Low-latency Machine learning for network traffic analysis

André Levi Ribeiro Tse

Mestrado em Engenharia de Redes e Sistemas Informáticos
Departamento de Ciência de Computadores
2022

Orientador
João Nuno Vinagre Marques da Silva, Professor Auxiliar Convidado, Faculdade de Ciências da Universidade do Porto

Todas as correções determinadas pelo júri, e só essas, foram efetuadas.

O Presidente do Júri,

Porto,______/______/________

MSC

Sworn Statement

I, André Levi Ribeiro Tse, enrolled in the Master Degree Network Engineering and Computer Systems at the Faculty of Sciences of the University of Porto hereby declare, in accordance with the provisions of paragraph a) of Article 14 of the Code of Ethical Conduct of the University of Porto, that the content of this dissertation reflects perspectives, research work and my own interpretations at the time of its submission.

By submitting this dissertation, I also declare that it contains the results of my own research work and contributions that have not been previously submitted to this or any other institution.

I further declare that all references to other authors fully comply with the rules of attribution and are referenced in the text by citation and identified in the bibliographic references section. This dissertation does not include any content whose reproduction is protected by copyright laws.

I am aware that the practice of plagiarism and self-plagiarism constitute a form of academic offense.

André Levi Ribeiro Tse

14 November 2022

Acknowledgments

First of all I would like to thank my closest family who are my parents, sisters and grandmother, who have been with me since day one and have provided me everything so that I could get this far. Without them none of this would be possible. Furthermore, I would also like to thank my remaining family for accompanying me on this journey.

I would also like to thank all my closest friends, who I will not list them one by one, but they all know how valuable they are to me and how much their contribute in my life led me to be where I am today. Additionally, I would also like to thank my girlfriend who, although she has not been involved in the whole journey, is a very important person to me and has helped me a lot.

The assistance provided by my supervisor Professor João Vinagre was greatly appreciated and I would like to thank him for believing in me, accepting me in this project and for all the support he has given me for almost a year. Besides that, I wish to thank the whole DAnon project team for the opportunity to be part of it.

In addition, I would like to extend gratitude to Lino Oliveira, my research supervisor at INESC TEC, who introduced me to the job market and has helped me a great deal, and also to all my colleagues at INESC TEC.

Last, but certainly not least, I would like to leave a special thanks to my grandfather António Ribeiro whom I called "Bu". Unfortunately, I lost him on this journey of getting my master's degree, but without him I am sure I would not be the man that I am today and I know that he would be proud of me.

To all the people mentioned here, a closing thank you, because without you none of this would be possible.

This work was partially supported by the Fundação para a Ciência e Tecnologia (FCT) under grants UIDB/50021/2020 and (DAnon) CMU/TIC/0044/2021.

Abstract

Convolutional Neural Networks, better known as CNNs, are a class of feed-forward artificial neural networks that are mostly used for image classification. However, these networks have other application areas, such as object detection, biometric authentication, machine translation, autonomous cars, among other applications. In addition, this type of networks can also be used in time series.

With an increasingly constant use in the real world it is imperative to ensure that CNNs are optimized, ensuring prediction times as low as possible and their accuracy as high as possible, to thus provide a better experience and results for those who use them. With this increasing use, several studies have been carried out in recent years to make the training of Machine Learning models faster, however, little research has been done on reducing latency times.

The Onion Router (TOR) software, that offers its users anonymity when surfing the Internet, which has caused some users to take advantage of this anonymity to engage in illegal activities. The processing of each user's traffic on TOR is done using CNNs and if these are sufficiently optimized it is possible to know which user is responsible for committing illegal activities using TOR. This example is just one of several that exist in today's world that reveal the importance of studying and optimizing CNNs for the best functioning of the various applications that depend on them.

This dissertation describes a series of several tests performed to understand which factors positively and negatively influence the execution time of a CNN, without ever forgetting the efficiency values. Furthermore, a final table was also built to reveal which of these factors are the best for network optimization and which meet the purpose of this dissertation.

Resumo

As Redes Neuronais Convolucionais, mais conhecidas como CNNs, são uma classe de redes neuronais artificiais do tipo feed-forward que são usadas na maior parte das vezes para classificação de imagens. Contudo, estas redes têm outro tipo de áreas de aplicação, tais como deteção de objetos, autenticação biométrica, tradução automática, automóveis autónomos, entre outras aplicações. Este tipo de redes pode também ainda ser usado em séries temporais.

Com uma utilização no mundo real cada vez mais constante é imperativo garantir que as CNNs são otimizadas, garantindo tempos de execução tão baixos quanto possíveis e a eficácia das mesmas o alta possível, para assim proporcionar uma melhor experiência e resultados a quem usufrui delas. Com esta crescente utilização, vários estudos foram realizados nos últimos anos de modo a tornar o treino de modelos de Machine Learning mais rápido, no entanto, pouca investigação foi feita no que toca à redução dos tempos de latência.

O software The Onion Router (TOR), que oferece anonimidade aos seus utilizadores na navegação da internet, o que fez com que alguns utilizadores se aproveitassem desta anonimidade para a prática de atividades ilegais. O processamento do tráfego de cada utilizador no TOR é feito com recurso a CNNs, e se estas estiverem suficientemente otimizadas é possível saber que utilizador é responsável pela prática de atividades ilegais com recurso ao TOR. Este exemplo, é apenas um dos vários que existem no mundo atual que revelam a importância de estudar e otimizar CNNs para o melhor funcionamento das várias aplicações que dependem delas.

Esta dissertação descreve uma série de diversos testes efetuados para perceber que fatores influenciam positivamente e negativamente o tempo de execução de uma CNN, sem nunca esquecer os valores de eficácia. Para além disso, foi também construída uma tabela final que revela quais destes fatores são os melhores para a otimização da rede e que vão ao encontro do propósito desta dissertação.

Contents

List of Tables

List of Figures

Listings

Acronyms

ML	Machine Learning
CNN	Convolutional Neural Network
TOR	The Onion Router
AI	Artificial Intelligence
NAS	Neural Architecture Search
DL	Deep Learning
FC	Fully Connected
IPD	Inter-Packet Delays
RNNs	Recurrent Neural Networks
CNNs	Convolutional Neural Networks
OS	Onion Services
ANNs	Artificial Neural Networks
LEAs	Law Enforcement Authorities
ISPs	Internet Service Providers

Chapter 1

Introduction

1.1 Motivation

Model accuracy is the main topic of Machine Learning (ML) research, however, many systems in the world must adhere to rigid latency specifications at the time of prediction. While much work has gone into making model training faster using more dependable and user-friendly software libraries that can fully utilize the available computational resources, comparatively little research has gone into prediction latency. Convolutional Neural Network (CNN) and Recurrent Neural Networks (RNNs) are two examples of complex deep neural architectures that have delays at prediction time that are incompatible with many high throughput low latency applications.

In this context, we present The Onion Router (TOR) which is a free software that uses the onion routing mechanism to offer anonymity. With the use of several encryption layers, users' transmitted and received data can be hidden by using an internet traffic redirection technique called onion routing that sends their traffic to specific servers. These techniques are used to anonymize the identity and data of the sender and the recipient. In addition, TOR was primarily developed to assist internet users with issues related to their right to privacy and freedom of speech, such as network surveillance that jeopardizes their personal identity and privacy and unlawful traffic tapping [40].

However, the fact that TOR provides this sort of privacy also has its downside, as it allows users to access illicit content and markets, such as child pornography or firearms dealing. With the use of TOR, these users can easily bypass governmental or ISP restrictions. Furthermore, TOR can also be used for other negative purposes besides the previous examples mentioned, such as spreading malware and hacking illegally while remaining anonymous [40].

With more than 2 million daily users [2], this means that TOR is already very well known and used by internet users which makes this problem quite worrisome. Motivated by the problem described, we are inserted in a project with a team that seeks to develop tools to deanonimize TOR Onion Services (OS) sessions. Such tools can potentially expose TOR's vulnerabilities and help acquire knowledge that can contribute to benefit licit uses such as privacy, censorship circumvention and mitigate potential criminal activities such as human traffic and child pornography.

The main method employed in a wide variety of attacks researched against TOR is flow correlation. This means that an attacker must see a portion of flows entering and leaving the TOR network in order to perform flow correlation. If the attacker is successful in intercepting both the ingress and egress portions of a given TOR connection he can then deanonymize that connection. Therefore, by attempting to intercept a larger percentage of TOR's entry and egress flows, an adversary can raise his chances of deanonymizing TOR connections [36].

Now, since Convolutional Neural Networks (CNNs) are used to learn a correlation function for TOR's noisy network, as network flow features can be modeled as time series, and the CNNs are known to have good performance on time series. If these networks are as optimized as possible, it will undoubtedly help in solving the problem.

With the whole problem and motivation explained, this dissertation project will put a strong emphasis on CNNs while concentrating on ML model inference optimization and through this, numerous applications can benefit from CNNs ability to reduce latency at inference time. Within a diverse team working on illicit hidden service discovery on TOR networks, this work will be especially focused on models for high throughput network traffic analysis.

1.2 Main goals

In this dissertation, we defined the following goals to be accomplished by the end of the project:

- To study the state of the art on latency optimisation of ML models;
- To define a methodology to develop and evaluate ML models under strict latency constraints;
- To study complexity-accuracy trade-offs of different ML methods;
- To apply the methodology to multiple use cases, with special emphasis on network traffic analysis with CNN;
- To optimize DeepCorr, a flow correlation system;
- To evaluate multiple approaches and report the results.

1.3 Contributions

The execution of this dissertation has achieved the following contributions:

- A methodology to develop and evaluate ML methods within strong latency constraints;

- Study of the various hyperparameters that make up a CNN, to understand which of these positively influence the prediction time of the network;
- Study of various architectures of a CNN, in order to find architectures that are beneficial in terms of CNN optimization;
- Software contributions to the DAnon project.

1.4 Dissertation layout

This dissertation is divided into six chapters which are going to be briefly explained. In Chapter 2 are presented the fundamental concepts necessary to understand the work being presented, namely ML, Deep Learning (DL), a detailed description of CNN and the layers that make up this network and also time series forecasting.

In Chapter 3 a comprehensive state-of-the-art and literature review is presented to sustain the decisions made. The studies already carried out in the field of CNN optimization as well as a detailed description of them is what can be found in this chapter.

Chapter 4 presents the proposed methodology used in this dissertation, as well as an explanation of the dataset used for the experiments that were performed. In this chapter there is also a small test using this methodology that proves its reliability.

In chapter 5, are all the tests and experiments performed using the proposed methodology described in the previous chapter. A careful analysis of these results and explanations for the collected results are also given.

Finally, the conclusions obtained are reported in Chapter 6, along with possible lines of development that may be applied in future work.

Chapter 2

Fundamental concepts

In this chapter, we will present fundamental concepts for a clearer understanding of this whole dissertation. A brief description of Machine Learning (ML), Deep Learning (DL), Convolutional Neural Network (CNN) and other important concepts is what you will find here.

2.1 Machine Learning

ML is a branch of Artificial Intelligence (AI) which was born from the premise that computers could look at some specific data and generate rules in order to perform a task, all this by itself[12].

In classical programming, the user obtains answers by inputting data and some rules to be processed by this data. ML also takes as basis data, rules and answers, however, these are used in different orders. It is only given the machine data and the answers to the problem that one wants to solve and the system will automatically define rules that will autonomously solve the problem based on a statistical structure found on the data and answers given. ML is typically used when tasks are too complex to program or also when tasks are beyond human capabilities.

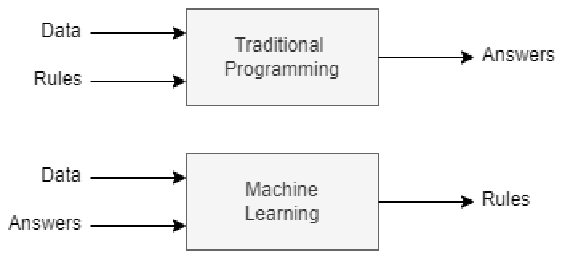

Figure 2.1: Machine Learning Vs Traditional Programming

This may be difficult to understand, but, take this as an example: You want to convert degree Celsius to Fahrenheit, using ML. So you will give the system data (numbers representing degree Celsius) and answers (numbers that correspond to the conversion of the degree Celsius to Fahrenheit). Using these two elements the system will then create a formula (rules), thus the next time you input a number, corresponding to a degree Celsius, it will give you the analogous Fahrenheit number.

Nowadays, ML has become very popular and it is widely used in various fields that demand information extraction from large datasets. There are numerous examples of that such as Tesla's car that use ML to detect obstacles, traffic lights and others cars in order to allow the car to drive itself. There is also Amazon Alexa which was trained to recognize voice commands and execute orders and tasks, and many other examples of the application of ML in real life.

Learning, however, is a extremely vast subject. Therefore, ML has branched into multiple sub-fields that deal with various sorts of learning problems. So, four parameters were defined in order to categorize learning paradigms.

2.1.1 Supervised learning

Similarly to the previous example, in supervised learning, an input data is provided to known targets, and from here, the algorithm will then adjust the parameters, iteration after iteration with the purpose of making the value of the loss function as close to zero as possible [12][6]. The open parameters of the ML model are then calibrated using the pairs of input and output data from the training set. Following a model's successful training, it can be used to forecast the target variable y using fresh or previously unobserved data points of the input attributes x [24]. To help better understand this concept, in figure 2.2 is illustrated an example of how supervised learning works.

To shed some light on what the loss function is, it's a technique for assessing how well a particular algorithm models the supplied data. This function would produce an extremely big number if forecasts differ too far from the actual outcomes. The loss function gradually gains the ability to lower prediction error with the use of some optimization function [38].

In supervised learning, it is important to avoid overfitting, which takes effect when a model learns the detail and noise in the training data exhaustively that impacts the performance of the model on new data, in an adversely way. Moreover, it's possible to split supervised learning algorithms into classification and regression algorithms. On one hand, regression it is based on predicting continuous variables, in other words an uncountable set of values, like age or salary. On the other hand, classification algorithms are used to predict or classify the discrete values, that is to say that information can only take certain values such as Male or Female, True or False, and other examples. Although classification and regression make up the majority of supervised learning, there are also more unusual variations, such as sequence generation, object detection, image segmentantion, and others.

