Universidade do Minho
Escola de Engenharia

Nuno Miguel Azevedo Pereira

Real-Time Multi-Network Deep Learning Facial Recognition for Service Robots

Tese de Mestrado

Mestrado Integrado em Engenharia Eletrónica Industrial e Computadores

Trabalho efetuado sob a orientação do

Professor Doutor António Fernando Macedo Ribeiro

Julho de 2021

ACKNOWLEDGMENTS

This Dissertation represents the culmination of my academic education, filled with the best and moments of my life. Nevertheless, it is important to acknowledge the people that made this road possible and enjoyable.

Thank you to my parents Alberto and Lurdes, for the love and for never cease to believe in me even when I was failing. For always supporting me and care to all my needs, making this graduation possible. To my sister Célia, for showing support and companionship. To the rest of my family, for constant support and understanding over the years. A special thank you to my cousin Joana Martins for the helpful review and corrections.

Thank you to all my friends whom I had the pleasure spent the last years with and helped me through the master's degree. I will fondly remember the times passed in university for the rest of my life.

A special thanks to my supervisor, Professor Fernando Ribeiro, for the guidance, support and constant availability during the dissertation, as well as letting me work the laboratory. A big thanks to my friend Tiago Ribeiro, for always being available to help at the making of this dissertation.

DECLARAÇÃO DE INTEGRIDADE

Declaro ter atuado com integridade na elaboração do presente trabalho académico e confirmo que não recorri à prática de plágio nem a qualquer forma de utilização indevida ou falsificação de informações ou resultados em nenhuma das etapas conducente à sua elaboração.

Mais declaro que conheço e que respeitei o Código de Conduta Ética da Universidade do Minho.

Resumo

Avanços recentes em robôs de serviço genéricos permitiram a sua utilização inovadora em vários meios ambientes, como residências domésticas, hospitais ou centros de saúde. O projeto CHARMIE (Collaborative Home / Healthcare Assistant Robot da Minho Industrial Electronics) é um robô antropomórfico que realiza tarefas em configurações não convencionais usando algoritmos de aprendizagem de máquina, que permitem ao robô tomar decisões racionais com base nas variáveis do meio ambiente que o rodeia.

Com a intenção de criar uma melhor interação entre o robô e seus operários, esta dissertação apresenta um sistema de Deep Learning para reconhecimento facial de várias instâncias. Ao identificar a pessoa com a qual está a interagir, o robô adapta-se para realizar as suas ações pré-definidas atendendo às necessidades dela. O sistema é composto por várias redes neuronais de modo a compor os três módulos constituintes. Uma rede convolucional em cascata de várias instâncias procura zonas faciais, como olhos, nariz ou boca, prevendo as coordenadas que permitem isolar as caras detetadas, detetando todos os rostos exibidos ao usar uma câmara. A rede neuronal Inception-ResNet foi treinada num dataset extenso apropriado, funcionando como extrator de características faciais. Isso gera os vetores característicos de pessoas que serão usados como input para um classificador SVM, um modelo de aprendizagem máquina que irá prever a qual das identidades esses vetores correspondem. Essa implementação de princípio a fim dos três módulos permitiu a deteção e reconhecimento facial para ser usado como um método de identificação de usuários em tempo real. O método resultante foi implementado no robô de serviços CHARMIE.

Essas tarefas foram realizadas com alunos e professores do Laboratório de Automação e Robótica (LAR) da Universidade do Minho. Para isso, foi usado um conjunto de imagens obtidas a partir de vídeos individuais dessas pessoas, invertendo ou rodando algumas dessas imagens aleatoriamente de modo a criar um sistema mais robusto

Palavras-Chave

Deep Learning, Deteção facial, Multi-Rede, Reconhecimento Facial, Robôs de serviço

ABSTRACT

Recent advances in generic service robots have shown their introduction in various novel environments such as domestic and healthcare facilities. The CHARMIE (Collaborative Home/Healthcare Assistant Robot by Minho Industrial Electronics) project is an anthropomorphic robot that performs generic service tasks in non-standardized environment settings using machine learning algorithms, which allow the robot to make rational decisions based directly on the surrounding environment.

In order to ensure an enhanced interaction between the robot and its specified users, this dissertation presents a real-time multi-stage deep learning pipeline for facial recognition algorithm. By detecting which of the pre-trained users the robot is interacting with, it can adapt its actions to best fit the user's needs. The multi-network system is composed of three modules. A multi-stage cascaded convolutional network searches for facial landmarks, such as eyes, nose, or mouth, predicting its bounding box coordinates, detecting all the faces showcased in the camera frame. An Inception-ResNet deep convolutional network was trained on a large dataset for facial feature extraction, comparing different hyperparameter settings and fine-tuning. That generates the feature vectors and provides an amplified network for facial recognition and an SVM classifier to categorize each of the faces recognized to the correct user. The combination of the three modules allowed an end-to-end facial detection and recognition that can be used as a real-time user-based identification method. The resulting method was furthermore implemented on the general service robot CHARMIE.

Those tasks were carried out on students and teachers from the Laboratory of Automation and Robotics (LAR) of Minho University. For that, a dataset containing those people was built using images fetch from individual videos, with random frames being rotated or flipped to create a more robust system.

Keywords

Deep Learning, Facial Recognition, Face Detection, Multi-Network, Service Robots,

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACRONYMS

2D - Two-Dimensional

3D - Three-Dimensional

AI - Artificial Intelligence

ANN - Artificial Neural Network

API - Application Programming Interface

CHARMIE - Collaborative Home/Healthcare Assistant Robot by Minho Industrial Electronics

CNN - Convolutional Neural Network

CPU - Central Processing Unit

CUDA - Compute Unified Device Architecture

DEI - Industrial Electronical Department of Minho University

DNN - Deep Neural Network

FIFO - First In First Out

GPU - Graphics Processing Unit

HOG - Histograms of Oriented Gradients

HRC - Human–robot collaboration

ILSVRC - ImageNet Large Scale Visual Recognition Competition

LAR - Laboratory of Automation and Robotics

LFW - Labeled Faces in the Wild

LPP - Locality Preserving Projections

ML - Machine Learning

MLP - Multilayer Perceptron

NMS - Non-Maximum Suppression

NN - Neural Network

O - Net-Output-Network

PCA - Principal Component Analysis

P-Net - Proposal-Network

ReLU - Rectified Linear Unit

R-Net - Refinement-Network

RoI - Region of interest

SVM - Support Vector Machine

t-SNE - T-Distributed Stochastic Neighbor Embedding

TSV - Tab-separated values file

1. INTRODUCTION

Visual perception and the information one collects from it is an essential ability in human behavior. The analysis of the information gathered allows humans to make structured decisions, with its interpretation being based on cognitive processes and previous knowledgeable interactions.

Thus, endowing robotic systems with the sense of vision to implement cognitive systems is quite a desirable yet challenging process. The concept of Robot Vision refers to the ability of a robot to understand the environment in which it is located and use that information to perform tasks through artificial vision algorithms (Kragic & Vincze, 2009). That was enabled by extensively researches and breakthroughs in detection and recognition tasks, with Deep Learning fields of study having a determinant role. Those benchmarked classification tasks involving various cases of detection as well as identification. Hence, being able to perceive the surrounding variables branched the focus into various relevant techniques of visual recognition by robots (Khaligh-Razavi, 2014). That culminates in machines capable of accurately evaluate and interact with a broad range of scenarios by observing different environments and recognizing valuable data.

State-of-the-art service robots provide aid in various novel environments, such as domestic and healthcare facilities, by performing generic service tasks based directly on the surrounding environment. This dissertation focused on providing a more collaborative and cooperative communication tool between the robot and its operators/patients/users.

1.1 General Motivation

Collaborative robots are designed to perform tasks with people in a shared work environment. It's mainly applied in the Industrial field, where its precision and repeatability are desirable factors, as well as the faster and more efficient manufacturing processes, while guaranteeing a safe interaction (Villani, Pini, Leali, & Secchi, 2018). Additionally, the flexibility and shift demand of new products or procedures provided these robots the advantage over fully automated ones, with the former being quickly performed with the guidance of a user.

Service robots are a type of machines that help in day-to-day tasks in different environments such as household, offices, or healthcare (Bauer, Wollherr, & Buss, 2008). The aim is to lower the staff workload by performing menial, dull tasks, but also to help solve complex chores with its users. They differ from the industrialized robots due to its form and function, having a more autonomous internal

control system, being mobile, and interacting with humans with a wide range of ways. These types of robots are used in a high spectrum of purposes, with recent advances uplifting their use in various novel environments, including domestic or healthcare facilities (Holland et al., 2021). Their introduction was possible due to technological improvements on manufacturing of cameras, sensors, and other robotic control tools such as speech recognition, as well as rapid advancements on Artificial Intelligence (AI), enabled by the availability of extensive big data analysis.

A desirable feature is the ability of service robots to recognize people in its surroundings, providing a more natural Human–robot collaboration, by detecting which of the users is interacting with, it can adapt its actions to best fit the user's needs. This thesis approaches this challenge using artificial vision and AI methods, namely deep supervised learning using multi-Convolutional Neural Networks (CNN) (Lecun, Bottou, Bengio, & Ha, 1998).

1.2 Objectives of the Dissertation

The main objective of this dissertation was the implementation of a real-time face detection and facial recognition system implemented on CHARMIE, recognizing a set of pre-determined people with confidence or being able to learn new identities previously unknown. This model was then tested on said members of the LAR of Minho University. For that, the following tasks were performed:

- Study of several NN architectures regarding state-of-the-art face detection and facial recognition, with different learning approaches used;
- Definition and installation of the necessary tools for the development of the model, including the most suitable programing language;
- Implementation of the defined NN used for face detection and recognition;
- Selection of the most suitable dataset and creation of a custom dataset containing students and teachers associated to the LAR;
- Training of the CNNs with the referenced datasets;
- Testing of the NNs and selection of the best parameters;

The aim is that the system can detect all the faces presented in a camera frame regardless of the position where it appears and different angles of the head. The system must then draw bounding boxes over the proposals, and predict the name of the shown person. That way, the robot can understand who it is interacting with and can act accordingly.

1.3 Scientific Contributions

This dissertation theme and work was implemented on a larger project named CHARMIE, developed by the LAR of Minho University of DEI. CHARMIE is an anthropomorphic robot that performs generic service tasks in non-standardized environment settings. Its conceptual shape (Figure 1) is designed in a way that its users can be more prone to interact with it by making it human-like. Additionally, its shape and dimensions make it more practical and easier to introduce CHARMIE into an environment designed for human work.

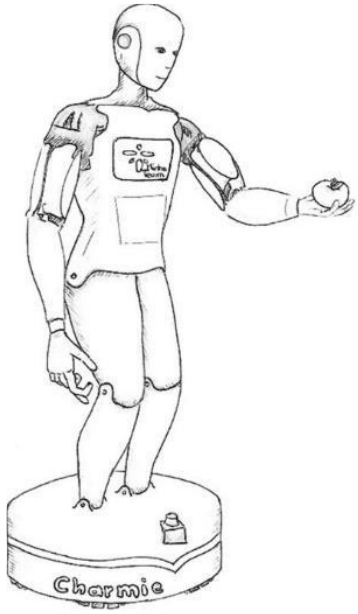


Figure 1 - Conceptual sketch of the anthropomorphic robot CHARMIE.

The system's design (Figure 2) is composed of Head, Robot Arms, Lifting Mechanism and Torso and a motion platform, allowing it to move dynamically in any direction (Gonçalves et al., 2019). A primary prototype was previously built (Figure 3) with additional parts of the project being under development at the time this dissertation is being written.

CHARMIE's goals are to provide aid in several chores in household and healthcare environments, from nursing homes to hospital facilities and including domestic homes. For that, the ability of understanding its surroundings and make rational decisions based on the information gathered is needed. To endow the robot with that ability, the LAR has been implementing different types of machine learning (ML) algorithms. For Autonomous Driving, such methods involve non-linear Navigation, Mapping, and Obstacle Detection and Avoidance. To provide communication tools with its users, the implemented processes include Person Detection and Recognition, and Speech and Gesture Recognition. For the

identification of certain items and being able to interact with them, such as pick-and-place and object delivery tasks, Object Detection and Recognition was also applied.

Figure 2 - Developed anthropomorphic design.

In order to benchmark new technologies developed for CHARMIE, service and assistive robotics competitions (Basiri, Piazza, Matteucci, & Lima, 2019) provide a common framework for a high-rigor benchmark of smart and autonomous systems. RoboCup@Home is regarded as a top competition in the field of domestic service and assistive robotics (Holz & Iocchi, 2013; Iocchi, Holz, Ruiz-Del-Solar, Sugiura, & Van Der Zant, 2015), where they must perform a set of pre-determined tests to aid in day-to-day realistic non-standardized home environment setting. Earlier versions of CHARMIE have already participated in RoboCup@Home, and the most recent version is focusing on introducing itself to the competition with qualification material in the next years.

