy that quantifies the uncertainty associated with a given RV [3]. Despite the vast scope of the concept of information, the entropy of a RV can be used in a wide range of applications to quantify it satisfactorily. The more uncertain the behavior of a RV is, the greater its entropy value will be and, consequently, the greater its informational content. Likewise, the amount of information contained in RVs with low entropy will be smaller. Naturally, if there is no uncertainty associated with the behavior of an RV (e.g., the value of RV $X$ that represents the result of tossing a coin whose faces are equal), its entropy will be zero, since there will be no information associated with it. Let $X$ be a random vector (RVEC) with PDF $p(\mathbf{x})$, $Y$ be a $v$-dimensional RVEC with PDF $p(\mathbf{y})$ and $p(\mathbf{x},\mathbf{y})$, their joint PDF. Since both $X$ and $Y$ are RVECs, each one of their entries is a RV (e.g., a pixel of an image). Mathematically, we define the entropy of $X$ as

$$H(X) = -\int p(\mathbf{x}) \log p(\mathbf{x}) \; d\mathbf{x}, \tag{2.16}$$

where $\mathbf{x} \in \mathbb{R}^u$, $p(\mathbf{x}) = p(x_1, \ldots, x_u)$ and $d\mathbf{x} = dx_1 \ldots dx_u$. The joint entropy of a pair of continuous RVECs $X$ and $Y$ with a joint distribution $p(\mathbf{x},\mathbf{y})$ is dened as

$$H(X,Y) = -\int p(\mathbf{x},\mathbf{y}) \log p(\mathbf{x},\mathbf{y}) \; d\mathbf{x}d\mathbf{y}, \tag{2.17}$$

where $\mathbf{y} \in \mathbb{R}^v$, $p(\mathbf{x},\mathbf{y}) = p(x_1, \ldots, x_u, y_1, \ldots, y_v)$ and $d\mathbf{x}d\mathbf{y} = dx_1 \ldots dx_u dy_1 \ldots dy_v$. The divergence (also known as the KullbackLeibler distance or relative entropy [3]) between two PDFs is another important statistical quantity of IT. It consists of a measure of the similarity between two statistical distributions $p(x)$ and $q(x)$, being defined as

$$D(p||q) = -\int p(\mathbf{x}) \log \frac{p(\mathbf{x})}{q(\mathbf{x})}\, d\mathbf{x}. \tag{2.18}$$

Although it is useful to think of $D(p||q)$ as a measure of the distance between $p(x)$ and $q(x)$, it is not a true distance measure between PDFs, because it is not symmetric and does not satisfy the CauchySchwarz inequality [2]. It can also be interpreted as a measure of the inefficiency of assuming that the distribution of a given RV is $q$ when in reality it is $p$ [3]. Mutual information (MI) is another important statistical quantity of IT, being the most used in the context of communications theory [2]. A special case of divergence, the MI $I(X;Y)$ is a measure of the amount of information that the RV $X$ contains about the RV $Y$, and corresponds to the reduction in the uncertainty of $X$ due to knowledge of $Y$ and vice versa [3]. It is defined as

$$\begin{aligned} I(X;Y) = I(Y,X) = \int p(\mathbf{x},\mathbf{y}) \log \frac{p(\mathbf{x},\mathbf{y})}{p(\mathbf{x})q(\mathbf{x})}\, dx = D[p(\mathbf{x},\mathbf{y})||p(\mathbf{x})q(\mathbf{x})] = \\ = H(X) - H(X|Y) = H(Y) - H(Y|X) = H(X) + H(Y) - H(X,Y). \end{aligned} \tag{2.19}$$

it is worth noting that, by fixing $u = v = 1$, expressions (2.16), (2.17), (2.18) and (2.19) become those of unidimensional RVs.

### 2.5.1 Rényi's Entropy and Mutual Information

Originally proposed and formalized in Claude Shannon's seminal work, Entropy and Mutual Information (MI) are commonly used measurements of information [3]. While there are multiple definitions for these quantities, the family of parametric entropies formalized in Alfréd Rényi's work has been widely used in ITL [2]. Let $X$ be a $u$-dimensional RVEC with PDF $p(\mathbf{x})$, $Y$ be a $v$-dimensional RVEC with PDF $p(\mathbf{y})$ and $p(\mathbf{x},\mathbf{y})$, their joint PDF. The $\alpha$-Rényi entropy of $X$ is defined as

$$\begin{aligned} H_\alpha(X) &= \frac{1}{1-\alpha}\log\int p^\alpha(\mathbf{x})\ d\mathbf{x} = \frac{1}{1-\alpha}\log\mathbb{E}[p^{\alpha-1}(\mathbf{x})] = \\ &= \frac{1}{1-\alpha}\log[V_\alpha(X)], \end{aligned} \tag{2.20}$$

where $V_\alpha(x)$ is the $\alpha$-information potential (IP$_\alpha$) of $X$. When $\alpha > 1$, $H_\alpha$ is a monotonic decreasing function of IP$_\alpha$, which implies that minimizing $H_\alpha$ is equivalent to maximizing IP$_\alpha$ and vice versa. When $\alpha \to 1$, expression (2.20) converges to Shannon's entropy, which means that the latter is a special case of the former [2]. The $\alpha$-Rényi joint entropy of $X$ and $Y$ is defined as

$$H_\alpha(X,Y) = \frac{1}{1-\alpha}\log\int p^\alpha(\mathbf{x},\mathbf{y})\ dx_1\dots dx_u dy_1\dots dy_v, \tag{2.21}$$

and the $\alpha$-Rényi MI between $X$ and $Y$ can be expressed as

$$I_\alpha(X;Y) = H_\alpha(X) + H_\alpha(Y) - H_\alpha(X,Y). \tag{2.22}$$

As done previously, expressions (2.20), (2.21) and (2.22) become those of unidimensional RVs by fixing $u = v = 1$. The exact evaluation of these expressions is often impossible in practice, because it requires the knowledge of $p(\mathbf{x})$, $p(\mathbf{y})$ and $p(\mathbf{x},\mathbf{y})$, which are usually unknown. The estimation of these PDFs in practice is often difficult as well, since it usually involves high-dimensional data, for which PDF estimation can be both unreliable and computationally unfeasible. In order to bypass this issue, it is common in the literature to use the quadratic Rényi entropy, obtained by making $\alpha = 2$ in expression (2.20), which gives

$$H_2(X) = -\log\int p^2(x)\ dx = -\log\mathbb{E}[p(x)] = -\log[V_2(X)]. \tag{2.23}$$

It is important to note that, unlike Shannon's definition of entropy in expression (2.16), the logarithm in expression (2.20) lies outside the integral. This particularity of $H_\alpha$ is extremely important because it allows for the estimation of the IP$_2$ of $X$ in a non-parametric manner, using only the differences between pairs of samples, even if the user has no previous knowledge of its PDF $p(x)$. This is extremely useful since not only this is the most common case in practice, but also because the use of parametric models in these situations is not recommended due to the risk of introducing substantial errors to the estimates. By using $H_2$, we aim to avoid unnecessary computations by estimating $\mathbb{E}[p(x)]$ directly from the samples, instead of doing so after estimating $p(x)$ [2]. To do so, we proceed as follows. Let $\{x_1,\dots,x_N\}$ be a set of $N$ independent and identically distributed (iid) samples of the RV $X$. The kernel (Parzen) estimate of the PDF using an arbitrary kernel function $\kappa_\sigma(.)$ is given by

$$\hat{p}_X(x) = \frac{1}{N\sigma}\sum_{i=1}^{N}\kappa_\sigma\left(\frac{x-x_i}{\sigma}\right), \tag{2.24}$$

where $\sigma$ is the kernel size, also known as bandwidth parameter [13]. In expression (2.24), the kernel function $\kappa_\sigma(.)$ is such that $\kappa_\sigma(x) \geq 0$, $\int_{-\infty}^{\infty}\kappa_\sigma(x)\ dx = 1$ and $\lim\limits_{x\to\infty}|x.\kappa_\sigma(x)| = 0$. For computational and analytical reasons, it is common to use Gaussian kernels, denoted $G_\sigma(.)$, where the kernel size $\sigma$ is the square root of the distribution's variance, $\sigma^2$. By using this kernel in expression (2.24) and plugging the result in expression (2.23), we obtain

$$\begin{aligned}\hat{H}_2(X) &= -\log\int_{-\infty}^{\infty}\left[\frac{1}{N}\sum_{i=1}^{N}G_\sigma\left(\frac{x-x_i}{\sigma}\right)\right]^2 dx\\ &-\log\frac{1}{N^2}\int_{-\infty}^{\infty}\left[\sum_{i=1}^{N}\sum_{j=1}^{N}G_\sigma\left(\frac{x-x_j}{\sigma}\right).G_\sigma\left(\frac{x-x_i}{\sigma}\right)\right] dx\\ &-\log\frac{1}{N^2}\sum_{i=1}^{N}\sum_{j=1}^{N}\left[\int_{-\infty}^{\infty}G_\sigma\left(\frac{x-x_j}{\sigma}\right).G_\sigma\left(\frac{x-x_i}{\sigma}\right)\right] dx\\ &-\log\left[\frac{1}{N^2}\sum_{i=1}^{N}\sum_{j=1}^{N}G_{\sigma\sqrt{2}}(x_j-x_i)\right] = -\log[\hat{V}_2(X)],\end{aligned} \tag{2.25}$$

where $\hat{H}_2$ is the estimation for $H_2$ and

$$\hat{V}_{2,\sigma}(X) = \frac{1}{N^2}\sum_{i=1}^{N}\sum_{j=1}^{N}G_{\sigma\sqrt{2}}(x_j-x_i). \tag{2.26}$$

The integral in expression (2.25) was calculated by using the fact that the result of the product between two Gaussian functions is another Gaussian computed at the difference of the arguments and whose variance is the sum of the variances of the two original Gaussians. By calculating expression (2.26), which can be directly estimated from the available samples in a non-parametric fashion, one can estimate expression (2.25). This result is extremely useful because it bypasses the need to estimate PDFs, using only the pairs of the already available samples. The value of $\sigma$ is important to make the estimation of the entropy meaningful, due it's role in expression (2.24). It is common in the literature to obtain it by using Silvermans rule,

$$\sigma_{opt} = \sigma_X(4N^{-1}(2d+1)^{-1})^{\frac{1}{d+4}}, \tag{2.27}$$

where $N$ is the number of samples, $d$ is the data dimensionality and $\sigma_X$, the data standard deviation. Although expression (2.27) is sufficient for most of our applications, it is also possible to fix the value of $\sigma$, as exemplified by the authors of [7], in order to avoid numerical difficulties that can appear due to the use of Silvermans rule (e.g., non-convergence of the estimations due to values of $\sigma_X$ that are either too large of too small).

## 2.6 MATRIX-BASED ESTIMATORS FOR RÉNYI'S ENTROPY

As mentioned in Section 2.5, the estimators derived for the quadratic Rényi entropy greatly simplifies the estimation of the information quantities. However, these estimators can become inefficient when used in applications with a large amount of samples due to the increase of their computational cost [9]. To circumvent this issue, recent works have used the matrix-based Rényis $\alpha$-order entropy functional, proposed in [9], to estimate these information quantities directly from the data, without explicitly evaluating its underlying PDF. Let $\mathbf{X} = \{\mathbf{x}_i\}_{i=1}^N$, $\mathbf{x}_i \in \mathcal{X}$, be a set of independent and identically distributed (iid) samples of $X$ (e.g., a mini-batch). The Gram matrix $\mathbf{K} \in \mathbb{R}^{N\times N}$ is obtained through the evaluation of a real valued positive definite kernel $\kappa : \mathcal{X} \times \mathcal{X} \mapsto \mathbb{R}$ on all the pairs of samples from $\mathbf{X}$, i.e. $(\mathbf{K})_{ij} = \kappa(\mathbf{x}_i,\mathbf{x}_j)$. Then, $\mathbf{K}$ is normalized in order to create a matrix $\mathbf{A}$ such as $(\mathbf{A})_{ij} = \frac{1}{N}\frac{(\mathbf{K})_{ij}}{\sqrt{(\mathbf{K})_{ii}(\mathbf{K})_{jj}}}$ and $tr(\mathbf{A}) = 1$, which results in the matrix-based Renyis $\alpha$-order entropy functional

$$S_\alpha(\mathbf{A}) = \frac{1}{1-\alpha}\log_2\left[\sum_{i=1}^N \lambda_i(\mathbf{A})^\alpha\right], \tag{2.28}$$

where $\lambda_i(\mathbf{A})$ is the $i$-th eigenvalue of $\mathbf{A}$. Also in [9], a matrix-based estimation of the joint entropy between two RVs $H_\alpha(X,Y)$ was defined as

$$S_\alpha(\mathbf{A},\mathbf{B}) = S_\alpha\left[\frac{\mathbf{A}\circ\mathbf{B}}{tr(\mathbf{A}\circ\mathbf{B})}\right], \tag{2.29}$$

where $\circ$ denotes the Hadamard product and the matrix $\mathbf{B}$ is obtained analogously to $\mathbf{A}$, but given $\mathbf{Y} = \{\mathbf{y}_i\}_{i=1}^N$ samples of the targeted layer from the same realizations of $\mathbf{X}$. Using expressions (2.28) e (2.29), it is possible also obtain a matrix-based estimation for the MI $I(X,Y)$,

$$I_\alpha(\mathbf{A},\mathbf{B}) = S_\alpha(\mathbf{A}) + S_\alpha(\mathbf{B}) - S_\alpha(\mathbf{A},\mathbf{B}), \tag{2.30}$$

and for the conditional entropy $H(X|Y)$,

$$S_\alpha(\mathbf{A}|\mathbf{B}) = S_\alpha(\mathbf{A},\mathbf{B}) - S_\alpha(\mathbf{B}). \tag{2.31}$$

The main advantage of the estimators above is that their computation is very efficient when compared to those derived in 2.5 because they are matrix-based. However, it is worth noting that the larger the dimensions of $\mathbf{X}$ and $\mathbf{Y}$, the higher will be the computational costs of the estimation process due to the larger amount of eigenvalues required for it.

### 2.6.1 Extension of the Matrix-Based Estimators for CNNs

The matrix-based estimators in expressions (2.28), (2.29), (2.30) and (2.31), although adequate for densely connected NNs, cannot be directly applied to CNNs. This limitation is due to the fact that the $\chi$ FMs inside a given convolutional behave as $\chi$ information sources for the following layer [10]. To extend their use to CNNs, the estimators proposed in [9] were extended to the ones in [10] in order to estimate the multivariate mutual information (MMI) between a single RV and a group of RVs (e.g. the input of a CNN and the FMs of one of its layers). Let $\{\mathbf{s}_i = (\mathbf{x}_1^i, \ldots, \mathbf{x}_C^i)\}_{i=1}^N$, $C \geq 2$, be a collection of $N$ samples obtained from the same realization containing $C$ measurements $\{\mathbf{x}_p \in \mathcal{X}_p\}_{p=1}^C$ each and $\{\kappa_p : \mathcal{X}_p \times \mathcal{X}_p \mapsto \mathbb{R}\}_{p=1}^C$, positive-definite kernels. A matrix-based analog to Rényis $\alpha$-order joint entropy among $C$ RVs is defined as

$$S_\alpha(\mathbf{A_1},\ldots,\mathbf{A_C}) = S_\alpha\left[\frac{\mathbf{A_1}\circ\cdots\circ\mathbf{A_C}}{tr(\mathbf{A_1}\circ\cdots\circ\mathbf{A_C})}\right], \tag{2.32}$$

where $\{(\mathbf{A_p})_{ij} = \kappa_p(\mathbf{x}_p^i,\mathbf{x}_p^j)\}_{p=1}^C$. Let $\{T^p\}_{p=1}^C$ be the RVs associated with the $C$ FMs in a CNN's convolutional layer. The MMI between these FMs and the input $X$ is given by

$$I(X;\{T^1,\ldots,T^C\}) = H(X) + H(T^1,\ldots,T^C) - H(X,\{T^1,\ldots,T^C\}). \tag{2.33}$$

By inspecting expressions (2.28), (2.32) and (2.33), it is immediate that the value of $I(X;\{T^1,\ldots,T^C\})$ in a mini-batch of size $N$ can be estimated with

$$\begin{aligned} I_\alpha(\mathbf{B},\{\mathbf{A_1},\ldots,\mathbf{A_C}\}) = {} & S_\alpha(\mathbf{B}) + S_\alpha\left[\frac{\mathbf{A_1}\circ\cdots\circ\mathbf{A_C}}{tr(\mathbf{A_1}\circ\cdots\circ\mathbf{A_C})}\right] \\ & - S_\alpha\left[\frac{\mathbf{A_1}\circ\cdots\circ\mathbf{A_C}\circ\mathbf{B}}{tr(\mathbf{A_1}\circ\cdots\circ\mathbf{A_C}\circ\mathbf{B})}\right], \end{aligned} \tag{2.34}$$

where $\mathbf{B}$ and $\mathbf{A_1},\ldots,\mathbf{A_C}$ denote the Gram matrices evaluated on the input tensor and $C$ FM tensors, respectively.

By estimating the MMI between these RVs using expression (2.34), the authors of [7] measured the amount of information about $X$ that was captured by all the FMs inside the bottleneck layer. They also used the partial information decomposition (PID) framework to understand how the redundancy and synergy between different FMs evolved during training. According to this framework, the MMI $I(X;\{T^i,T^j\})$ between the input $X$ and the pair of FMs $T^i$ and $T^j$ can be written as the sum of four nonnegative IT components: the unique information associated with $T^i$ and $T^j$, redundancy and synergy. The first two, denoted by $\mathbf{Unq}(X;T^i)$ and $\mathbf{Unq}(X;T^j)$, measure the information about $X$ that can be exclusively provided by $T^i$ and $T^j$ respectively. Therefore, the higher the unique information associated with a given feature map, the more important it will be for the overall CNN's performance. The redundancy between the pair of FMs $T^i$ and $T^j$, denoted by $\mathbf{Rdn}(X,\{T^i,T^j\})$ measures the shared information about $X$ that can be provided by either $T^i$ or $T^j$. A larger redundancy between pairs of FMs in a given convolutional layer may suggest that it possesses more FMs than the necessary to capture the relevant information from inputs. This means that the representations created by this layer will be contaminated with redundant information. This is an issue because, by containing layers with more FMs than necessary, the CNN memory and computational costs increase needlessly. If, for example, the bottleneck layer of a CAE is substantially larger than the inputs dimension, it will store redundant information alongside relevant one. Therefore, the redundancy between its FMs will be high. On the other hand, if the bottleneck layer's size is equal to the inputs dimension, it will tend to store mostly relevant information while ignoring redundancies which, in turn, will decrease the redundancy between its FMs. Lastly, the synergy between a pair of FMs $T^i$ and $T^j$, denoted by $\mathbf{Syn}(X,\{T^i,T^j\})$, measures the information about $X$ provided by the combination of $T^i$ and $T^j$ (i.e., the information that cannot be captured by either $T^i$ and $T^j$ alone). A larger synergy between pairs of FMs in a given convolutional layer indicates that their combination contains significant non-redundant information from the inputs that cannot be obtained the individual FMs. This can be viewed as an indicator that eliminating a given FM may have a negative impact on the CNN's overall performance. Therefore, differently from redundancy, its preferable for the synergy between pairs of FMs in a given convolutional layer to be as large as possible.