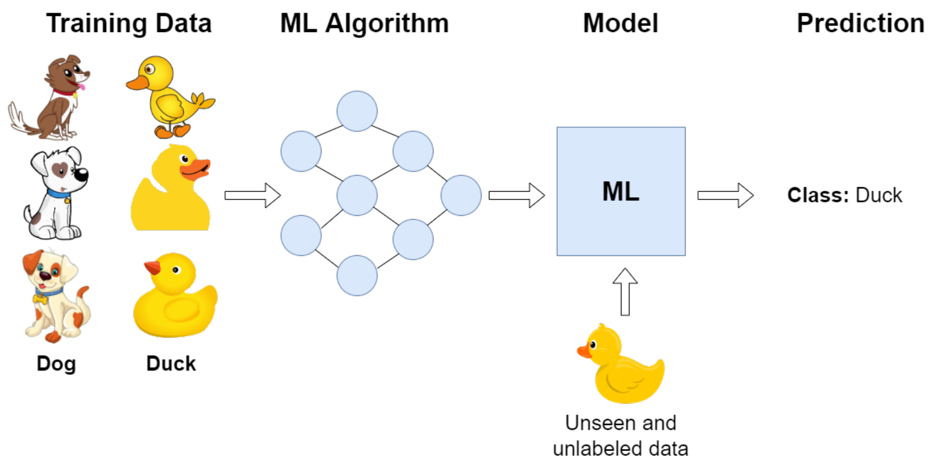

Figure 2.2: Brief example of how Supervised learning works

2.1.2 Unsupervised learning

Another field of ML is Unsupervised learning, and unlike supervised learning, here there is the input of data without the providing of targets, this approach is used to the better understanding and comprehension of a dataset before seeking to solve a supervised learning problem. As a result, training data only includes variables x with the intention of discovering structural information of relevance, such as collections of pieces with similar characteristics (known as clustering) or data representations that are projected from a high-dimensional space into a lower one (known as dimensionality reduction) [24].

Given the previous information, it is important to mention that the purpose of clustering is to find some structure in the dataset, for example, looking at the image below, it is easy to identify 3 groups. In clustering, similarity is a crucial concept, with this being said, points that are on the same group are similar, consequently they share some properties. A good example of the application of unsupervised learning, is for example in electronic markets where clustering techniques are applied to group customers or markets into segments for the purpose of a more target-group specific communication [24].

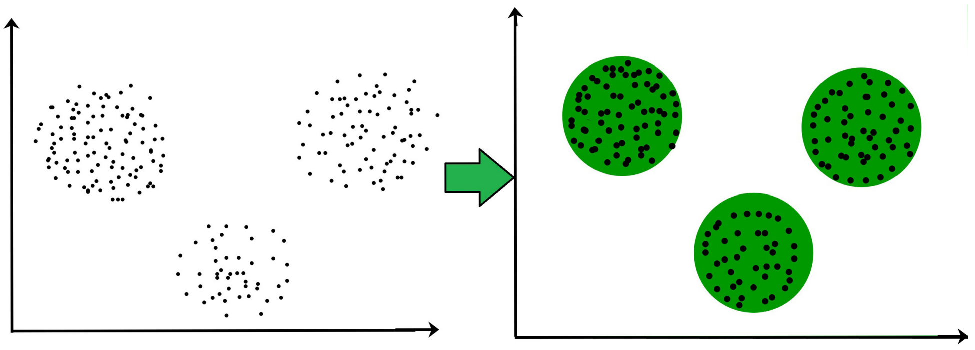

Figure 2.3: Example of how clustering works (https://www.geeksforgeeks.org/clustering-in-machine-learning/, accessed on 7 April 2022).

2.1.3 Self-supervised learning

Self-supervised learning or semi-supervised learning, is very similar to Supervised learning, however it learns mainly from unlabeled data as the number of unlabeled examples is far superior to the number of labeled examples. The unlabeled examples are produced from the input data using normally a heuristic algorithm and the goal is the same as the last two methods described, which is to help the learning algorithm to find a better model [10] [12].

The ability of self-supervised learning to make use of the massive volumes of unlabeled data that become accessible in the big data era is essential to its success. Now is the moment for deep learning algorithms to stop relying on human supervision and resume self-supervision of the data. Inherently occurring co-occurrence correlations in the data are used as the self-supervision in self-supervised learning, which has the potential to be flexible. For instance, a well-trained language model would predict "eating" for the blank in the incomplete sentence "I like ______ apples" because it regularly co-occurs with the context in the corpora [32].

Despite demonstrating encouraging results when compared to fully-supervised procedures, purely self-supervised techniques provide much worse visual representations. As a result, they have a limited range of practical applications and self-supervision is still insufficient [49].

2.1.4 Reinforcement learning

The environment's feedback is another foundation for reinforcement learning. The information is more qualitative in this instance though and does not aid the agent in calculating an exact measurement of its inaccuracy. This feedback which is frequently referred to as a reward in reinforcement learning (sometimes a negative one is referred to as a penalty), is helpful in determining if a certain action carried out in a condition is positive or not. To constantly make the best choice in terms of the highest immediate and cumulative reward the agent must understand the policy of the most useful sequence of activities. In other words, even if a course of action is flawed, it must nevertheless provide the best overall reward in the context of global policy. This approach is predicated on the notion that a rational agent always seeks out goals that will improve his or her income. Reinforcement learning is especially effective when the environment is not totally deterministic, when it is frequently very dynamic and when a precise error measure is unattainable [6]. In figure 2.4, is a schematic representation of how reinforcement learning works as well as the interaction between the agent and the environment.

Although, Reinforcement learning is currently mostly a research area and has not yet had significant practical successes beyond games, it is a matter of time before it is applicable in various real world applications such as self-driving cars, robotics, resource management, education, and so on [12].

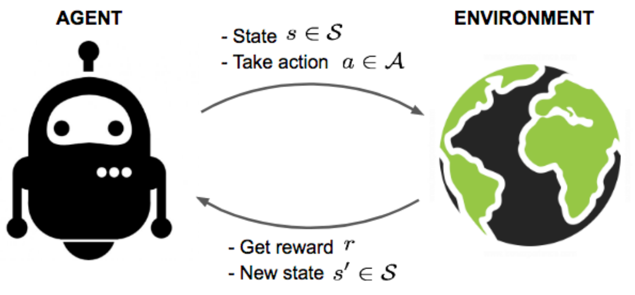
AGENT
ENVIRONMENT
- State $s \in \mathcal{S}$
- Take action $a \in \mathcal{A}$
- Get reward r
- New state $s' \in \mathcal{S}$

Figure 2.4: Illustrative picture of how reiforcement learning works where an agent interacts with the environment, trying to take smart actions to maximize cumulative rewards (https://lilianweng.github.io/posts/2018-02-19-rl-overview/ , accessed on 12 September 2022).

2.2 Overfitting and underfitting

In every ML situation overfitting occurs and for ML to be mastered, dealing with overfitting must be learned [12]. When a model has too much capacity and can no longer properly generalize given the initial dynamics provided by the training set, this is called overfitting. When an unknown input is given the related prediction error can be very high, yet it can virtually flawlessly link all known samples to the appropriate output values [6].

The conflict between generalization and optimization is the core problem in ML. Generalization, as opposed to optimization, refers to how well a trained model performs on data it has never seen before. Optimization refers to the process of changing a model to attain the greatest performance possible on the training data. Naturally, the objective of the game is to achieve good generalization but there's control on generalization and all that is to do modify the model using the training data. Generalization and optimization are associated from the beginning of training: the smaller the loss on training data, the lower the loss on test data. The model is considered to be underfit (which occurs when the model is unable to adequately represent the dynamics revealed by the same training set [6]) while this is taking place because there is still work to be done and the network hasn't fully modeled all pertinent patterns in the training set.

However, generalization stops improving after a certain number of iterations on the training data and validation measures stagnate before declining, indicating that the model is beginning to overfit [12].

As mentioned, overfitting is a very common problem in ML, fortunately, there are already several techniques that have been studied to avoid this type of problem.

2.3 Neural networks

A Neural network is a directed structure that connects an input layer with an output layer. It is named this way because it is a reference to neurobiology and also an attempt to simulate the decision-making process in the human brain consisting of a biological neural network made of neutrons. Furthermore, Neural networks can learn and model relationships between inputs and outputs that are nonlinear and complex making problems that are difficult easy to solve using fairly simple computational operations, such as additions, fundamental logic elements and multiplication [17].

This type of networks has three different types of layers which are:

- Input Layer;
- Hidden Layer, that could be one or more layers;
- Output Layer.

Now that we have knowledge of the three layers that make up a neural network, it will be easier to understand how these networks operate.

Starting from the input layer, each node is connected to the other nodes and has a weigh and threshold linked with it. Every node also produces an output, and if that output exceeds an established threshold value the node is activated and data is sent to the next tier of the network. Otherwise, no data is sent on to the network's next tier.

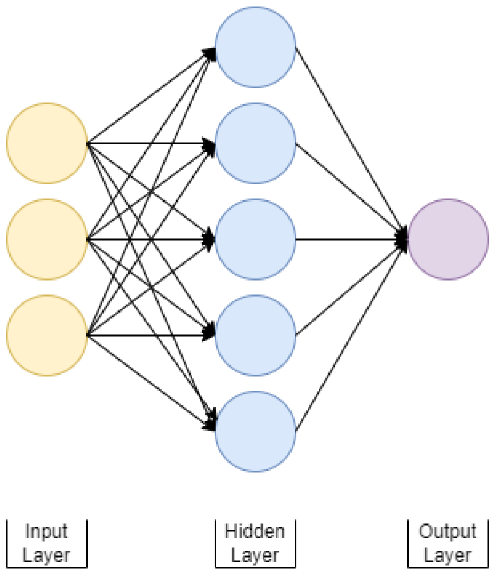

Figure 2.5: Diagram of a Neural Network

In terms of real life application, Neural networks are very useful tools in AI and computer science as they allow to classify and cluster data at high speeds and it also play an important part in speech and image recognition [14].

The training process of a neural network is built around the following components:

- The **input data** and and the respective associated **targets**;
- Several **layers**;
- The **Loss function** that gives the prediction error of the network, which then will give feedback to the optimizer;
- The **optimizer** receives feedback from the loss function and then update the network's weight in order to improve the network.

2.4 Deep Learning

The Machine Learning's offspring, Deep Learning, is a larger family of methods and algorithms that aim for general intelligence while achieving specialized intelligence. Deep Learning typically suggests that the approach is being used to handle issues that require more generalization, such as computer vision and speech recognition, or that the approach is trying to solve a problem in a more generic fashion, like spatial reasoning. People are good at finding solutions to general problems, like us humans can, for instance, match visual patterns in virtually any situation. Along with supervised learning, unsupervised learning, and reinforcement learning, Deep Learning also addresses these issues. Typically, Deep Learning systems use numerous layers of Artificial Neural

Networks (ANNs) and this number of layers is what defines the so-called depth of the model. The layers work together to solve difficult problems in order to achieve a broader purpose. Each layer handles specific challenges by utilizing distinct levels of intelligent components. For instance, recognizing any object in an image is a general issue but it can be divided into comprehending color, recognizing object forms, and recognizing relationships between objects to accomplish a task [20].

In figure 2.6, the difference between ML and DL is illustrated, however, it is worth to enunciate in written form some of the differences that separated DL (a subsection of ML) from ML:

- For ML to produce results more constant human engagement is needed. Although more difficult to set up, DL requires less intervention once it is running;
- DL systems require extra setup time but produce results immediately (although the quality is likely to improve over time as more data becomes available). ML systems can be swiftly set up and run but their effectiveness may be constrained;
- ML typically uses conventional techniques like linear regression and calls for organized data. Neural networks are used in DL, which is designed to handle enormous amounts of unstructured data;
- While DL systems need far more powerful hardware and resources, ML applications are frequently less sophisticated and can run on standard computers [34].

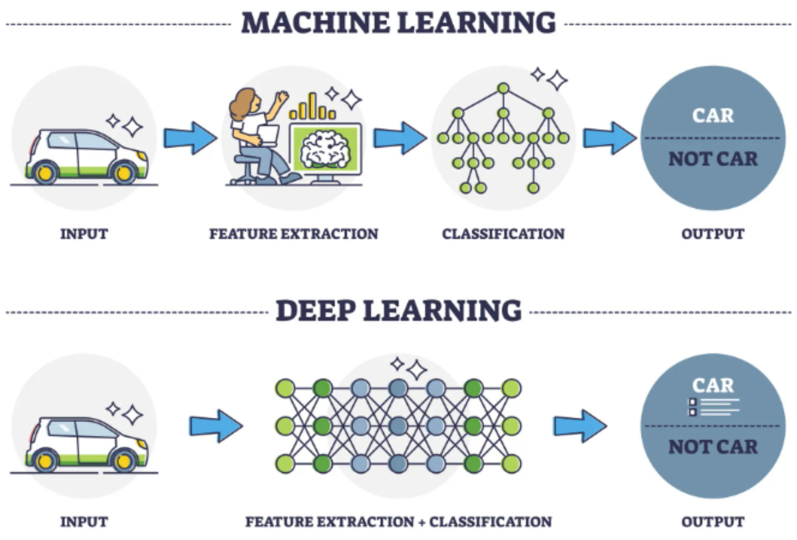

Figure 2.6: The difference between Deep Learning and Machine Learning (https://www.turing.com/kb/ultimate-battle-between-deep-learning-and-machine-learning, accessed on 16 August 2022).

2.5 Convolutional Neural Networks

Convolutional Neural Networks (CNNs) appear as one of the types of DL networks which are used for several areas such as image identification and classification, face detection, self-driving or autonomous cars, document classification and many other useful areas [33]. CNNs key advantage over its forerunners is that it recognises the pertinent elements automatically and independently of human intervention [4].

Equivalent representations, sparse interactions, and parameter sharing were the three main advantages that Goodfellow et al. [16] cited for the CNN. In contrast to typical fully connected networks, the CNN uses shared weights and local connections to fully exploit 2D input-data structures, such as visual signals. This technique uses a remarkably minimal amount of parameters which speeds up the network while also making training easier.

There are different architectures of CNNs, however the basic components are quite similar which are three types of layers. These are convolutional layers, pooling layers and dense layers (or Fully Connected (FC) layers) [37]. In the figure 2.7 it is possible to see an illustrative scheme of the standard architecture of a CNN. On a side note, it should be noted that the big difference between a neural network and a CNN is that whereas in a neural network each neuron in the network is connected to every other neurons, in a CNN only the last layer is fully connected.

Using the CNN shown in the figure 2.7 we can divide it into 3 areas:

1. The first and the main building piece of a CNN is the **convolutional layer** and it is responsible for the majority of the network's computational burden. This layer does a dot product between two matrices, one of which is a set of learnable parameters known as a kernel and the other is the limited section of the receptive field [35].
2. The **pooling layer** will just downsample the input along its spatial dimension, bringing the number of parameters in that activation even lower [37].
3. The **Dense layers** will then carry out the ANNs normal functions and try to create class scores from the activations for categorization.