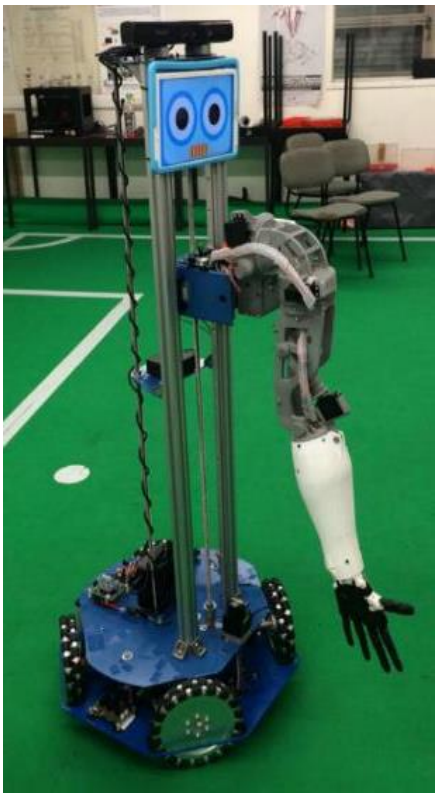


Figure 3 - Primary prototype assembled.

1.4 Structure of the Dissertation

This document is divided into 6 chapters: Introduction, Literature Review, Methodologies, Results, Conclusions, and Future Work.

On the Introduction chapter, a brief description of the work is presented, as well as the motivation as to why it was produced, outlining the objectives necessary to accomplish the result and the scientific contribution of it.

Throughout the Literature Review chapter, the study of the state-of-the-art is presented. First, the study of NN is presented, from the earlier and basic concepts to the more complex architectures used nowadays. After that, methods used for human face detection and recognition are introduced, demonstrating how the introduction of AI improved the accuracy of those fields, describing notorious works and algorithms.

The Methodologies chapter describes the implementation of the end-to-end framework, starting by describing the software tools used. Then, the schematic structure of the work is displayed, with every part being analyzed with the methods implemented and the respective theoretical fundaments that corroborate them.

Then, the Results chapter compares the different variations and outcomes obtained regarding the previous chapter's algorithms. Those include the detection of human faces in photos and videos, as well as in real time processing, the acquisition and pre-processing of individual images of members of LAR (to create a dataset necessary to train the NNs), and the results of the training of the NN using different data visualization tools. Finally, all those parts are connected, displaying the result of the dissertation. Also, a discussion regarding the obtained results is always presented after every module of the framework.

The Conclusions chapter presents the outcomes of the work done and its analysis regarding the outline objectives of the dissertation, with the Further Work chapter evaluating improvements that can be made.

2. Literature Review

In this chapter an overview of the state of the art is presented, from basic concepts to the more complex architectures used nowadays. After that, methods used for human face detection and recognition are presented and compared, demonstrating how the introduction of AI improved the accuracy of those fields, describing notorious works and algorithms.

2.1 Introduction and basic definitions to Neural Networks

ML refers to an Artificial Intelligence field of study that promotes the ability of machines to make decisions autonomously, i.e., without being programed explicitly for that purpose. This is done by the development of algorithms based on mathematical and statistical approaches, where the system learns by being exposed to data, the same way as humans. A popular branch of this field, and the foundation of deep learning algorithms, is Neural Networks (NNs) (Nielsen, 2018). Simply put, NNs are ML models inspired by the way the human brain works, where the model maps the input data into desirable outputs. For example, for classification purposes, such as the topic of this dissertation, it tries to predict the correct output pair of the data provided into these models by pattern recognition.

The first NN as an electrical circuit was proposed in 1943 by Warren McCulloch, and a learning algorithm was proposed by Donald Hebb six years later (Capaldi, 1992). This introduced the concept of perceptron (or node), which consists on a basic unit of the neuronal network that simulates a neuron (Figure 4). Despite its simplicity, the perceptron is a binary classifier that can learn and solve complex problems.

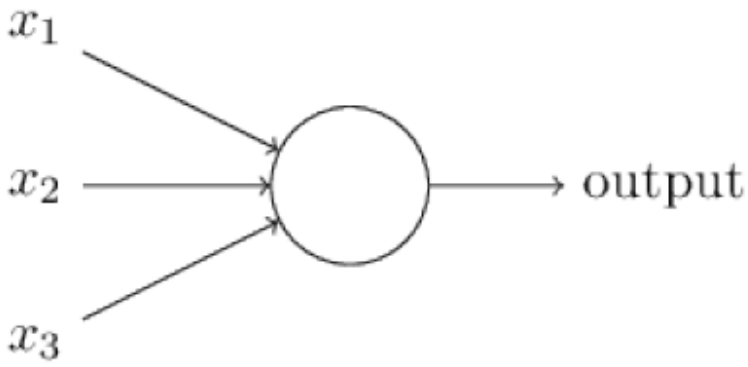


Figure 4 - Perceptron module.

The learning process can be summarized in equation (1):

$$output(x) = \begin{cases} 0, & if \sum_k x_k w_k + b < threshold \\ 1, & if \sum_k x_k w_k + b \geq threshold \end{cases} \quad (1)$$

where K is the number of the connection, X is the input value and W is the weight of the inputs. The full model can be seen in Figure 5.

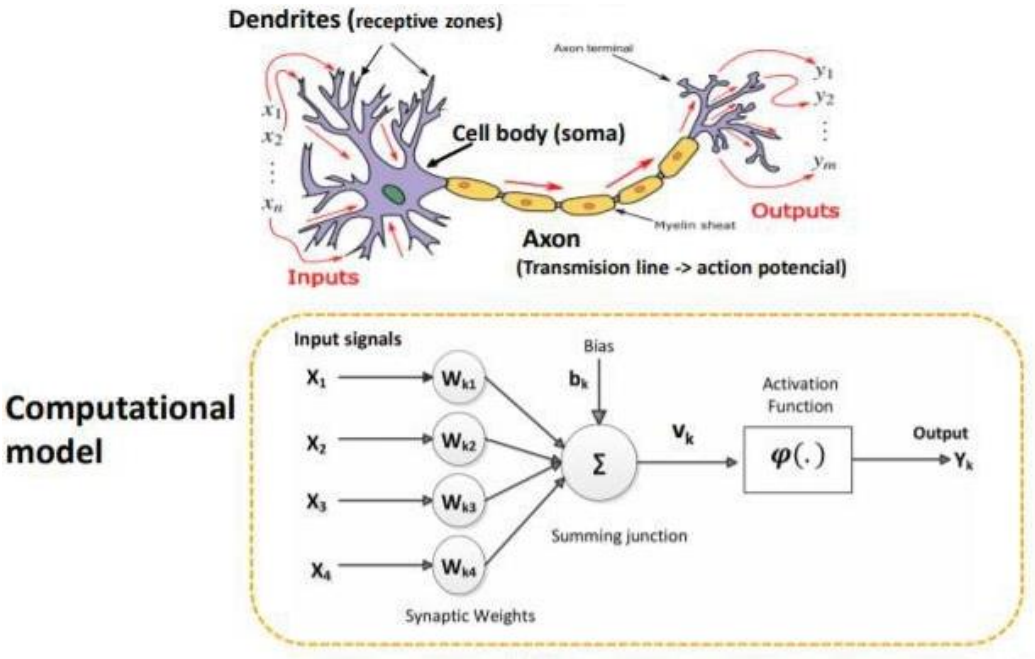


Figure 5 - Perceptron Learning Module.

The model receives input values by multiplying each value by a characteristic weight. Once all inputs are received, coming from other perceptrons or from the data itself sent to the network, they are added, together with a Bias (value for which the output of the neuron converges). Then the model passes the information through an activation function, which limits the output value to acceptable values, getting output value of each perceptron.

Activation functions, or Transfer Function, defines how the weighted sum of the inputs converts into the output. Normally, those functions are non-linear, limiting the value into presentable values. The most commonly used function is the ReLU (Schmidhuber, 2015), defined by Equation (2):

$$f_{(x)} = \max(0, x) \quad (2)$$

where x is the input value. The output value is linear if positive or 0 if negative, as observed in Figure 6.

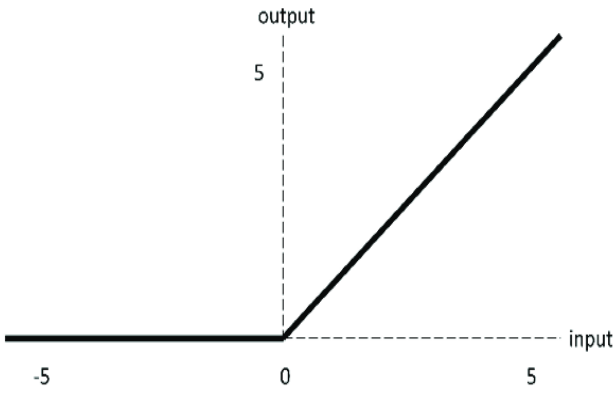


Figure 6 - ReLU activation function.

Nair & Hinton *et al.* (Nair & Hinton, n.d.) proved that activation functions perform better than sigmoid or hyperbolic tangent functions in Restricted Boltzmann Machines by preventing saturation of gradients, which popularized its use. Glorot, Bordes & Bengio (Glorot, Bordes, & Bengio, 2011) showed that these activation functions speed up the training algorithms.

Shallow NNs (Figure 7) group the perceptrons into three types of processing layers: input layer, output layer and, between them, hidden layers (Nielsen, 2018). These are fully connected layers, so that all nodes are connected to every node of the next layer, from start to finish. These models can already perform classification tasks, with Min & Chung *et al.* (Min & Chung, 2019) achieving 100% accuracy with shallow sigmoid-type NNs for personalized datasets following a linear separability condition.

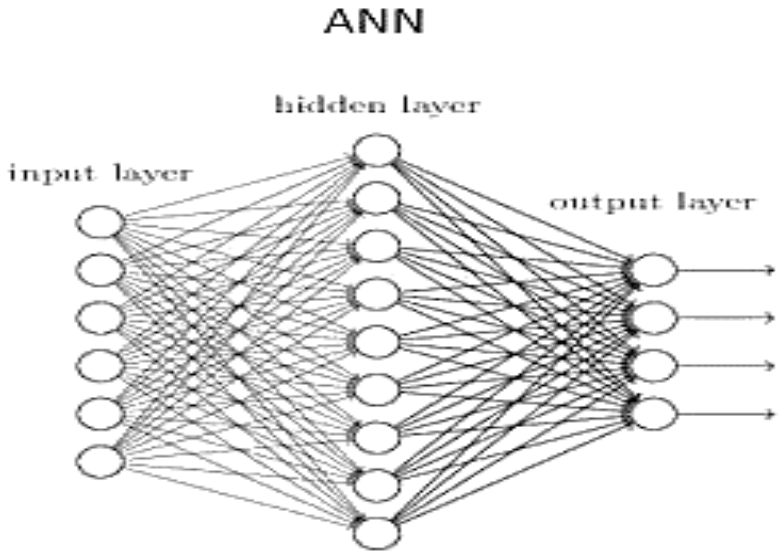


Figure 7 - Shallow Neural Network.

2.2 Deep Learning

Deep learning methods revolutionized the way robotic systems recognize patterns through the data to which they are exposed (Lecun, Bengio, & Hinton, 2015). These methods derive from ANNs, but with several layers of processing, allowing better recognition. That was possible due to the availability of extensive training datasets for deep learning solutions, as well as larger CPUs and faster GPUs. Thus, this research field branched into Deep Neural Networks and CNN (O'Shea & Nash, 2015). The learning process is called 'training': the model receives inputs of a certain desirable task, altering itself over time accordingly to correctly predict what the input means. This method can be applied to different applications, with Classification, Regression and Clustering problems being some of the more researched and successful branches. The two main categories for training are Supervised and Unsupervised Learning, with the first being used more for classification and regression methods, and the second being mostly used on clustering.

2.2.1 Deep Neural Networks

DNNs (Figure 8) have the same structure as ANNs, but with more than one hidden layer being used for nonlinear patterns, requiring further information processing. DNNs are Feedforward networks, where the flow of information is passed from layer to layer from the input up to the output, with no feedback between the layers.

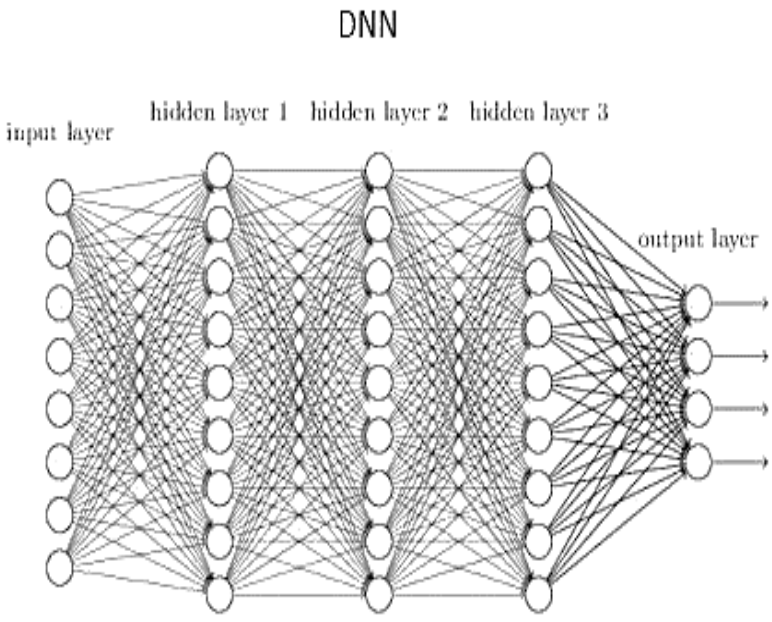


Figure 8 - Deep Neural Network (DNN).