All the aforementioned information quantities satisfy

$$I(X;\{T^i,T^j\}) = \mathbf{Syn}(X,\{T^i,T^j\}) + \mathbf{Rdn}(X,\{T^i,T^j\}) + \mathbf{Unq}(X;T^i) + \mathbf{Unq}(X;T^j), \quad (2.35)$$

$$I(X;T^i) = \mathbf{Rdn}(X,\{T^i,T^j\}) + \mathbf{Unq}(X;T^i) \quad (2.36)$$

and

$$I(X;T^j) = \mathbf{Rdn}(X,\{T^i,T^j\}) + \mathbf{Unq}(X;T^j). \quad (2.37)$$

From their definitions, it is evident that, during a CAE's training, it should maximize $\mathbf{Unq}(X;T^i)$, $\mathbf{Unq}(X;T^j)$ and $\mathbf{Syn}(X,\{T^i,T^j\})$ while minimizing $\mathbf{Rdn}(X,\{T^i,T^j\})$. However, direct estimation of these quantities is not currently possible. To circumvent this problem, the authors of [7] proposed two new information quantities to characterize intrinsic properties of the CNN layer representations: redundancy-synergy trade-off (**RST**) and weighted non-redundant information (**WNRI**). The $\mathbf{RST}_{ij}$ between $X$, $T^i$ and $T^j$ measures the redundancy-synergy trade-off between these RVs, being defined as

$$\begin{aligned}\mathbf{RST}_{ij} &= I(X;T^i) + I(X;T^j) - I(X;\{T^i,T^j\}) \\ &= Rdn(X;\{T^i,T^j\}) - Syn(X;\{T^i,T^j\}).\end{aligned} \tag{2.38}$$

During training, the **RST** between the input layer and a given convolutional layer can be estimated by averaging the sum of the $\mathbf{RST}_{ij}$ of each pair of FMs in the latter, which gives

$$\mathbf{RST} = \frac{2}{C(C-1)} \sum_{i=1}^{C} \sum_{j=i+1}^{C} \mathbf{RST}_{ij}. \tag{2.39}$$

Although a negative **RST** during training would be ideal, meaning that synergy dominates over redundancy, practical experimentation done so far have always shown positive values [7]. Therefore, in practical applications, it should be as close to zero as possible, indicating that, although redundancy dominates over synergy, it does so weakly. The $\mathbf{WNRI}_{ij}$ between $X$, $T^i$ and $T^j$, in turn, measures the amount of non-redundant information about $X$ that is captured by the pair of FMs $T^i$ and $T^j$, being defined as

$$\mathbf{WNRI}_{ij} = 2.I(X;\{T^i,T^j\}) - I(X;T^i) - I(X;T^j). \tag{2.40}$$

As the **RST**, the **WNRI** between the input layer and a given convolutional layer can be estimated by averaging the sum of the $\mathbf{WNRI}_{ij}$ of each pair of FMs in the latter, which results in

$$\mathbf{WNRI} = \frac{2}{C(C-1)} \sum_{i=1}^{C} \sum_{j=i+1}^{C} \mathbf{WNRI}_{ij}. \tag{2.41}$$

Expressions (2.39) and (2.41) are important because they grant access to the insights provided by both redundancy and synergy without their explicit estimation. It is evident that, to maximize the amount of non-redundant information present in a given convolutional layer, the **WNRI** between pairs of its FMs should increase during training. On the other hand, in order to prevent redundancy from limiting the capacity of these FMs to store non-redundant information, the **RST** between pairs of them (always positive in practice) should be kept as close to zero as possible during training. It is also worth noting that, by adding expressions (2.38) and (2.40), we obtain

$$\mathbf{RST}_{ij} + \mathbf{WNRI}_{ij} = I(X; \{T^i, T^j\}) \therefore \frac{\mathbf{RST}_{ij}}{I(X; \{T^i, T^j\})} + \frac{\mathbf{WNRI}_{ij}}{I(X; \{T^i, T^j\})} = 1, \tag{2.42}$$

which suggests that, from expressions (2.39) and (2.41), we obtain

$$\mathbf{RST}_{\%} = \frac{2}{C(C-1)} \sum_{i=1}^{C} \sum_{j=i+1}^{C} \frac{\mathbf{RST}_{ij}}{I(X; \{T^i, T^j\})} \tag{2.43}$$

and

$$\mathbf{WNRI}_{\%} = \frac{2}{C(C-1)} \sum_{i=1}^{C} \sum_{j=i+1}^{C} \frac{\mathbf{WNRI}_{ij}}{I(X; \{T^i, T^j\})}. \tag{2.44}$$

Expressions (2.43) and (2.44) show the percentages that the **RST** and **WNRI** account for the MMI in each pair of FMs, respectively. From expression (2.42), it is clear that

$$\mathbf{RST}_{\%} + \mathbf{WNRI}_{\%} = 1, \tag{2.45}$$

which means that any increase in $\mathbf{RST}_{\%}$ during training implies in the decrease of $\mathbf{WNRI}_{\%}$ and vice-versa.

## 2.7 RELATED WORKS: UNDERSTANDING AES AND CAES WITH INFORMATION THEORETIC CONCEPTS

The current scarcity of proven, well established and widely accepted methods for designing DNNs and evaluating their performance often leads them to be labeled as “black boxes” [2]. This scenario severely hinders the use of DNNs in a wider range of applications because, without said methods, it is not possible to optimally design them and evaluate the quality of their training. In order to tackle this issue, the authors of [6] used the ITL framework to study a SAE architecture. By doing so, they aimed to use the informational quantities' estimators defined in 2.5 to understand their behavior during these NNs' training. As discussed in 2.4, the SAEs are feedforward DNNs whose basic learning mechanism is the backpropagation, where the inputs are propagated from the input layer to the output one and the errors are backpropagated in the reverse direction. Both propagations are unidirectional and depend exclusively on the previous variables, therefore obeying the Markov assumption and forming a Markov chain. This translates in existence of two different data processing inequalities (DPIs) in feedforward DNNs:

$$I(X,R_1) \geq I(X,R_2) \geq \cdots \geq I(X,R_L) \tag{2.46}$$

and

$$I(\delta_L,\delta_{L-1}) \geq I(\delta_L,\delta_{L-2}) \geq \cdots \geq I(\delta_L,\delta_1), \tag{2.47}$$

where $R_1, R_2, \ldots, R_L$ are successive hidden layer representations from the first hidden layer to the output one and $I(\delta_L,\delta_{L-1}), I(\delta_L,\delta_{L-2}), \ldots, I(\delta_L,\delta_1)$ are errors from the output layer to the first hidden one [6]. Noteworthy is the fact that expression (2.46) indicates that the successive layer representations in the AE's encoder should form a simple Markov chain [6, 8]. In the case of SAEs' architectures such as the one depicted in 2.9, due to its symmetric structure, expressions (2.46) and (2.47) will translate into the next two DPIs:

$$I(X,E_1) \geq \ldots I(X,E_Q) \text{ and } I(Y,D_1) \geq \ldots I(Y,D_Q) \tag{2.48}$$

and

$$I(X,Y) \geq I(E_1,D_1) \cdots \geq I(E_Q,D_Q). \tag{2.49}$$

Expressions (2.46) and (2.48) essentially state that information can only decrease due to its flow through the DNN. In turn, expression (2.49) indicates that a monotonically non-increasing trend (as the number of layers increases) of the MI between the layer output and their "symmetric" counterparts is to be expected. This means that MI cannot be gained when the original information is processed in one of the SAE's deeper layers, which imposes an upper limit to the number of layers in practical situations. The equality on expression (2.49) will only hold if, for example, $E_1$ and $D_1$ are the sufficient statistic with respect to $\{X,Y\}$ [6].

The authors of [6] used the information plane to visualize the information quantities' evolution during the training process. Among the various conclusions drown from the work, the authors stated that the results obtained validated the DPI in expression (2.49), which indirectly corroborates the appropriateness of the non-parametric estimators used. They also suggested that, since information is lost while it flows through a DNN, the existence of an upper bound on the number of layers in DNNs that achieves optimal performance can be expected. By using the information quantities, a designer could evaluate how deep a DNN can be for a given application without loosing so much information that it hinders its representation capacity.

Motivated by the positive results obtained in [6], its authors extended the proposed methodologies to CNNs in [7], albeit it was necessary to utilize the multivariate extension of matrix-based Renyis entropy functional defined in [10]. They also used new information quantities, **RST** and **WNRI**, whose definition and derivation was detailed in 2.5. Based on the results obtained, they concluded that the most promising application concerning the estimators proposed in [7] is to determine and discard kernels that are less important.

Worth mentioning is the fact that the authors of [6] and [7] noted the issue of adequately sizing DNNs. As previously mentioned, due to the lack of fixed rules or widely acknowledged methods currently available to this end, the optimal design of DNNs topology is still a challenge that prevents their use in a wider array of practical applications. In the case of CNNs, they emphasized that the results obtained did not allow the specification of a rule to determine the optimal number of filters in a given convolutional layer. However, the results obtained in the aforementioned works suggest that the proposed methodologies could be an useful tool to address these issues.

## 2.8 METHOD FOR AUTOMATIC SIZING OF THE BOTTLENECK LAYER DIMENSION

As previously mentioned, there is a lack of methods to determine $size(Z)$, as well as to evaluate its adequacy. In order to address this issue, the authors of [11] proposed a method for automatic estimation of the optimal value of $size(Z)$, denoted by $D_{opt}$, in a densely connected SAE architecture. In order to do so, they used the efficient matrix-based estimators proposed in [9] to estimate the bottleneck layer's entropy, $H(Z)$, and the MI between input and output of the SAE, $I(X;Y)$, during its training process. The SAE's training was done via back-propagation and stochastic gradient descent (SGD) by minimizing the MSE between the DNN's predictions and its inputs. Concurrently, the estimated values of $H(Z)$ and $I(X;Y)$ are compared to each other. The goal of this comparison was to verify if $H(Z)$ converged to $I(X;Y)$ during the SAE's training.

The method is fundamentally based on the fact that the flow of information through the DNN can only cause its loss. This is due to the fact that the SAE is a feedforward DNN and, consequently, the existence of a data processing inequality (DPI) that guarantees said fact can be demonstrated [6]. Consequently, the amount of information (i.e., entropy) in a given hidden layer cannot be larger than $I(X;Y)$ at any given point during training. Therefore, $I(X;Y)$ is the upper bound for $H(Z)$. From a purely theoretical standpoint, it is expected that $H(Z) = I(X;Y)$ in these DNNs. However, in practice, this is not the case at the beginning of a SAE's training because their weights are initially randomly set. Only after enough training is performed that $H(Z) = I(X;Y)$ will hold for a well-trained SAE. If these two information quantities converge during the SAE's training process, it means that $size(Z)$ is large enough to allow the DNN to adequately capture the statistical information of its inputs. On the other hand, if they do not converge, it means that $size(Z)$ is too small to do so, which in turn will hinder the DNN's performance. Thus the method's goal is to make $size(Z) \simeq D_{opt}$, that is, make the bottleneck as small as possible while enabling the convergence between $H(Z)$ and $I(X;Y)$. The proposed method is as follows. First, an initial value of $size(Z)$ is set (e.g., a fraction of the input's size), and the SAE's training starts. After training with a certain amount of mini-batches, the values of $H(Z)$ and $I(X;Y)$ are estimated, averaged accordingly and compared. If their difference eventually becomes smaller than a given threshold (i.e., they converge), it means that the initial value of $size(Z)$ is enough to allow the code to adequately capture the important information contained in the DNN's input. In order to verify if a lower value of $size(Z)$ would also be enough to do so, the SAE's training is interrupted, the value of $size(Z)$ is reduced, and the training process restarts. However, if $H(Z)$ and $I(X;Y)$ do not converge during the SAE's training, it means that the initial value of $size(Z)$ is insufficient to allow the code to adequately capture the important information contained in the DNN's input. To verify if a higher value of $size(Z)$ would be able to do so, the SAE's training is interrupted, the value of $size(Z)$ is increased and the NN's training restarts. This process is repeated until $D_{opt}$ is estimated.

This approach is heavily reliant on the use of $H(Z)$ works as a KPI for the densely connected SAE studied. The authors argue that this can be done because, when the SAE is well-trained in terms of generalization to reconstruct its input through the generated codes, the bottleneck layer must contain most of (ideally, all of) the useful information that define the DNN's input. In other words, after the training process, $H(Z) \simeq I(X;Y)$. If $H(Z)$ and $I(X;Y)$ did not converge by the end of the SAE' training, it means that $size(Z)$ isn't large enough to contain all the information necessary to reconstruct inputs through the codes generated, which will negatively impact the DNN's overall performance. By using $H(Z)$ as a KPI, the results obtained showed that the proposed method was able to estimate $D_{opt}$ for the studied SAE.

However, the results obtained in [11] cannot be directly extrapolated to CAEs. This is due to the fact that $H(Z)$ alone is an insufficient KPI for these DNNs. That is because, if there are $K$ kernels inside a convolutional layer, its representation of an input will be stored in $K$ FMs, each one characterizing particular properties of the input (e.g., edges, contours). This suggests that the amount of information that the convolutional layer gains from its input is preserved in $K$ information sources. Among these, some will likely contain a mixture of relevant and redundant information about the input, others may contain mostly redundant information and others, no relevant information at all [7]. This means that, among the FMs inside a the bottleneck layer, there may be some that store mostly redundant and/or uninformative information. There may be also pairs of FMs with substantial redundancy (as defined in Section 2.6) between them.

A possible indicator of this phenomena is the continuous evolution of the CAE's bottleneck's **WNRI** after the convergence between $H(Z)$ and the MMI. Defined in Section 2.5, the **WNRI** of a given layer can be interpreted as a measurement of the amount of information about the input provided by the pairs of its FMs (i.e., the information that cannot be captured by the pairs' individual FMs). As the CAE's training process takes place, it is reasonable to expect that its layers' **WNRI** will increase alongside $H(Z)$. If, however, the bottleneck's **WNRI** continues to increase even after the between $H(Z)$ and the MMI, it may suggest indicate the CAEs effort to maximize the amount of non-redundant information inside its codes. If, on the other hand, its bottleneck's **WNRI** converges, it may be an indicator of the interruption of this effort. Nevertheless, this scenario may be problematic because, even if convergence $H(Z)$ is achieved during the training process, it is meaningless if the CAE's codes are filled with redundancies. After all, this goes against one of the primary goals of these CNNs, which is exactly to create representations of their inputs with the smallest amount of redundancies and noise as possible. As will be shown in 3, differently from what was verified in [11], changing $size(Z)$ is often not enough to determine $D_{opt}$ due to the structural peculiarities of the CAEs.

CHAPTER 3

# METHOD AND EXPERIMENTS

In this chapter, we will discuss the results yielded by the experiments performed using the CAE architectures and the setup described in Section 3.2. These results consist on curves depicting the evolution of the information quantities defined in Section 2.5 inside Conv_1, Conv_2, Conv_3 and Z of the CAEs described in Table 3.1.

## 3.1 INITIAL EXPERIMENTS: MNIST DATASET

These initial experiments were conducted with the goal of reproducing the results presented in [11]. By doing so, we aimed to more thoroughly comprehend them, as well as the proposed method for the automatic sizing of the bottleneck's dimension as a whole. We also aimed to validate our implementations of the estimators proposed in [9]. All the codes used in the experiments were written in Python, and were structured in order to reproduce the pseudo-code presented in [11]. For this purpose, the Keras library, which is an API based on Tensorflow that provides tools that simplify the design and training of NNs, was used. Using $H(Z)$ as the KPI for the SAE's performance, the experiments aimed to determine $D_{opt}$ by making $size(Z)$ as close to it as possible during the training of an SAE with four hidden layers, as explained in Section 2.8. The widely known dataset MNIST was used, and its images were randomly subdivided into 400 mini-batches of size 100 for training [11].

As done in [11], the matrix-based estimators defined in Section 2.5 were used in the experiments. Gaussian kernels were utilized, whose bandwidths were selected by applying Silverman's rule. As explained in Section 2.8, after each epoch, the bottleneck's entropy $H(Z)$ and the MI between input and output were estimated and compared. If the absolute difference between them was less than a fixed threshold, the training was interrupted, $size(Z)$ was reduced, a new model was created and the training process restarted. If the absolute difference between

$H(Z)$ and the MI was larger than the fixed threshold, $size(Z)$ was increased, a new model was created and the training process restarted. The value of $10^{-2}$ was fixed for the aforementioned threshold. Two control variables were used in order to set the upper and lower bonds for the value of $size(Z)$, and the value of $size(Z)$ was set to average. Initially, the upper bond was arbitrarily set to 25% of the input size and the lower bond, to 1. So, for example, if the input's size was 400, the upper bond for $size(Z)$ was initially set to 100, meaning that it was assumed that the other 300 dimensions contained mostly redundancies and noise. By initially setting the lower bond at 1, it was assumed that, at minimum, all the non-redundant information about the input occupied only one of its 400 dimensions. In this example, the value of $size(Z)$ would be initially assumed to be 50 (i.e., the average between the upper and lower bonds). The value of these variables (and, consequently, of $size(Z)$) was updated throughout the experiment, in order to progressively reduce the range of possible values for $size(Z)$. In this example, if it was determined that the input's non-redundant information occupied more than 100 of its dimensions (i.e., $size(Z)$ needed to be larger than 25% of the input's size), the upper bond for $size(Z)$ was increased, and vice-versa. In turn, if it was determined that the input's non-redundant information occupied more than one of its dimensions (i.e., $size(Z)$ needed to be larger than 1), the lower bond for $size(Z)$ was increased, and vice-versa. The experiments were concluded when the values of these control variables converged.

Due to the incompleteness of the experimental information registered in [11] (e.g., how the kernels' normalization was conducted, condition for the interruption of the NN's training), several adaptations of the pseudo-code provided were necessary for the implementation of the code used in the experiments. Due to these adaptations, the adapted code consistently determined the value of $size(Z)$ between 18 and 21, which somewhat diverged from the fixed value of $D_{opt} = 19$ determined for the MNIST in [11]. Despite these divergences, considering the goal of these initial experiments, the implemented code's estimations for $D_{opt}$ were satisfactorily close to the value estimated in [11]. The results yielded by the adapted code thereby corroborate the ones obtained in [11], indicating that the proposed method is a potentially useful tool for the automatic sizing of $size(Z)$. They also further validate the matrix-based estimators defined in Section 2.8.

## 3.2 EXPERIMENTAL PROCEDURES: WIKI-ART DATASET

As explained in Section 2.7, the lack of methods and rules for optimally designing DNNs is an issue that prevents their use in a wider array of practical applications. Given the positive results yielded by the method proposed in [11], corroborated by the ones reported in Section 3.1, it is pertinent to investigate if it can be extended to a wider variety of DNNs in order to tackle the aforementioned issue. Based on the multivariate extension of matrix-based Renyis entropy functional defined in 2.5, we aimed to investigate the possibility of extending the method proposed in [11] to symmetric CAEs.

The real-world dataset “Wiki-Art: Visual Art Encyclopedia” [16] was selected for training and evaluation of the CAEs used in the experiments. It has a size of 37 GB and consists of 72.619 RGB images of various works of art, with distinct qualities and dimensions, distributed between 14 different classes according to their classification (e.g, abstract and animal paintings), such as the ones depicted in Figure 3.1. This dataset was chosen due to the substantial differences between the themes of its images and the peculiarities of their respective author’ painting styles, which make it very rich from an information standpoint. Before the experiments, the images contained in the dataset were shuffled and partitioned so that 75% of them were used for training and 25%, for testing. For most experiments, the conversion from RGB to grayscale and resizing of the images to 128x128 pixels was performed, without any other image preprocessing. In the experiments in which the conversion from RGB to grayscale and resizing of the images to 128x128 pixels was not performed, no image preprocessing was conducted besides the resizing of the images to 128x128 pixels. The codes were written in Python using the Keras API and various Python libraries.