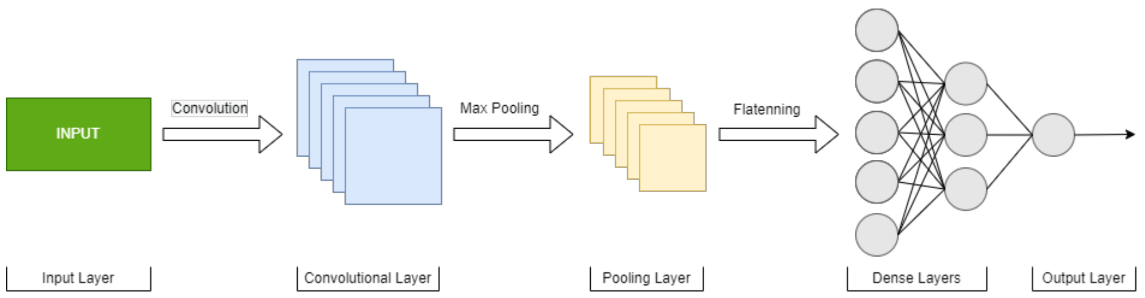

Figure 2.7: Illustrative schema of the functioning of a CNN

2.5.1 Convolutional Layer

How CNNs function depends critically on the convolutional layer that performs feature extraction, which combines linear and nonlinear processes, or more specifically activation functions and convolutional operations [47].

Convolution is a form of linear operation used for feature extraction in which a tiny array of numbers called a kernel is applied over the input, which is an array of numbers called a tensor. A feature map, also known as an output value in the corresponding place of the output tensor, is obtained by computing an element-wise product between each element of the kernel and the input tensor at each point of the tensor and summing it [47]. An illustrative image of this process can be seen in the figure 2.8.

This process is repeated while applying various kernels to create an infinite number of feature maps that represent various properties of the input tensors; various kernels can thus be thought of as various feature extractors. It should also be noted that two hyperparameters that define the convolution operation are size and number of kernels. Another important hyperparameter is the stride which also describes the convolution operation and is the distance between two subsequent kernel points [47].

There is also the zero-padding hyperparameter which is a quick and efficient way to give more control over the dimensionality of the output volumes by padding the input's boundary [37].

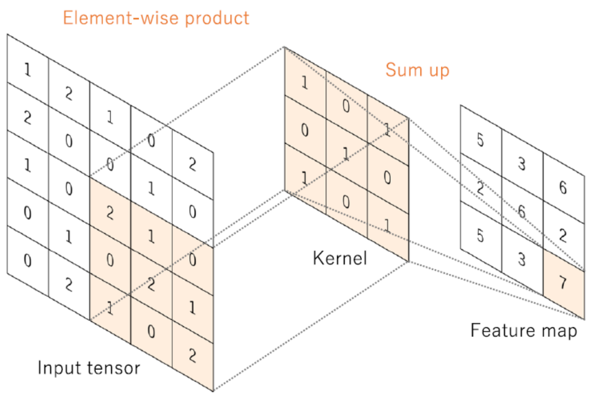

Figure 2.8: An example of convolutional operation with a kernel size of 3x3, no padding and stride of 1 [47].

2.5.2 Pooling Layer

A pooling layer offers a standard downsampling method that lowers the feature maps' in-plane dimensions to introduce translation invariance to slight shifts and distortions and limit the number of ensuing learnable parameters. It is important to notice that none of the pooling layers have a learnable parameter, whereas filter size, stride, and padding are hyperparameters in pooling operations, much like convolution operations [47].

Max pooling is the most widely used type of pooling operation and it takes patches from the input feature maps, outputs the largest value in each patch and discards all other values as can be observed in the figure 2.9.

It is also crucial to realize that CNN architectures may also include general-pooling in addition to max-pooling. L1/L2-normalization, average pooling, and other common operations can all be carried out by the pooling neurons that make up general pooling layers [37].

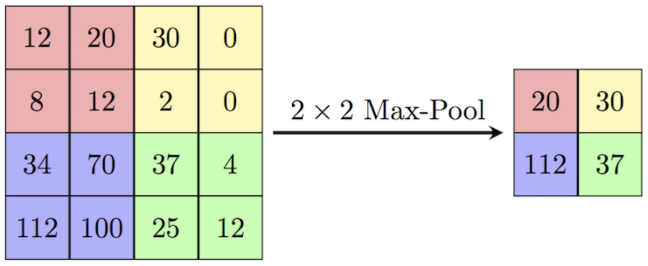

Figure 2.9: An example of of how max pooling works (https://computersciencewiki.org/index.php/File:MaxpoolSample2.png accessed on 19 September 2022).

2.5.3 Dense Layer

Following the pooling layer, the final convolution or pooling layer's output feature maps are often flattened, which is converting the data into a one-dimensional array for inputting it to the following layer [26]. After flatenned they are connected to one or more dense layers, also known as FC layers, in which each input is connected to each output by a learnable weight. A subset of fully connected layers then maps the features provided by the convolution layers and the downsampling layers to the network's final outputs, such as the probabilities for each class in classification tasks. The number of output nodes in the final fully connected layer normally equals the number of classes. A nonlinear function, such as ReLU (or any other), is followed by each completely linked layer.

2.6 Time series forecasting

The practice of studying time series data using statistics and modeling to produce forecasts and inform strategic decision-making is known as time series forecasting. It is not always an accurate prediction, but forecasting can give us a real prediction on what events will occur most and less likely compared to other possible outcomes [1].

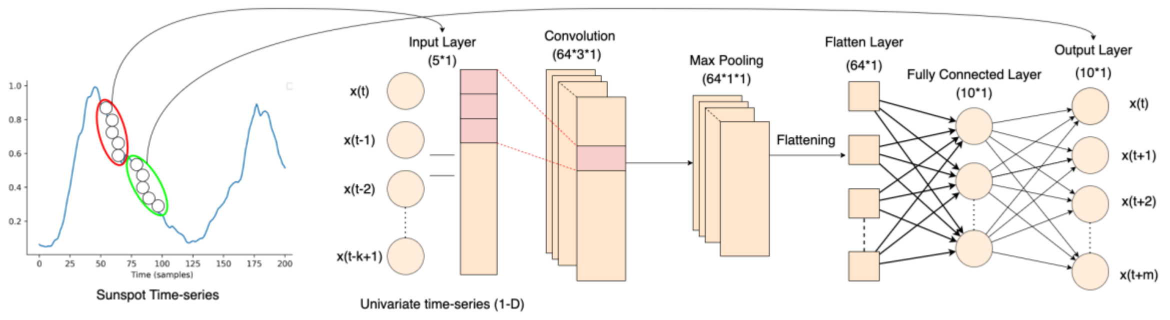

Figure 2.10: One-dimensional convolutional neural network for multi-step ahead time series prediction [11]

Although CNNs are known to be applied mostly to computer vision problems this type of network shows excellent results when it comes to time series forecasting. Because CNN offers dilated convolutions in which filters may be used to compute cell dilations, it is useful for forecasting time series data. The neural network can better comprehend the correlations between the various observations in the time series thanks to the magnitude of the gap between each cell [46].

Its robustness but also low training times compared to other more complex architectures also make it a good reason to use this architecture for time series problems [22].

2.7 Summary

This chapter has introduced several concepts and topics such as ML, Neural Networks, Deep Learning and CNN as well as a detailed description of what happens at each layer of a CNN. The information obtained in this chapter will be extremely important for the understanding of the rest of the dissertation.

In chapter 3 we will be showing some research already developed on the optimization of CNN and use that knowledge throughout the rest of the dissertation.

Chapter 3

State of the Art

Convolutional Neural Network (CNN) have been shown to have an impact in several areas through good implementation. A factor that may be determinant for these networks to be better and achieve better results is the response times they present.

In this case, is intended to use Convolutional Neural Networks (CNNs) in the context of time series, as these present quite positive results in the indicated context. With the optimisation of this type of network, it is possible to contribute to the resolution of some current problems, such as, for example, the greater ease in identifying users of private networks (for example The Onion Router (TOR)) that may be using the same private networks to commit illegal acts [36].

This chapter contains literature review related to the optimization of CNNs through several studied methods.

3.1 Width Vs Depth

An approach aimed at optimising CNNs would be the adjusting of width and depth values of the network. Questions on whether network models should be wider or deeper have been going on and on for years [29] and opinions vary as to which of these two attributes is more relevant for the model to show the best results.

It is worth mentioning that the depth of the net corresponds to the number of layers in the net while the width refers to the number of neurons in a given layer [7].

In 2007, Bengio and Le Cun [5] proved that models with few depth and a larger width, the so-called shallow models, demand exponentially more components than models with a higher depth.

In the same year, Larochelle et al. [29] performed multiple experiences using different models with deep architectures on the MNIST dataset, which is a large database of handwritten digits that is commonly used for training various image processing systems [3], and came to the conclusion the deep architectures models have shown great results. In the vast majority of cases these type of architectures outperformed shallow architectures like SVMs, NNET and others. The only example where shallow architectures were superior, was when used on images from the MNIST to which rotation motion was applied. Larochelle et al. justify this admirable result due to the fact that were a large number of samples in the training set and also because, there was only one factor applied.

A more recent study done by Zagoruyko and Komodakis [48] contradicts what the previous article mentioned attaching more importance to width than to depth. In a study where multiple image datasets were used, they tried to adjust the depth and width of residual networks also taking into account the number of parameters and the results were pretty interesting:

- The first conclusion to draw is that unless the number of parameters is unreasonably high it does not affect the effectiveness of the model;
- Secondly, it was proved that the widening factor is more relevant since increasing its value almost always showed improvements in residual networks with different depths. The only example where this was not the case was when the number of parameters was too high;
- Finally, when the number of parameters becomes too high, stronger regularization is required.

In terms of prediction times this study also concludes that wide networks also have better prediction times, even giving the example of two different networks. Comparing a thin and a wide network, for both to achieve the same efficiency, the thin had an prediction time 8 times slower than the wide, thus revealing the great advantage of the wide networks [48].

3.2 Dropout

Adding a dropout layer to a CNN might be another way of ensuring that they have better results. Dropout is basically an approach to reduce overfitting and an extremely efficient way of operating model averaging with neural networks [18].

Even though there have been no studies done on the prediction times when adding a dropout layer to a CNN, its accuracy has been proven with a decrease in test error in the majority of the cases this layer is applied but especially when the optimal value of depth and width is found and then this layer is applied [48].

3.3 Different architecture designs

Lebedev and Lempitsky [31] have conducted a study on the different approaches already in place to speed up CNNs. Therefore, there are several architectures that have been studied and developed, but I will only mention a few.

AlexNet is the name of one of the best-known architectures in the world, for the extraordinary results it proved to have at the time it was developed. Moreover, the SqueezeNet architecture [21] has shown to have the same performance as AlexNet with 50 times less parameters using three essential rules:

- Use 1x1 filters whenever possible, this will allow the model to be small and fast as smaller filters need fewer operations and have fewer parameters;
- Reduce the number of input channels when 3x3 filtering is required, just like in the previous rule, this allows the model to be small and fast;
- Late in the network, downsample to allow most layers to function with large high-resolution maps. This will boost the acurracy of the model.

Putting these three rules together, this model has a shorter inference time than AlexNet.

Another architecture is MobileNet [19] which is based on changing the the network width and input resolution, proving to be a simple method for managing trade-offs between speed and quantity of parameters on the one hand and precision on the other.

In comparison to the previous architecture, with the same model size and one tenth of the mult-add operations MobileNet can achieve greater accuracy. However, because depth-wise convolution is not as effectively implemented on GPU as standard convolution, this advantage in operation number is difficult to transfer into real inference speed-up on GPU.

The EffNet architecture [15] is a variation of MobileNet building block and is based on the following principles:

- Use depthwise separable convolutions, then takes it even farther by breaking a 3x3 convolution into two 1x3 and 3x1 kernel convolutions;
- Apply downsampling along one dimension is performed with strided convolution and along other using 2x1 pooling;
- Reject convolutions with filters larger than 1x1.

Table 3.1 gives a short summary of the different CNN architectures, containing their main characteristics.

Architecture Design	**Main Principles**
AlexNet	- 8 layers with learnable parameters; - 5 convolutional layers with a combination of max-pooling layers; - 3 fully connected layers; - Activaction function used in all layers is ReLU; - 2 Dropout layers; - Activaction function used in the output layer is Softmax.
SqueezeNet	- Use 1x1 filters whenever possible; - Reduce the number of input channels when 3x3 filtering is required; - Late in the network, downsample to allow most layers to function with large high-resolution maps.
MobileNet	- Use depthwise separable convolutions; - Use 1 by 1Point-wise convolution.
EffNet	- Use depthwhise separable convolutions; - Apply downsampling along one dimension is performed; with strided convolution and along other using 2x1 pooling; - Reject convolutions with filters larger than 1x1.

Table 3.1: Summary table of the different architecture designs

3.4 Automatic architecture search

CNN brings the problem that the evaluation function is too expensive due to the fact that the number of hyperparameters is too large. Given this problem, automatic architecture search arises, which is basically a hyperparameter optimization.

The Neural Architecture Search (NAS) algorithm, that uses reinforcement learning, was the first effective attempt at automatic architectural search. The NAS biggest problem is the fact that even in tiny datasets it requires a significant amount of computational resources, making this architecture impossible for datasets that are much too large.

Larger datasets can be tackled in two different ways: either by developing more efficient search algorithms or by guaranteeing that results achieved on smaller datasets can be applied to larger datasets. This method requires only the architecture of a single block to be predicted rather than the entire network, resulting in a substantially smaller search area making NAS faster [31].

It is possible to further speed up the NAS either anticipating the architecture's eventual performance based on the initial epochs in the training process or exchanging parameters between multiple designs. Nevertheless, all different approaches that split evaluation and architecture construction demand evaluating a high number of CNNs whose resource consumption can be reduced if both the architecture and the CNN training is done at a time [31].

3.5 Quantization

Quantization is a simple approach to compress and decrease the memory requirements of a neural network while also speeding up computations [31].

Hinton et al. [18] pointed out that neural networks are not only resistant to noise but that noise insertion can also improve training performance. Therefore, if the quantization process is appropriately adjusted, noise-like distortions introduced by it may not be harmful. With this being said, quantization is a potential strategy for speeding up neural networks if the hardware enables it.

Compression generally goes hand-in-hand with speed enhancement and certain strategies are applicable to both convolutional and fully connected layers, making the study conducted by Jegou et al. [25] extremely important for speeding up CNNs where they present binarization and scalar quantization. Simple scalar quantization and product quantization showed great results and to mention also that binarization is a very auspicious approach due to its simplicity and high compression rate.

Following the same line of thought, it is feasible to reduce the CNN capacity (number of nodes and layers) using regularization, dropout or by reducing the number of filters which will lead to compression and ultimately speed-up. However, there is a great dependence on the size of CNN, because accelerating smaller CNNs is always easier than in larger ones [31].