DNNs, or MLPs, are specified by its hyperparameters and parameters. Hyperparameters are the variables related to the architecture, such as the number of layers and nodes per layer, activation functions, etc., and the specifications of the learning process, such as the lr. These hyperparameters are set manually before the training process commences. The parameters are the configuration variables whose values are toned by the model, like the weights of the composed perceptrons. MLPs are used for classification and regression when the data used is presented in a tabular form, namely as a spreadsheet. Although DNNs can recognize images, they would usually have to be reduced into an array, normalized so that the desired information was in the center and with its entire data being the same size. Also, the fact that an image on a computer is represented by a two-dimensional matrix in which each pixel represents an input node with a corresponding weight would cause the training of that network to use a lot of computational power.

2.2.2 Convolutional Neural Networks

Each pixel of an image is considered an input neuron, and its value is given by its intensity. As an example, the MNIST dataset used in (Lecun, Bottou, et al., 1998) is a 28 x 28 pixel image of handwriting numbers from 0 to 9, containing 60.000 images for training and 10.000 for testing. That means that the network would have 784 input nodes (784 = 28 x 28). Having fully connected layers, DNNs connect all neurons from one layer to the next. This causes pixels at various points in an image to be treated in the same way. Given the spatial structure of an image, pixels in a given location have very similar characteristics. Knowing this, it is possible to map locations in the image, facilitating processing and improving the collection of characteristics.

Convolutional Neuronal Networks (CNNs), developed by LeCun (Lecun, Bottou, et al., 1998), benchmarked NNs in terms of image processing and vision by computers, approaching the drawbacks of their fully connected counterparts. Being more robust and automatic, they have the advantage of relying on the spatial structure of an image (height, with and depth), removing the need to interconnect all neurons between layers. The depth does not refer to the total number of layers within the ANN, but to the third dimension of an activation volume.

CNNs were inspired by the work on the visual cortex in the brain of mammals (Hubel & Wiesel, 1959)(Hubel & Wiesel, 1968), assessing how they process information. These studies concluded the existence of several layers of neurons between the fable and cortical areas, with local connections. That way, neurons of a given layer perform the same type of processing, and the size of the neuron's

reception fields (i.e., the region of the cortex whose signal affects the operation of a neuron) increases from layer to layer.

Lecun et al. (Lecun, Bottou, et al., 1998) states that these networks are based on three premises: local connections, shared weights, and spatial sub-sampling or pooling. The term "local connections" means that these networks extract the features of an area in an image, taking advantage of correlated pixels representing the same feature with the same local values. Shared weights allow the pattern to be discovered to be invariant to the location in the image, given that the extraction filter result comes from the same connection regardless of the position on an image. These networks are built by stacking several layers of three different types, commonly known as Convolutional layer, which performs spatial sub-sampling, resulting in Pooling Layer and Fully Connected Layer.

The first layer of a CNN will usually be a Convolutional Layer. They extract the desired characteristics in an image using filters on regions of the image. The convolution is carried out using a flexible filter or kernel, and shifting it along the entirety of the layer, also called receptive field, extracting the features. Each kernel, which mathematically consists on a matrix array of a set of weights, calculates the scalar product by convolving the weights by the pixels of the input. From this, the network learns by acknowledging activations, i.e., kernels that understand specific features in an image. By repeatedly sliding the filter across the spatial dimensionality of the input, the completed process results in a 2D activation/feature map, represented in Figure 9.

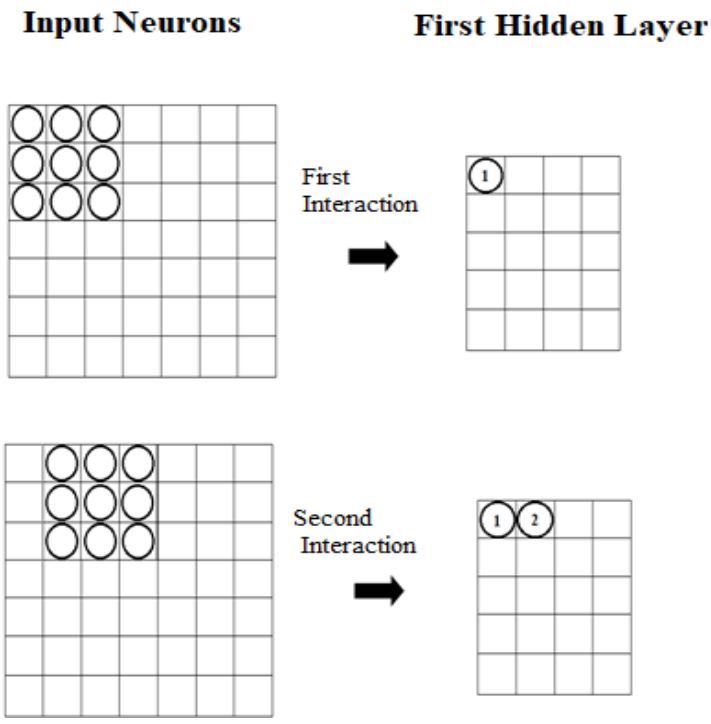


Figure 9 - Activation map.

Convolutional layers are also able to reduce the complexity of the model, optimizing its output with three hyperparameters: depth, stride and zero pathing. The depth is manually set through the number of nodes within the layer in the same input. Reducing this hyperparameter significantly reduces the number of neurons of the network but can also reduce the pattern recognition ability. The number of units in which the filter slides is called stride. As seen in Figure 10, the filter travels across the input one unit at a time.

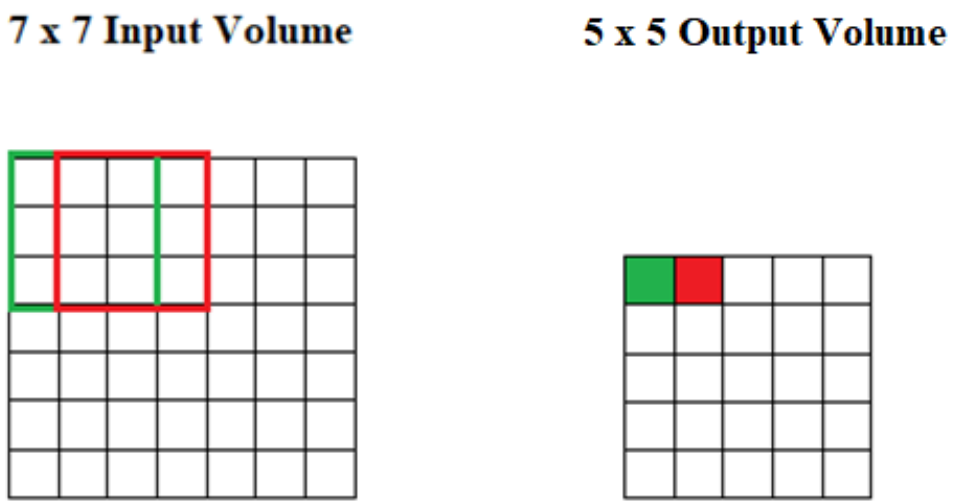


Figure 10 - Stride of 1 on a 7 X 7 input volume.

The stride is always chosen so that the output volume is an integer and not a fraction. In Figure 11, it is possible to see that, when using a stride of 2, the output will be of a spatial dimension much smaller than the input from which it resulted. A small stride results in a heavily overlapped receptive field, producing a large number of activations, whereas increasing the stride reduces overlapping and produces lower spatial dimension outputs.

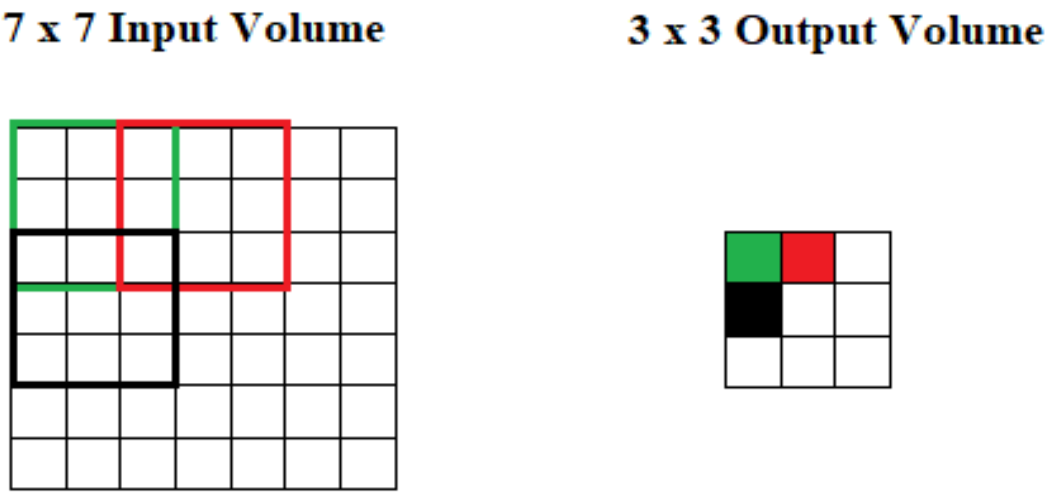


Figure 11 - Stride of 2 on a 7 X 7 input volume.

The output of the Convolutional layer can be calculated through equation (3):

$$O = \frac{(W - K) + 2P}{S + 1} \tag{3}$$

where O represents the output size, W the input dimension (height x width x depth), K the filter dimension, S the stride and P the zero-padding quantity. Zero-padding the process implies applying a combination of pixels of value 0 around the input, as represented in Figure 12.

With this, the output spatial reduction will be smaller, while no important input information is lost, and more convolutions can be applied to be able to remove more features.

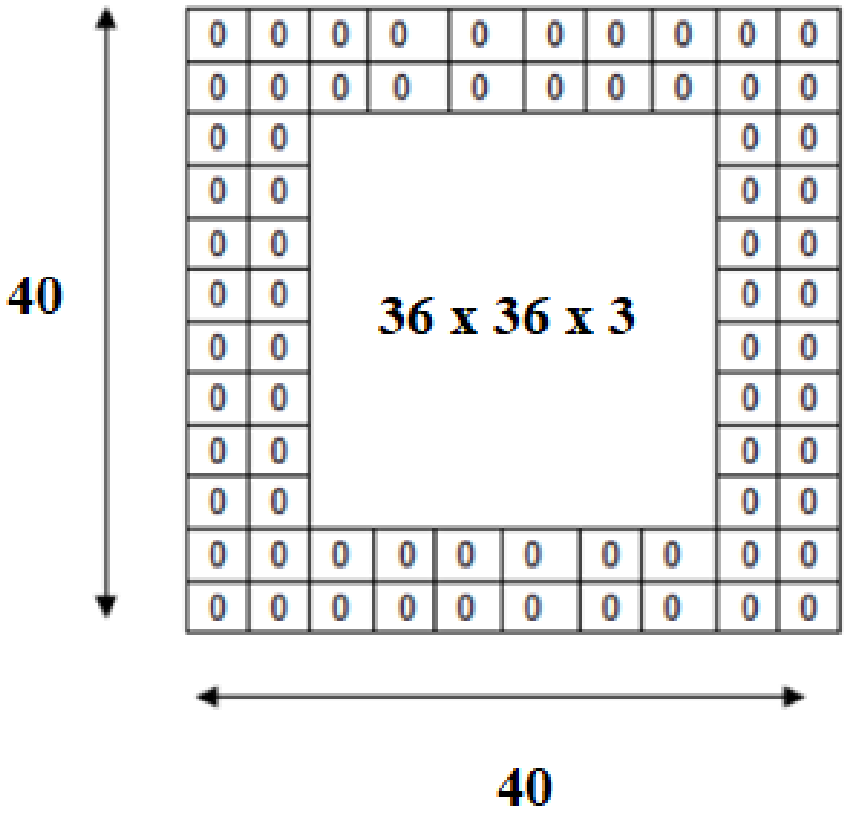


Figure 12 - Zero-pathing technique.

Pooling Layers are applied to perform a down sample of the input, when the dimensionality of the layers needs to decrease, further reducing the computational complexity by lowering the parameters.

These layers are used when a set of specific features are extracted through the activation maps of previous convolutional layers, and their exact spatial location is no longer important to their position relative to the other features. With this, the layers will be drastically reduced, resulting in fewer parameters, thus lowering computational resources. The most common Pooling methods are Max and Average Pooling, as seen in Figure 13 and Figure 14 respectively.