Multiple symmetric CAEs were used in the experiments. All of them consisted of seven hidden convolutional layers: three were located in the encoder, the bottleneck one and three in the decoder. Table 3.1 contains the amount of FMs inside the bottleneck (Z), first (Conv_1), second (Conv_2) and third (Conv_3) convolutional layers of the CAEs’ encoders. All layers used the rectified linear unit (ReLU) function and had stride 1. After each layer in the encoder, a max pooling layer with stride 2 was used. This implies that the FMs inside a given layer have half the size of its predecessor’s FMs. Since all inputs were $128 \times 128$ images, the dimensions of the FMs inside Conv_1, Conv_2 and Conv_3 of all the CAEs were $64 \times 64$, $32 \times 32$ and

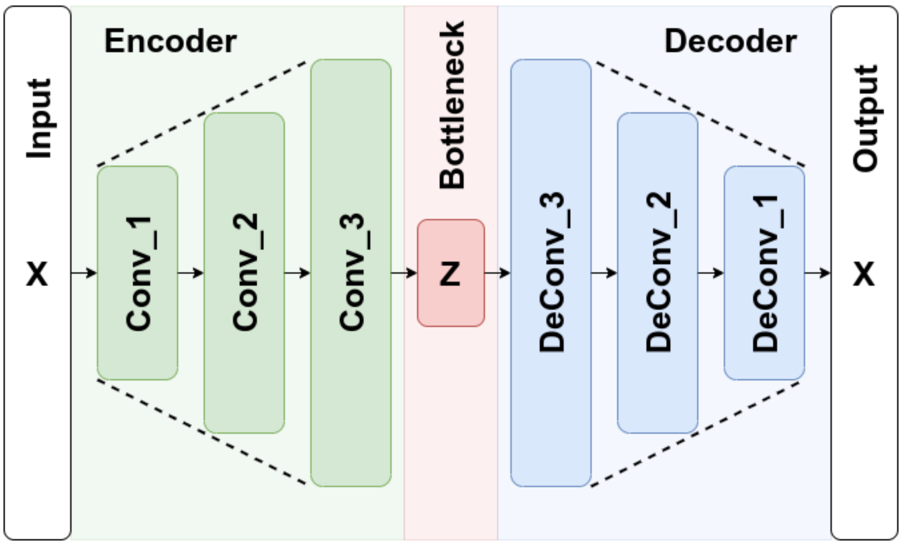
**Figure 3.1.** Images extracted from the “Wiki-Art: Visual Art Encyclopedia” dataset.

**Source:** Own authorship.

16 × 16, respectively. After the bottleneck layer and each layer in the decoder, an upsampling layer whose interpolation function could be selected was used. To investigate the effect of this selection on the evolution of the CAEs’ information quantities, two CAE architectures used the nearest neighbor (or simply nearest) function, while the others used the bilinear one. The nearest function selects the nearest pixel while ignoring the values of its neighbors, duplicating it to create new pixels as the image grows. The bilinear function takes a weighted average of the four neighborhood pixels to calculate its final interpolated value, which produces an image that is smoother than the original one. It yields better results than the nearest function, but has a higher computational cost [15].

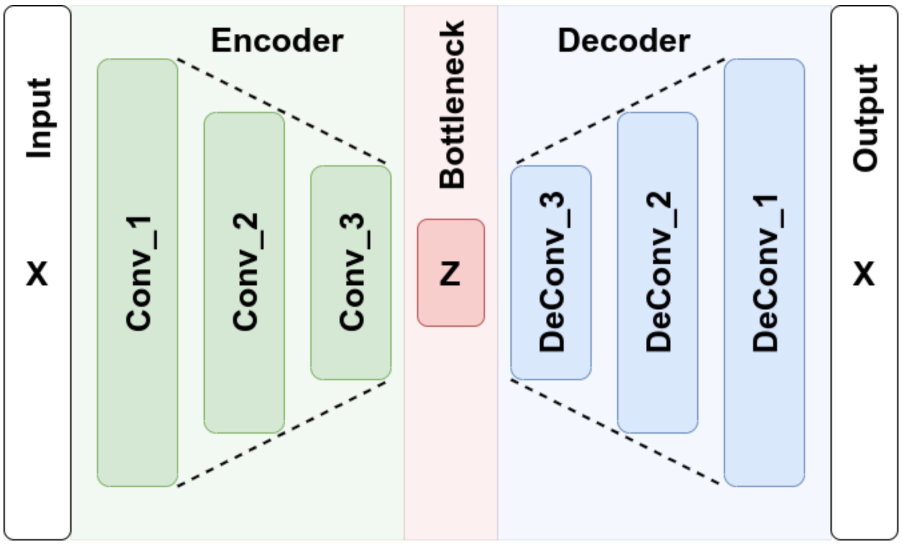
The CAE architectures were separated in two groups: CAE_1 and CAE_2. All the CAEs in group CAE_1, whose structure is depicted in Figure 3.2, have 16 FMs in the layer Conv_1, 32 in Conv_2 and 64 in Conv_3. In turn, all the CAEs in group CAE_2, whose structure is depicted in Figure 3.3, have 64 FMs in the layer Conv_1, 32 in Conv_2 and 16 in Conv_3. Both

**Figure 3.2.** Illustration of the architectures in CAE_1.



**Source:** Own authorship.

**Figure 3.3.** Illustration of the architectures in CAE_2.



**Source:** Own authorship.

Figure 3.2 and 3.3 highlight the direction in which the number of FMs inside the convolutional layers decreases. The differences between the architectures from the same group reside on their respective bottleneck's sizes and the interpolation function used in their decoders.

**Table 3.1.** Specifications of the CAEs' architectures

| | Conv_1 | Conv_2 | Conv_3 | Z | Interpolation |
|---|---|---|---|---|---|
| CAE_1_4 | 16 | 32 | 64 | 4 | bilinear |
| CAE_1_6 | 16 | 32 | 64 | 6 | bilinear |
| CAE_1_7 | 16 | 32 | 64 | 7 | bilinear |
| CAE_1_8 | 16 | 32 | 64 | 8 | bilinear |
| CAE_1_8_N | 16 | 32 | 64 | 8 | nearest |
| CAE_2_8 | 64 | 32 | 16 | 8 | bilinear |
| CAE_2_12 | 64 | 32 | 16 | 12 | bilinear |
| CAE_2_12_N | 64 | 32 | 16 | 12 | nearest |

**Source:** Own authorship.

These CAEs were chosen to investigate the effects caused by the FMs' placement inside the convolutional layers, as well as the interpolation function selected, on the evolution of the information quantities defined in Section 2.5 during their training. The experiments were conducted using the Python 3 Google Computer Engine backend (GPU) with 83,49 GB of RAM and 166,83 GB of Disk. The CAEs were trained via SGD using the Adam optimizer and the MSE loss function. A mini-batch size of 128 was adopted, and the CAE was trained for 50 epochs in order to allow the information quantities' stabilization. For their estimation, we fixed $\alpha = 2$ and used the radial basis function (RBF) kernel to obtain the Gram matrices. The kernel size $\sigma$ was determined based on Silverman's rule of thumb $\sigma = h.N^{-1/(4+d)}$, where $N$ is the mini-batch's size, $d$ is the sample dimensionality, and $h$ is an empirical value selected experimentally. Similarly to [7], we pick $h = 5$. Differently from them, however, no vector rastering was done in order to preserve the spatial relationships between neighboring pixels.

## 3.3 EXPERIMENTAL RESULTS

The results regarding each one of the CAEs involved in the experiments will be addressed in a separated subsection. Throughout this Section, as done in the previous ones, we will refer to the bottleneck layer's entropy by $H(Z)$, to its dimension (i.e., the amount of FMs inside it) by $size(Z)$ and to the MMI between the CAE's input and output simply by MMI. Based on the results obtained, we will state a series of assumptions which we believe can serve as general criteria to evaluate if the training process of a given CAE was successful or not.

### 3.3.1 CAE_1_4

This architecture was used in two different experiments whose results are depicted in Figures 3.4, 3.5, 3.6 and 3.7. In both experiments, grayscale images were used, and the information quantities were estimated only once, at the end of each epoch. The convergence $H(Z)$ to a value lower than the MMI observed in Figure 3.4 indicates that $size(Z)$ is insufficient to adequately capture the informational content of the dataset. However, it is worth noting that, in experiment 1, the $H(Z)$ displays a monotonically increasing behavior until its convergence is reached. This phenomenon is frequently observed in the experiments where the CAE training was successful. In view of this frequency, we state the first assumption regarding a characteristic of well-trained CAEs:

**Assumption 1** *In successfully trained CAEs, the curve of $H(Z)$ will display a monotonically increasing behavior until its convergence is reached.*

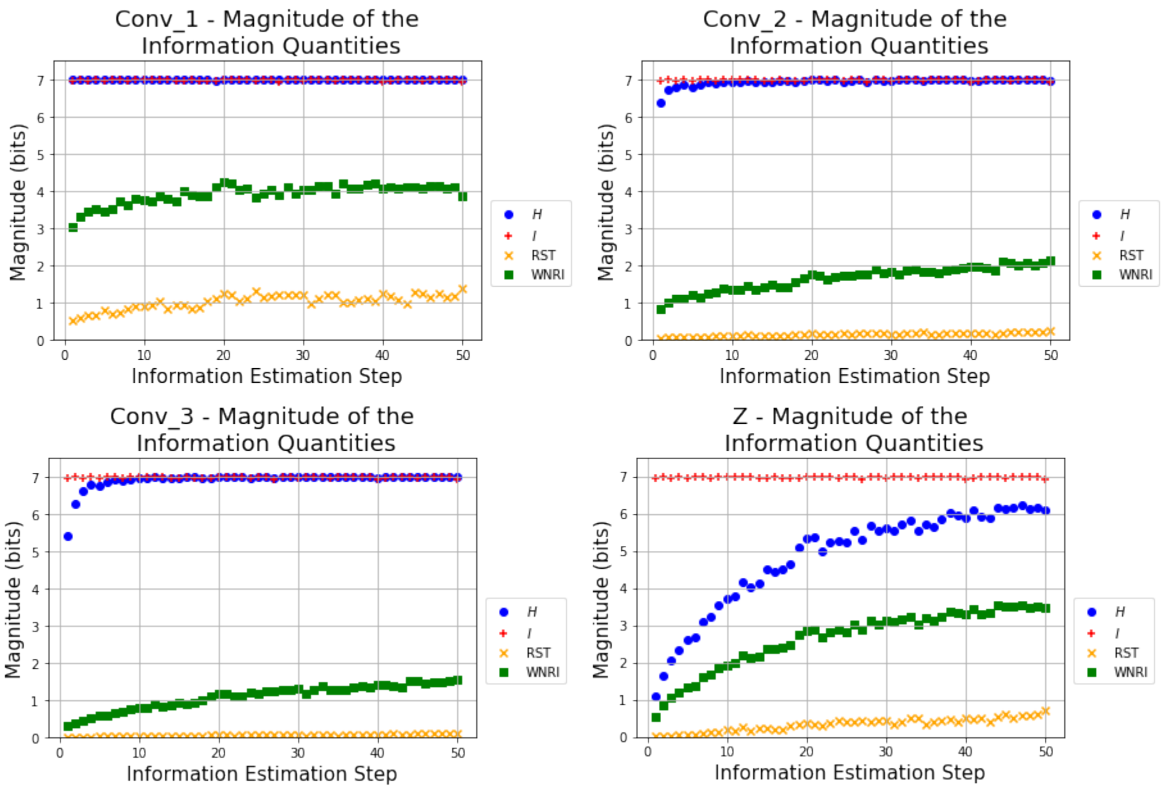
Another phenomenon frequently observed in experiments where the CAE was successfully trained can also be observed in 3.4. In such experiments, it can be verified that, in all the CAE's layers, the **WNRI** curve is always above the **RST** curve. One possible explanation for this phenomenon is the fact that, as discussed in 2.4, a well-trained CAE is expected to filter redundancies while retaining non-redundant information. In view of this frequency, we state the second assumption regarding a characteristic of well-trained CAEs:

**Assumption 2** *In successfully trained CAEs, the* **WNRI** *always stays above the* **RST** *curve.*

Despite the potential usefulness of Assumption 2, the computational costs associated with the estimation of **RST** and **WNRI** in the encoder's hidden layers are usually extremely high, specially when they contain a large amount of FMs. Therefore, in practical applications, the verification of 2 can be only possible for the bottleneck layer.

Figure 3.5 shows the evolution of $\mathbf{RST}_{\%}$ and $\mathbf{WNRI}_{\%}$ during experiment 1. It can be seen that, the deeper a given encoder layer is, the higher will be the value to which its $\mathbf{WNRI}_{\%}$ converges and, consequently, the lower will be the value to which its $\mathbf{RST}_{\%}$ converges. Also noteworthy is the fact that the $\mathbf{WNRI}_{\%}$ is considerably higher than the $\mathbf{RST}_{\%}$ at the end of epoch 1, when the training process has barely started to take place. We suggest that this somewhat counterintuitive behavior may be attributed to the random initialization of the CAE's weights and kernels before the start of the training process. Since the initial weights are drawn according to a probabilistic distribution (i.e., gaussian), they naturally tend to have little to no correlation to each other. This makes so that there is almost no redundant information between the FMs inside the encoder's layers and, therefore, their $\mathbf{WNRI}_{\%}$ is considerably higher than their $\mathbf{RST}_{\%}$. However, during the CAE's training, its weights and kernels increasingly reflect the statistical nature of the CAE's inputs, which include the redundancies between them. This, in turn, causes the decrease of $\mathbf{WNRI}_{\%}$ and the increase of $\mathbf{RST}_{\%}$ during the CAE's training.

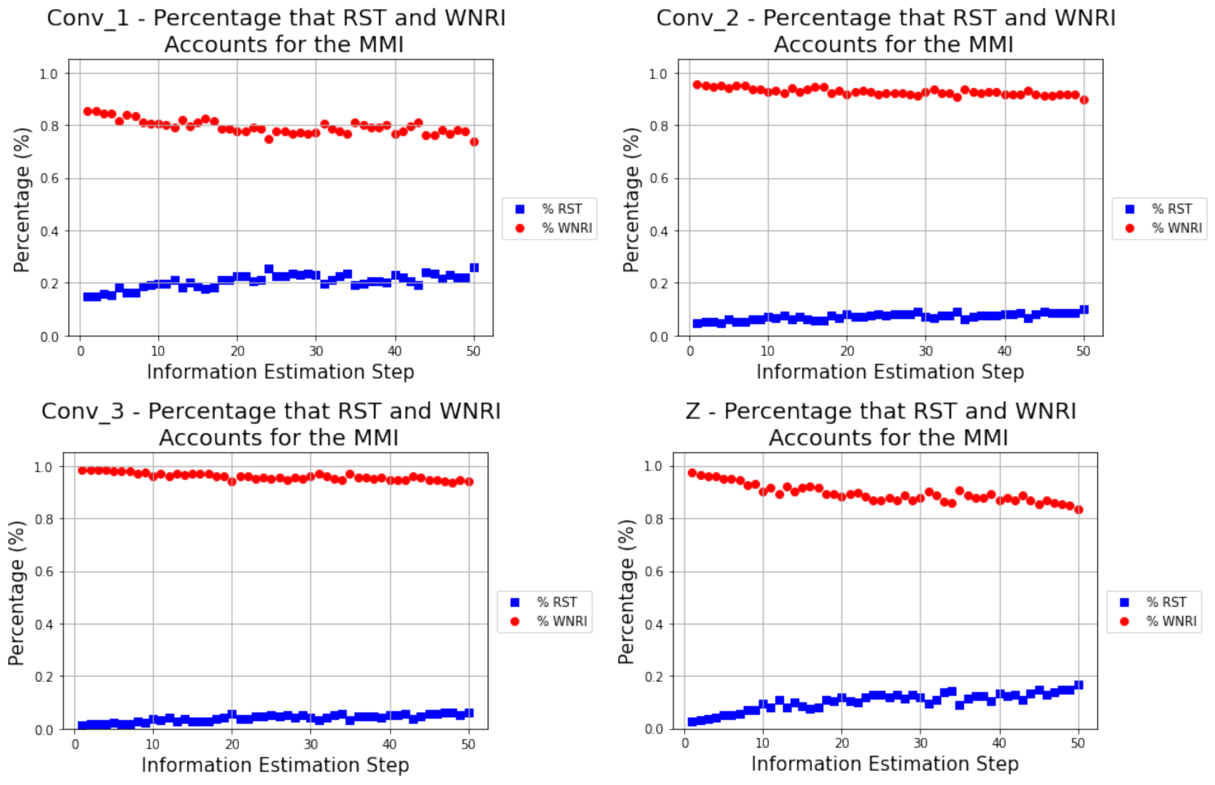
**Figure 3.4.** Evolution of the CAE_1_4's information quantities' magnitudes during experiment 1.



**Source:** Own authorship.

Using Assumptions 1 and 2 to evaluate the results yielded by experiment 2, it can be argued that the CAE's training was also unsuccessful. The curves in Figures 3.26 and 3.27 exemplify what can happen when the information quantities' curves' behavior differ from Assumptions 1 and 2. Not only $H(Z)$ didn't converge to the MMI, but both it and the bottleneck's **WNRI** displayed a decreasing behavior during a few epochs after epoch 22. Another interesting fact is that, unlike what is verified in successful experiments, the **RST** curve of the Conv_1 layer

**Figure 3.5.** Evolution of the CAE_1_4's $\mathbf{RST}_{\%}$ and $\mathbf{WNRI}_{\%}$ during experiment 1.

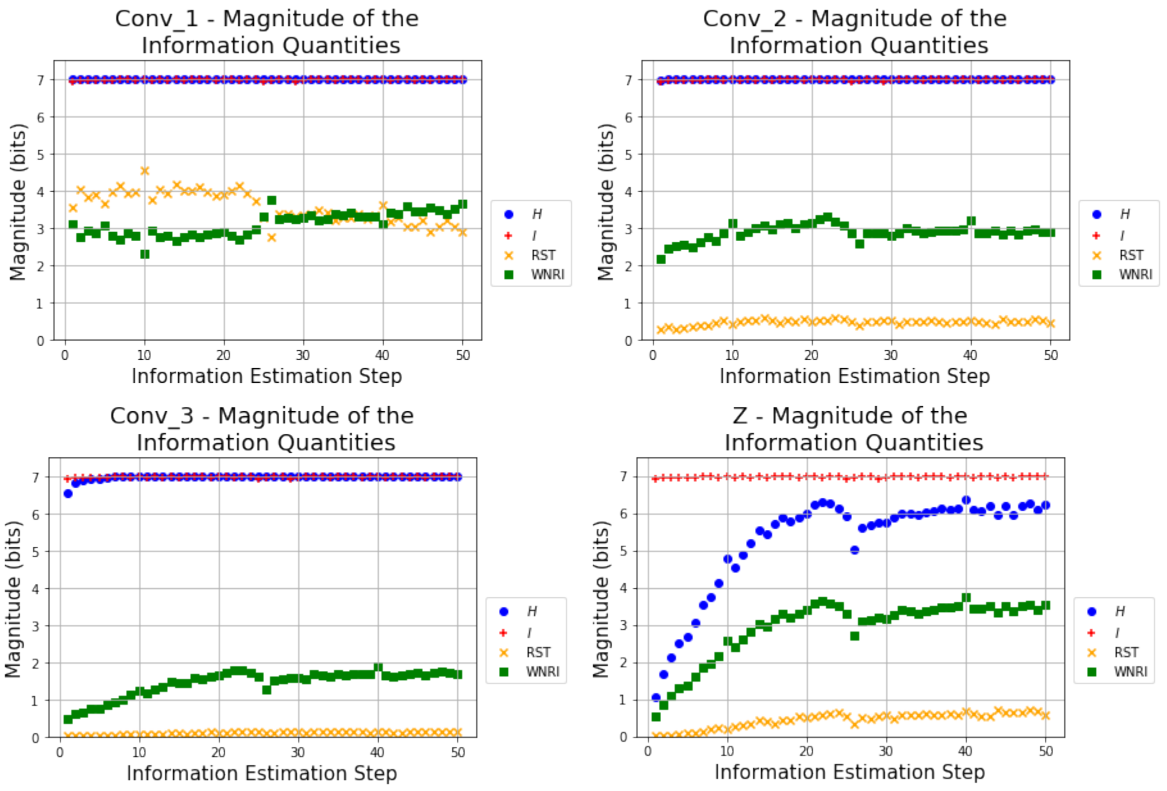


**Source:** Own authorship.

stays above the **WNRI** one until epoch 22. After this epoch, the **RST**'s and **WNRI**'s values rapidly change until both curves overlap and, from epoch 40 onward, the **WNRI** curve stays above the **RST** one, as expected. Moreover, the **WNRI** curves of the Conv_2 and of the Conv_3 layers also display a decreasing behavior after epoch 22. The curves depicted in 3.27 also display the aforementioned behavior.