Courbariaux et. al [13] suggests producing binarized weights numerous times as a test time to generate an ensemble of models, however, assembly is incompatible with the speed-up target, and keeping full-precision weights is incompatible with compression. Forward propagation with binarized weights can be much faster, but once again, the results are only reasonable for small datasets. An efficient attemp at binarizing largen CNNs is the XNOR-Net [39]. However, comparing, for example, a full precision AlexNet and a XNOR-Nets AlexNet, the accuracy drops dramatically from the first to the second, suggesting that binarization suggesting that although binarization gives us significant gains in speed, accuracy is also significantly compromised which in some cases may not be the most advisable.

To conclude, both weight quantization and binarization are efficient techniques for CNN compression. Quantization of CNNs weights or activations clearly speeds up calculations, although the amount of time saved varies on the low-level implementation. Most researchers do not disclose such implementations, and when they do, it appears that existing floating point implementations are quite highly tuned and the real speedups offered by quantization approaches are not nearly as large as the operation-count based forecast predicts [31].

3.6 Pruning

Pruning is an approach used to reduce the complexity of convolution operations, thus leading to acceleration of CNNs.

Although there are many different ways of prunning most of them follow the same pipeline that consists of:

1. The significance of neurons is determined using a criteria. The neurons that are not as vital are pruned;
2. Fine-tune the network and the accuracy decrease is somewhat recovered.

In addition to what was explained above, it is also required to make 3 choices when it comes to implementing the pipeline, which are:

- Choose the desired sparsity structure;
- Select the importance pruning norm;
- If the select approaches uses one, determine a sparsity-inducing regularizer [31].

3.7 Teacher-student approaches

The concept behind the teacher-student technique is that instead of training on labeled data, a CNN model might be trained on the outputs of another model (the so-called teacher). This method allows knowledge to be transferred from one model to another and unlabeled or synthetic data to be incorporated into the training process.

Another natural application of the teacher-student technique would be to transfer information from big, slow, and accurate models to small, rapid models.

Based on these principles, a paper published by BuciluÂ et al. [9] suggests that an ensemble of models be compressed into a single neural network. When compared to a single model, an ensemble is supposed to be more resistant to overfitting, so, the first step is to train an ensemble of classification models on a certain annotated dataset. This ensemble is used to label a huge quantity of synthetic data created by various basic random sampling techniques and then a single model is taught on the synthetic dataset as a consequence.

Although the authors of this paper point out that this approach can alleviate the overfitting problem for neural networks, this approach is limited if the datasets are too large or the models are too large to build ensembles.

3.8 Fine-Tuned CP-Decomposition

Fine-Tuned CP-Decomposition is a tensor decomposition scheme which aims to accelerate convolutions through, as the name implies, Fine-Tuning and CP-Decomposition. It should be noted that this approach is more complex than the previous ones mentioned.

On one hand, Fine-Tuning is based on transfer learning which consists of using features used to solve a previous problem and using them to solve a new similar problem. Fine-Tuning is an optional step in the transfer learning process that achieves very positive results and does not take long execution times [44] [27]. On the other hand, CP-Decomposition basically factorizes a tensor into a linear combination of rank one tensors [28].

Having said this, Fine-Tuned CP-Decomposition method can be split into two stages. In the first stage, CP-Decomposition is used to decompose the kernel of a select a selected convolutional layer, it is also possible to do the CP-Decomposition with non-linear least squares. In the second stage, fine-tuning is applied to the all network using backpropagation [30].

When it comes to results, Lebedev et al. [30] have tried to apply this method to already constructed CNNs. They applied their method to the layers of the CNN described in (Jaderberg et al. [23]) and the results were a success as they achieved a network 8.5 faster than the previous one with just a drop of 1% in accuracy.

However, applying their method to the second convolutional layer of AlexNet, a way larger network, the results were not that appealing since they were unable to find a good learning rate due to the instability of the low-rank CP-decomposition.

It was therefore, concluded that this approach while excellent for smaller networks is not a good approach for large networks.

3.9 Summary

In this chapter several studied techniques for CNN optimization have been presented, that will serve as inspiration to approach the problem of this dissertation. Although several techniques have been presented and studied in this chapter, we will only apply Width Vs Depth and Dropout to the set of tests to be performed because we believe they are the most appropriate for the problem and dataset in question.

In the next chapter, the methodology used for this dissertation will be presented and described, as well as a short demonstration of the methodology.

Chapter 4

Proposed Methodology

This chapter contains a brief description of the methodology used throughout this dissertation to reduce the latency times of a Convolutional Neural Network (CNN). In addition, it includes a short test run as well as a description of the dataset to be used.

4.1 Dataset

As introduced in chapter 1, the goal will be to optimize CNNs which in this particular case receive traffic data as input. Similar to the DeepCorr [36], the CNN built for this problem will receive as input raw flow features and will use them to derive complex features. A bidirectional network flow, i, is represented by the following array:

$$F_i = [T_i^u; S_i^u; T_i^d; S_i^d]$$

where T represents the vector of Inter-Packet Delays (IPD) of the flow i, S is the vector of i'th packet sizes, and u and d superscripts represent "upstream" and "downstream" sides of bidirectional flow i.

In the instance where we want to correlate two flows i and j (which is what happens in section 5.3), where for example i was intercepted by a malicious Tor guard relay and j was intercepted by an accomplice exit relay. This pair of flows with the following two-dimensional matrix consisting of 8 rows is represented as follows:

$$F_{i,j} = [T_i^u; T_j^u; T_i^d; T_j^d; S_i^u; S_j^u; S_i^d; S_j^d]$$

where the lines of the array are taken from the flow representations F_i and F_j.

In summary, the traffic is processed and transformed into a time series of 2 variables being the size of the packets and the time interval between packets, the latter being what turns the traffic into a time series and serves as input to the CNN.

4.2 Methodology

The architecture of a CNN can be seen in the figure 4.1 and as it turns out, it is quite complex since it consists of several layers and the input undergoes several transformations until it is translated into an output.

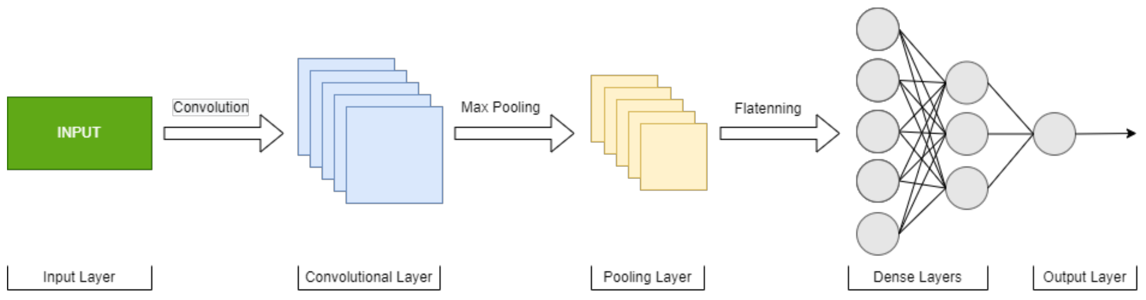

Figure 4.1: Simple scheme of how a CNN works

Although in the figure 4.1 there is only one convolution and pooling layer, most networks have 2 or more of both layers mentioned, making networks even more complex. When looking at a CNN the main goal of its optimization is to get a good trade-off between prediction time and accuracy, i.e., the shortest possible time and an efficiency as close to the value 1 as possible. This becomes possible by changing some hyperparameters that make up the network or changing the architecture of the network itself.

In order to define the methodology for this dissertation, a table (table 4.1) of the hyperparameters that constitute a CNN was made and on the constructed table it is important to highlight that a hyperparameter is a variable that must be pre-set, whereas a parameter is a variable that is automatically optimized during training.

After having the information provided by table 4.1, the goal will be to make individual tests of each hyperparameter by changing that same hyperparameter in an already designed model. After training the model and consequent testing we will proceed to build a comparative graph with two axes, one representing the model's accuracy and the other the model's execution time, making it much easier to analyze results. If it is not possible to construct a graph, given the attributes that the hyperparameter can have we will proceed to construct a table that will compare the different attributes of that hyperparameter. It should also be noted that after training the built model we will use the test dataset to test both the prediction time and the accuracy of the model. Note that we will repeat the test 10 times and choose the minimum time in order to avoid interference problems from other processes on the machine, which are noticeable because the prediction times obtained are relatively short (any simultaneous process, however small, potentially has a high impact on the time measurement). In listing 4.1, the code describing how we obtain this prediction time is presented.

	Parameters	**Hyperparameters**
Convolutional layer	Kernels	Kernel Size Number of Kernels Stride Padding Activaction Function
Pooling Layer	None	Pooling Method Filter Size Stride Padding
Dense Layer	Weights	Number of Weights Activaction Function

Table 4.1: List of hyperparameters of a CNN [47]

Listing 4.1: Example of building a CNN model

```
this_execution = np.zeros(10)
l3s, labels3 = generateDataset(testPairsFolders)
for j in range(10):
    start_test = time.time()
    test_loss, test_accuracy = model.evaluate(l3s,labels3,
    batch_size=BATCH_SIZE, verbose=1)
    end_test = time.time()
    this_execution[j] = end_test - start_test
print('Prediction time = ', np.min(this_execution))
```

In addition to the tests that will be done regarding the hyperparameters, experiments involving the CNN architecture will also be done, such as with increasing or decreasing convolution/max pooling layers, increasing or decreasing dense layers, the use of dropout, and others.

In the next section Preliminary Studies, we will show a preliminary study that was done, in order to better exemplify the methodology used throughout this dissertation and also to better understand some of the attributes to be modifiable in a CNN.

4.3 Preliminary Studies

Using the Fashion MNIST dataset (which is a slight variation of the classic MNIST dataset) that contains articles of clothing, a model of a CNN was created which goal would be to correctly identify as many articles of clothing as possible. We used 60000 images to train the network and 10000 images to evaluate how accurately the network learned to classify images

The model built consisted of 2 pairs of convolutional/Max Pooling layers and 2 dense layers as you can see in Listing 4.2.

Listing 4.2: Example of building a CNN model

```
model = tf.keras.Sequential([
    tf.keras.layers.Conv2D(32, (3,3), padding='same', activation=tf.nn.relu,
                           input_shape=(28, 28, 1)),
    tf.keras.layers.MaxPooling2D((2, 2), strides=2),
    tf.keras.layers.Conv2D(64, (3,3), padding='same', activation=tf.nn.relu),
    tf.keras.layers.MaxPooling2D((2, 2), strides=2),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(128, activation=tf.nn.relu),
    tf.keras.layers.Dense(10, activation=tf.nn.softmax)
])
```

The next step was to test various kernel sizes in the convolution layer and to understand what the trade-off would be between efficiency and prediction time whose code used can be seen at listing 4.3. So, a graph was produced for a better analysis which can be seen at figure 4.2.

Listing 4.3: Code used to test the accuracy and prediction time of different kernel values

```
this_execution = np.zeros(10)
l3s, labels3 = generateDataset(testPairsFolders)
for j in range(10):
    start_test = time.time()
    test_loss, test_accuracy =
    model.evaluate(l3s,labels3, batch_size=BATCH_SIZE, verbose=1)
    end_test = time.time()
    this_execution[j] = end_test - start_test
print('Accuracy on test dataset:', test_accuracy)
print('Execution time = ', np.min(this_execution))
```

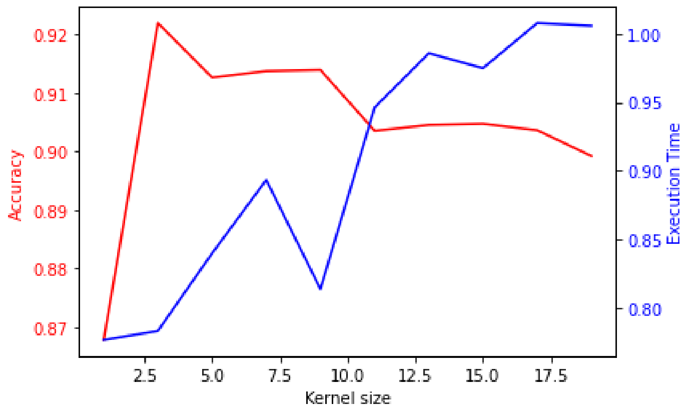

Figure 4.2: Trade-off of execution time and accuracy depending on the size of the kernel

Analyzing the graph, it is possible to conclude that the best value for the kernel size would be 3 since it has the best accuracy of all sizes and the prediction time is the second lowest.

Moving on to the next layer, the pooling layer, there is one attribute that can greatly influence the quality of the model, and that is stride. Similar to what was done with the kernel size, here too several models were tested with different stride values, to try to understand what the optimal value would be.

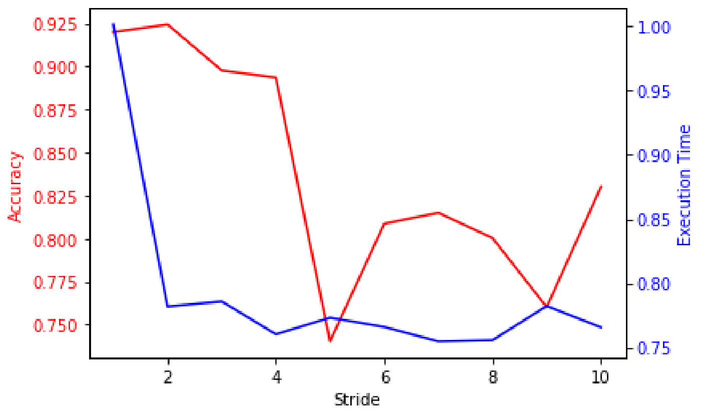

Figure 4.3: Trade-off of prediction time and accuracy depending on the stride value

According to the graph, we can state with certainty that the optimal stride value will be between 2 and 3 since it presents great accuracy and an extremely positive prediction time.

4.4 Summary

In this chapter, was presented the methodology to be used in this dissertation as well as a small demonstration of how this methodology and its application will work. In the next chapter, this methodology will be put into practice by perform several studies and understand which hyperparameters or network structure is most beneficial for a more optimal CNN.

Chapter 5

Results and analysis

In this chapter, the various experiments performed with the purpose of optimizing Convolutional Neural Networks (CNNs) will be presented as well as a detailed analysis of the results obtained. Finally, a conclusion will be made illustrating the best results obtained and contributing factors.