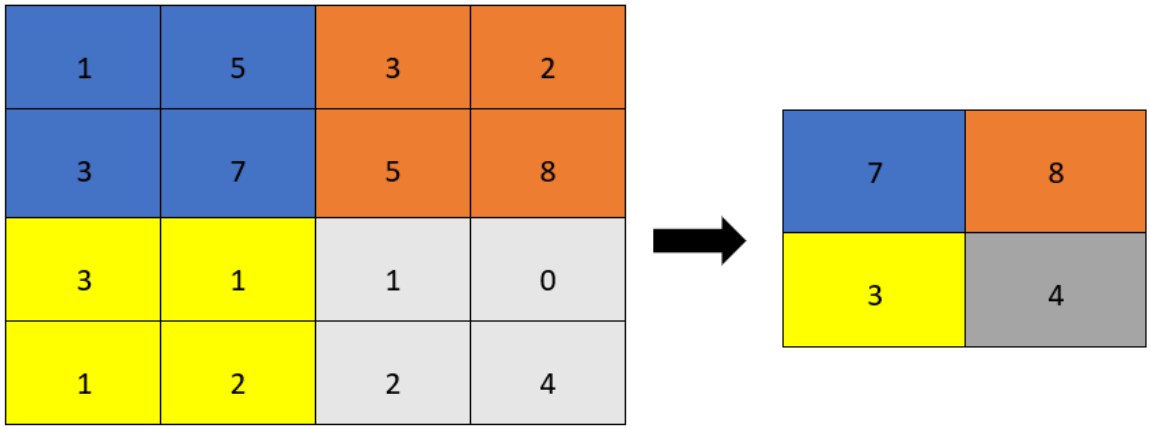


Figure 13 - Max Pooling on an 8 X 8 activation map with stride of 4.

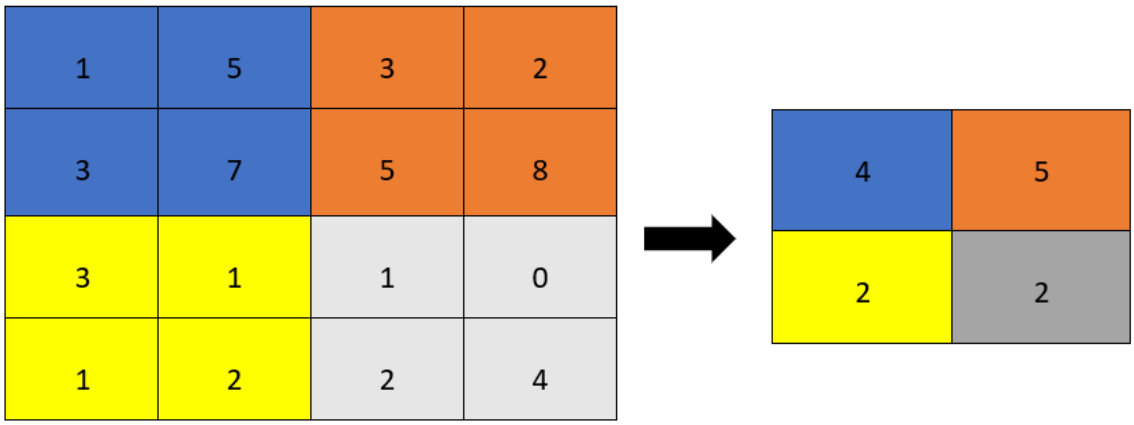


Figure 14 - Average Pooling technique on an 8 X 8 activation map with stride of 4.

Through a filter and a stride, Max Pooling method checks the highest pixel value, eliminating the rest. The final output is then left with the larger value of each sub-region, not changing the features of the image, but considerably decreasing its spatial value. Average Pooling calculates the mean value between the nodes.

Fully Connected Layers will be used at the end of the network after the desired features have been extracted through convolutions and pooling. The output of those layers is a one-dimensional array. It

represents the inputs of the network with reduced dimension. Like DNNs, all these layer's neurons are interconnected to all the previous layer's neurons, with its size being manually determined. A Convolutional network can have various Fully-connected layers, reducing the dimensional size without the need of creating more activations nor searching for more features in the spatial dimension of the images, to avoid create overfitting.

2.3 Training of Neural Networks

For the neural network to be able to classify the input of data sent, it needs to be able to recognize and identify patterns in it and understand their meaning. This is achieved through learning and training methods, perform a classification validation with precision tests. For that, a training set is necessary, with inputs of data of a desired subject grouped into batches and fed into the network.

Typically, training a network is divided into training cycles, called epoch, with a batch of data being fed into the network by a feed forward manner, where the model then tries to predict what this data means. The learning process modifies the weights of the connections between the perceptrons, initially random, so that it can achieve expected results. The measurements of the errors in the performance of the model are estimated by the cost/loss function. Having a small cost function is the objective of the model, so that it has minimal losses and maximum precision. The weight values are updated with Backpropagation. Rumelhart *et al.* (Rumelhart, Hinton, & Williams, 1986) first used Backpropagation on neural algorithms by repeatedly adjusting the weights of the connections in the network so as to minimize the difference between the actual output vector of the net and the desired output vector. In CNNs, the algorithm updates the kernel values that extract the features. The weights are updated via an optimization process, where the gradients are calculated into a gradient descent method (Bottou, 2010). That way, by back forwarding onto the network from the output layer into the input layer, the model finds the minimum values gradients in the direction needed to reduce errors. Then, the weights are optimized in the opposite way of calculated gradients.

The training method can be divided into two major approaches: supervised and unsupervised learning. Supervised Learning focuses more on classification and regression problems, whereas Unsupervised Learning focus on Clustering issues or dimension. These two approaches are explained below.

2.3.1 Supervised Learning

Supervised Learning is a training method where the dataset used for training is labeled, i.e., every point of data is paired with an individual predefined ground truth, namely the output class desirable of that

data. That way, the training algorithm can assess if it got the correct answer. This method focuses on learning through the training process to correctly predict new instances of data without the attached label, intending to enable the model with the capacity of generalizing the problem outside of previously seen examples. For that, it needs to map the data with a function to recognize output features to categorize them by correctly predicting its samples.

For classification problems, the final output are discrete labels, like fruits, animals, and people. Those can be binary classifiers for True or False problems like detection, or multiclass classifiers, such as the MNIST dataset previously discussed (Lecun, Cortes, & C.J.C., 1998). MNIST dataset is a 0 to 9 handwritten digits dataset, used in various classification learning problems as an introductory exercise. Nowadays, various databases for image classification are available, which is instrumental for computer vision and deep learning research improvements. ImageNet project (Russakovsky et al., n.d.) offers a large visual database designed for visual object recognition software research, with more than 14 million images hand-annotated indicating what topics are considered as pictures in more than 20000 categories. This project also hosts a yearly software competition since 2010 called the ImageNet Large Scale Visual Recognition Challenge. This competition evaluates algorithms for object detection and image classification at large scale, allowing researchers to compare progress in detection across a wider variety of objects, and was crucial to various benchmarking progresses over the last years.

Regression is a process of predicting a continuous value by finding the correlation between dependent and independent variables. These techniques are used to predict, for instance, pricings and market trends.

2.3.2 Unsupervised Learning

Contrary to the previous approach, unsupervised learning systems use untagged data, discovering the patterns and information without assistance. This allows users to perform more complex tasks, although its unpredictability can be a drawback, resulting in a more expensive training process, when compared to supervised learning.

The main category of this method is Clustering. It analyses the structural components of the data and segregates them by similarity. Entities in each cluster are more similar to each other than to entities of different clusters. For that, the training algorithm needs to understand how the data is structured. Then, considering the data located in an n-dimensional space, intra-variance between samples needs to be minimal while having maximum inter-variance between different classes.

Another powerful method used is dimension reduction of data. Input variables are referred as dimensionality. As an example, feeding the network with an array of 10 positions means the input is a 10-dimension input. As such, using an image as input can result in a hundred or thousand dimensions. By reducing its complexity, while retaining the meaningful properties of the original data, it is represented with fewer dimension, and the model tends to consequently be less sparse and not overfit. This technique is also used for visualization purposes by reducing the data into a 2D or 3D Euclidean space. The PCA (Jollife & Cadima, 2016) technique performs a linear projection by calculating the Principal Components, new variables that are linear combination of the original data, and creating new uncorrelated variables that successively maximize variance. Additionally, the t-SNE (Van Der Maaten & Hinton, 2008a) is a non-linear method based on Stochastic Neighbor embedding that computes K-dimensional data into a lower dimensional mapping by performing several region based transformations.

2.4 Convolutional Network Architectures

This section provides an overview of some CNN structures that were important for the development for the field with different benchmarking records in recognition tasks, as well as recognizing the improvements those methods and ideas had in architecture designing.

The first notorious architecture of a CNN was conceived by Lecun *et al.* (Lecun, Bottou, et al., 1998) who named it Lenet-5, and was a pioneer of this type of networks. It was used to test written handwritten digits for zip codes at post offices, now known as the MNIST dataset (Lecun, Cortes, et al., 1998). Being the first structural architecture for these types of networks, Lenet-5 introduced the concept of convolutional networks, incorporating the spatial mapping in an image, where it extracted the features present through designated filters.

The architecture was designed of several layers, namely:

1. Input layer at the beginning of the network – considered as initial image;
2. Convolutional Layer - created from the convolutions of the initial input, where 5x5 filters with stride of 1 were applied;
3. Average Pooling Layer – created by subsampling with a 2x2 filter with stride of 2, reducing the spatial size of the initial image;
4. Fully-connected Layer – sorts the data received from the network, predicting the input.

Although the method proved to be efficient, being acknowledged by its innovation at the time, its expensive computational cost required the model to be trained for 2 to 3 days until it could predict accordingly a simple dataset. Combined with lack of training data for real life problems, this approach proved unsustainable for better purposes than recognizing simple features in small images.

With better performance CPUs and GPUs, multi-core parallel processing architecture allows the computer to perform expensive computational calculations, which shortened the time spent by a lot. Additionally, it led to a significant increase in labeled datasets publicly available, resulting in a big breakthrough in this field of study. Alex Krizhevsky *et al.* (Krizhevsky, Sutskever, & Hinton, 2017) proposed the AlexNet architecture in 2012 to participate in the ILSVRC competition (Russakovsky et al., n.d.). Hinton *et al.* (Hinton, Srivastava, Krizhevsky, Sutskever, & Salakhutdinov, 2012) states that small datasets with deeper network tend to overfit the model, the state where the framework adapts and performs well to the training data, 'memorizing' its details, but struggles to generalize outside of it. AlexNet presented three techniques for that problem: 1) use of Data augmentation, 2) making the data more diverse, and 3) using the concept of Dropout. That way, this method prevents complex co-adaptations on the training data by temporarily removing a percentage of random weights pixels in the hidden layers, along with all its incoming and outgoing connections. The AlexNet architecture is similar to Lenet-5, but has a larger number of layers, therefore more complex or, with the correct neural network nomenclature, deeper. Having a simple layout compared to more modern architectures, AlexNet architecture design was composed by 5 convoluted layers, max-pooling layers, dropout layers, and 3 fully-connected layers, with the non-dropout variation having only one fully-connected layer, using 60 million parameters. The major difference in architecture features was the introduction of overlapping on the pooling layers. With s being the space between the pooling grid and z x z the center of the pooling grid location, the authors set $s < z$, reducing the top-1 and top-5 error rates by 0.4% in comparison to the $s = z$ counterpart with equivalent output dimension. It was also observed that overlapping pooling was less prone to overfit.

For training dataset, the network used the ImageNet data for the ILSVRC challenge, a subset of a larger dataset (15 million images with a total of 22,000 categories), containing 1000 subjects, with over 1.2 million training images, 50000 validation images and 150000 testing images. For this, ReLU was used as activation function to prevent saturation, achieving 25% error rate on CIFAR dataset (Krizhevsky & Hinton, 2009), which is six times faster in relation to the tanh activation function previously used, as well as SoftMax function as classifier. The training was also done using 2 parallel GPUs, given that, at the time, processing 1.2 million images was not conceivable with only one. This architecture was

revolutionary in the area of deep learning. The top-5 error was reduced from 26.2% from the previous years to 15.3%, in the 5 classes where the model predicted the worst solution. (*"ImageNet Large Scale Visual Recognition Competition 2012 (ILSVRC2012)." [Online]. Available:image-net.org/challenges/LSVRC/2012/*, n.d.) That demonstrated that CNNs obtain better performance in image recognition tasks than previous studied algorithms, making this the paradigm for this field of study.

Introduced by Simonyan & Zisserman *et al.* (Simonyan & Zisserman, 2014), VGG-Net explored the concept of depth on CNNs in order to increase the model's accuracy in image recognition settings.

With various configurations proposed, highlighting the 16 and 19-layers layout, represented in Table 1, the convolution filter was modified to a 3 x 3 design with a stride of 1 in order to reduce the millions of parameters used. To compensate the reduction of parameters while maintaining the performance, the framework stacked 2 convolutional layers in a row. For instance, using 2 layers with a 3 X 3 filter results in 2 X 3 X 3 = 18 parameters, which has the same receptive field to one convolutional layer with a 5 X 5 filter with 1 X 5 X 5 = 25 parameters, decreasing substantially the number of parameters by 28%. Even with a significant decrease, those models were still very computational heavy. The trade-off in a better performance network was the use of about 3 times the parameters used in comparison to the AlexNet counterpart. VGG-Net participated in the ILSVRC in 2014 on the same dataset as AlexNet, where it obtained a top-5 error-rate of only 7.3%, placing 2nd in classification and 1st in location (*ImageNet Large Scale Visual Recognition Competition 2014 (ILSVRC2014)." [Online]. Available:image-net.org/challenges/LSVRC/2014/*, n.d.). This reinforced the notion that the deeper and easier CNNs, understand representations in images, the more accurate the results will be.