These phenomena may suggest that there is a link between the hidden layers' information quantities behaviors. This is corroborated by the fact that, after epoch 20, the behavior of the information quantities of all the hidden layers started to display visible changes. Particularly, it is worth noting that, although only the information quantities of Conv_1 displayed an undesirable behavior throughout the training process until epoch 20, it had a negative impact on the evolution of the other layer's quantities. Moreover, it is worth noting that, from epoch 27 onward, after the **WNRI** and entropy curves of the bottleneck stopped decreasing, their behavior started to resemble that of their correspondents in Figure 3.4, even converging to approximately the same values by the end of the training process. This further corroborates the hypothesis that, in order to be successfully trained, the information quantities of the CAE's

**Figure 3.6.** Evolution of the CAE_1_4's information quantities' magnitudes during experiment 2.

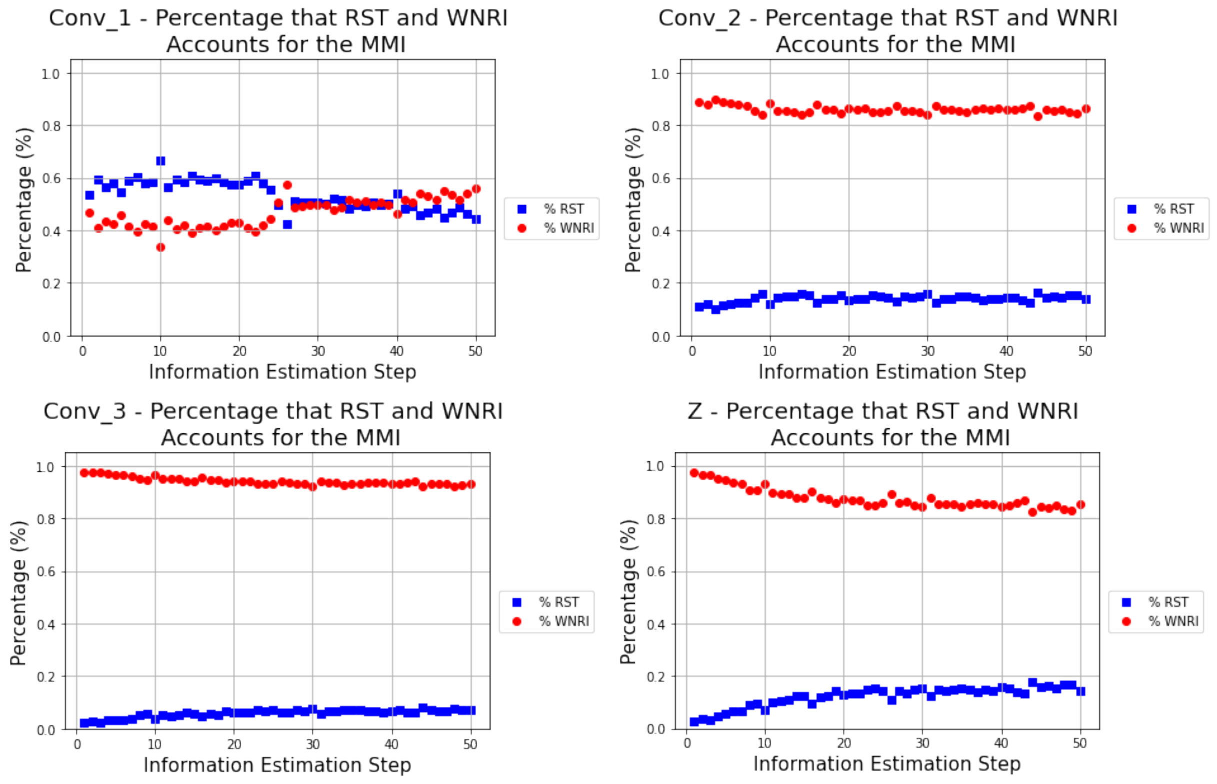


**Source:** Own authorship.

layers should obey Assumptions 1 and 2.

Lastly, the fact that the **WNRI**'s and entropy's curves of the bottleneck layer in both Figures 3.4 and 3.6 converged to the same value by the end of the training process also supports this hypotheses. It may suggest that, even if Assumptions 1 and 2 are not verified in all of the encoder's layers during the training, if trained for a sufficient amount of epochs, the CAE can adjust its parameters in order to modify its layers' information quantities such that they reflect these Assumptions. However, despite these observations, more experimentation with different CAE architectures and datasets are necessary in order to further support these hypotheses.

**Figure 3.7.** Evolution of the CAE_1_4's $\mathbf{RST}_{\%}$ and $\mathbf{WNRI}_{\%}$ during experiment 2.

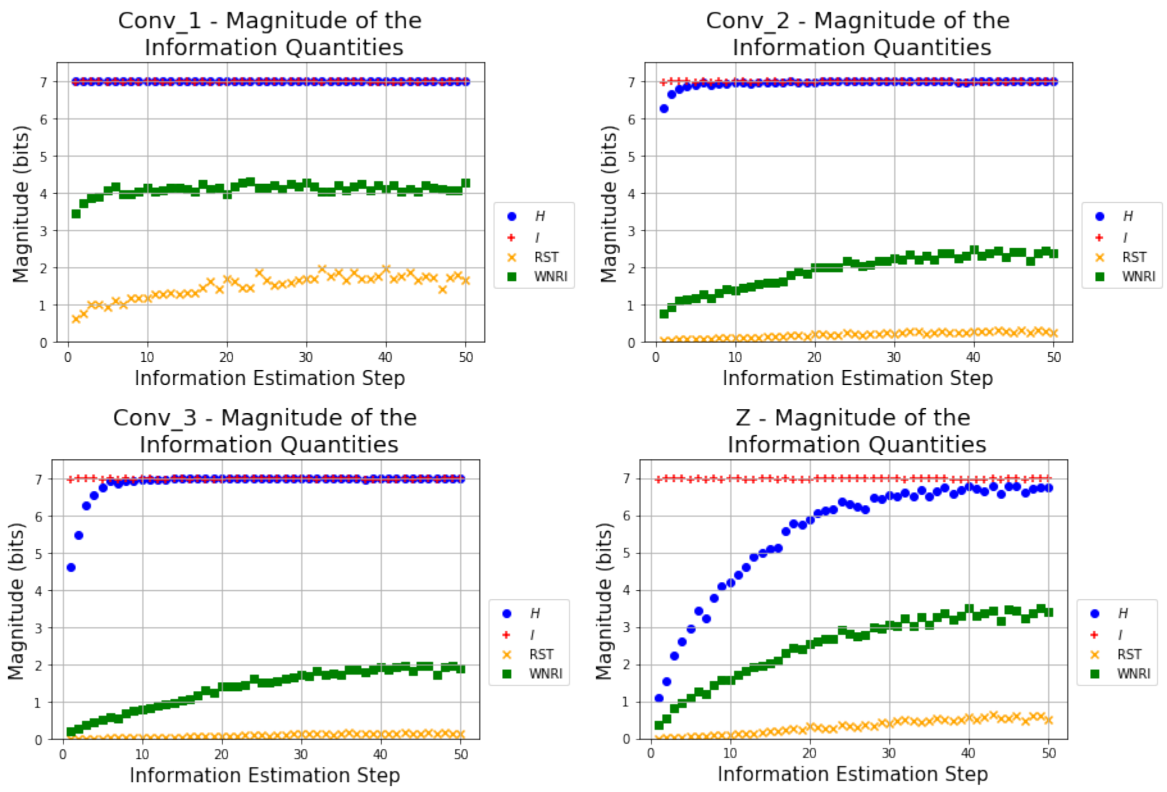


**Source:** Own authorship.

### 3.3.2 CAE_1_6

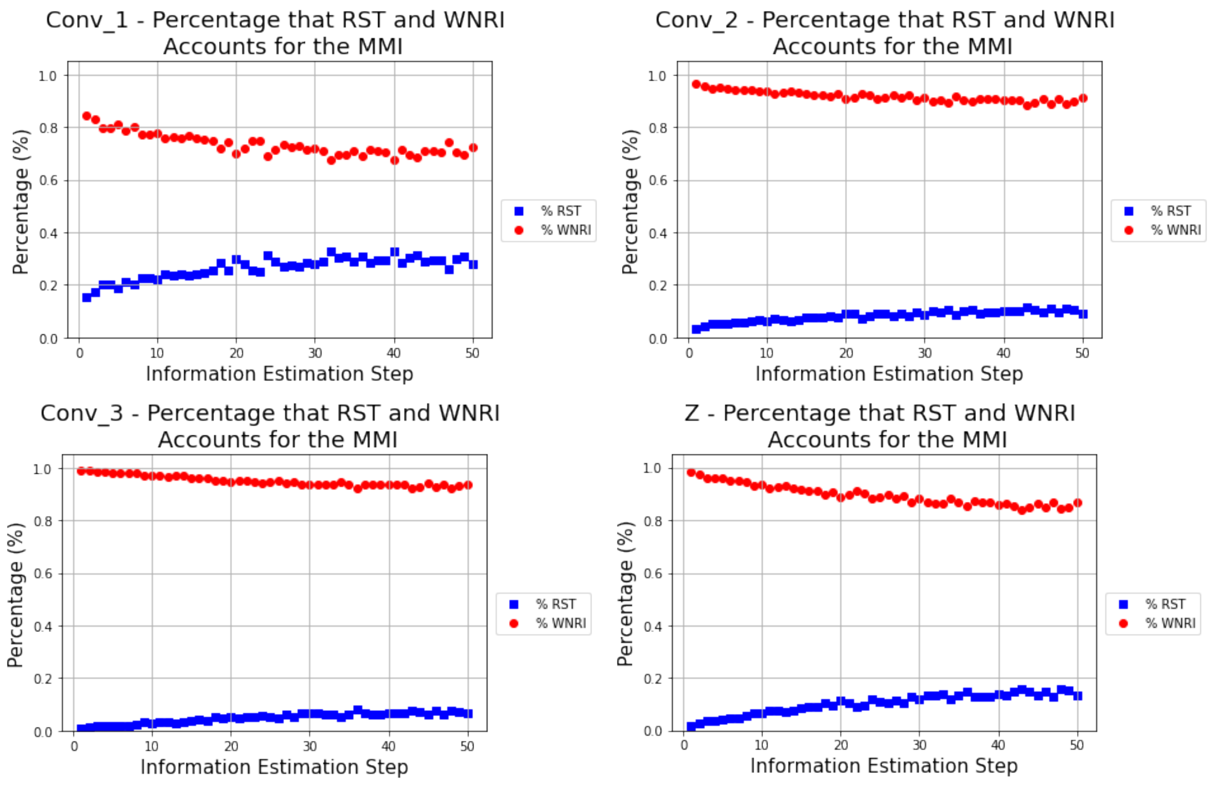
The results generated by using this architecture in one of the experiments are depicted in Figures 3.8 and 3.9. Grayscale images were used. During training, the information quantities were estimated only once, at the end of each epoch. As it was the case for CAE_1_4, the convergence of $H(Z)$ to a value lower than the MMI indicates that $size(Z)$ may be insufficient to adequately capture the informational content of the dataset. However, the fact that $H(Z)$ converged to a value closer to the MMI may suggest that its bottleneck's dimension ($size(Z)$) is closer to the minimum necessary to adequately fully capture the dataset's information structure in its codes. Also worth noting is the fact that the curves depicted in Figure 3.8 displayed a similar behavior to those in Figure 3.4. This is expected because the training of both CAEs were successful. Also, the curves' behaviors in the aforementioned Figures corroborated Assumptions 1 and 2, further corroborating them.

**Figure 3.8.** Evolution of the CAE_1_6's information quantities' magnitudes during the experiment.



**Source:** Own authorship.

**Figure 3.9.** Evolution of the CAE_1_6's $\mathbf{RST}_{\%}$ and $\mathbf{WNRI}_{\%}$ during the experiment.

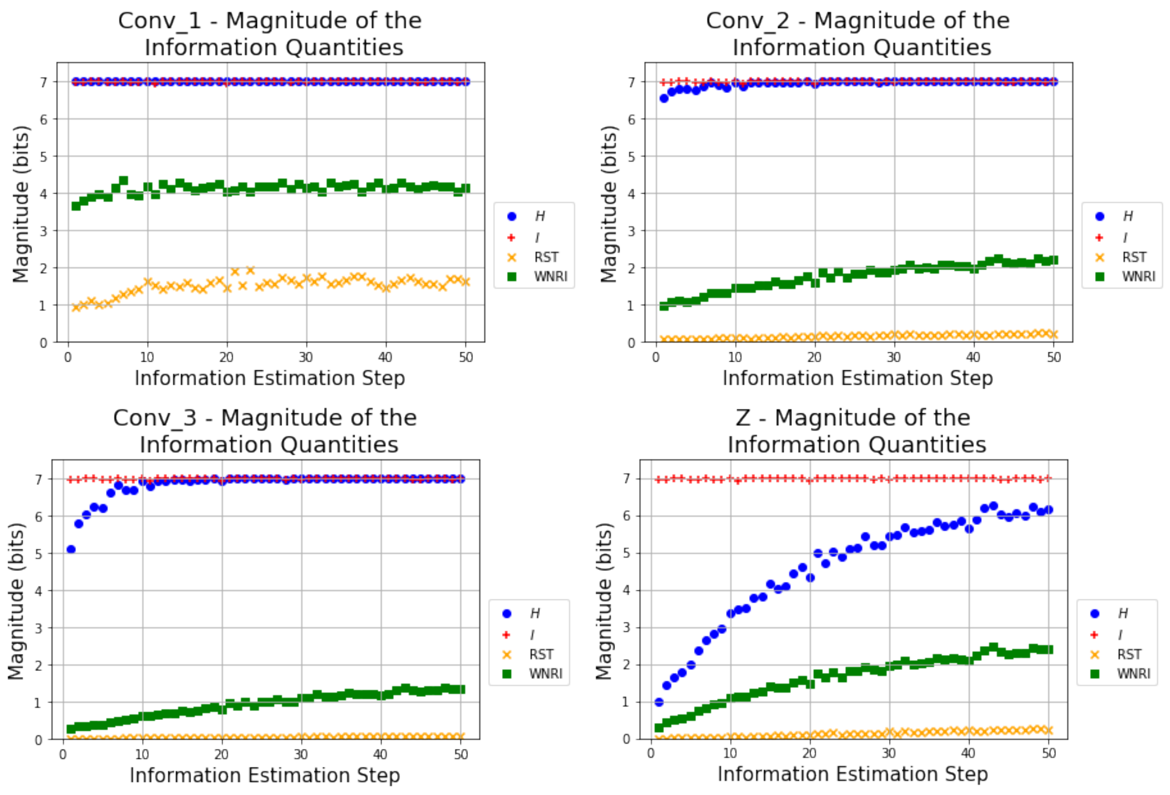


**Source:** Own authorship.

### 3.3.3 CAE_1_7

The results generated by using this architecture in one of the experiments are depicted in Figures 3.10 and 3.11. Grayscale images were used. During training, the information quantities were estimated only once, at the end of each epoch. The training of this CAE was also performed successfully. As it was the case for both CAE_1_4 and CAE_1_6, the convergence of $H(Z)$ to a value lower than the MMI indicates that the bottleneck layer has an insufficient amount of FMs to adequately capture the informational content of the dataset.

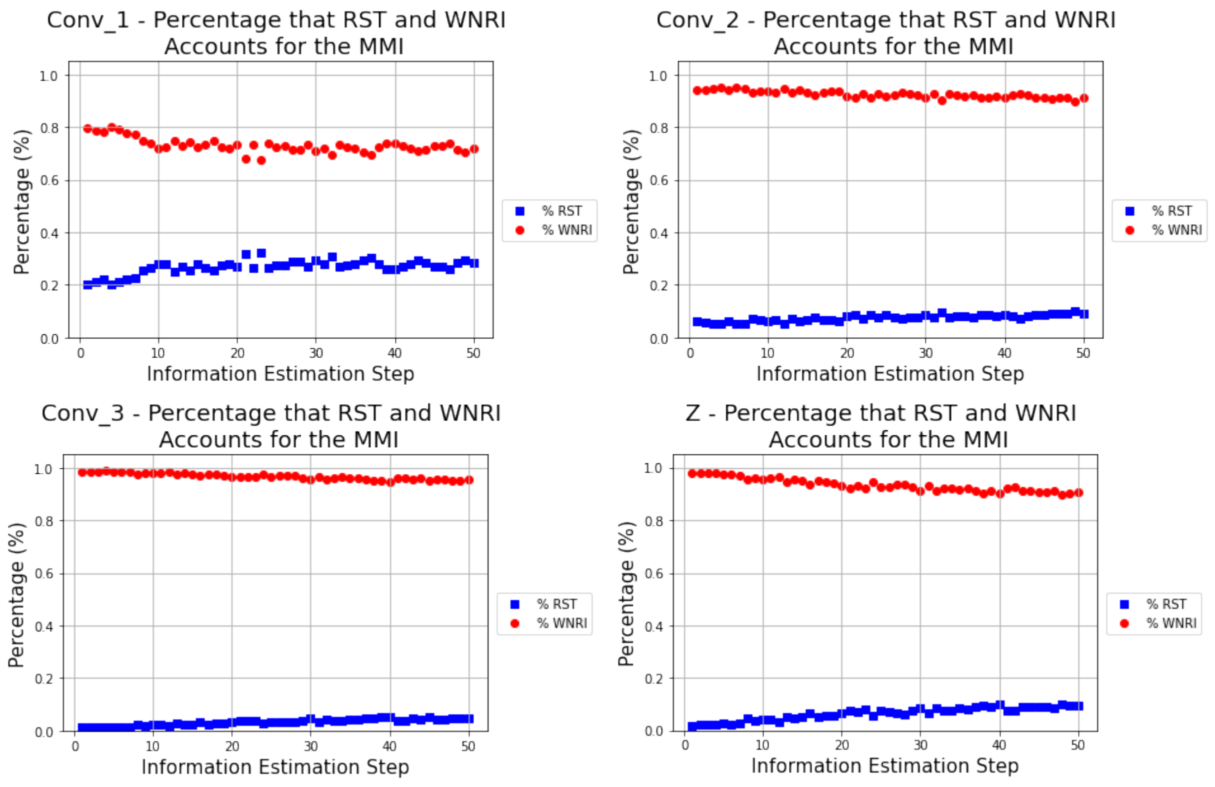
**Figure 3.10.** Evolution of the CAE_1_7's information quantities' magnitudes during the experiment.