All code used in this work is available at the following repository: https://github.com/andretse77/Notebook-Thesis

5.1 Torpedo

The experiments performed in this dissertation were done in the scope of the DAnon project using Torpedo. Torpedo is a distributed system that offers a global service for the deanonymization of The Onion Router (TOR) Onion Services (OS) sessions with the goal of assisting Law Enforcement Authorities (LEAs) in the investigation of cybercrime. The deanonymization inquiries that LEAs send as part of criminal investigations are processed by Torpedo which is a system built on a federated architecture of network monitoring servers operated by Internet Service Providers (ISPs). Torpedo uses a multi-stage pipeline based on various Machine Learning models for filtering and correlating OS sessions in order to analyze OS traffic at scale.

The architecture of Torpedo's prediction pipeline is quite extensive and consists of different sections. For the purpose of this dissertation we only focused on two specific parts of this pipeline, which was the Ranking Stage and the Correlation Stage, while the rest of the team focused on their part of the pipeline.

Ranking Stage is the first step in torpedo's matching phase and its role is to quickly and roughly scan the flow pairs, reducing the search space for the more complex but slower Correlation Stage. In a nutshell, the Ranking Stage will examine a little amount of metadata for each flow pair and produce a score between 0 and 1. This score can be read as the confidence that a given pair is correlated.

The Correlation Stage, the second stage in Torpedo's matching pipeline, is in charge of

Layer	Details
Input	Size: 3
Fully Connected 1	Size: 300, Activation: ReLU
Fully Connected 2	Size: 80, Activation: ReLU
Fully Connected 3	Size: 10, Activation: ReLU
Output Layer	Size: 1, Activaction: Sigmoid

Table 5.1: Network architecture used in the Ranking Stage

ingesting and processing a collection of potentially correlated flow pairs chosen by the Ranking Stage. A score between 0 and 1 represents the likelihood that a client-generated flow would be correlated with a specific OS-generated flow is the result of the Correlation Stage. In this context, it should be noted that there is a balanced proportion between 0's and 1's in the training dataset, consisting of traffic samples corresponding to the launch and startup of 1000 Tor clients and 1000 OSes.

5.2 Ranking Stage

The purpose of the Ranking Stage has already been described in the previous section. For this part of the pipeline, a very simple neural network was constructed (which can be seen in table 5.1) to produce the confidence scores on the potential correlation of flow pairs.

With the model already built, we started by basing ourselves on one of the methods studied in chapter 3 and tested the approach of preferring a wide net to a deep net, and to do so we removed one layer from the original model. The next step would be to choose the indicated number of neurons to place between the input and output layer. Since the input layer provides 300 neurons and the output layer has only 1, in our opinion, the layer between them should have a number that would be roughly a similar distance apart. Therefore, we chose to do a small test with values between 150 and 60 and for these values, we chose intervals of 10 in 10 in each experiment and the results can be seen in the figure 5.1 and in table 5.2.

Accuracy and prediction time of different models		
Model	Accuracy	Prediction Time
Original model	**0.8**	**0.004953**
3 layers - 150 neurons	0.8006	0.004366
3 layers - 140 neurons	0.7996	0.0041423
3 layers - 130 neurons	0.7998	0.004203
3 layers - 120 neurons	0.8002	0.004233
3 layers - 110 neurons	**0.8032**	**0.004198**
3 layers - 100 neurons	0.8011	0.004566
3 layers - 90 neurons	0.8006	0.004444
3 layers - 80 neurons	0.8011	0.004483
3 layers - 70 neurons	0.8009	0.004473
3 layers - 60 neurons	0.8004	0.004564

Table 5.2: Accuracy and Prediction time of the different models tested

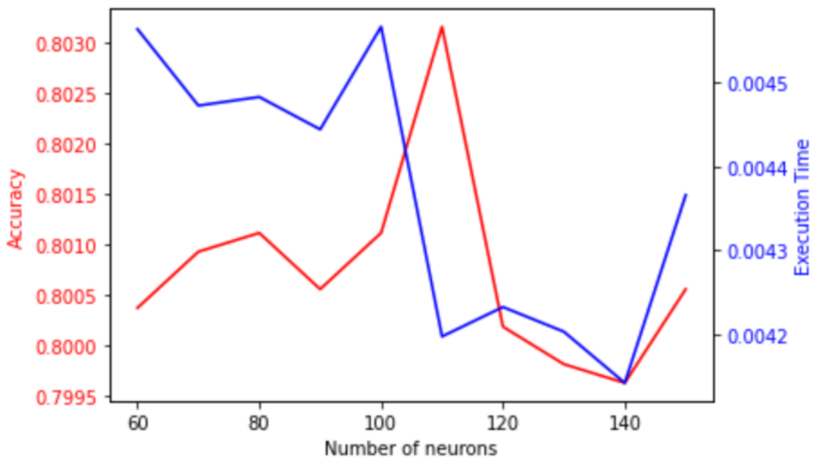

Figure 5.1: Trade-off of prediction time and accuracy using only three dense layers

Looking at the graph there is clearly one value that stands out which is 110 neurons that besides having the best accuracy compared to the other competitors, the prediction time is close to being one of the lowest. It only remained to be seen whether this result would only be better than the others compared, or whether it would be better than the original model that is built.

Since this network is quite small and is intended to be very fast, it is not possible to obtain very significant conclusions regarding the purpose of this thesis although slight improvements are seen in the removal of a dense layer. Since if we look at the results in table 5.2, all tested models had a shorter prediction time than the original model and we also point out that the accuracy of these models is very similar to the original one.

Although the gains are not very significant it is possible to state that there are gains in terms of prediction time when making the network wider instead of deeper, as can be seen in table 5.2

because all tested models had a shorter prediction time than the original model. Furthermore, we also point out that the accuracy of these models is very similar to the original one.

As was stated this network was of low complexity, so this small test was only performed as a small study for chapter 5.3, where further and more complex experiments will be performed.

5.3 Correlation Stage

Also embedded in the Torpedo, there was the Correlation Stage. Shortly, the neural network receives as input a collection of features that were taken from two flows, comprising eight vectors of length N, where N is the number of training packets, in this case 100. These eight vectors may be divided into quadruples (one for each flow), where each quadruple contains time-series for:

- Incoming packets inter-arrival time;
- Outgoing packets inter-arrival time;
- Incoming packet sizes;
- Outgoing packet sizes for that particular flow.

Layer	Details
Convolutional Layer 1	Kernel num: 2000 Kernel size: (2,30) Stride: (2,1) Activation: ReLU
Max Pool 1	Window size: (1,5) Stride: (1,1)
Convolutional Layer 2	Kernel num: 1000 Kernel size: (4,10) Stride: (4,1) Activation: ReLU
Max Pool 2	Window size: (1,5) Stride: (1,1)
Fully connected 1	Size: 3000, Activation: ReLU
Fully connected 2	Size: 800, Activation: ReLU
Fully Connected 3	Size: 100, Activation: ReLU

Table 5.3: DeepCorr's hyperparameters optimized to correlate Tor traffic. [36]

In this section of the pipeline there was also a built model which is a slightly modified version of DeepCorr [36], an advanced deep learning architecture that learns a flow correlation function tailored to TOR's network. Therefore, the structure of the model can be seen in Table 5.4.

Input vectors that are anticipated to be correlated for linked TOR flows are captured by the first convolution layer. The second convolution layer uses a combination of timing and size information to capture aspects of the overall traffic flow.

Layer	**Details**
Input Layer	Size: 8 * Flow Length
Convolutional Layer 1	Kernel num: 2000/reduce_factor Kernel size: (2,30) Stride: (2,1) Activation: ReLU
Max Pool 1	Window Size: (1,5) Stride: (1,1)
Convolutional Layer 2	Kernel num: 1000/reduce_factor Kernel size: (4,10) Stride: (4,1) Activation: ReLU
Max Pool 2	Window Size: (1,5) Stride: (1,1)
Fully Connected 1	Size: 3000, Activation: ReLU
Fully Connected 2	Size: 800, Activation: ReLU
Fully Connected 3	Size: 100, Activaction: ReLU
Output Layer	Size: 1, Activation Sigmoid

Table 5.4: Network architecture used in the Correlation Stage

To have a method of comparison between the two models the DeepCorr model is described in the table 5.3. Although looking at the tables the models look similar there is one rather important factor. In the model in the table 5.4, a reduce_factor with a value of 16 has been set, this significantly reduces the complexity of the network which allows us to make the experiments feasible on the available hardware and also faster. In the next subchapters we will show the study that was done and analyze the importance of the various hyper-parameters and Convolutional Neural Network (CNN) architecture that can significantly influence the prediction time of these networks.

5.3.1 Kernel

By using the network architecture described in table 5.4 we start by varying the kernel width in the first layer between the values 15 and 40 and then in a second test in the second layer, we varied the kernel width between 5 and 15. In both cases, the range of values mentioned was chosen, because it was a range that contained values before and after the original value of the kernel width. It should also be noted that a larger range of values was not chosen because from the graph, a certain tendency is already observed.

It is important to note that when changes were made in the first layer, the second remained the same as the original architecture and vice versa, as will happen in the remaining tests throughout this dissertation. The results of both tests can be seen in figure 5.2 and 5.3, respectively.

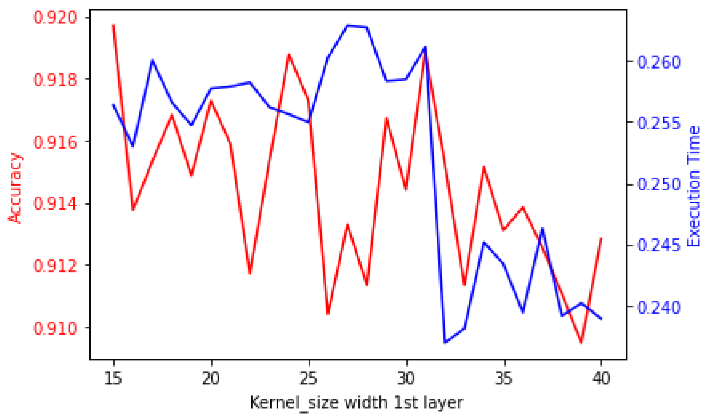

Figure 5.2: Trade-off between accuracy and prediction time by varying the kernel width in 1st the layer

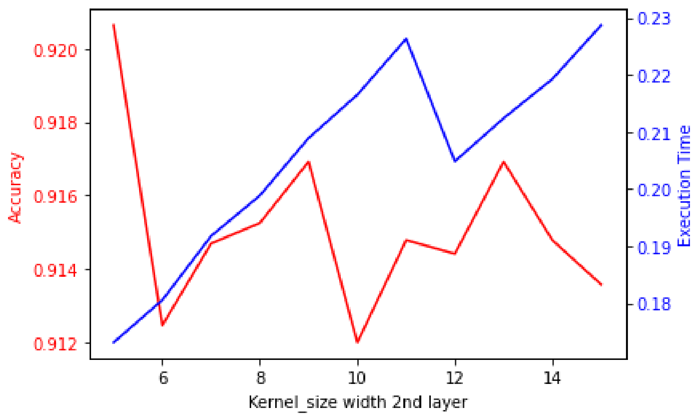

Figure 5.3: Trade-off between accuracy and prediction time by varying the kernel width in 2nd the layer

Analyzing the figure 5.2, we can observe that the prediction time values when the kernel width in the first layer is between 15 and 31 are quite high, however, after that the prediction time drops dramatically (compared to the other points) while its accuracy remains quite high during all tests. These results are an indicator that the kernel width values should be as high as possible, as they present the lowest prediction times without ever having a too significant loss in model accuracy. The kernel size refers to the height and width of the filter mask to be applied in the convolution, the bigger the kernel, the fewer convolution operations that need to take place and consequently the convolution process will be faster.

Moving on now to the analysis of figure 5.3, compared to the previous graph (figure 5.2), here it is possible to see that the kernel width influences the prediction time in a more significant way, since while in figure 5.2 the lowest prediction time reached was around 0.24s. In this graph (figure 5.3) we are analyzing that is possible to reach 0.18s, and in both cases the accuracy values remain quite similar.

Given the results presented, the figures and values suggest that the influence of the kernel on prediction time is more significant in the second convolution layer than in the first. One of the facts that can help explain this phenomenon is related to the size of the kernel and the input shape that each layer receives.

As far as the first convolution layer is concerned, it receives an input with the size 8x100 and since the kernel has the size of 2x30 and knowing that the strides in this layer is 2x1, the time required to go through the entire input will be relatively large. This is why we see a tendency for the prediction time to drop as the kernel width size increases.

Nevertheless, if we now look at the second convolution layer, the same logic of the first layer does not apply here because in the graph in figure 5.3 the exact opposite is observed. This can be explained by the following fact: the second convolution layer receives the output of the first max pooling layer which has a size of 4x67 that corresponds to the same height as the kernel in this layer 4x10. This may seem good but with a kernel with the same height as the input it receives, it will take a long time to perform the convolution operation, since it is a very large matrix and therefore many operations will have to be done.

Concluding the study on this hyperparameter, it seems like a good approach to vary the kernel size to reduce the prediction time of a CNN, but special care must be taken not to have a kernel that is too large and takes too long to perform the convolution operations and also not to have one that is too small and takes too long to go through the entire input.

5.3.2 Strides in Convolutional layer

Still inside the convolution layer, there is the strides parameter that is an integer, tuple, or list of two numbers that specifies the height and width of the convolution's strides. In table 5.5, the results of this same experiment are displayed. We varied the strides between 1 and 2, both in height and width, the already defined architecture of the network, did not allow us to vary these values even more in this first convolutional layer, however, it has already allowed us to extract some elations.

Strides in 1st convolutional layer	**Accuracy**	**Prediction Time**
(1,1)	0.9178	0.2512
(1,2)	0.9167	0.1662
(2,1)	0.9157	0.2136
(2,2)	0.9082	0.1423

Table 5.5: Results obtained when changing strides in the 1st convolutional layer

The results obtained are quite interesting, we clearly notice a drastic decrease in prediction time when we use higher strides in width, which in this case is using the value of 2. When using this stride value for both attributes the best result was obtained. Regarding accuracy, the values remain relatively close and high so we can focus solely on prediction time.

These results have their logic, since as we can see, the worst prediction times are when the window is smaller (1x1) which makes sense, since the window is smaller it will take longer to scroll through the entire input. Conversely, the lowest prediction time is when the window is larger (2x2) and so it will be much faster to scroll through the input. Between these two examples there is a difference of about 0.11s, which in this context is 44% less prediction time, a value that we consider significant. There is also a significant difference when you increase either the width or the height of the stride. In this case, we suspect that these results are due to the structure of the dataset we are using. Overall, however, we assume that a plausible hypothesis to obtain better prediction time is to increase the strides value in the convolutional layer.

5.3.3 Filters

Looking at the convolution process, there is a very important argument that is filters consisting of an integer, which gives the dimensionality of the output space or in other words the number of output filters in the convolution [41]. Since it was a hyperparameter that we considered extremely relevant, we experimented varying the values above and below the original value of the initial model (seen in table 5.4). We started by performing experiments on the first convolution layer given that the tests performed saw the integer filters vary between 4000 and 1100 and the results of this experiment can be seen in figure 5.4.