Table 1 - VGG-Net 2 best performance architecture configurations.

VggNet Convolution configuration	
16 weight layers	19 weight layers
Input (224 X 224 RBG image)	
Conv3-64 **Conv3-64**	Conv3-64 Conv3-64
Max Pool	
Conv3-128 **Conv3-128**	Conv3-128 Conv3-128
Max Pool	
Conv3-256 **Conv3-256** **Conv3-256**	Conv3-256 Conv3-256 Conv3-256 Conv3-256
Max Pool	
Conv3-512 **Conv3-512** **Conv3-512**	Conv3-512 Conv3-512 Conv3-512 Conv3-512
Max Pool	
Conv3-512 **Conv3-512** **Conv3-512**	Conv3-512 Conv3-512 Conv3-512 Conv3-512
Max Pool	
FC- 4096	
FC- 4096	
FC- 4096	
SoftMax	

Also in 2014, Szegedy *et al.* (Szegedy et al., 2015) introduced the GoogleNet, also known as Inception architecture. Szegedy highlighted that, at the time, the most straight forward way of improving the performance of the networks was increasing the depth of the framework. Although practical, two major drawbacks could not be ignored: increasing cost of the model, which required more computational resources for the training and fine tuning, and the increase in parameters to calculate, making it more prone to overfit with smaller/low quality datasets produced.

As stated, CNNs had a standard layout structure of stacking processing layers, varying in number and size. The authors theorized that a network architecture that made use of extra sparsity at filter level could be the next step to lessen the computation behind the network learning process. Thus, the Inception module was presented, with its configuration using convolution and pooling operations in a parallel way, concatenating them into the next level layer of the network, as seen in Figure 15.

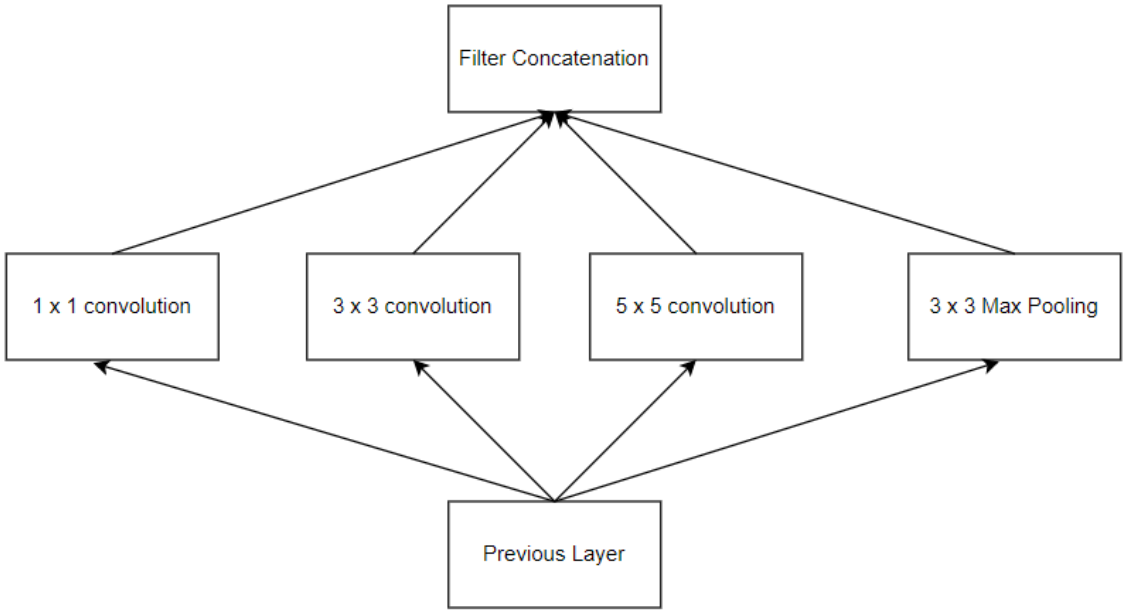


Figure 15 - Example of a 'naïve' Inception Module schematic performing 3 convolutions and max pooling with different spatial size filters.

Hence, the model has different size kernels finding various features at the same time, and increasing the width of the network, the number of units at each level. The main problem with these modules is that even modest number of 5 X 5 convolutions can be prohibitively expensive on top of the large number of filters already used, leading to a blow up of parameters.

To tackle that problem, Szegedy incorporated 1 X 1 convolutional layers, also known as 'Bottleneck Layers', firstly introduced in Network in Network by Lin *et al.* (M. Lin, Chen, & Yan, 2014). This simpler convolutional layer aims to reduce the dimension of its inputs. As an example, seen in

Figure 16, convoluting a 1 X 1 X 44 filter on a 24 X 32 x 190 input reduces the output depth of the output to only 44 channels without compromising the height and width. Those are used before the more complex convolutions, resulting in a less computational output compared to the 'naïve' first implementation, as well as increasing the non-linearity of its outputs, since the convolutions included ReLU functions which do not affect dimensions and avoid losing feature representation. The final Inception module can be visualized on Figure 17.

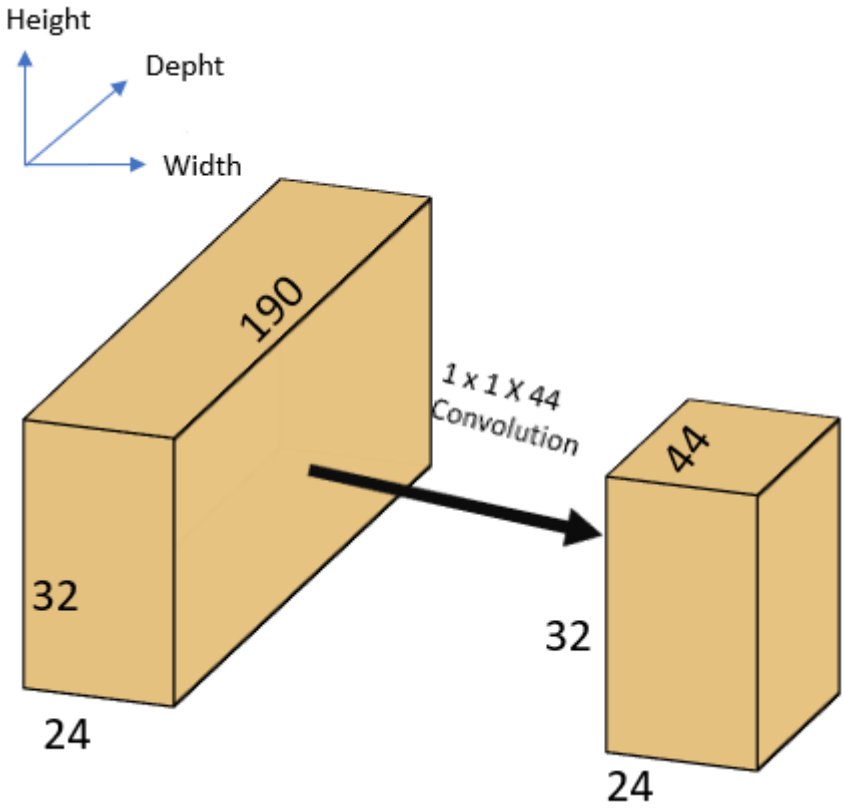


Figure 16 - Dimension Reduction using 1 X 1 convolutional layer.

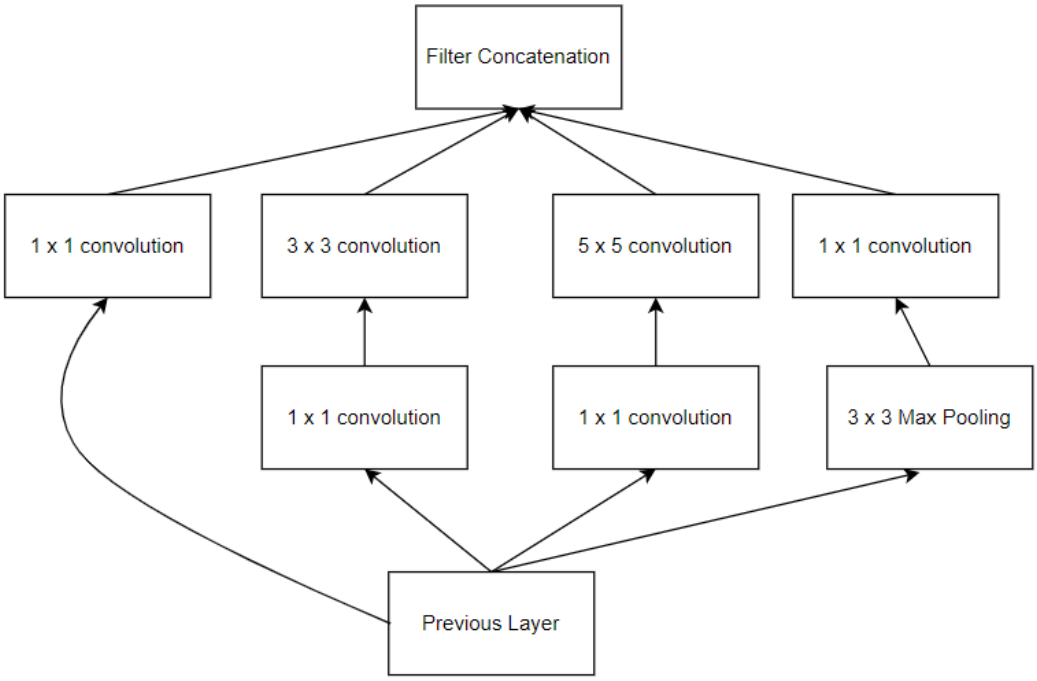


Figure 17 - Inception Module schematic with bottleneck layers for dimension reduction.

GoogleNet architecture is represented in Table 2. All the convolutions had ReLU activation functions. #3 X 3 reduce and #5 X 5 reduce showcase the number of 1 X 1 filters in dimension reduction before the 3 X 3 and 5 X 5 convolutions. The network was 22 layers deep and had 27 counting Pooling layers. It used auxiliary classifiers connected to intermediate layers, if considered that propagating gradients through all layers was a concern, given its large depth of the network.

Table 2 - GoogleNet architecture.

Layer Type	Patch size/ stride	Output size	Depth	# 1 X 1	#3 X 3 reduce	#3 X 3	#5 X 5 reduce	#5 X 5	Parameters	Operations
Convolution	7 X 7/ 2	112 X 112 X64	1						2,7 K	34 M
Max pool	3 X 3/ 2	56 X 56 X 64	0							
Convolution	3 X 3/ 1	56 X 56 X 192	2		64	192			112 K	360 M
Max pool	3 X 3/ 2	28 X 28 X 192	0							
Inception module		28 X 28 X 256	2	64	96	128	16	32	159 K	128 M
Inception module		28 X 28 X 480	2	128	128	192	32	96	380 K	304 M
Max pool	3 X 3/ 2	14 X 14 X 480	0							
Inception module		14 X 14 X 512	2	192	96	208	16	48	364 K	73 M
Inception module		14 X 14 X 512	2	160	112	224	24	64	437 K	88 M
Inception module		14 X 14 X 512	2	128	128	256	24	64	463 K	100 M
Inception module		14 x 14 x 528	2	112	144	288	32	64	580 K	119 M
Inception module		14 X 14 X 832	2	256	160	320	32	128	840 K	170 M
Max pool	3 X 3/ 2	7 X 7 X 832	0							
Inception module		7 X 7 X 832	2	256	160	320	32	128	1072 K	54 M
Inception module		7 X 7 X 1024	2	384	192	384	48	128	1388 K	71 M
Avg Pool	7 X 7/ 1	1 X 1 X 1024	0							
Dropout		1 X 1 X 1024	0							
Linear		1 X 1 X 1000	1						1000 K	1 M
SoftMax		1 X 1 X 1000	0							

This network was important because it resulted in a significant increase in performance, with its structural creativity at a modest increase of the computational cost compared to deeper networks. By reaching 1st place with a top-5 error-rate of 6.6% in classification on the 2014 ILSVRC, it outperformed the already reviewed VGG-Net in that regard, while also having 12 X fewer parameters (5 million) than AlexNet (60 million).

The following year, Szegedy *et al.* (Szegedy, Vanhoucke, Ioffe, Shlens, & Wojna, 2016) managed to improve its approach by introducing 2 new frameworks: Inception-v2 and Inception-v3.