**Source:** Own authorship.

However, despite CAE_1_7 having a bottleneck with higher dimension than CAE_1_6, the $H(Z)$ of the former converged to a lower value than the one of the latter. This somewhat counterintuitive result may be provoked by differences in the quality of the training processes. If the quality of the CAE_1_6's training was superior than that of the CAE_1_7, it can make it more capable of storing the most important information from the dataset, despite having a bottleneck with less capacity for it. Another possible explanation is the random initialization of the CAEs' weights and kernels at the beginning of their training, which may have impacted

**Figure 3.11.** Evolution of the CAE_1_7's $\textbf{RST}_{\%}$ and $\textbf{WNRI}_{\%}$ during the experiment.



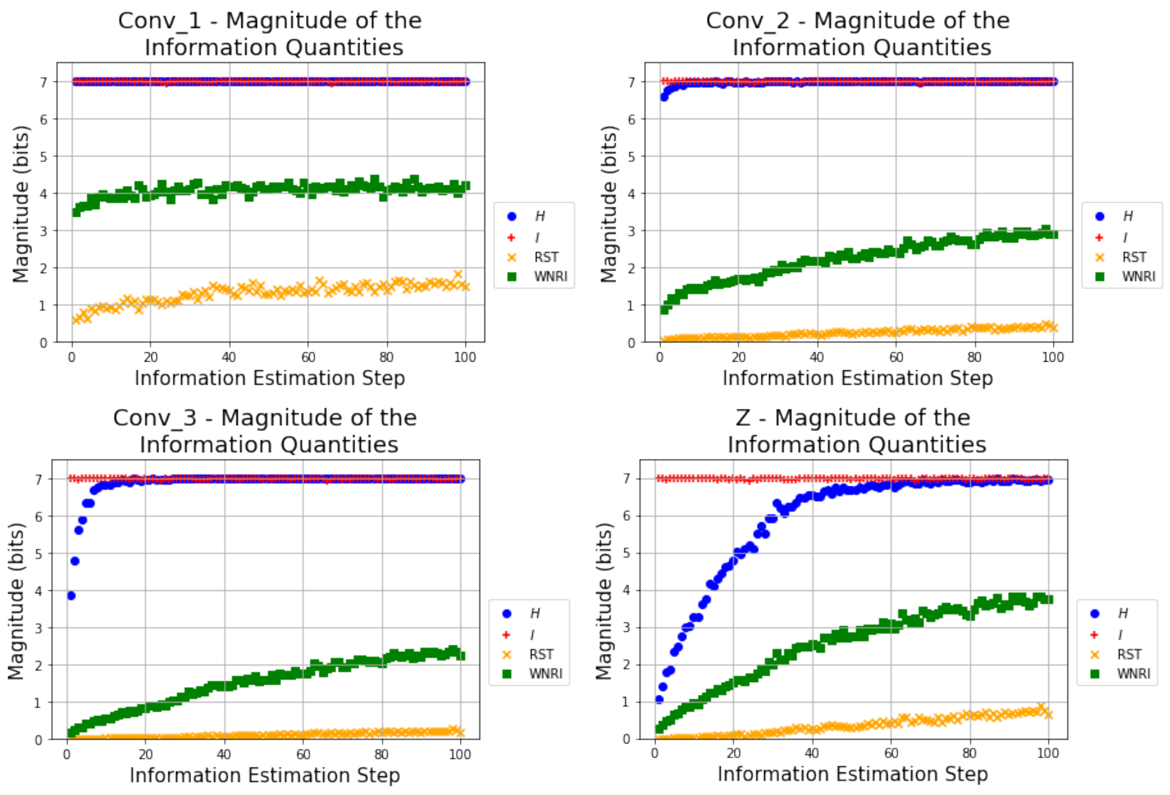
**Source:** Own authorship.

their quality. That is, maybe the random weights and kernels initially assigned to CAE_1_6 had a positive impacted on its training quality, and those assigned to CAE_1_7 had a positive impact. Despite this possible explanations, it is unclear what exactly causes the difference of quality between the training processes that may be behind this phenomenon.

### 3.3.4 CAE_1_8

This architecture was used in two different experiments whose results are depicted in Figures 3.12, 3.13,3.14 and 3.15. Grayscale images were used for both. In experiment 1, the information quantities were estimated two times each epoch: one in the middle and one in the end. In experiment 2, they were estimated only once, at the end of each epoch. This was done to evaluate if there was any substantial difference between the amount of times the estimations were performed and, if so, if the costs associated with more estimations were justified or not by any advantage. A comparison between Figures 3.12 and 3.14 shows that, despite the curves depicted in the former containing more datapoints, this does not translate in a better unders-

**Figure 3.12.** Evolution of the CAE_1_8's information quantities' magnitudes during experiment 1.

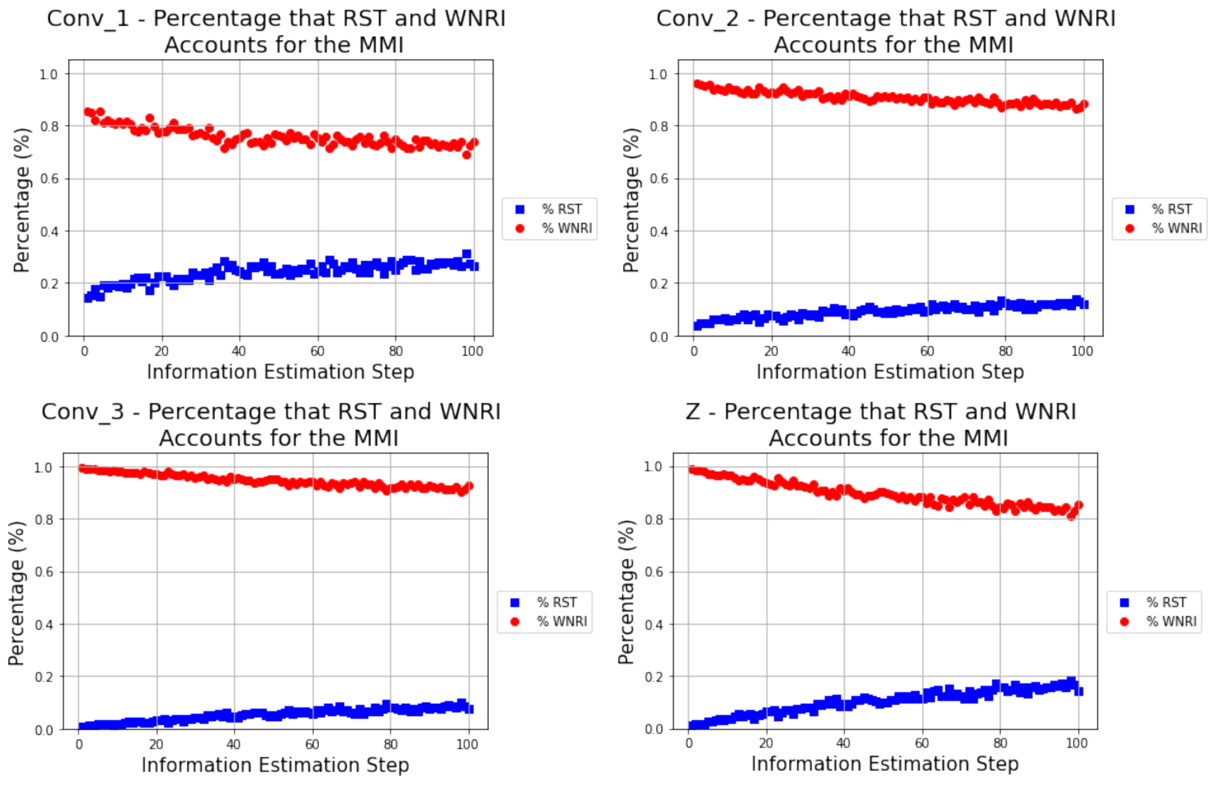


**Source:** Own authorship.

tanding of the overall behavior of the curves. Therefore, it indicates that, for this particular CAE, there are no benefits in estimating the information quantities more than once per epoch. Different from the CAE architectures previously investigated, the $H(Z)$ of CAE_1_8 converged to the MMI in both experiments, which indicates that $size(Z)$ is sufficient to adequately capture the informational content of the dataset. As verified in the other successful experiments, the information quantities of the CAE's corroborated Assumptions 1 and 2, further supporting the previously hypotheses raised.

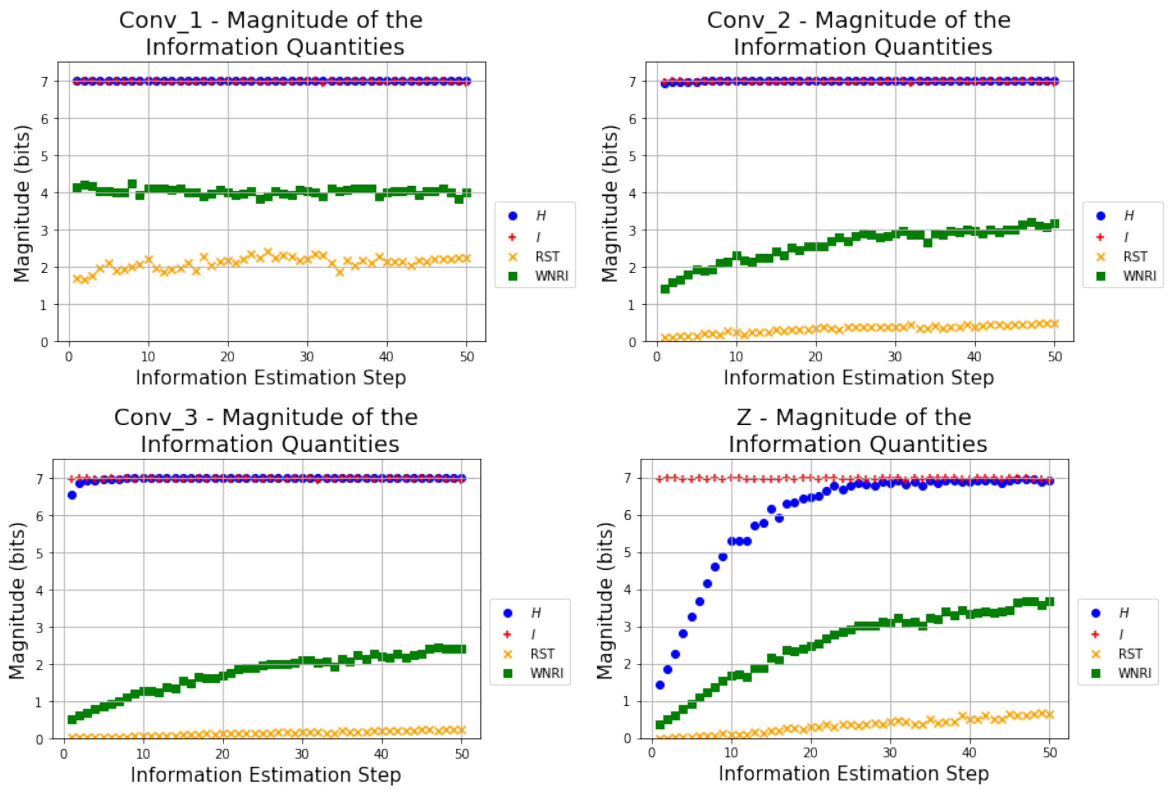
Finally, it is worth noting that the curves depicted in Figures 3.12 and 3.14 display a very similar behavior, and the information quantities depicted in both converged to very similar values. This phenomenon may suggest that, if two CAEs with the same architecture are trained using the same dataset, and both training processes are successful (i.e., corroborate with Assumptions 1 and 2), the behavior of the information quantities of their respective layers can be expected to be very similar. This is an important result because this phenomenon is necessary in order to use these information quantities as evaluation criteria.

**Figure 3.13.** Evolution of the CAE_1_8's $\mathbf{RST}_{\%}$ and $\mathbf{WNRI}_{\%}$ during experiment 1.



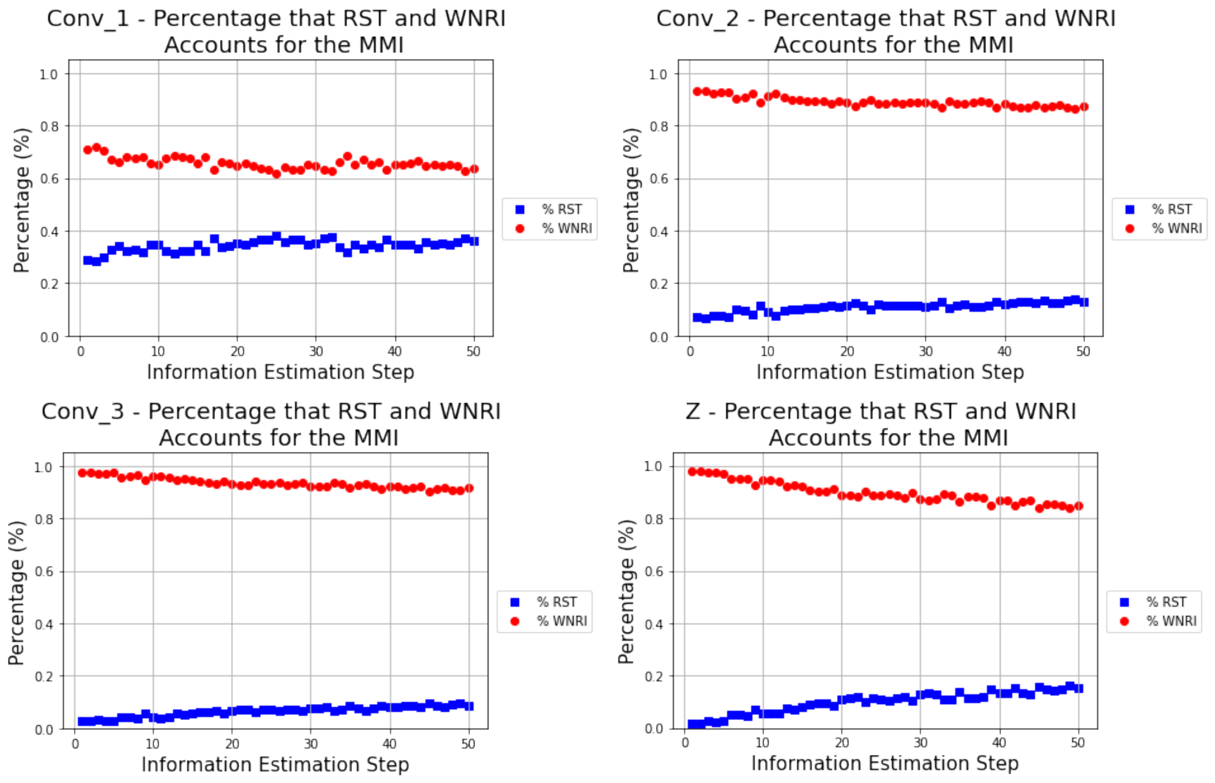
**Source:** Own authorship.

**Figure 3.14.** Evolution of the CAE_1_8's information quantities' magnitudes during experiment 2.



**Source:** Own authorship.

**Figure 3.15.** Evolution of the CAE_1_8's $\mathbf{RST}_{\%}$ and $\mathbf{WNRI}_{\%}$ during experiment 2.

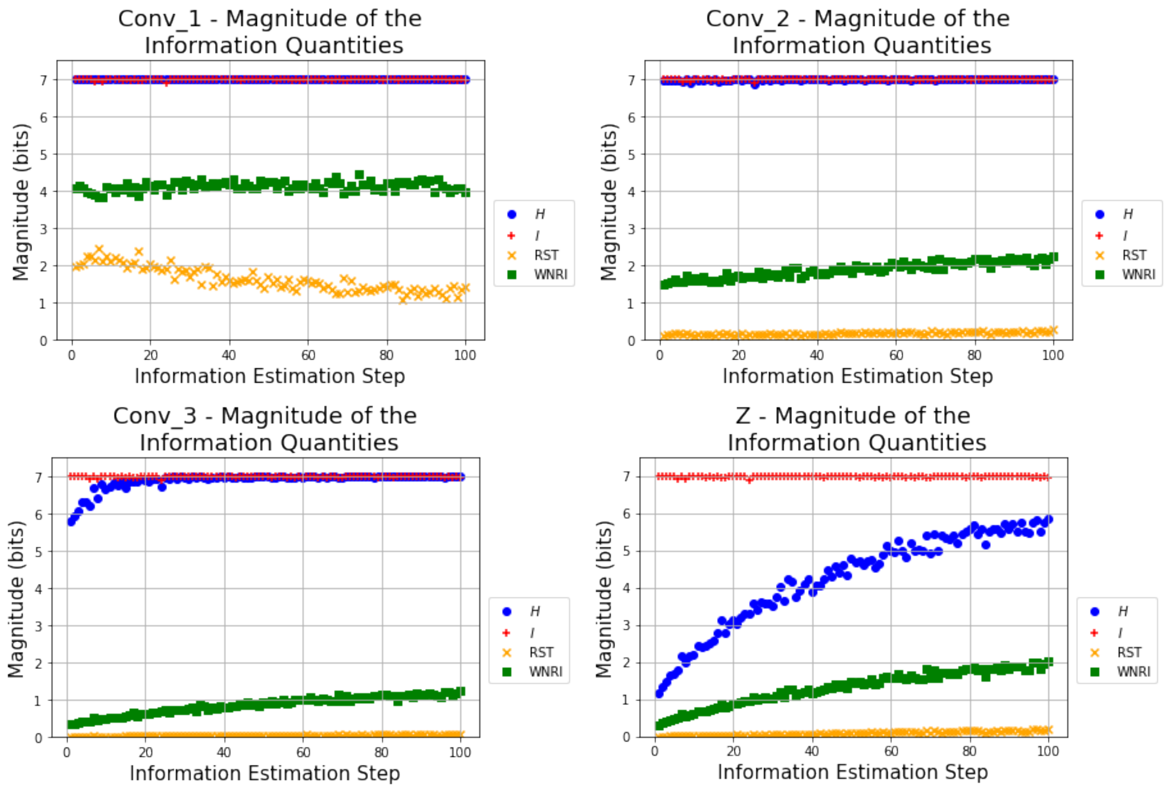


**Source:** Own authorship.

### 3.3.5 CAE_1_8_N

This architecture was used in two different experiments whose results are depicted in Figures 3.16, 3.17, 3.18 and 3.19. In both experiments, grayscale images were used, and the information quantities were estimated two times each epoch: one in the middle and one in the end. This was done to further evaluate if there were any benefits to this procedure and, again, it was verified that no additional information was provided by the larger amount of points in the curves depicted.