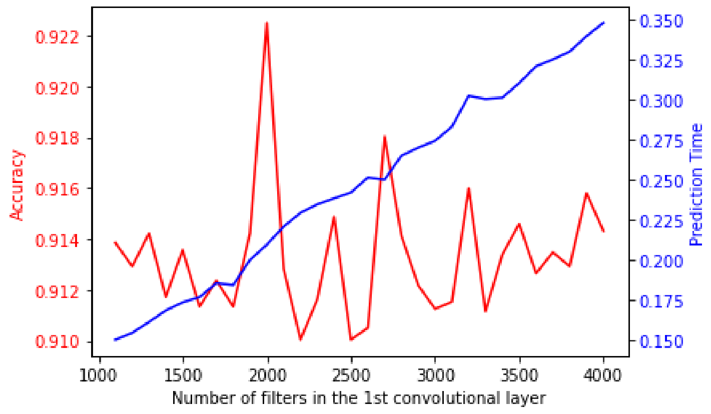

Figure 5.4: Trade-off between accuracy and prediction time by varying the size of the filters in the 1st convolutional layer

After a carefull analysis of the figure 5.4 it was possible to conclude that the results were actually quite interesting. There is a large variation in prediction times with the change of the integer filters value and if we look specifically between the maximum value and the minimum value there is a difference of 0.2s which translates to a 42% difference. In addition to this, we observe an increasing trend in prediction time as this value increases. Looking now at the accuracy in figure 5.4 the values vary inconsistently, however, they all remain above 0.91 accuracy, which is still a quite high accuracy. If we look at the value of the initial model, that value has the highest accuracy, however, the prediction time is not the lowest at all.

Performing the test now on the second convolution layer we have varied the filters value between 1900 and 100.

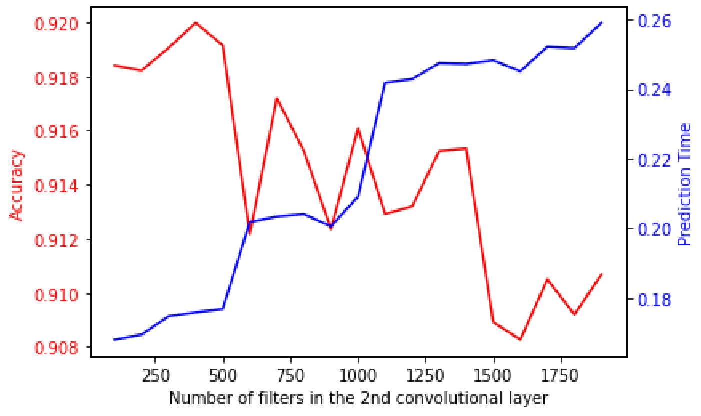

Figure 5.5: Trade-off between accuracy and prediction time by varying the size of the filters in the 2nd convolutional layer

Similar to figure 5.4, here in figure 5.5 we also observe an increasing trend in prediction time as the number of filters increases, with minimum values of 0.18s and maximum values of 0.26s, a not very significant difference. Looking at the accuracy values they turn out to be a little inconstant although there is a slight tendency for the accuracy to decrease as the number of filters increases, despite this, the accuracy values always remain high.

In summary, in a model with 2 convolution/max pooling layers changing the number of filters, either in the first or second layer, appears to us to be a viable approach to reduce the prediction time of a CNN.

5.3.4 Pool_Size

When entering into the max pooling layer, one of the arguments the model can take is pool_size, which is an integer or tuple of two numbers, the size of the maximum-taking window [43]. We therefore experimented to perform a set of tests and understand the relevance of this parameter in the latency time of CNNs. In the original architecture, the pool_size was set to "(1,5)", in this experiment we will try varying the width of the pool_size, in both layers, between 1 and 31 and the height will have to stay at 1, since its height never exceeds 1. The width value was limited to 31, since, as we were changing the width value in both max pooling layers, if we set a width value of 32 in the first layer, the output_shape, as far as width is concerned, in the second layer would be equal to 31 (which would make it impossible to enter a pool_size value of

"(1,32)"). In this way, we ensure that we cover as much of the pool_size widths as possible for better completion of the results.

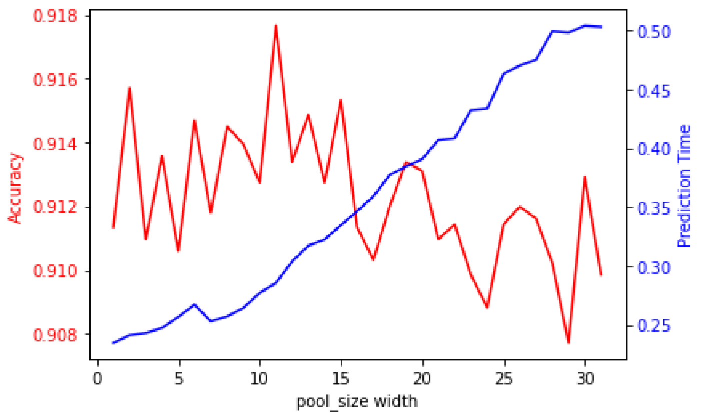

Figure 5.6: Trade-off between accuracy and prediction time by varying the pool_size width in both max pooling layers

The results obtained of the experiment conducted are shown in figure 5.6. On one hand, it is possible to observe a clear tendency for the prediction time to increase as the pool_size width increases. On the other hand, the accuracy shows inconstant and rather less significant values than the prediction time. According to what was described in the first paragraph of this subsection this tendency can be easily justified, according to our thinking. Well, as mentioned, pool_size takes the maximum value from a window of defined size, i.e., in our view it seems trivial that if this window is small, the minimum value will be found faster than in a larger window because there is less "space" to travel to find this minimum value along the window. Since only one value is chosen within the pool_size, our idea would be that the larger the pool_size, the less effective the model would be since many features that could be relevant would be lost. Although this happens in a certain way since there is a slight tendency for the accuracy value to decrease as the pool_size width increases, it was not as significant as we thought it would be, since the accuracy of all the tested models was always above 0.9, a value that we consider very reasonable.

In conclusion, and looking at concrete values in figure 5.6, within this experiment, there is a difference of about 0.25s between the maximum value (about 0.50s) and the minimum value (about 0.25s), which represents twice the time between these extremes. Therefore, we consider that the hyperparameter is quite important in optimizing the time, so a plausible hypothesis to optimize a CNN is to keep the pool_size as low as possible.

5.3.5 Strides in Max Pooling layer

After analyzing all the hyperparameters associated with the convolution layer, we move on to the max pooling layer. We start by performing two tests by changing the stride values, being that in the first one we change the stride values in the first layer and in the second one we change the stride values in the second layer and the results can be seen in figure 5.7 and figure 5.8, respectively.

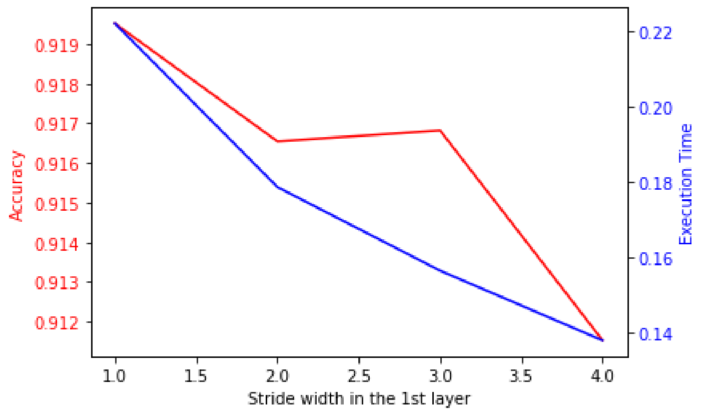

Figure 5.7: Trade-off between accuracy and prediction time by varying the stride width in 1st the layer

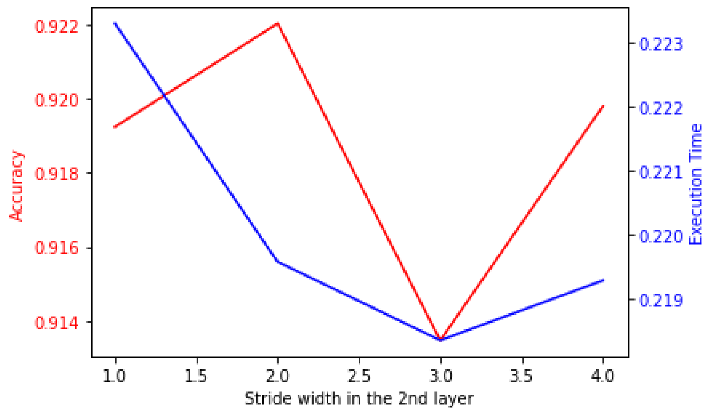

Figure 5.8: Trade-off between accuracy and prediction time by varying the stride width in 2nd the layer

When examining both results we can observe an interesting detail. While in the first layer the stride value influences the prediction time more significantly and the accuracy is almost null, in the second layer the opposite occurs since the variance of the prediction time is almost null and the accuracy of the model is affected more relevantly.

From our point of view, as far as figure 5.7 is concerned, the results shown seem to make a lot of sense, this is because strides specifies how far the pooling window moves for each pooling step, and as such, the larger the stride in total, the fewer translations will be made and as such, the prediction times will be faster.

When it comes to figure 5.8, the results in terms of prediction time were very low with a difference of only about 0.004 seconds between the maximum and minimum time, however, if you think about the entire CNN pipeline it is a result that appears to make sense. By getting a summary of the model (table 5.6), we can see that the output shape that reaches the second convolution layer is quite small, and as such, whether the stride is large or small, it will never make much difference because there is only a small "area" to go around.

Layer (type)	**Output Shape**	**Param #**
conv2d_4 (Conv2D)	(None, 4, 18, 125)	7625
activaction_7 (Activaction)	(None, 4, 18, 125)	0
max_pooling2d_4 (MaxPooling2D)	(None, 4, 14, 125)	0
conv2d_5 (Conv2D)	(None, 1, 5, 62)	310062
activaction_8 (Activaction)	(None, 1, 5, 62)	0
max_pooling2d_5 (MaxPooling2D)	(None, 1, 1, 62)	0

Table 5.6: Part of the table obtained by executing the command that gives a summary of the model

Finally, we ran one last test that combined changing the strides width value in both max pooling layers, the results of which are shown in table 5.7. The results are in line with what we studied and analyzed in the previous results because as can be seen, when we have strides with larger values the prediction time values are significantly lower. The acuraccy of the models does not prove to be a problem with the change of stride values because as these values vary, the accuracy always remains quite high.

Strides 1st max pool	**Strides 2nd max pool**	**Accuracy**	**Prediction Time**
(1,1)	(1,1)	0.9185	0.2635
(1,1)	(1,2)	0.9157	0.2635
(1,1)	(1,3)	0.9151	0.2659
(1,1)	(1,4)	0.9217	0.2652
(1,2)	(1,1)	0.9170	0.1893
(1,2)	(1,2)	0.9223	0.1889
(1,2)	(1,3)	0.9186	0.1878
(1,2)	(1,4)	0.9190	0.1913
(1,3)	(1,1)	0.9142	0.1673
(1,3)	(1,2)	0.9155	0.1659
(1,3)	(1,3)	0.9145	0.1655
(1,3)	(1,4)	0.9187	0.1673
(1,4)	(1,1)	0.9193	0.1554
(1,4)	(1,2)	0.9189	0.1555
(1,4)	(1,3)	0.9208	0.1560
(1,4)	(1,4)	0.9125	0.1585

Table 5.7: Results obtained by varying the stride value in both max pooling layers

Focusing on the problem of this dissertation, the practice of increasing the stride away from values that are too low appears to be a good solution for optimizing CNNs since a downward trend in prediction time is observed as the stride value increases. At the same time, there is also a decrease in the accuracy value of the model, so we would say that it is at personal discretion how much of the accuracy can be sacrificed to achieve a lower prediction time.

5.3.6 Padding

Regarding the convolutional and the max pooling layer, one of the arguments it receives is padding. The padding has 2 values "valid" or "same" where valid means no padding and "same" means results in padding that is uniformly distributed to the input's left, right, or up and down, such that the output has the same height and width dimensions as the input [43].

Tests were performed on the 2 convolution layers we had, changing the padding values, to see what effect they had on the model's accuracy and prediction time and the results can be seen in table 5.8.

Padding	**Accuracy**	**Prediction Time**
'VALID' in both layers	0.9130	0.2141
'VALID' in the first layer 'SAME' in the second layer	0.9117	0.2305
'SAME' in the first layer 'VALID' in the second layer	0.9143	0.2784
'SAME' in both layers	0.9190	0.2988

Table 5.8: Results obtained by changing padding in the convolutional layers

Looking at the table with the results, one can see a relatively significant difference between not using padding ('VALID') and using padding ('SAME') in both convolution layers. Without using padding, we obtained the shortest prediction time for the tests performed, and on the opposite side of the coin, the longest prediction time was obtained with the use of padding. Following this logic, there is also an observable difference between not using padding in the first layer and not using it in the second layer. Since we have already seen that not using padding is more beneficial for a shorter prediction time, the explanation we found for this difference between layers has to do with the output shape since it is obviously much larger in the first layer than in the second one. Furthermore, its relevance turns out to be more important in a larger shape than in a smaller one, and the results prove it. Mentioning also the accuracy values, they have always remained practically equal and above 0.91 which we consider to be a very solid accuracy.

In addition to the first test, the influence of padding was also tested but this time on the max pooling layers. Looking at the table 5.9 which constitutes the results and comparing it to those obtained in the first test, there is the same tendency to get better results without the use of padding ('VALID') than with its use. Besides this, there is also a very interesting fact, the discrepancy between values is much smaller and the minimum values are lower than the ones registered in table 5.8. The accuracy, once again, always remains at values above 0.91.

Padding	Accuracy	Prediction Time
'VALID' in both layers	0.9156	0.2114
'VALID' in the first layer 'SAME' in the second layers	0.9148	0.2089
'SAME' in the first layer 'VALID' in the second layers	0.9172	0.2203
'SAME' in both layers	0.9182	0.2237

Table 5.9: Results obtained by changing padding in the max pooling layers

Overall, we cannot state an absolute truth about whether or not to use padding when it comes to optimizing CNNs, as it will always depend on the datasets we have been dealing with and analyzing. However, from the results obtained, it seems to be much more plausible to change the padding value in the max pooling layer than in the convolution layer.