Inception-v2, or Batch-Normalized Inception, targets what the authors call internal covariate shift - the phenomenon of changes in the distributions of internal nodes of a deep network due to changes in network parameters during training. That means that the NN must continuously adapt to input

distribution of the learning system in the distribution of layer's inputs, making the training slower, requiring low learning rates and careful parameter initialization. Addressing the problem, Batch-Normalization (Ioffe & Szegedy, 2015) was applied into the layer's inputs, allowing much higher lr without risk of divergence and less careful initialization. This network acted as a regularizer, having a beneficial effect on gradient flow through the network. Batch-normalization computes the mean and standard-deviation before inputting the next layer of the network layer, normalizing the response with these values. These practices achieved the same accuracy as the previous Inception Module with 14 less fewer training steps. Another improvement made was the replacement of larger kernels with 2 smaller ones, the same way as VGG-Net (Simonyan & Zisserman, 2014) on Inception modules (Figure 18), where the 5 X 5 convolutions layer was replaced by two 3 X 3 convolutions.

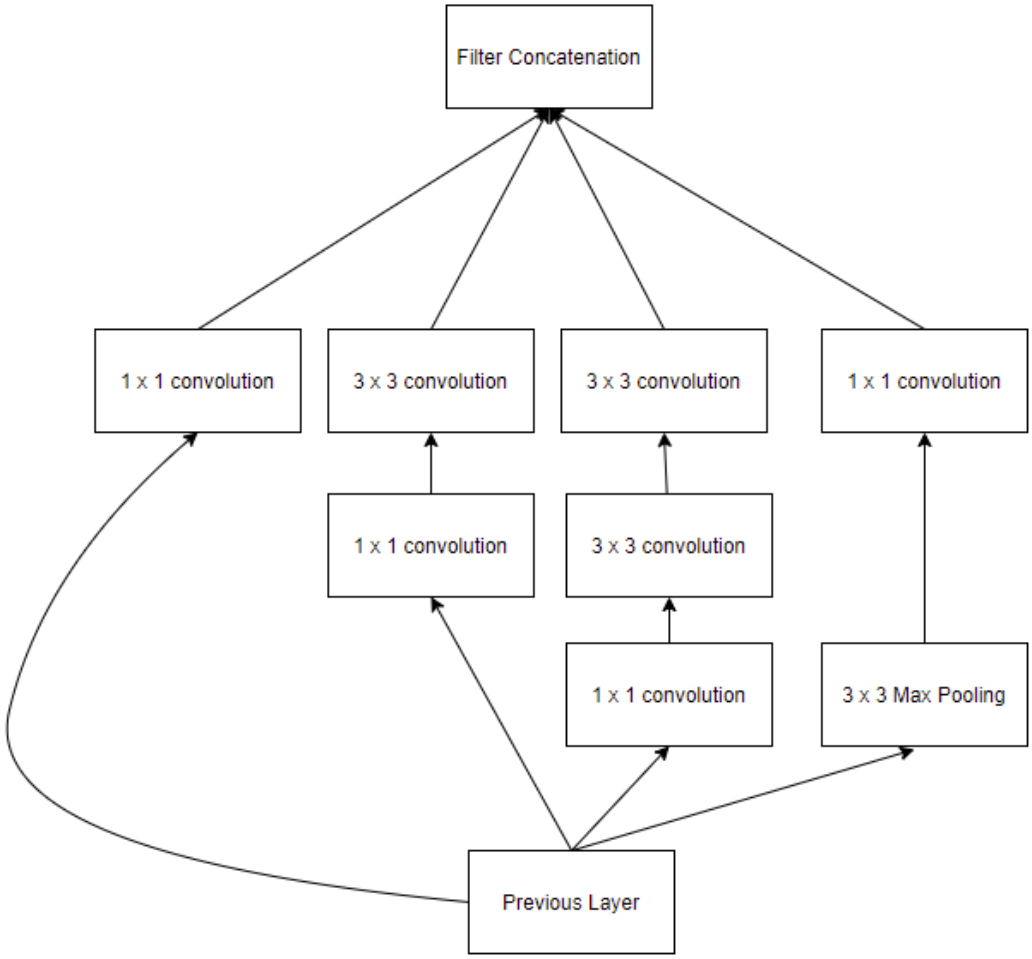


Figure 18 - Schematic Inception Module of Inception-v2, where 5 X 5 convolution layers were replaced by two 3 X 3 convolutions layers.

Inception-v3 continued with optimization processes for lowering the computational cost. That was accomplished with the application of spatial factorization into asymmetric convolutions, i.e., factorized n

Inception-v3 continued with optimization processes for lowering the computational cost. That was accomplished with the application of spatial factorization into asymmetric convolutions, i.e., factorized n X n convolutions into n X 1, followed by 1 X n. E.g., using 3 X 1 and 1 X 3 convolution has the same receptive field as the already known 3 X 3, using less parameters:

- 3 X 3 filter, number of parameters = 3x3 = 9;
- 3 X 1 and 1 X 3 filter, number of parameters = 3x1 + 1x3 = 6;

Resulting in a 33% reduction of parameters. Using the Inception module in Figure 17 as reference, its new factorized and more efficient counterpart can be seen on Figure 19.

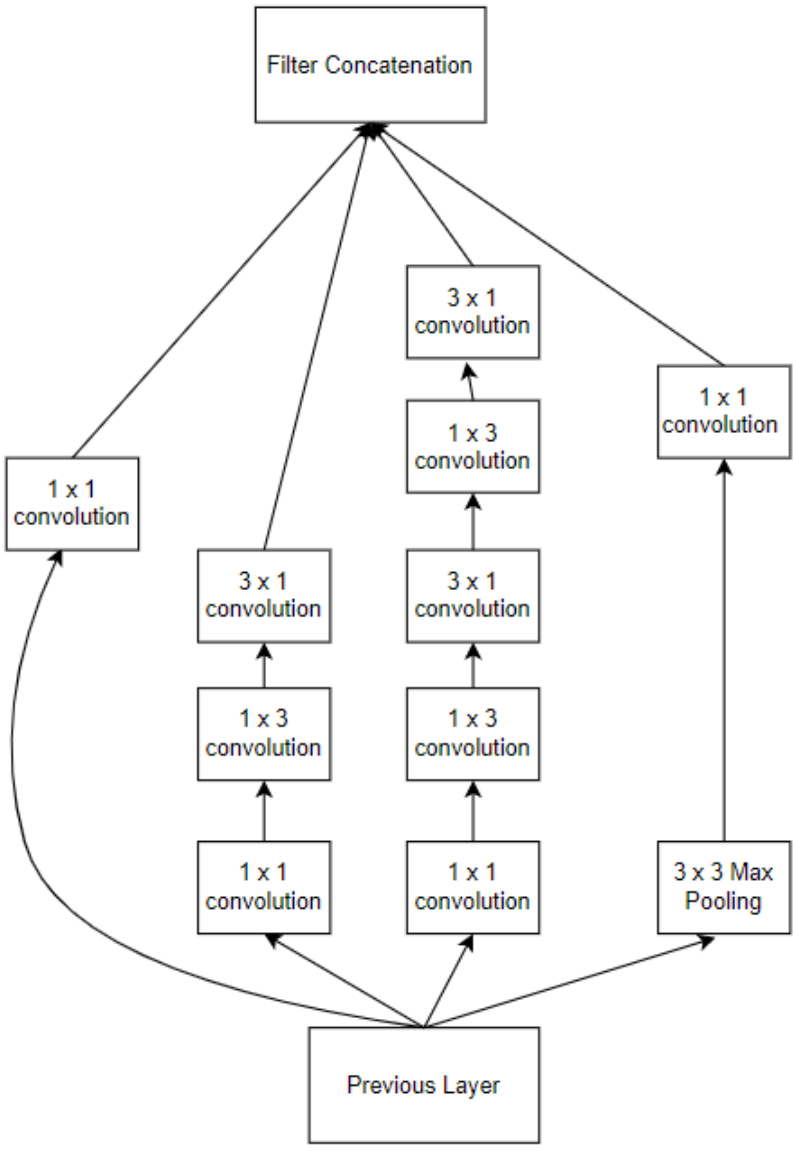


Figure 19 – Schematic of Inception Module of Inception-v3, using factorize filter and asymmetric convolutions.

This framework also proposed a different module with expanded filter bank outputs, believing that higher dimensions are easier to process locally within a network, where increasing activations allow for more disentangled features, which results in faster training. The module is represented in Figure 20. Another feature was the removal of the two auxiliar classifiers of the original architecture, where its loss was added to the final model's loss. Instead, only one was used, but as a regularizer, performing Batch-Normalization similar as Inception-v2. That results in 0.4% absolute gain in top-1 accuracy.

Other ideas presented were label smoothing as regularization, preventing disparities of output sizes and efficient grid size reduction, in order to avoid representational bottleneck.

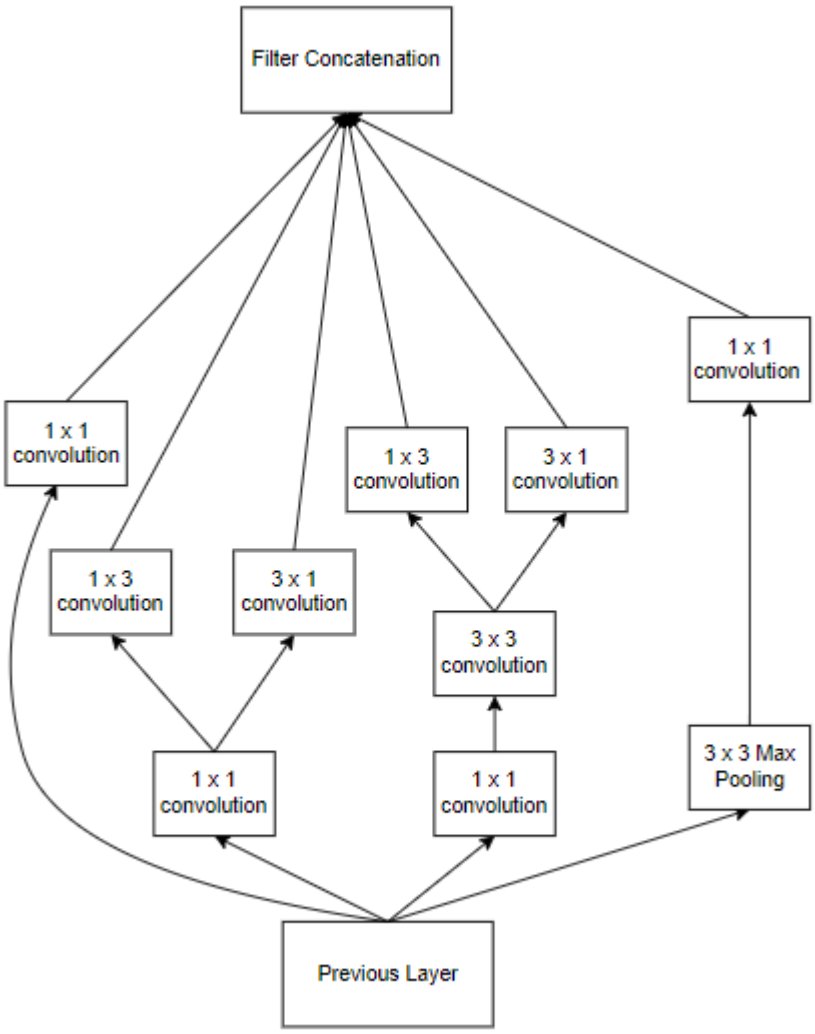


Figure 20 - Inception Module of Inception-v3 for high dimensional representations.

The final architecture had 42 layers deep, with its computational cost being only about 2.5 fold higher than GoogleNet (Szegedy et al., 2015). The outlined of the network is presented in Table 3, where grid size reduction was used between the Inception modules.

Table 3 - Inception-v3 Network Architecture.

Type	Patch size /stride	Input size
Convolutional	3 X 3 / 2	229 X 299 X 3
Convolutional	3 X 3 / 1	149 X 149 X 32
Convolutional (with 0-paddin)	3 X 3/ 1	147 X 147 X 32
Pooling	3 X 3 / 2	147 X 147 X 64
Convolutional	3 X 3 / 1	73 X 73 X 64
Convolutional	3 X 3 / 2	71 X 71 X 80
Convolutional	3 X 3 / 1	35 X 35 X 192
3 x Inception Module	As seen in Figure 18	35 X 35 X 288
5 x Inception Module	As seen in Figure 19	17 X 17 X 768
2 x Inception Module	As seen in Figure 20	8 X 8 X 1280
Pooling	8 X 8	8 X 8 X 2048
Linear	Logits	1 X 1 X 2048
Softmax	Classifier	1 X 1 X 1000

Inception-v3 was the runner up of ILSVRC in 2015 (*ImageNet Large Scale Visual Recognition Competition 2015 (ILSVRC2015)." [Online]. Available: image-net.org/challenges/LSVRC/2015/*, n.d.), reaching 21.2% top-1 and 5.6% top-5 error for single frame evaluation. With a 4-model ensemble and multi-crop evaluation, the accuracy was reported to be 3.5% top-5 and 17.3% top-1 error.