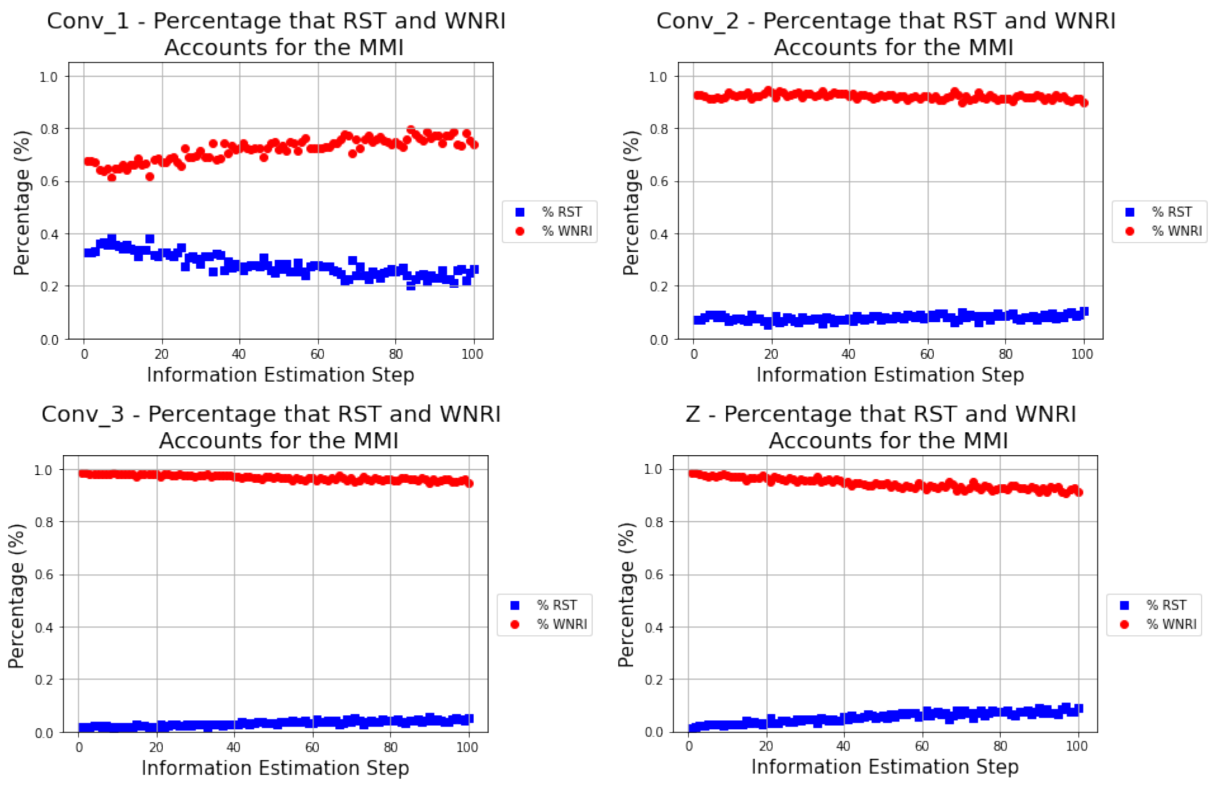
**Figure 3.16.** Evolution of the CAE_1_8_N's information quantities' magnitudes during experiment 1.



**Source:** Own authorship.

The only difference between this architecture and that of the CAE_1_8 is that, instead of using the bilinear function as the interpolation function for the decoder's layers, the nearest function, widely used in practical applications, was used instead. This was done to verify if the choice of this function had any impact on the evolution of the information quantities of the encoder's layers and. The differences between the curves depicted in Figures 3.12 3.14, 3.17 and 3.19 suggest that the choice of the interpolation function used in the decoder has an impact on the evolution of the information quantities of the encoder's layers.

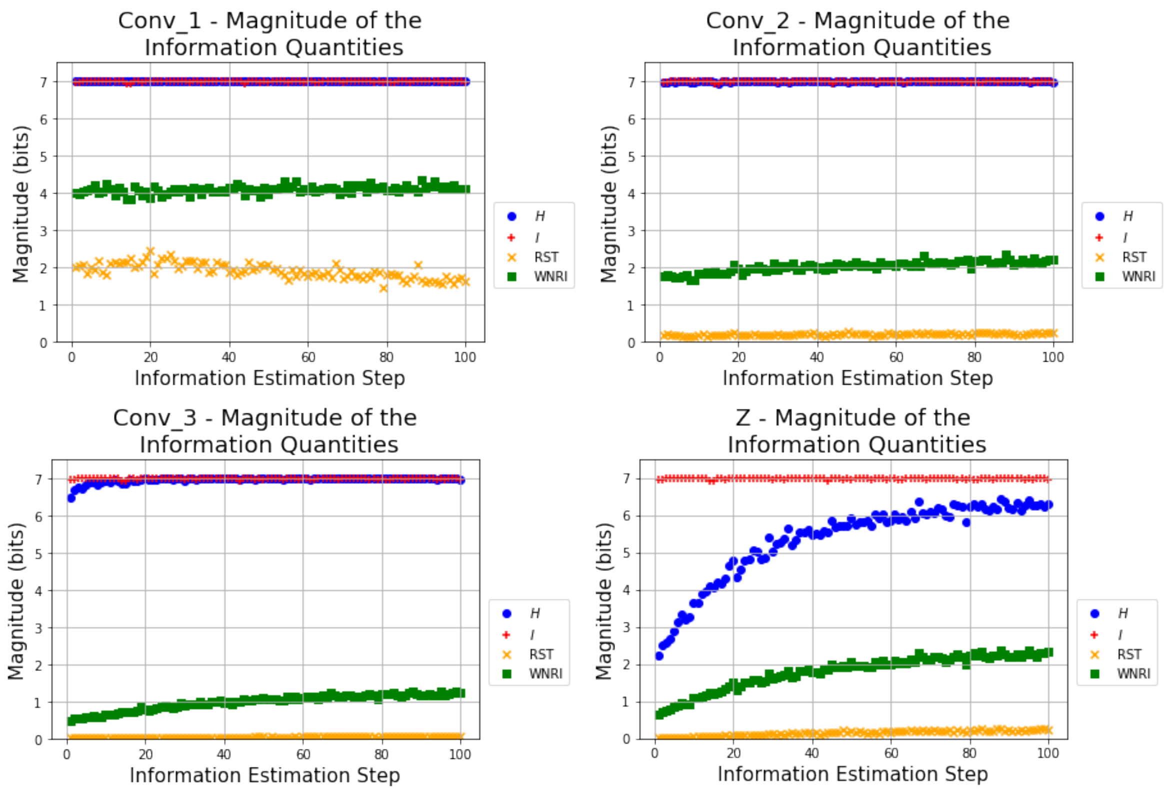
**Figure 3.17.** Evolution of the CAE_1_8_N's $\mathbf{RST}_{\%}$ and $\mathbf{WNRI}_{\%}$ during experiment 1.



**Source:** Own authorship.

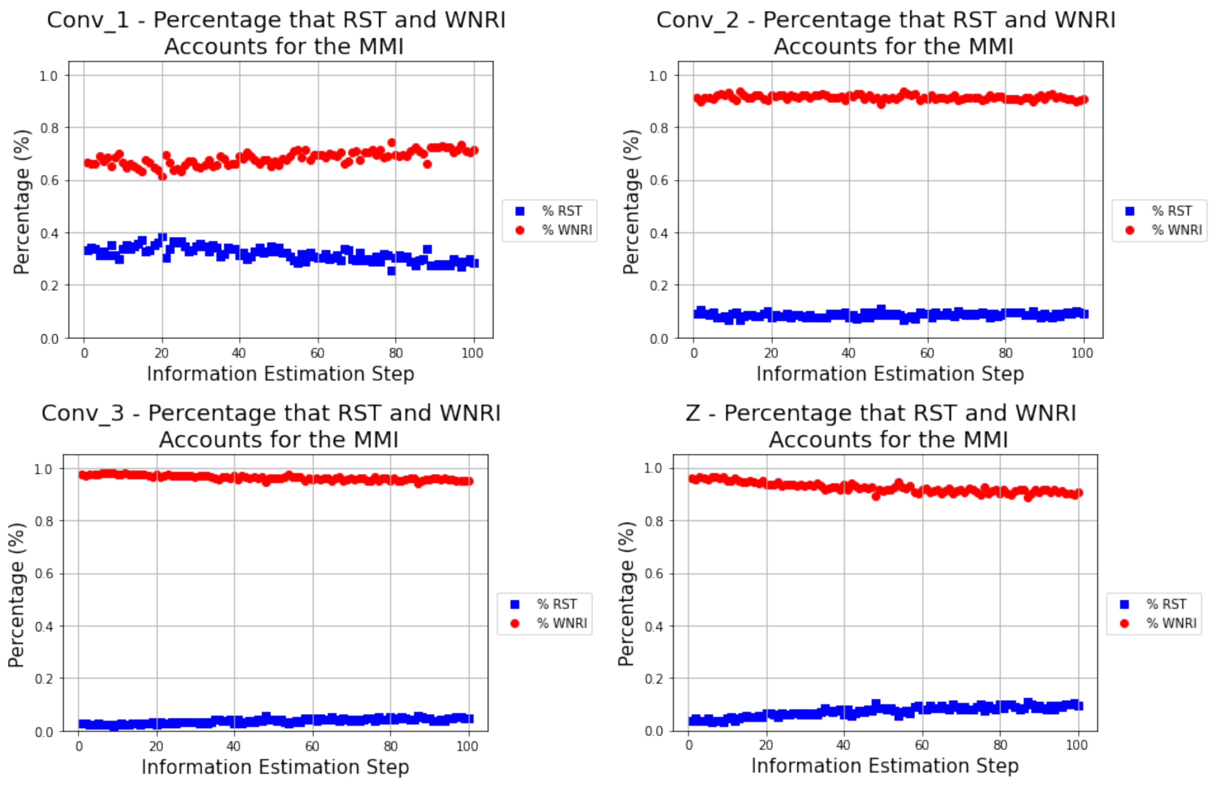
Despite the CAE_1_8_N having the same amount of FMs in the bottleneck layer as CAE_1_8, its entropy converged to a value lower than the MMI. Therefore, from an informational standpoint, CAE_1_8 had a superior performance when compared to CAE_1_8_N. This is an important result because it reveals that the choice of the aforementioned function can single-handedly impact on the value to which $H(Z)$ will converge. One possible reason for this result is the importance of the decoder's outputs for the quality of the training process. Since they are constructed from the codes stored in the bottleneck and are used to compute the MSE, their quality is crucial for the overall training process. Since their construction heavily depends on the interpolation function used, it suggests that the use of better interpolation functions from an informational standpoint will yield better results.

**Figure 3.18.** Evolution of the CAE_1_8_N's information quantities' magnitudes during experiment 2.



**Source:** Own authorship.

**Figure 3.19.** Evolution of the CAE_1_8_N's $\mathbf{RST}_{\%}$ and $\mathbf{WNRI}_{\%}$ during experiment 2.

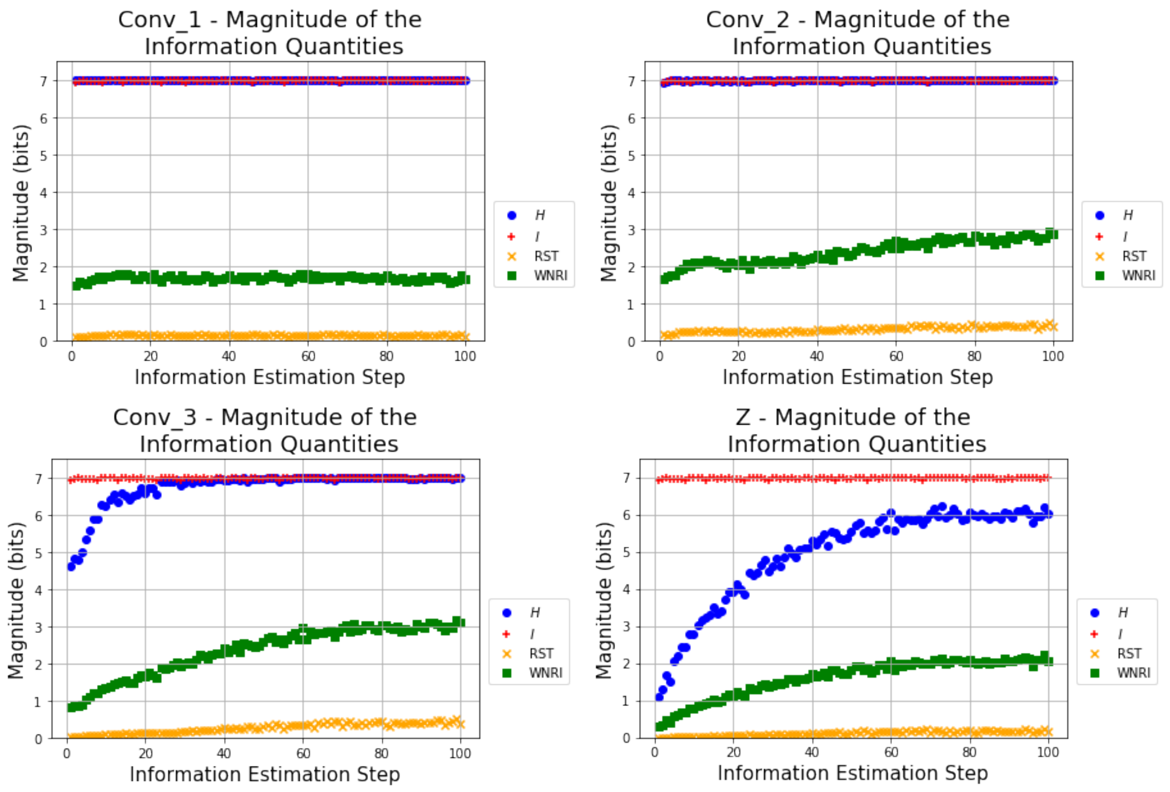


**Source:** Own authorship.

### 3.3.6 CAE_2_8

The results generated by using this architecture in one of the experiments are depicted in Figures 3.20 and 3.21. Grayscale images were used. During training, the information quantities were estimated only once, at the end of each epoch. The convergence of $H(Z)$ to a value lower than the MMI may indicate that $size(Z)$ is insufficient to adequately capture the informational content of the dataset. However, this CAE's bottleneck size is the same as that of CAE_1_8 and, as previously discussed, its $H(Z)$ converged to the MMI in both experiments where it was used.

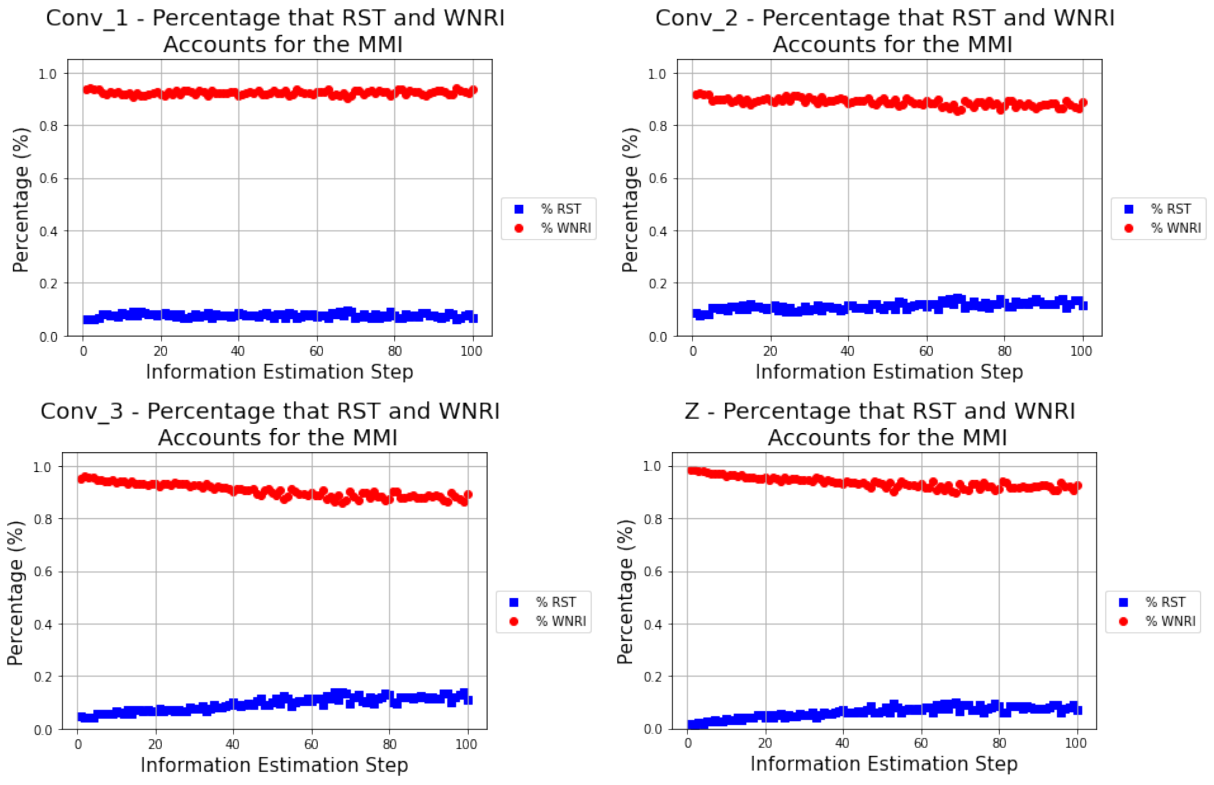
**Figure 3.20.** Evolution of the CAE_2_8's information quantities' magnitudes during the experiment.



**Source:** Own authorship.

This suggests that the order in which the FMs of the hidden layers are arranged can impact the evolution of the CAE's information quantities and, therefore, the quality of its training. This result is interesting because it may suggest that designing CAE's in the same fashion as those in group CAE_2 can lead to architectures that have more difficulties to achieve an optimal from an information standpoint.

**Figure 3.21.** Evolution of the CAE_2_8's $\mathbf{RST}_{\%}$ and $\mathbf{WNRI}_{\%}$ during the experiment.



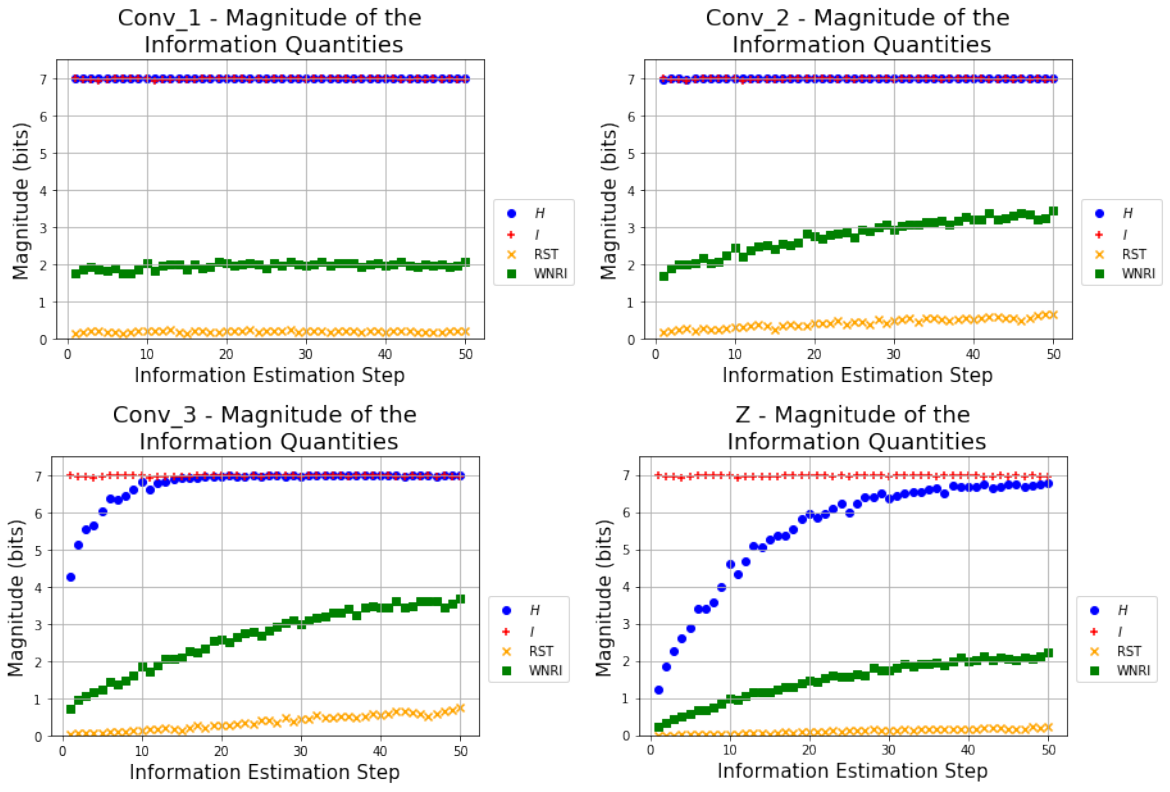
**Source:** Own authorship.