5.3.7 Activaction Functions

When working with CNN it is very important to introduce activation functions into their architecture. Data can be mapped into dimensions more effectively with an appropriate activation function. A neural network model's expression capacity is further increased by using an activation function, which can provide the deep neural network the true importance of Artificial Intelligence (AI) [45].

With the information obtained and mentioned above, we decided to test whether the different activation functions would have any effect on the prediction time or the accuracy of the model, so we experimented in addition to the original model activation function ("relu"), two more activation functions, being "sigmoid" and "tanh". The fact that they are the most commonly used [8] is what led us to choose these activation functions over others.

Looking at the table 5.10 with the results of the different activation functions, they all seem quite identical, both in model accuracy, with values around 0.9 and in prediction time, where all the values are within 0.21s. Although certain types of problems require a certain type of activation function, the results obtained lead us to believe that these functions have practically no influence on improving the prediction time of a CNN, so, according to the problem addressed in this dissertation, changing the activation function of a network does not seem to be a good option when it comes to optimizing it.

Activaction Function	Accuracy	Prediction Time
ReLU	0.9110	0.2110
Sigmoid	0.9	0.2171
Tanh	0.9181	0.2148

Table 5.10: Results obtained by using different activation functions

5.3.8 Dropout

As already mentioned in section 3.2, the use of dropout can be one of the techniques used to reduce latency time in a CNN. To prove this theory, we first ran two tests in which we compared the original model, which has the use of a dropout of 0.4, and a model that has no dropout, as can be observed in the table 5.11. It should be clarified that in order to avoid overfitting, the Dropout layer randomly sets input units to 0 with a frequency of rate at each step during training. The sum of all inputs is maintained by scaling up non-zero inputs by 1/(1 - rate) [42].

Dropout	**Accuracy**	**Prediction Time**
Yes (value of 0.4)	0.9165	0.2182
No	0.9182	0.2111

Table 5.11: Results obtained by using or not Dropout

Although we did not observe any major differences in the results between the two tests, we decided to manipulate the dropout values and see if we would come to any conclusions that would be relevant and constructive to the study being done.

We therefore ran 9 different tests varying the dropout value on a scale of 0.1 with each experiment and the results can be seen in figure 5.9. The maximum value that the dropout can have is 1 and the minimum is 0, and as such, we decided to use this scale of 0.1 among the various tests performed so that it would be possible to observe a possible trend from the generated graph(figure 5.9).

Through our analysis of the graph in figure 5.9 the prediction time has a tendency to increase as the dropout probability increases. There is also another interesting perspective, although a little inconstant in the initial values, the trend of the model's accuracy values tends to decrease as the probability of dropout values increase. The maximum time value found in the graph is around 0.222s while the minimum value is around 0.212s, which is a difference of only 0.01s, a difference that we do not consider to be very relevant. If we now compare with the values in table 5.11, in the sense that we can compare the difference between using dropout or not, there is still no significant difference when it comes to improving prediction times.

In a general conclusion, we consider that although the use of dropout is a good technique to avoid overfitting, in terms of improving prediction times it does not seem to be a plausible hypothesis.

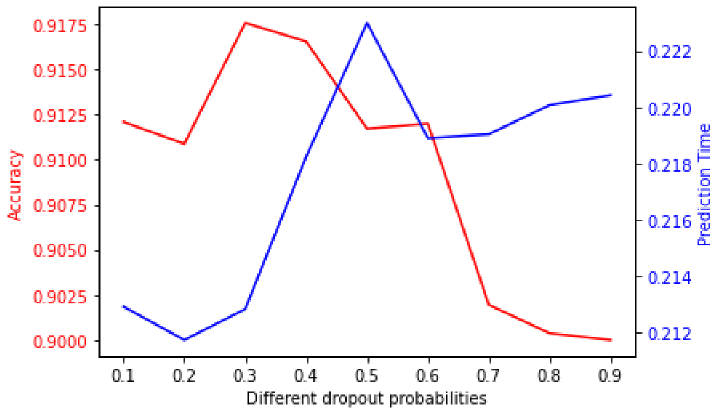

Figure 5.9: Trade-off between accuracy and prediction time using different dropout probabilities

5.3.9 Number of neurons

Table 5.4 shows the architecture of the network that is being used to perform various experiments and, as can be seen, there are 4 dense layers in this architecture. Since the first and the last are unchanged because they correspond to the input and output layer, we experimented with varying the number of neurons in the second dense layer, since there was a large distance of neurons from the input layer to the input layer and even to the third dense layer. The number of neurons was then manipulated between 500 and 2000 which is a range that we consider reasonable to draw good insights.

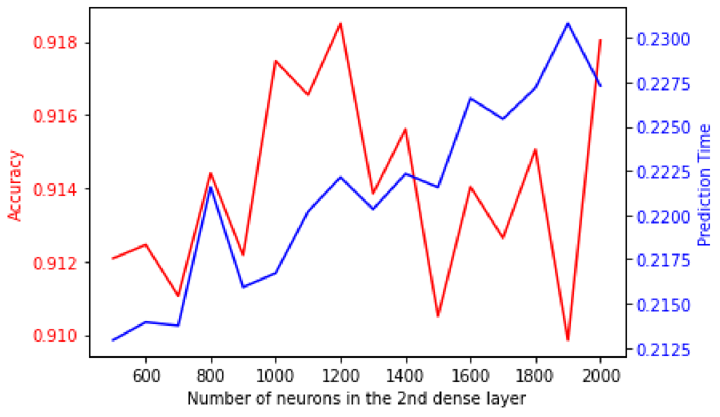

Figure 5.10: Trade-off between accuracy and prediction time varying the number of neurons in the 2nd dense layer

The prediction time and accuracy of the tested models can be seen in figure 5.10, and as can be seen there is a tendency for the prediction time to increase as the number of neurons increases, while the accuracy remains quite inconsistent but with very positive values above 0.91. If we look at concrete values, there is a difference of about 0.0175s between the maximum and minimum values, which is quite a tiny difference. Therefore, from our point of view, we understand that varying the number of neurons does not appear to be the best option in optimizing CNNs.

5.3.10 Number of Convolutional/Max Pooling Layers

After having done several experiments involving the hyperparameters of a CNN, we move on to experiments involving changes in the network architecture. It started by removing a convolution and max pooling layer and after that, varying in individual tests the kernel and stride values and seeing if there was any trend.

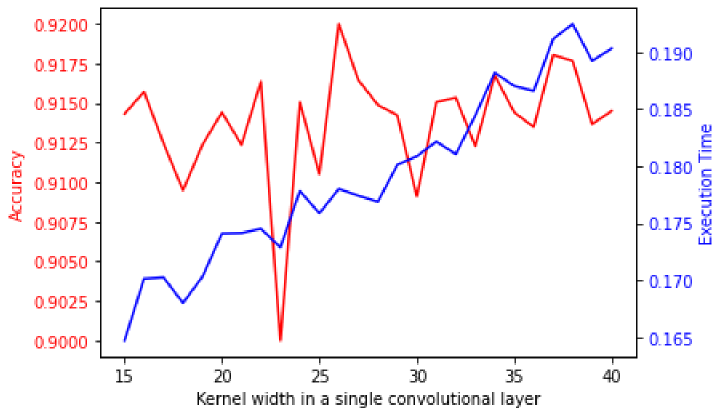

Figure 5.11: Trade-off between accuracy and prediction time using only one convolutional/pooling layer and varying the kernel

When it comes to tests involving the kernel the results were quite surprising, as we can see in figure 5.11. To have a comparison, in the similar test performed in figure 5.2, the prediction time values were between 0.24 and 0.26, and in this test performed with only one layer the values are between 0.19 and 0.165 which is a drop in prediction time of between 31% and 27%. This may not seem like high values but on larger datasets with much longer prediction times this drop can be crucial for better results. There is also another interesting fact, the accuracy values, except for one loose case, are quite similar to those obtained with a model with 2 convolution and max pooling layers with the nuance that in this experiment we have much lower prediction times. Finally, it should also be mentioned that figure 5.11 also shows the tendency for the prediction time to increase as the kernel width increases.

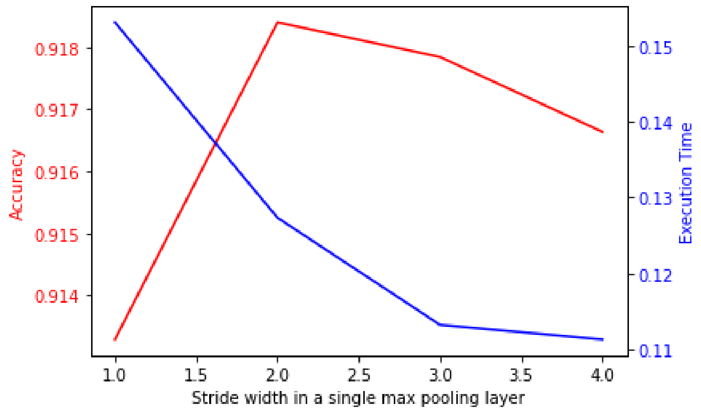

Figure 5.12: Trade-off between accuracy and prediction time using only one convolutional/pooling layer and varying the stride

Regarding the tests performed involving stride that can be seen in figure 5.12, one clearly notices a downward trend in prediction time as stride increases. As for accuracy, it doesn't oscillate much, being within a short but high value range. However, when the stride value is 1 the accuracy reaches minimum values but from then on this value goes up and remains relatively constant.

Again, compared to the stride test observable in figure 5.7, where the prediction times were around minimum values of 0.14s and maximum values of 0.22s, in this graph (figure 5.12), we get minimum times around 0.11s and maximum times around 0.15s. On the one hand this difference may not seem significant, but on the other hand, in this experiment the time values are in a more concentrated and smaller time interval which can guarantee low prediction times. In terms of accuracy values, the graph in the figure 5.7 appears to have slightly higher accuracy values than in the experiment we are analyzing, although in both cases the values are acceptable.

Comparing now the influence of the strides and kernel hyperparameters, it is concluded that similar to the tests that were performed on the original model, the strides hyperparameter in the max pooling layer influences the prediction time more significantly than the kernel hyperparameter in the convolution layer. Nevertheless, in both cases there were improvements in prediction times and relatively similar accuracy values, and this leads us to believe that when it comes to the optimization of a CNN a permissible scenario for optimizing the network would be to prefer a network with only one convolution layer and max pooling rather than two.

On the reverse side, another experiment that was tested was to increase the number of convolution/max pooling layers to 3. This being said, as mentioned, the tested model had three convolution layers and for this test to be possible the network architecture had to be slightly changed. For this purpose, the architecture of this new network can be seen in table 5.12

The results presented by this network were not very encouraging, as can be seen in table 5.13, the accuracy values of the models proved to be similar although a slightly higher value was noted for the accuracy of the original model. If we analyze the time, however, the model with 3 layers

Layer	Details
Input Layer	Size: Flow Length * 8
Convolutional Layer 1	Kernel num: 2000/reduce_factor Kernel size: (2,30) Stride: (2,1) Activaction: ReLU
Max Pool 1	Window Size: (1,5) Stride: (1,1)
Convolutional Layer 2	Kernel num: 1000/reduce_factor Kernel size: (2,30) Stride: (2,1) Activaction: ReLU
Max Pool 2	Window Size: (1,5) Stride: (1,1)
Convolutional Layer 3	Kernel num: 500 Kernel size: (2,30) Stride: (4,1) Activaction: ReLU
Max Pool 3	Window Size: (1,5) Stride: (1,1)
Fully Connected 1	Size: 3000, Activaction: ReLU
Fully Connected 2	Size: 800, Activaction: ReLU
Fully Connected 3	Size: 100, Activaction: ReLU
Output Layer	Size: 1, Activaction: Sigmoid

Table 5.12: Network architecture used by using 3 convolutional/max pooling layers

Model	Accuracy	Prediction Time
Original Model	0.9110	0.2110
Model with 3 convolutional/max pooling layers	0.9	0.2643

Table 5.13: Comparison table between the original model and a model with 3 convolutional/max pooling layers

of convolution/max pooling, shows a higher prediction time by about 0.05s, which is an increase of roughly 19%, which is not much but proves to be higher. Through the results obtained, and comparing with the results of having only one layer, we understand that for an optimized CNN, an admissible hypothesis seems to be to reduce the number of layers in the network instead of increasing them.

Model	Accuracy	Prediction Time
Original Model	0.9110	0.2110
Model with 3 dense layers	0.9167	0.2075

Table 5.14: Comparison table between the original model and a model with only 3 dense layers

5.3.11 Dense layer

As seen in table 5.4 the network architecture has 4 dense layers. It was observed that the gap between the third layer and the output layer was relatively short and as such, we decided to remove this layer and observe the results that can be seen in table 5.14. As the results, both for accuracy and prediction were quite similar, after removing a dense layer we experimented varying the number of neurons, similar to how it was in subsection 5.3.9, to try to see if the results could be any better with the results of this experiment being shown in figure 5.13.

If we compare the results of this experiment with those of figure 5.10, there is a common trend, the prediction time decreasing as the number of neurons increases and an inconstant but always high accuracy (above 0.9). Even so, the results for figure 5.10, where there is one more dense layer show to be slightly better, as the maximum and minimum values are both lower than the maximum and minimum values in figure 5.13.

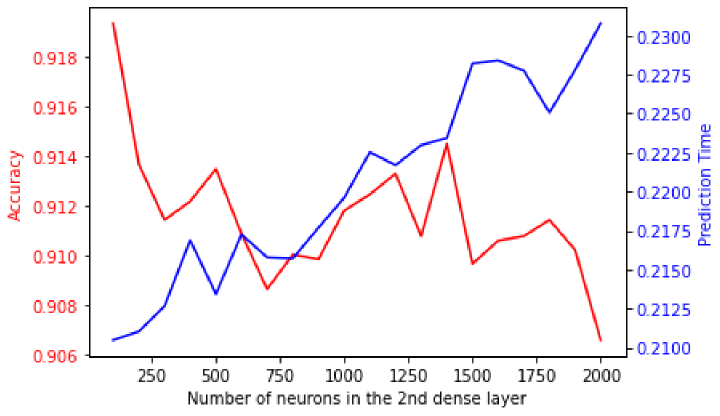

Figure 5.13: Trade-off between accuracy and prediction time varying the number of neurons in the 2nd dense layer and with 1 layer less

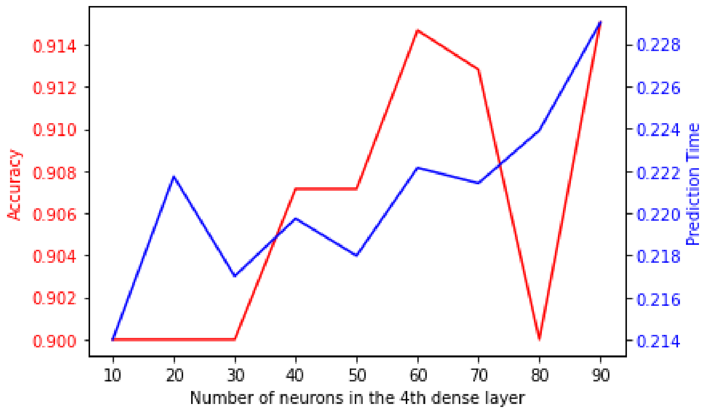

Figure 5.14: Trade-off between accuracy and prediction time varying the number of neurons in the 4th dense layer

In the same way that we removed a dense layer to test results, we proceeded to add another dense layer to test the results that would be obtained. The network was then left with 4 dense layers plus the output layer and it was in this last dense layer that we decided to vary the number of neurons. As in the previous layer the value was 100 neurons, we decided to vary in this layer the number between 10 and 90 for a better analysis of the results, which are registered in figure 5.14. Similar to figure 5.10, here too there is an upward trend in prediction time as the number of neurons increases, and in accuracy, despite inconsistency there is also an upward trend. If we compare in terms of values, although they have different maximum and minimum values, they are still quite similar. With that said, when it comes to improving the prediction times of a CNN removing or adding dense layers does not appear to be a good approach, from our perspective.