Developed in 2015 by He *et al.* (K. He, Zhang, Ren, & Sun, 2016), ResNet focused on deepening networks without affecting its performance, given that deeper NN are more difficult to train given the vanishing gradient problem. ResNet solved this problem by introducing residual blocks into the architecture, explicitly reformulating the layers as learning residual functions with reference to the layer

inputs, instead of learning unreferenced functions. Those were easier to optimize, gaining accuracy with considerably increased depth.

The vanishing gradient problem (Glorot & Bengio, 2010) refers to the saturation of the accuracy of the network when adding more layers, degrading rapidly. (Glorot & Bengio, 2010) verified that a 20-layer plain network got lower training error and test error than 56-layer plain network. That experimentation refuted the premise that the same network has better accuracy the deeper it gets. Such degradation was not associated to overfitting, but to backpropagation issues, computed from the back of the network to the front. When training a deep model, the weight of each layer receives an update proportional to the partial derivative of the error function but in some cases, the gradients disappear when vanishingly small multipliers are computed. This effectively prevents the weight from changing its value, proving that not all systems are similarly easy to optimize by stacking more layers. The proposed solution for the degradation problem was the introduction of a deep residual learning framework, presented below in Figure 21.

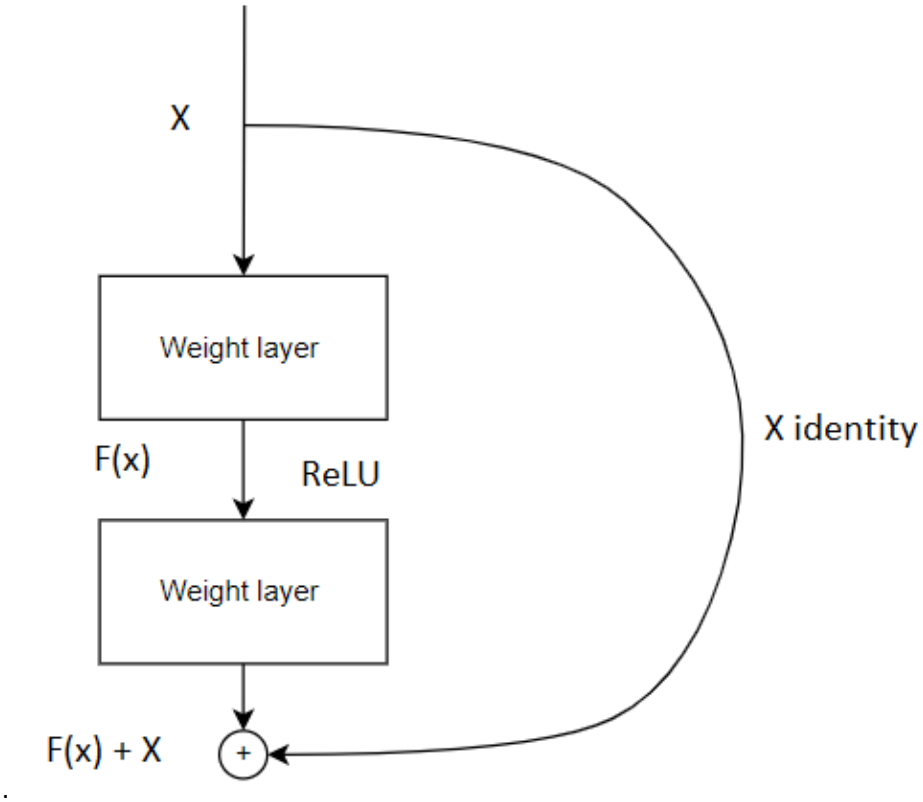


Figure 21 - Residual Block of ResNet model.

By attaching skip/shortcut connections and performing identity mapping, the input x was added to the output after a few weight layers. Formally, denoting the output as $H(x)$, the new the underlying

mapping summed with the identity was $H(x) = F(x) + x$. That meant weight layers fitted a residual mapping of $F(x) = H(x) - x$, being easier to optimize residual mappings than to optimize the original, unreferenced mappings. Identity shortcut connections did not add any extra parameter nor computational complexity. With that, He *et al.* (K. He, Zhang, Ren, & Sun, 2016) proved, on the ImageNet dataset, that:

- Extremely deep residual nets with skip connections were easier to optimize in comparison with its plain network counterparts (that simply stack layers), with the former exhibiting higher training error when the depth increases.
- Deep residual nets could easily enjoy accuracy gains from greatly increased depth, producing results substantially better than previous networks.

Four main architectures were proposed with different depths: ResNet-34, ResNet-50, ResNet-101 and ResNet-152, with 34, 50, 101 and 102 being the number of layers used in the architecture, respectively. The baseline models were stacked 3 X 3 convolutions, inspired by VGG-Net (Simonyan & Zisserman, 2014), using shortcut connections every 2 pairs, as seen in Figure 22.(a). The deeper models replaced those 2-layer building blocks with a 3 layer bottleneck block, with the same premise use in Inception architectures (Szegedy et al., 2015), in order to lower the number of parameters, and reducing the complexity whilst deepening the layout, as seen in Figure 22.(b).

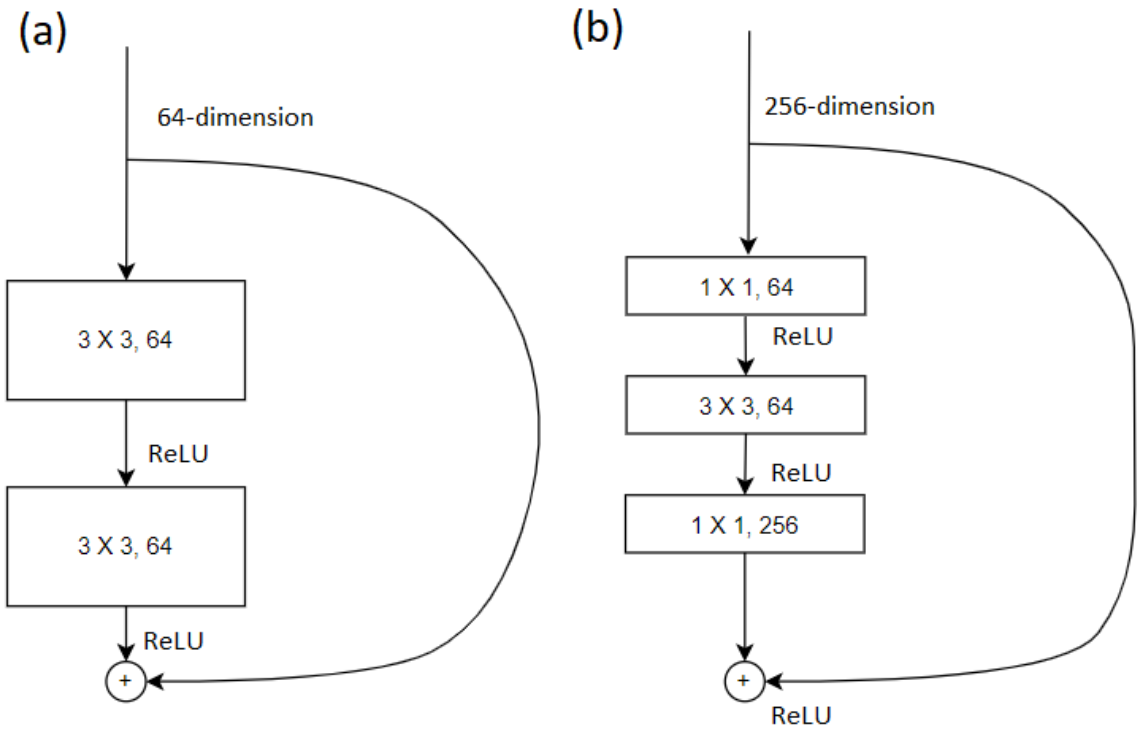


Figure 22 - Residual building blocks schematic used on ResNet architectures (a) Residual building blocks used on ResNet–34; (b) Bottleneck building block used on ResNet-50/101/152.

As notable results, a plain 34-layer and 18-layer networks were trained on the ImageNet dataset, the same way as ResNet architectures with the same number of layers. The top-1 error on ImageNet validation dataset is presented in Table 4.

Table 4 - Top-1 error % on ImageNet validation comparison between plain and ResNet architectures.

	Plain Network	ResNet Network
18-Layer error %	27.94	27.88
34-Layer error %	28.54	25.03

The error percentage was higher on deeper networks on plain layouts, presenting the gradient descent problem. With ResNet networks, the error decreased, solving the problem with shortcut connections. Compared with the state-of-the-art at the time, ResNet won ILSVRC in 2015 (*ImageNet Large Scale Visual Recognition Competition 2015 (ILSVRC2015)." [Online]. Available: image-net.org/challenges/LSVRC/2015/*, n.d.), with some results Table 5. This architecture was a breakthrough to deep learning algorithms, proving that residual blocks optimized the deeper network training in comparison to plain counterparts, where better performances could be achieved with the same number of parameters with only skipping connections between some layers.

Table 5 - Results of ResNet Architectures.

	VGG-16	GoogleNet	ResNet-34	ResNet-50	ResNet-101	ResNet-152
top-1 error % on ImageNet validation	28.07%	-	24.19%	22.85%	21.75%	21.43%
top-5 error % on ImageNet validation	9.33%	9.15%	7.40%	6.71%	6.05%	5.71%
top-1 error % on single model ImageNet validation	8.43%	7.89%	5.60%	5.25%	4.60%	4.49%
top-5 error % on single model ImageNet validation	24.4%	21.99%	21.53%	20.74%	19.87%	19.38%

2.5 Face Detection and Facial Recognition

Face detection and facial recognition are active computer vision fields of study researched for over twenty years. Being a powerful biometric for a user's identity authentication, its widely use covers many different areas such as law enforcement (Kalal, Mikolajczyk, & Matas, 2010), airport and boarder

control or marketing (Masi, Wu, Hassner, & Natarajan, 2019). As many computers vision problems, the research paradigm shifted with the blow up of Deep Learning. Convolutional Neural Networks and the availability of extensive training datasets in unconstrained capture conditions allowed for quicker and more robust systems regarding previously implemented methods.

Face detection can be regarded as an object-class detection problem, where its algorithms check for facial landmarks in order to identify faces in images or videos, isolating the proposals via bounding boxes. This method is the cornerstone for several facial applications and analysis, such as facial recognition, being the initial step to modern vision-based human/robot interactions. According to Zafeiriou *et al.* (Zafeiriou, Zhang, & Zhang, 2015), the first efforts were mainly based on hand-crafted features extracted from images of upright frontal faces, fed into classifiers to detect likely face regions. The first landmark was the Viola & James framework (Viola & Jones, 2001) in 2001, providing a real-time facial detector using Haar Cascade classifiers to detect promising face-like regions, fed into an Adaboost learning algorithm to select critical features from a very large set of potential options. This method explored cascade chain to reject false face regions. The work had high accuracy with great detection speed, with its principles still being used into new algorithms. This led to the widespread adoption of such scale-invariant face detection frameworks. Another popular approached was the use of regional statistics like histograms. Histograms of Oriented Gradients (HOG) (Dalal & Triggs, 2005) reviewed existing edge and gradient based descriptors, with the classification being performed with an SVM classifier, significantly outperforming existing features sets for human detection at the time. However, those methods proved to still be limited on multiple variations of faces like scale, pose or illumination.

Facial recognition is a multi-label classification problem, focusing on the analysis of faces in order to predict a specific identity. A survey (M. Wang & Deng, 2021) highlighted three big milestones before Deep learning techniques, presented in Figure 23.

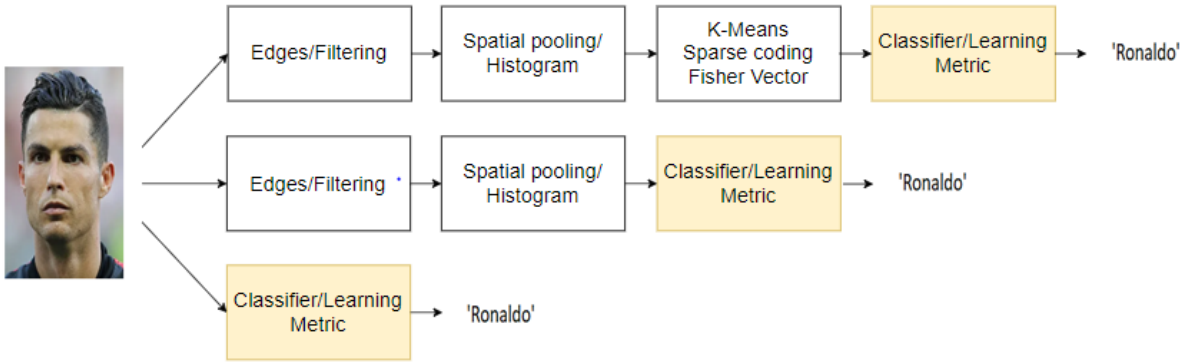


Figure 23 - Milestone of face representation for facial recognition.