### 3.3.7 CAE_2_12

The results generated by using this architecture in one of the experiments are depicted in Figures 3.22 and 3.23. Grayscale images were used. During training, the information quantities were estimated only once, at the end of each epoch.

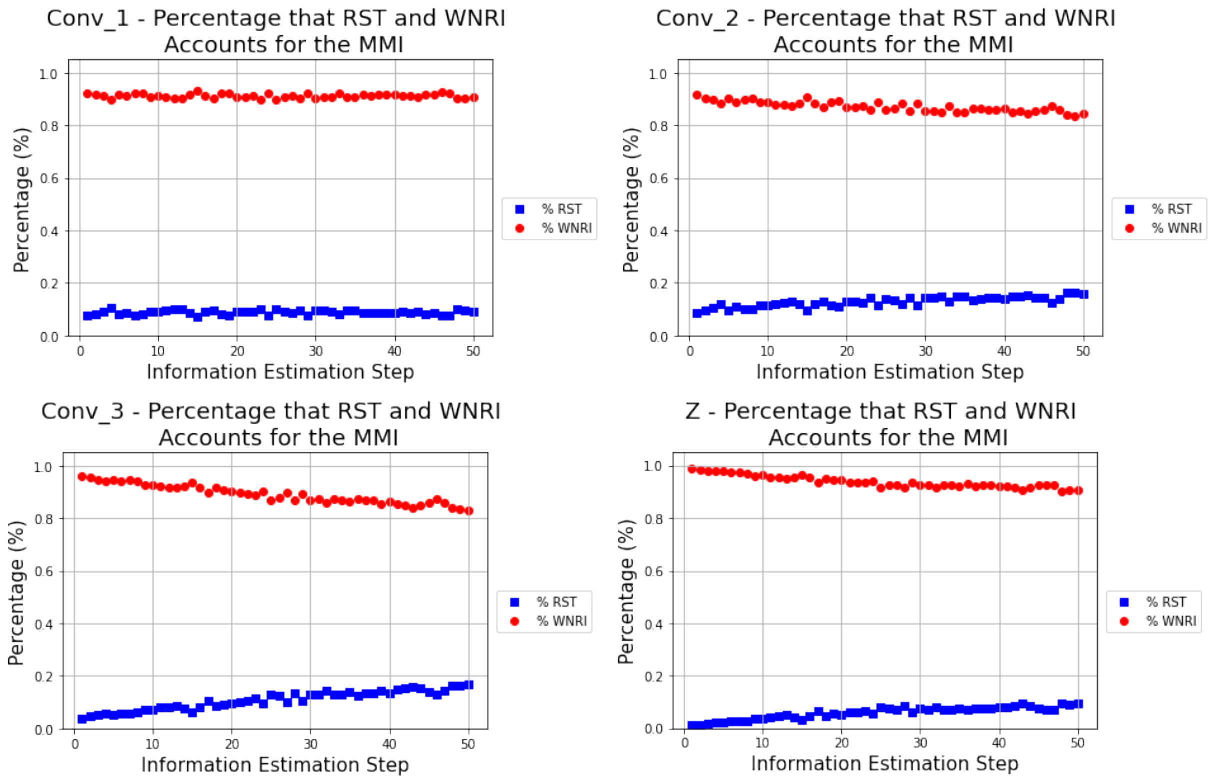
As it was the case for CAE_1_8, the $H(Z)$ of CAE_2_12 converged to the MMI during the experiment. This indicates that its $size(Z)$ is sufficient to adequately capture the informational content of the dataset. Moreover, the curves depicted in Figure 3.22 corroborate Assumptions 1 and 2. The results depicted in Figures 3.12, 3.14, 3.20 and 3.22 show that, differently from what was observed in [11], modifying the value of $size(Z)$ is not enough to properly determine a CAE's size during training. Despite their similarities, for this particular dataset, CAE_1_8 yielded better results than CAE_2_8. Although increasing $size(Z)$ of CAE_2_8 by 50% (thus creating CAE_2_12) made its $H(Z)$ converge to the MMI, it did so considerably slower than the CAE_1_8's $H(Z)$, which can be viewed as an indicator of inferior performance.

**Figure 3.22.** Evolution of the CAE_2_12's information quantities' magnitudes during the experiment.



**Source:** Own authorship.

**Figure 3.23.** Evolution of the CAE_2_12's $\mathbf{RST}_{\%}$ and $\mathbf{WNRI}_{\%}$ during the experiment.

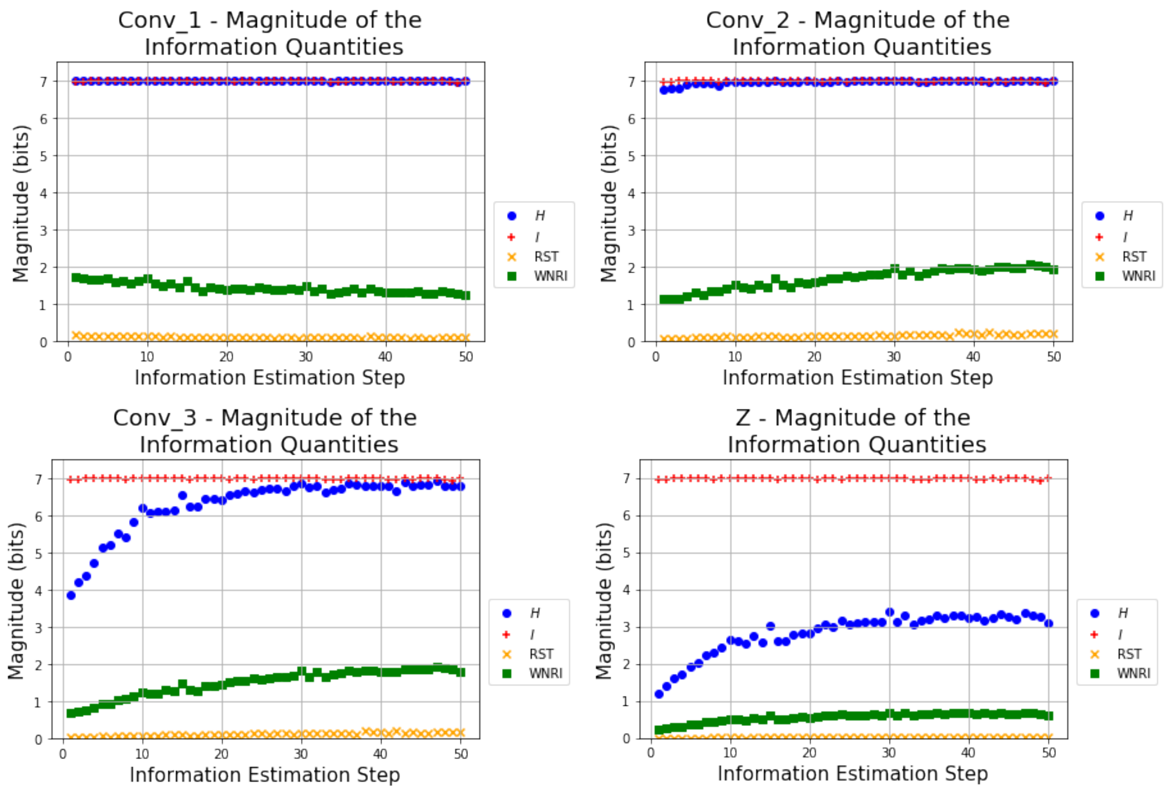


**Source:** Own authorship.

### 3.3.8 CAE_2_12_N

This architecture was used in two different experiments whose results are depicted in Figures 3.24, 3.25, 3.26 and 3.27. Grayscale images were used. In experiment 1, the information quantities were estimated only once, at the end of each epoch. In experiment 2, they were estimated two times each epoch: one in the middle and one in the end. Similarly to what was verified previously, no benefits were provided by estimating the information quantities more than once per epoch, although the computational cost of the estimation process increased considerably.

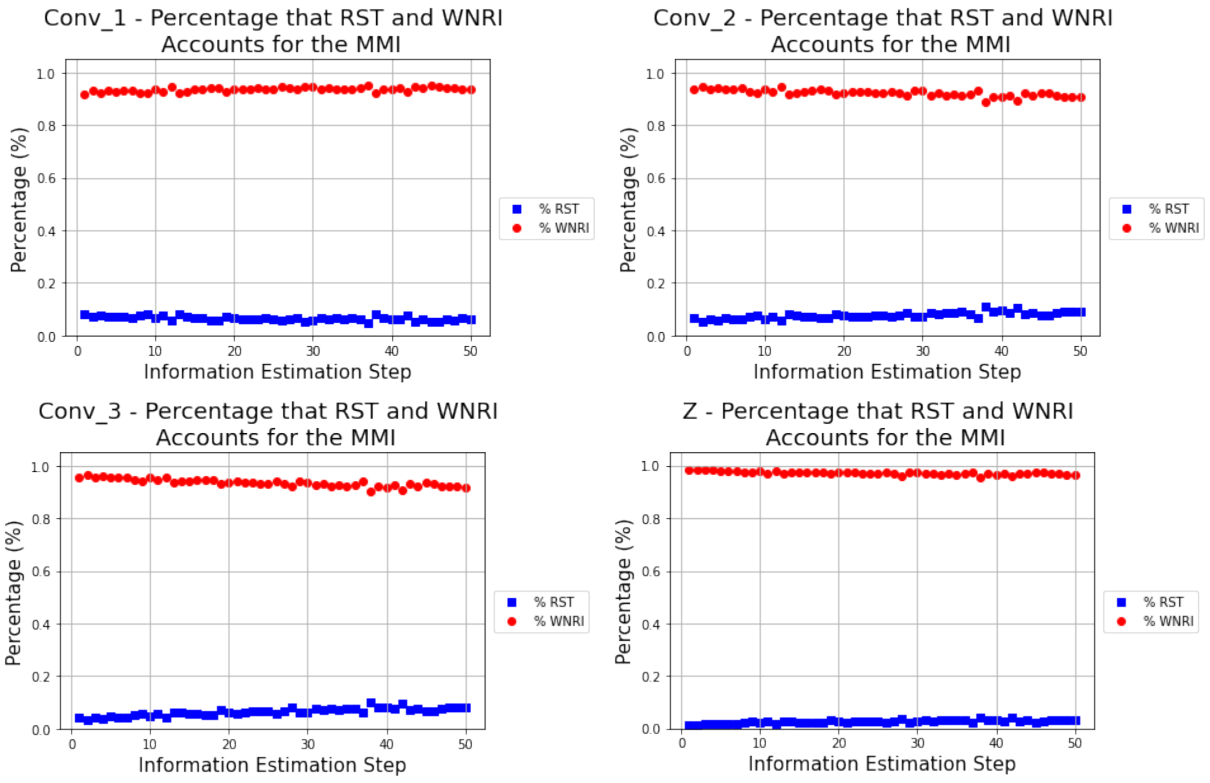
**Figure 3.24.** Evolution of the CAE_1_12_N's information quantities' magnitudes during experiment 1.



**Source:** Own authorship.

During both experiments, $H(Z)$ converged to a value considerably lower than the MMI, yielding worst results (from an information standpoint) than even CAE_1_4, which has a bottleneck layer with four times less capacity. This lackluster behavior, similar to that verified for CAE_1_8_N, further supports the possibility that the "nearest" function is inferior to the "bilinear" one from an information point of view. The fact that the convergence of $H(Z)$ to the MMI was observed during the training of both CAE_1_8 and CAE_2_12 also corroborates

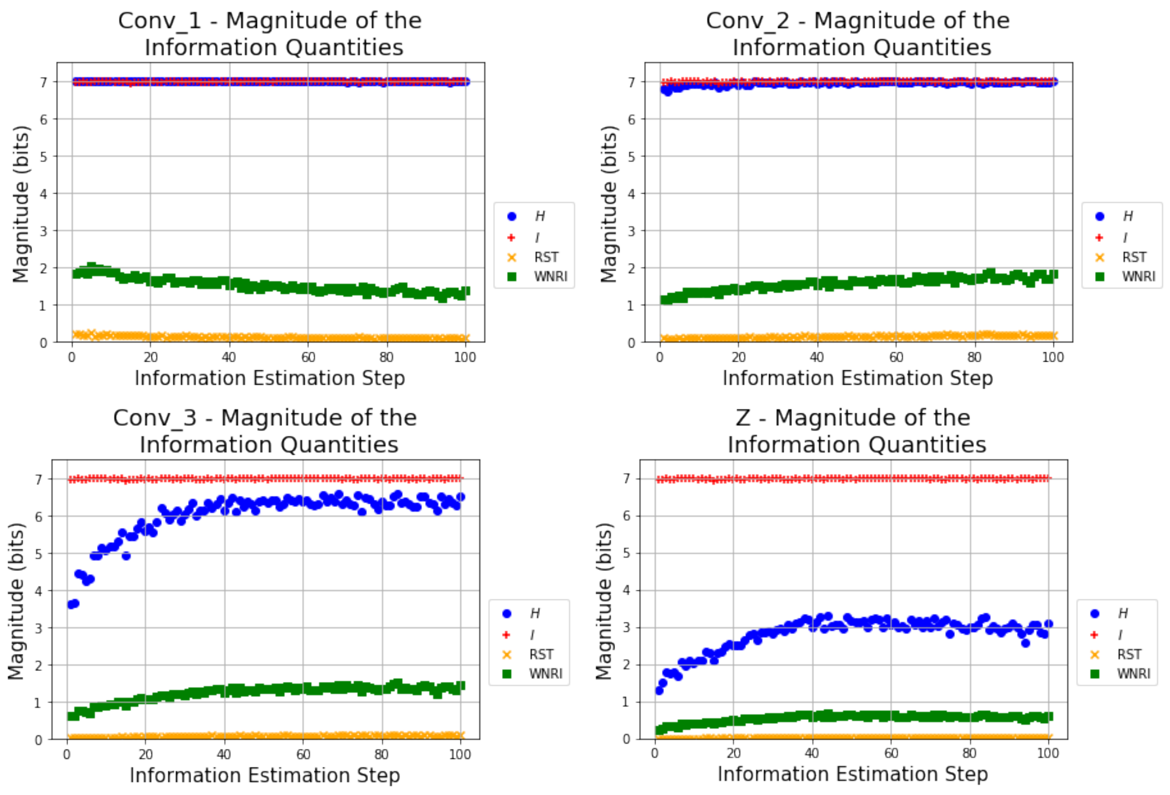
**Figure 3.25.** Evolution of the CAE_1_12_N's $\mathbf{RST}_{\%}$ and $\mathbf{WNRI}_{\%}$ during experiment 1.



**Source:** Own authorship.

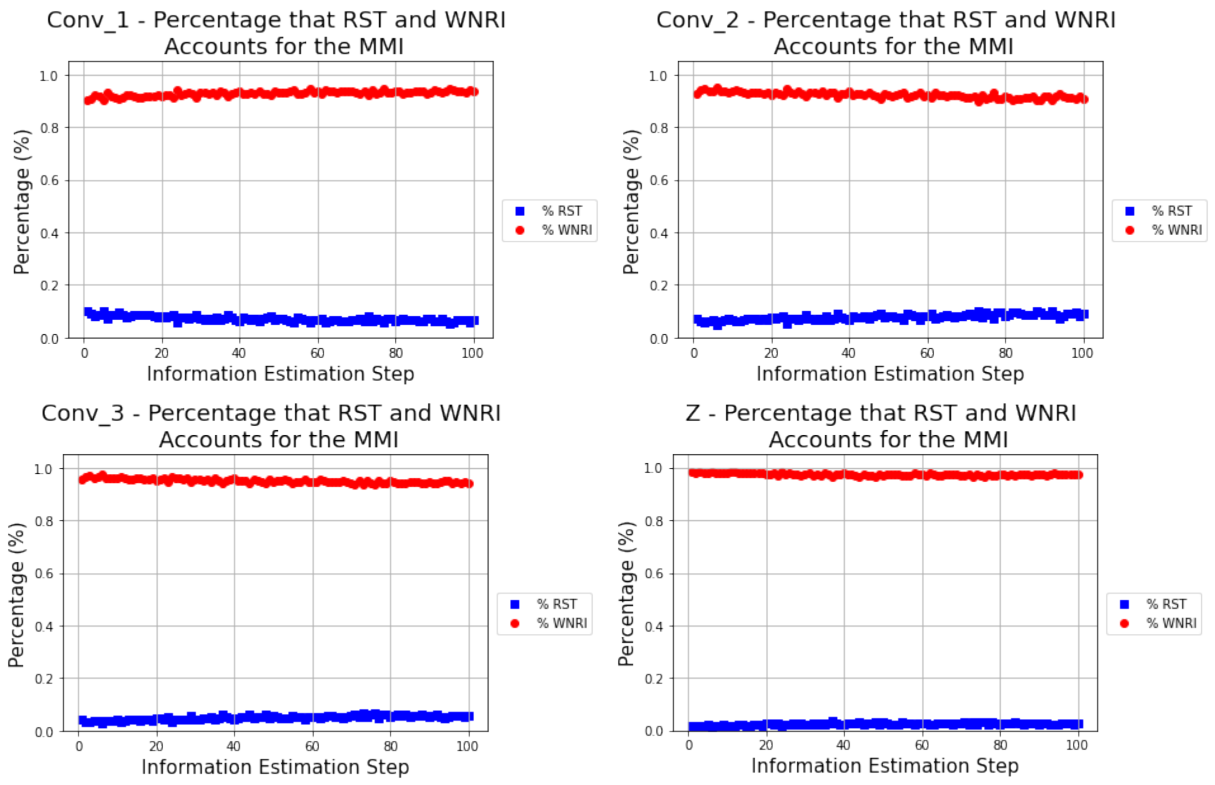
with this possibility.

**Figure 3.26.** Evolution of the CAE_1_12_N's information quantities' magnitudes during experiment 2.



**Source:** Own authorship.

**Figure 3.27.** Evolution of the CAE_1_12_N's $\mathbf{RST}_{\%}$ and $\mathbf{WNRI}_{\%}$ during experiment 2.

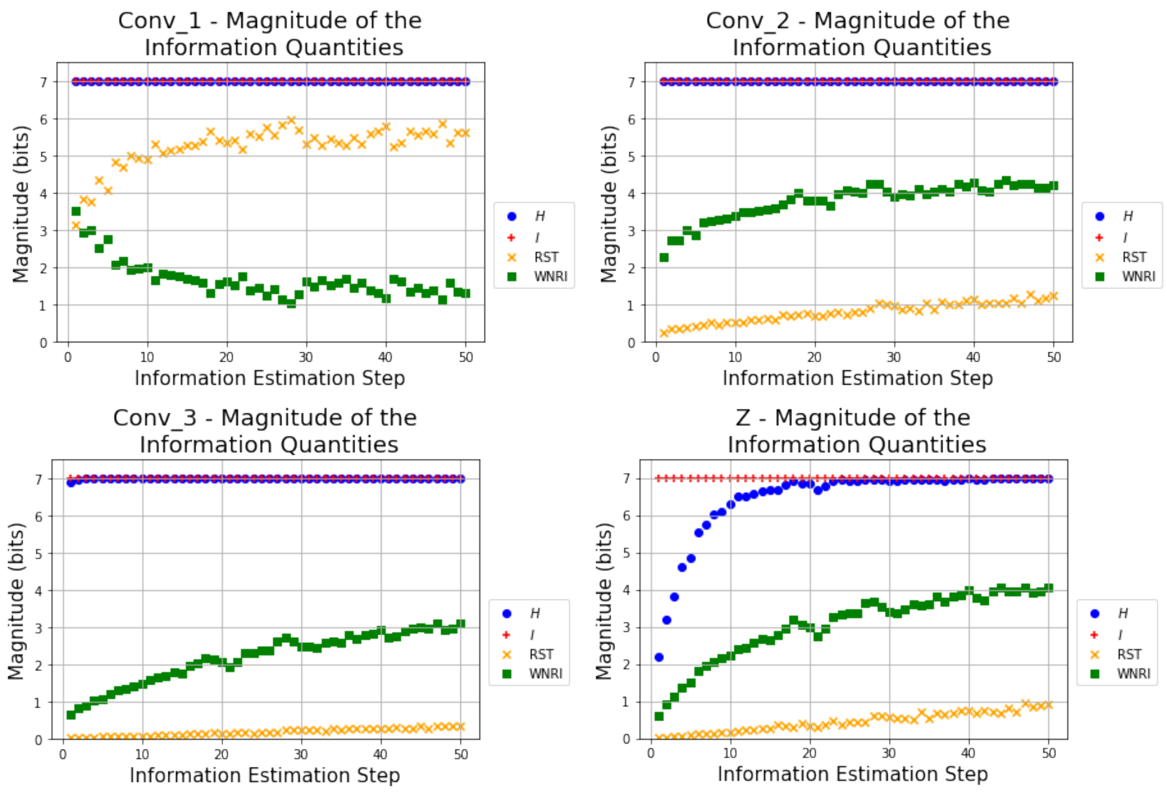


**Source:** Own authorship.

### 3.3.9 RGB Images

Architecture CAE_1_8 was used in two different experiments whose results are depicted in Figures 3.28, 3.29, 3.30 and 3.31. RGB images were used for both, and the information quantities were estimated only once, at the end of each epoch. This was done to evaluate if using RGB images instead of grayscale ones provided tangible benefits and, if so, if they outweighed the higher computational cost associated with the use of three input channels instead of a single one.