5.3.12 Absence of the Max Pooling layer

The use of pooling layers is not strictly necessary when building a CNN model. Without the existence of a layer that does pooling there is no downsampling, and the only repercussion this has on a CNN is that it can make it harder to train. Since the goal of this dissertation is not focused on training time but rather on prediction time, this was an approach we sought to take. As mentioned, we removed the two max pooling layers constituting the original network and tested the results of the model that we recorded in table 5.15, with the results of the original model so that a comparison could be made.

The values obtained show a time gap of 0.03s, a small value that represents the difference of 15%, which can be justified by the reduction in the number of layers that the input has to pass through. Overall, and although we have seen better approaches so far, removing the max pooling layers from a network seems to be a reasonable approach for optimizing CNNs if there is not the problem of spending more resources on training the network.

Model	**Accuracy**	**Prediction Time**
Original Model	0.9110	0.2110
Model without max pooling layers	0.9120	0.1864

Table 5.15: Comparison table between the original model and a model with no max pooling layers

5.4 Analysis of the results

Once all the experiments were done, we decided to build a table that could summarize these same experiments and classify which hyperparameters or changes in the network architecture could benefit more or less from the optimization process of a CNN.

Hyperparameters and CNN Architecture		**Positive Influence on Prediction Time**
Kernel		Medium
Strides in Convolutional Layer		High
Filters		High
Pool_Size		Low*
Strides in Max Pooling layer		High
Padding		Low
Activaction Functions		None
Dropout		Low
Number of neurons		Low
	Add Convolutional/ Max Pooling layers	None
	Remove Convolutional/ Max pooling layers	High
	Add dense layers	None
	Remove dense layers	Low
	Remove Max Pooling layer	Medium

*depends on the circumstance

Table 5.16: Summary of performed tests with corresponding influence in prediction time

Table 5.16, therefore, summarizes all the experiments performed, dividing them into hyperparameters and CNN architecture and further classifying them into four degrees of relevance for improving the prediction time of a CNN, which are:

- None, which means that in the tests performed there was no improvement over the initial model.
- Low, there has been an improvement in the best experience of less than 10% over the initial model
- Medium, there was an improvement in the best experience between 10% and 20% over the original model.
- High, there was an improvement in the best experience of more than 20% over the original model.

It is important to emphasize that this assignment of relevance degrees is based on the dataset used for this dissertation and has the original model of the table 5.4 as a comparison.

There is, however, a special case in the table 5.16 that we would like to address and that was properly marked with an asterisk. In the original model we had, the pool_size was set to "(1,5)" and since the graph in the figure 5.6 showed an increasing trend, the time gains that were obtained in the best experiment were minimal. But observing that same figure 5.6 the times varied between about 0.25s and 0.50s which represents a significant difference of 50%, i.e., if we had an original model with a larger pool_size the gains would be greater and the relevance attribution would be different. What we mean by this is that the pool_size may not be significant in the circumstances used but since there is a large difference between the minimum and maximum values, on other occasions it may prove to be more significant.

5.5 Summary

In this chapter, the methodology described in chapter 4 was applied to conduct several experiments involving CNNs.

Within the tests performed, very interesting and positive results were obtained that contribute positively to the study of CNN optimization in the context of network traffic analysis. In addition, a table was created that summarizes and qualifies the different tests performed. In chapter 6 all the conclusions reached with this dissertation will be addressed as well as a table that summarizes the entire study that was done in this chapter.

Chapter 6

Conclusions

In this chapter the development and results accomplished throughout this dissertation is going to be described. Firstly, a brief overview of the research and development made during the dissertation is made. Secondly, a description of the results obtained is done, comparing all the tests performed and concluding which ones achieved the best trade-off in accuracy and prediction time. Finally, some directions on future work are given.

6.1 Research and development

This dissertation described a study on the optimization of Convolutional Neural Networks (CNNs). A methodology has been developed to evaluate which factors, whether hyperparameters or network architectures, positively or negatively influence the prediction time and accuracy of a Convolutional Neural Network (CNN).

A careful study was also done that analyzed the importance of each hyperparameter of a CNN that influences its optimization, and this study was summarized in a table made in chapter 5. In addition to this, some changes in the network architecture were also studied which will also be represented in this table.

After performing all the tests and observing the results, we understand that when it comes to hyperparameters, the strides in both the convolution layer and the max pooling layer, and the filters proved to be the most decisive in the study of this dissertation. When it comes to changes in the architecture of a CNN, the removal of layers proved to be the most beneficial approach in reducing the latency times of these networks.

Finally, it is also important to emphasize that this entire dissertation contributes to the DAnon project from Torpedo but above all to the study of CNN optimization in the context of network traffic analysis.

6.2 Future work

As already mentioned in this chapter, the results obtained were quite interesting and contribute to the optimization of CNNs in the context of network traffic analysis. However, it would be interesting to test this study done in datasets from other areas, such as in image classification, recommendation systems and among others. Furthermore, it would be interesting to test other approaches in changing the architecture of a CNN and to analyze the results, using a machine with much more computational power than the one used in this dissertation so that more complex and demanding tests could be done. On top of all this, it would be interesting to test several changes of hyperparameters and architectures at the same time and see what results would go.

Although the analysis of the results was relatively simple it would also be interesting to create a metric that could provide a value that represents a good trade-off between accuracy and model prediction time for even more accurate analysis of a model.

Bibliography

[1] Time series forecasting: Definition, applications, and examples. https://www.tableau.com/learn/articles/time-series-forecasting. (Accessed on 11/08/2022).

[2] Users. https://metrics.torproject.org/userstats-relay-country.html. (Accessed on 21/09/2022).

[3] Mnist database. https://en.wikipedia.org/wiki/MNIST_database#cite_note-1, Jul 2022. (Accessed on 14/09/2022).

[4] L. Alzubaidi, J. Zhang, A. J. Humaidi, A. Al-Dujaili, Y. Duan, O. Al-Shamma, J. Santamaría, M. A. Fadhel, M. Al-Amidie, and L. Farhan. Review of deep learning: Concepts, cnn architectures, challenges, applications, future directions. *Journal of big Data*, 8(1):1–74, 2021.

[5] Y. Bengio, Y. LeCun, et al. Scaling learning algorithms towards ai. *Large-scale kernel machines*, 34(5):1–41, 2007.

[6] G. Bonaccorso. *Machine learning algorithms: popular algorithms for data science and machine learning*. Packt, 2018.

[7] J. Brownlee. How to configure the number of layers and nodes in a neural network. https://machinelearningmastery.com/how-to-configure-the-number-of-layers-and-nodes-in-a-neural-network/, Aug 2019. (Accessed on 18/03/2022).

[8] J. Brownlee. How to choose an activation function for deep learning. https://machinelearningmastery.com/choose-an-activation-function-for-deep-learning/, Jan 2021. (Accessed on 13/09/2022).

[9] C. Buciluǎ, R. Caruana, and A. Niculescu-Mizil. Model compression. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 535–541, 2006.

[10] A. Burkov. *The hundred-page machine learning book*. Andriy Burkov, 2019.

[11] R. Chandra, S. Goyal, and R. Gupta. Evaluation of deep learning models for multi-step ahead time series prediction. *IEEE Access*, 9:83105–83123, 2021.

[12] F. CHOLLET. *DEEP LEARNING WITH PYTHON*. OREILLY MEDIA, 2021.

[13] M. Courbariaux, Y. Bengio, and J.-P. David. Training deep neural networks with low precision multiplications. *arXiv preprint arXiv:1412.7024*, 2014.

[14] B. I. C. Education. What are neural networks? https://www.ibm.com/cloud/learn/neural-networks. (Accessed on 14/01/2022).

[15] I. Freeman, L. Roese-Koerner, and A. Kummert. Effnet: An efficient structure for convolutional neural networks. In *2018 25th IEEE International Conference on Image Processing (ICIP)*, pages 6–10. IEEE, 2018.

[16] I. Goodfellow, Y. Bengio, and A. Courville. *Deep learning*. MIT press, 2016.

[17] D. Graupe. *Principles of artificial neural networks*, volume 7. World Scientific, 2013.

[18] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*, 2012.

[19] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

[20] R. Hurbans. *Grokking Artificial Intelligence Algorithms*. Manning Publications, 2020.

[21] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and$<$ 0.5 mb model size. *arXiv preprint arXiv:1602.07360*, 2016.

[22] H. Ismail Fawaz, G. Forestier, J. Weber, L. Idoumghar, and P.-A. Muller. Deep learning for time series classification: a review. *Data mining and knowledge discovery*, 33(4):917–963, 2019.

[23] M. Jaderberg, A. Vedaldi, and A. Zisserman. Speeding up convolutional neural networks with low rank expansions. *arXiv preprint arXiv:1405.3866*, 2014.

[24] C. Janiesch, P. Zschech, and K. Heinrich. Machine learning and deep learning. *Electronic Markets*, 31(3):685–695, 2021.

[25] H. Jegou, M. Douze, and C. Schmid. Product quantization for nearest neighbor search. *IEEE transactions on pattern analysis and machine intelligence*, 33(1):117–128, 2010.

[26] J. Jeong. The most intuitive and easiest guide for cnn. https://towardsdatascience.com/the-most-intuitive-and-easiest-guide-for-convolutional-neural-network-3607be47480, Jul 2019. (Accessed on 19/08/2022).

[27] Joshinav. How to fine-tune your artificial intelligence algorithms. https://www.allerin.com/blog/how-to-fine-tune-your-artificial-intelligence-algorithms, Jan 2020.

[28] Krishan. Cp decomposition. https://iksinc.online/tag/cp-decomposition/, journal=From Data to Decisions, Jun 2020. (Accessed on 08/02/2022).

[29] H. Larochelle, D. Erhan, A. Courville, J. Bergstra, and Y. Bengio. An empirical evaluation of deep architectures on problems with many factors of variation. In *Proceedings of the 24th international conference on Machine learning*, pages 473–480, 2007.

[30] V. Lebedev, Y. Ganin, M. Rakhuba, I. Oseledets, and V. Lempitsky. Speeding-up convolutional neural networks using fine-tuned cp-decomposition. *arXiv preprint arXiv:1412.6553*, 2014.

[31] V. Lebedev and V. Lempitsky. Speeding-up convolutional neural networks: A survey. *Bulletin of the Polish Academy of Sciences. Technical Sciences*, 66(6), 2018.

[32] X. Liu, F. Zhang, Z. Hou, L. Mian, Z. Wang, J. Zhang, and J. Tang. Self-supervised learning: Generative or contrastive. *IEEE Transactions on Knowledge and Data Engineering*, 2021.

[33] V. Meel. Ann and cnn: Analyzing differences and similarities. https://viso.ai/deep-learning/ann-and-cnn-analyzing-differences-and-similarities/, Feb 2022. (Accessed on 08/04/2022.

[34] M. Middleton. Deep learning vs. machine learning - what's the difference? https://flatironschool.com/blog/deep-learning-vs-machine-learning/, Aug 2022. (Accessed on 23/09/2022).

[35] M. Mishra. Convolutional neural networks, explained. https://towardsdatascience.com/convolutional-neural-networks-explained-9cc5188c4939, Sep 2020. (Accessed on 17/08/2022).

[36] M. Nasr, A. Bahramali, and A. Houmansadr. Deepcorr: Strong flow correlation attacks on tor using deep learning. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 1962–1976, 2018.

[37] K. O'Shea and R. Nash. An introduction to convolutional neural networks. *arXiv preprint arXiv:1511.08458*, 2015.

[38] R. Parmar. Common loss functions in machine learning. https://towardsdatascience.com/common-loss-functions-in-machine-learning-46af0ffc4d23, Sep 2018. (Accessed on 20/09/2022).

[39] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks. In *European conference on computer vision*, pages 525–542. Springer, 2016.

[40] F. A. Saputra, I. U. Nadhori, and B. F. Barry. Detecting and blocking onion router traffic using deep packet inspection. In *2016 International Electronics Symposium (IES)*, pages 283–288. IEEE, 2016.

[41] K. Team. Keras documentation: Conv2d layer. https://keras.io/api/layers/convolution_layers/convolution2d/. (Accessed on 13/09/2022).

[42] K. Team. Keras documentation: Dropout layer. https://keras.io/api/layers/regularization_layers/dropout/. (Accessed on 28/08/2022).

[43] K. Team. Keras documentation: Maxpooling2d layer. https://keras.io/api/layers/pooling_layers/max_pooling2d/. (Accessed on 16/09/2022).

[44] K. Team. Keras documentation: Transfer learning &; fine-tuning. https://keras.io/guides/transfer_learning/.

[45] Y. Wang, Y. Li, Y. Song, and X. Rong. The influence of the activation function in a convolution neural network model of facial expression recognition. *Applied Sciences*, 10(5):1897, 2020.

[46] A. P. Wibawa, A. B. P. Utama, H. Elmunsyah, U. Pujianto, F. A. Dwiyanto, and L. Hernandez. Time-series analysis with smoothed convolutional neural network. *Journal of big Data*, 9(1):1–18, 2022.

[47] R. Yamashita, M. Nishio, R. K. G. Do, and K. Togashi. Convolutional neural networks: an overview and application in radiology. *Insights into imaging*, 9(4):611–629, 2018.

[48] S. Zagoruyko and N. Komodakis. Wide residual networks. *arXiv preprint arXiv:1605.07146*, 2016.

[49] X. Zhai, A. Oliver, A. Kolesnikov, and L. Beyer. S4l: Self-supervised semi-supervised learning. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 1476–1485, 2019.