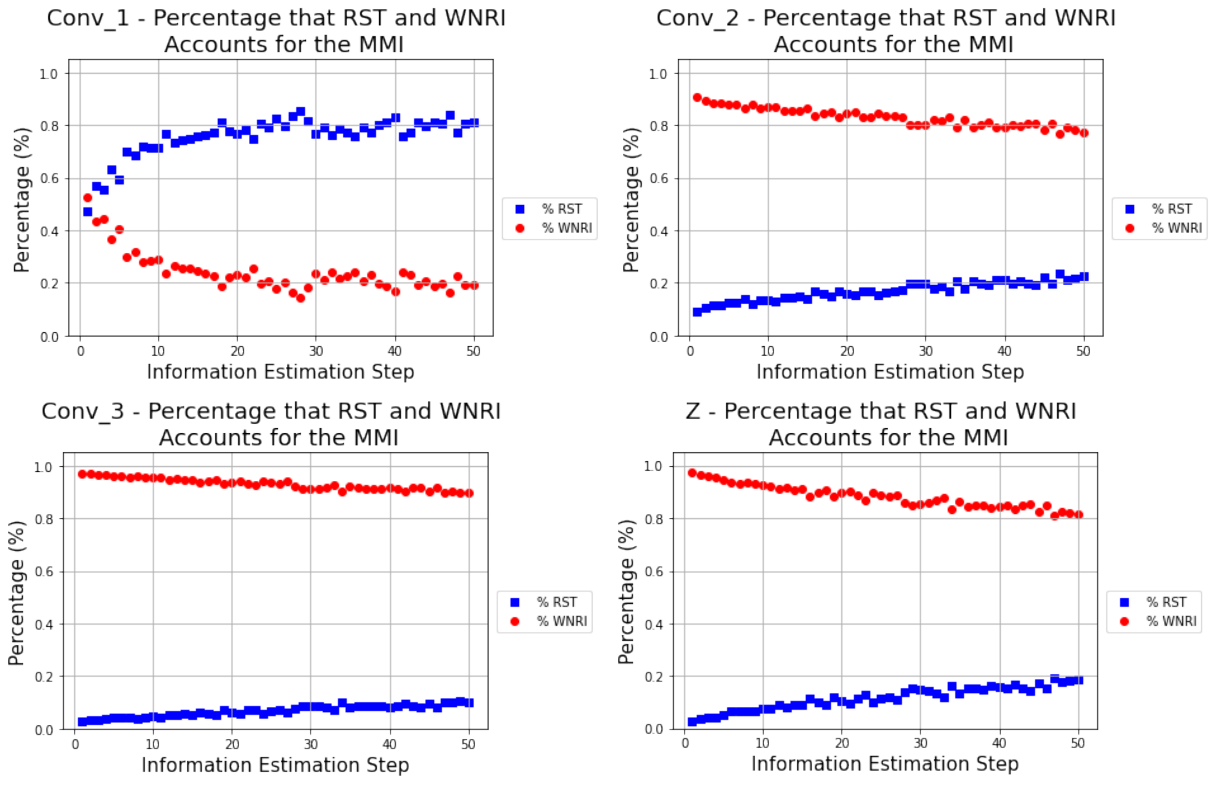
**Figure 3.28.** Evolution of the CAE_1_8's information quantities' magnitudes during experiment 1.



**Source:** Own authorship.

The curves depicted in Figures 3.12, 3.14, 3.28 and 3.30 show that the use of the three color channels considerably increased the speed with which $H(Z)$ converged to the MMI. This result is expected, since the information provided by three channels is intuitively larger than that provided by a single one. Despite this improvement in the speed of convergence, however, it is not clear whether it is worth the substantial increase in the computational costs of the estimations or not. Despite the increase of the time required to perform the estimations, it can be argued that it is offset by the earlier convergence of the information quantities, which in

**Figure 3.29.** Evolution of the CAE_1_8's $\mathbf{RST}_{\%}$ and $\mathbf{WNRI}_{\%}$ during experiment 1.

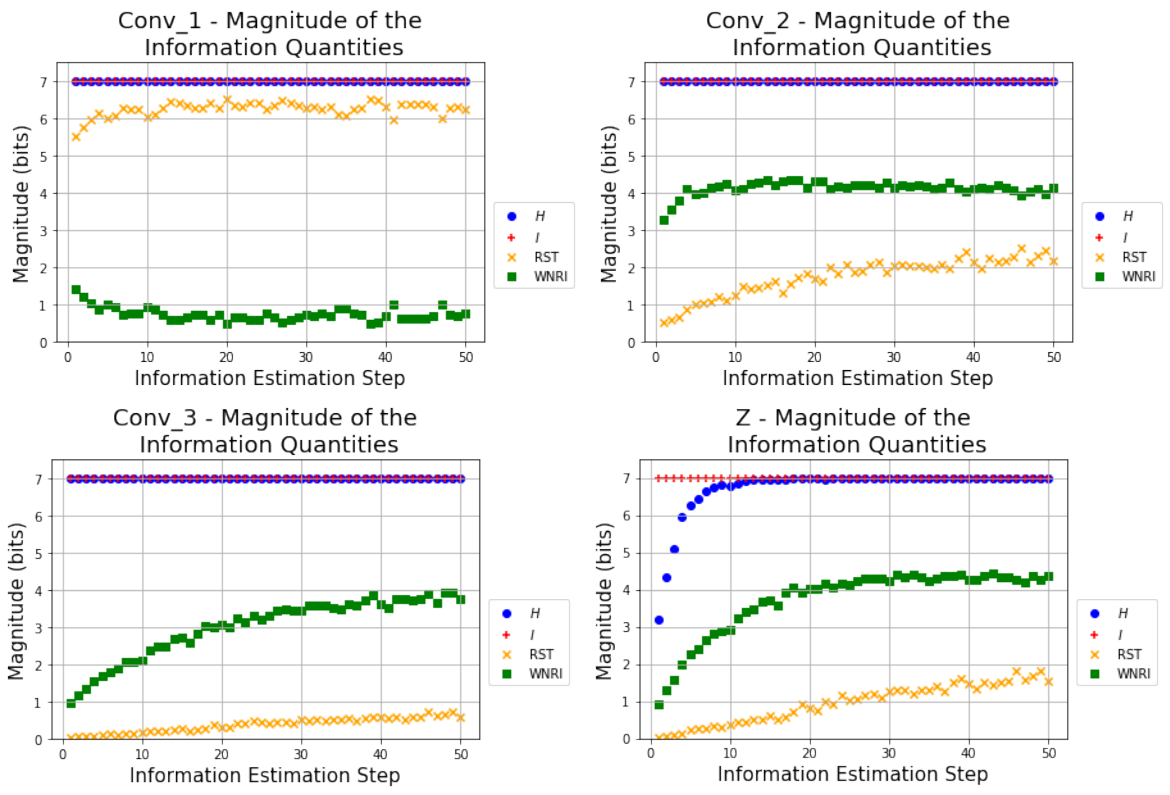


**Source:** Own authorship.

turn requires less epochs of training to happen. Also, this early interruption of the training has the additional benefit of reducing the amount of redundant information stored in the CAE's bottleneck. This is due to the fact that the bottleneck's **RST** tends to increase during training.

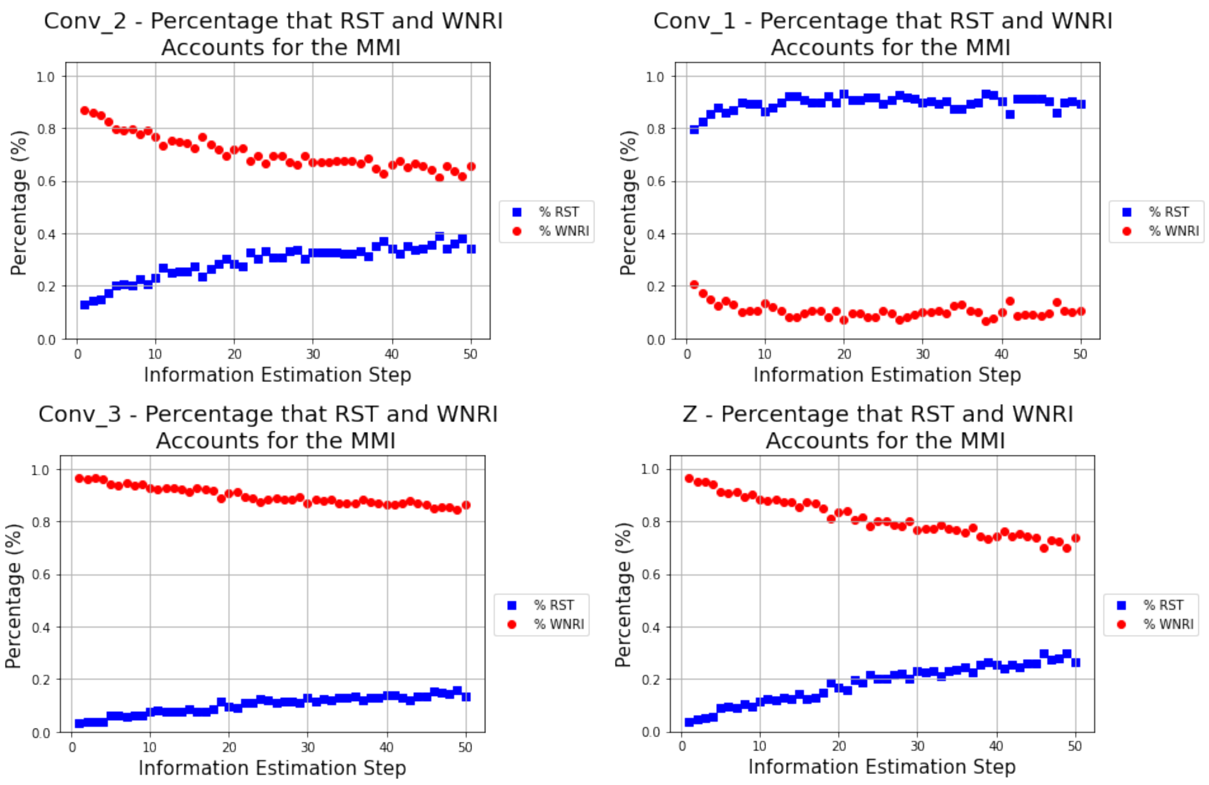
Lastly, it is worth noting that, differently from the other experiments in which such convergence was achieved, the **WNRI** and **RST** curves of the CAE-1-8's Conv_1 did not display the expected behavior discussed before. In both experiments, the **RST** curve associated with this layer stayed above the **WNRI** one, which was not verified in any other experiment that used grayscale images. Although there is no clear explanation for this phenomenon, we suggest that this can be a particularity of the type of image used.

**Figure 3.30.** Evolution of the CAE_1_8's information quantities' magnitudes during experiment 2.



**Source:** Own authorship.

**Figure 3.31.** Evolution of the CAE_1_8's $\mathbf{RST}_{\%}$ and $\mathbf{WNRI}_{\%}$ during experiment 2.



**Source:** Own authorship.

CHAPTER 4

# CONCLUSIONS

This work presents a study of multiple CAE architectures from an ITL perspective by using the efficient matrix-based information estimators proposed in [10]. The main scope of this study was to investigate if the method for automatic estimation of the optimal dimension of the bottleneck layer in densely connected SAEs proposed in [11] could be extended for the sizing of CAEs. In order to so, the evolution of different information quantities was regularly evaluated during their training with the goal of verifying if $H(Z)$ could be used as a KPI and, if not, if another information quantity would be a suitable substitute.

The results of the experiments conducted show that, due the different natures of CNNs and densely connected ones, $H(Z)$ cannot be used as the sole KPI for the former. This is due to the fact that, even if both $H(Z)$ and $I(X;Y)$ converge during the CAEs' training and the MSE is minimized, the continuous evolution of its bottleneck's **WNRI** may indicate that the CAE is actively trying to remove redundant information from its bottleneck. If so, it would be desirable to allow it to complete this procedure (i.e., to allow the **WNRI** to converge). This is because, if the information stored in the CAE's bottleneck is mostly redundant, it defeats one of its primary purposes and hinders their performance. However, the results also shed a light over role possibly played by the CAEs' hidden layers (i.e., their down and upsampling blocks) during their training process. We hypothesise that the CAEs' actively learn to store the most common redundant information in the FMs of their hidden layers. By doing so, it filters the redundant information present in the input, which results in the flow of mostly non-redundant information towards its bottleneck and, therefore, optimization of the codes stored there. By having the larger FMs on the CAEs shallower layers, we suggest that they are forced to learn to optimally store structurally large, recurrent redundancies in them, which may explain the better results yielded by architectures designed in this fashion. Although interesting, this hypothesis is very preliminary, needing to be validated through the study of deeper, more complex CNN architectures in order investigate its validity in them. We leave this as a suggestion for future works.

We close the present work by suggesting that the evolution of the estimated values of both **RST** and **WNRI** could potentially be used as KPIs for the training of the CAEs. The results showed that, by keeping the **RST** as small as possible while progressively increasing the **WNRI** during training, the CAEs maximize the amount of non-redundant information that arrives in their bottlenecks. This suggests that, by simultaneously making so that $H(Z)$ converges to $I(X;Y)$, the MSE between input and output is minimized, the **RST**(Z) is kept as small as possible and the **WNRI**(Z) becomes as large as possible during the training process, the CAE would be able to produce optimal results once fully trained. Therefore, its optimal size would be that with by the minimum amount of hidden layers, of FMs inside both them and the bottleneck layer that allow the aforementioned goals to be accomplished during the CAE's training. Thus, in order to extend the method proposed in [11] to CAEs, one should adapt it to monitor the evolution of the previously discussed information quantities during the training process, adapting the quantity of hidden layers and their FMs as necessary. We also leave this as a suggestion for future works. Finally, the analysis and results presented in this work culminated in the following article, which was accepted for publication in 2022 *Simpósio Brasileiro de Telecomunicações* (SBrT):

Amaral, F. C. F., e Silva, D. G.: Information-Theoretic Analysis of Convolutional Autoencoders: Initial Insights. Accepted at XL Simpósio Brasileiro de Telecomunicações e Processamento de Sinais - SBrT 2022.

# REFERENCES

Alpaydin, Ethem: Introduction to Machine Learning. 4th edn. The MIT Press (2020) Cited 8 times in pages 1, 3, 6, 8, 9, 11, 16, and 17.

Principe, Jose C.: Information theoretic learning: Renyi's entropy and kernel perspectives. 1st edn. Springer Science & Business Media (2010) Cited 9 times in pages 1, 3, 4, 9, 18, 19, 20, 21, and 27.

Thomas M. Cover, Joy A. Thomas: Elements of Information Theory. 2nd edn. John Wiley & Sons (2008) Cited 2 times in pages 18 and 19.

Ian Goodfellow, Yoshua Bengio, Aaron Courville: Deep Learning. MIT Press, <http://www.deeplearningbook.org> (2016). Last accessed 19 Mar 2022 Cited 8 times in pages 2, 3, 9, 12, 13, 15, 16, and 17.

Abraham, R., Marsden, J. E., Ratiu, T. S.: Manifolds, Tensor Analysis, and Applications. 2nd ed. Springer-Verlag (1991). Cited in page 12.

Yu, Shujian, Principe, Jose C.: Understanding autoencoders with information theoretic concepts. Neural Networks, vol. 117, 104–123 (2019) Cited 7 times in pages 1, 2, 4, 27, 28, 29, and 30.

S. Yu, K. Wickstrøm, R. Jenssen, J. C. Príncipe: Understanding Convolutional Neural Networks With Information Theory: An Initial Exploration. IEEE Transactions on Neural Networks and Learning Systems, vol. 32, no. 1, 435–442 (2021). Cited 9 times in pages 2, 4, 22, 24, 25, 26, 29, 31, and 38.

Shwartz-Ziv, Ravid, and Naftali Tishby: Opening the black box of deep neural networks via information. arXiv preprint arXiv:1703.00810 (2017). Cited in page 28.

Sanchez Giraldo, Luis Gonzalo, Murali Rao, Jose C. Principe: Measures of Entropy From Data Using Infinitely Divisible Kernels. IEEE Transactions on Information Theory, vol. 61, no. 1,, 535–548 (2015). Cited 4 times in pages 22, 23, 30, and 33.

Yu, Shujian, Giraldo, Luis Gonzalo Sánchez, Jenssen, Robert, Príncipe, José C.: Multivariate Extension of Matrix-Based Rényi's $\alpha$-Order Entropy Functional. IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 42, no. 11, 2960–2966 (2020). Cited 5 times in pages 4, 5, 23, 29, and 62.

G. Boquet, E. Macias, A. Morell, J. Serrano and J. L. Vicario: Theoretical Tuning of the Autoencoder Bottleneck Layer Dimension: A Mutual Information-based Algorithm. 28th European Signal Processing Conference (EUSIPCO), 1512–1516 (2021). Cited 12 times in pages 2, 4, 18, 30, 31, 32, 33, 34, 35, 54, 62, and 63.

Siradjuddin, Indah Agustien, Wardana, Wrida Adi, Sophan, Mochammad Kautsar: Feature Extraction using Self-Supervised Convolutional Autoencoder for Content based Image Retrieval. 3rd International Conference on Informatics and Computational Sciences (ICICoS), 1–5 (2019). No citation in text.

Silva, Daniel, et al.: An Introduction to Information Theoretic Learning, Part I: Foundations.Journal of Communication and Information Systems, vol. 31, no. 1 (2016). Cited in page 21.

Silva, Daniel, et al.: An Introduction to Information Theoretic Learning, Part II: Applications. Journal of Communication and Information Systems, vol. 31, no. 1 (2016). Cited in page 1.

Parsania, Pankaj and Virparia, Dr: A Comparative Analysis of Image Interpolation Algorithms. International Journal of Advanced Research in Computer and Communication Engineering (IJARCCE), vol. 5, no. 1, 29–34 (2016). Cited in page 36.

Wiki-Art: Visual Art Encyclopedia, <https://www.kaggle.com/datasets/ipythonx/wikiart-gangogh-creating-art-gan>. Last accessed 4 Apr 2022

Cited in page 35.