These holistic approaches derive from dimensional reduction algorithms as baseline. These dominated the face recognition community in the 1990's and 2000's through certain distribution assumption, such as linear subspace, manifold, or sparse representation. They were popularized by the Eigenface method (Turk & Pentland, 1991), where eigenvectors were calculated using principal components analysis (PCA) and then compared to known individuals. This framework proved to be efficient but has low accuracy with different scales of images, orientation invariance of the head, and lighting variation problems. Fisher Discriminant Analysis (Belhumeur, Hespanha, & Kriegman, 1997) performed better on illumination variance and expression in a linear subspace approach, while Laplacianfaces (X. He, Yan, Hu, Niyogi, & Zhang, 2005) used locality preserving projections (LPP), providing lower error rates. However, these methods are still very prone to misclassification when dealing with age and pose variation, as well as facial changes.

In 2000's, facial recognition converged to local feature-based filtering methods such spatial pooling or histograms. With these methods, Gabor-Fisher classifiers (C. Liu & Wechsler, 2002) and local binary patterns (Ahonen, Hadid, & Pietikäinen, 2006) techniques and its extensions achieved robust performances through some invariant properties of local filtering. Although handcrafted features were an important breakthrough, they suffered from a lack of distinctiveness and compactness.

In the early 2010s, facial recognition used shadow networks with learning-based local descriptors (Low, n.d.) (Chan et al., n.d.), where local filters are learned for better distinctiveness, and the subject's faces are compacted into encoders. With good results, these shallow techniques still had limitations regarding robustness and complex non-linear facial variations.

In general, non-deep learning methods used one or two processing layers in order to recognize human faces, focusing on filtering responses, histograms, or local-based distributions. Those methods almost only focused on one aspect of unconstrained facial changes, like lighting, pose, expression, while not approaching these integrally. Researchers studied intensively to improved said results, only to improve the accuracy slowly. Hence, that technical insufficiency made facial recognition frameworks unstable, with its real-world systems having untrustworthy performances with various false positives.

2.6 Related Work

Deep Learning based facial detection and recognition pipelines reached state-of-the-art performances compared to traditional methods, having more processing layers which incorporated previous filtering techniques. A key contribution was the assembly of large datasets of annotated unconstrained face data in order to train said algorithms.

CNNs have been successfully applied for face detection and location. The main benchmarks for this field of study are the Wider Face (Yang, Luo, Loy, & Tang, 2016), with 32.203 images and 393.703 labeled faces with a high degree of variability in scale, pose and occlusion, resulting in a very challenging dataset. For each event class, it randomly selected 40%, 10%, 50% data as training, validation and testing sets, accordingly. Other benchmarks include FDDB dataset, PASCAL Face (Yan, Zhang, Lei, & Li, n.d.), MALF, UFDD (Nada, Sindagi, Zhang, & Patel, n.d.) or VGGFace2 (Cao, Shen, Xie, Parkhi, & Zisserman, 2018).

According to Minaee *et al.* (Minaee, Luo, Lin, & Bowyer, 2021), the four main techniques used for facial detection were:

1. Cascade-CNN based models;
2. R-CNN based models;
3. Single shot detector models;
4. Feature Pyramid Detector based models.

Cascade-CNN based models grouped several architectures into one, where the output of the previous convolutional model is the input of the next. Those proved to be powerful multi-tasking frameworks, accurately performing facial regression as well as keypoint estimation, and managing to align the detected face proposals. Li *et al.* (Li, Lin, Shen, Brandt, & Hua, n.d.) proposed a 12-layer cascaded operating at different resolutions, quickly rejecting background regions on the early low-resolution stages, and having calibration methods after every detection phase. In the same order, Zhang *et al.* (K. Zhang, Zhang, Li, & Qiao, 2016) proposed a 3-phase cascaded architecture for joint face detection and alignment trained on the WIDER FACE dataset (Yang et al., 2016), where that training data presented large amounts of significant pose variation, lighting or expression.

Region proposal-based CNN algorithms used a two-step approach. First identified regions where objects are expected. Then, these algorithms used region proposal networks, extracting regions of interest in order to propose bounding box coordinates for a desired class. By being very successful for object detection, those algorithms were adapted to face detection by several works. Chen *et. al.* (Chen, Hua, Wen, & Sun, n.d.) used a multi-task Region proposal network, where it predicted the candidate faces along with its facial landmarks. Those regions were then normalized and fed into a Region-CNN where it validated if the regions are valid faces or not. Wang *et al.* (Y. Wang, Ji, Zhou, Wang, & Li, 2017) presented a ResNet convolutional network with as feature extractor, resulting in a batch of the RoI. Those were then further fed into two sibling positions sensitive RoI pooling layer to produce class score

maps and bounding box prediction map. That way with multi-scale training maps to a certain location of the output score maps.

Single shot detection models proposed bounding boxes over different aspect ratios and scales, generating scores accordingly to the certainty of the presence of a face. Additionally, the network combined predictions from multiple feature maps with different resolutions to naturally handle samples of various sizes, instead of rescaling the original image like the cascaded approaches. That way, single shot detects faces in a single stage directly. Najibi *et al.* (Najibi, Samangouei, Chellappa, & Davis, n.d.) achieved state of the art results by using a scale-invariant design in a single forward pass.

Feature Pyramid Network Based models (T.-Y. Lin et al., n.d.) combined semantically weak features with meaningly strong features based on skip-connections. Zhang *et al.* (J. Zhang et al., n.d.) proposed a feature agglomeration network, where inherent multi-scale features were exploited by aggregating semantic features maps with different scales to augment lower-level feature maps via a hierarchical agglomeration manner. Deng *et al.* (Deng et al., n.d.) introduced RetinaFace, which performed a pixel-wise face localization on various scales in a single-stage. This algorithm took advantage of joint extra-supervised and self-supervised multi-task learning. The authors manually annotated five facial landmarks on WIDER FACE dataset, observing significant improvement in hard face detection. It achieved state of the art performance in several benchmarks.

Facial recognition, being a multi-class classification, benefited greatly from deep learning techniques, having typical CNN architectures as structural baseline. Wang and Deng (M. Wang & Deng, 2021) categorized various novel frameworks by their lost functions, being grouped by Euclidean-distance-based loss, angular/cosine-margin-based loss and SoftMax Loss and its variations.

SoftMax Loss is a logistic function used for probability distribution. In multi-class classifications, SoftMax computes the probability vector of all the output classes. DeepFace (Taigman, Yang, Ranzato, & Wolf, 2014) represented the first breakthrough in facial recognition using neural networks, where it achieved state-of-the-art accuracy (97.35%) on LFW benchmark (Huang et al., 2008) in 2014. Using the SoftMax loss function and the AlexNet (Krizhevsky et al., 2017) model trained using four million facial images, approaching human performance (97.53%). After this breakthrough, various proposals achieved state-of-the-art performances regarding classification. However, SoftMax had the drawback that intra-variations could prevail over inter-variations, making the loss function insufficient for generalization of the facial recognition pipeline.

Euclidean-distance-based loss method maps facial features extracted from images into embedding arrays into a Euclidean space, that maximized the inter-variance (different classes/between group) distance and minimized the intra-variance (same class/within group).

Contrastive loss simultaneously compressed the Euclidean distance of image pairs of the same individual, while enlarging the distance between different identity pairs. DeepID2 (Sun, Wang, & Tang, 2015) combined the SoftMax algorithm for classification with contrastive loss for verification, reaching 99.15% accuracy on the LFW benchmark. Later, DeepID2+ (Sun, Chen, Wang, & Tang, 2014) added supervision to early stages of the CNN, as well as increasing the dimension of hidden representations. DeepID3 (Sun, Liang, Wang, & Tang, 2015) introduced the VggNet and GoogleNet architectures into the previous algorithms. Although powerful, the contrastive loss problems reside on the margin parameter used on the positive and negative pairs being difficult to choose in order to optimize accuracy. The main ideas were the Contrastive loss and the Triplet loss, providing great performances in facial classification functions.

Triplet loss algorithm used face triplets, anchor, positive sample, and negative sample. Anchor and positive sample are the same class identities whereas the negative sample is a different class sample. The formula tried to minimize the relative distance between the anchor and the positive while maximizing the anchor/negative distance. This concept was introduced by FaceNet (Schroff, Kalenichenko, & Philbin, 2015), which directly optimized the embedding itself, rather than using an intermediate bottleneck. Using the Inception architecture, Szegedy *et al.* (Szegedy et al., 2015) provided the capability to concatenate different filter sizes in the same processing layer. VGGface (Parkhi, Vedaldi, & Zisserman, 2015) built a large-scale dataset, training a VGG network pipeline and fine-tunning it via triplet loss. Triplet loss approaches can be difficult to optimize given the training instability induced by the selection of triplets, making it hard to choose effective samples for various classes.

In other Euclidean loss metrics proposed, Center loss (Wen, Zhang, Li, & Qiao, n.d.) proposed a center-invariant loss, calculating the 'centers' of each class, penalizing the distance between the features and their corresponding class centers. By reducing that distance of each data sample, it compresses the inter-variance. Combining it with a SoftMax loss, Center loss prevents embeddings from collapsing. Range loss (X. Zhang, Fang, Wen, Li, & Qiao, 2017) minimized the greatest ranges of harmonic means in one class and, maximizing the shortest inter-class distance within one batch.

Angular/Cosine-margin based loss studied angular similarity of discriminative face features, generating angular/cosine separability between the learned characteristics. Compared to the Euclidean-distance based loss, this method added constraints in a hypersphere manifold, represented in Figure 24.

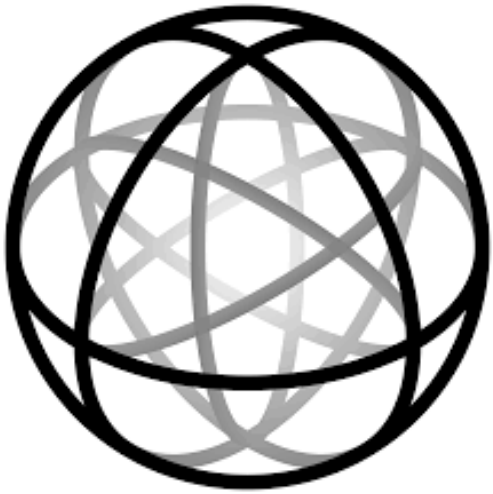


Figure 24 - Hypersphere manifold design.

L-SoftMax (W. Liu, Wen, Yu, & Yang, 2016) combined the SoftMax function with a large angular margin in a multiplicative manner. Based on that work, A-SoftMax normalized the extracted feature vectors with a L2. normalizer, where faces are represented into a hypersphere space manifold, learning by angular margin. ArcFace (Deng, Guo, Xue, & Zafeiriou, 2018) and CosineFace (Feng Wang, Cheng, Liu, & Liu, 2018) overcame the optimization problems by presenting additive angular/cosine margins. Those were easier to implement without the need of complex hyperparameters and can be converged without SoftMax. Angular/Cosine-margin based loss approaches tend to achieve better performance, although, according to Wang (Fei Wang, n.d.), these loss approaches are very vulnerable to noisy or unclean datasets.

The biggest prerequisite for an effective deep learning face recognition pipeline is a sufficiently large dataset. Early works of facial recognition trained with private datasets. Naming a few, Facebook's DeepFace (Taigman et al., 2014) trained on 4 million images of 4000 people. Google's FaceNet (Schroff et al., 2015) trained on 200 million images of 3 million classes. DeepID methods trained on 200.000 images of 10.000 people. Hence, performance results could not be replied given the lack of data necessary. Nowadays, public datasets were created, mainly by public institutions. CASIA-Webface (Yi, Lei, Liao, & Li, n.d.) provided the first public training dataset, containing 500.000 images of 10.000 celebrities collected online. Recently, larger datasets were created and made available publicly, such as the MS-Celeb-1M (Guo, Zhang, Hu, He, & Gao, n.d.), VGGface2 (Cao, Shen, Xie, Parkhi, & Zisserman, n.d.) or Megaface (Kemelmacher-Shlizerman, Seitz, Miller, & Brossard, n.d.).Table 6 presents some of the most used datasets to achieve a robust facial recognition.

Table 6 - Facial Recognition datasets used for training

Datasets	Publish Time	Images	Subjects
CASIA WebFace	2014	494.414	10.575
Facebook (private)	2014	4.4 million	40.000
CelebFaces+ (private)	2014	202,599	10.177
Google (private)	2015	>500 million	>10 million
VGGface	2015	2.6 million	2.622
VGGface2	2017	3.31 million	9.131
MegaFaces	2016	4.7 million	672.057
MS-Celeb-1M	2018	4 million	80.000

In order to compare the state-of-the-art facial recognition methods, Wang and Deng (M. Wang & Deng, 2021) compared the performance accuracy results on the LFW benchmark (Huang et al., 2008), which is presented in Table 7:

Table 7 - LFW benchmark results of deep learning facial recognition approaches.

Method	Public. Time	Loss	Number of Networks	Architecture	Training Set	Accuracy +/- Std (%)
DeepFace (Taigman et al., 2014)	2014	SoftMax	3	Alexnet	Facebook (4.4M,4K)	97.35 +/- 0.25
DeepID2 (Sun, Wang, et al., 2015)	2014	contrastive loss	25	Alexnet	CelebFaces+ (0.2M,10K)	99.15 +/- 0.13
DeepID3 (Sun, Liang, et al., 2015)	2015	contrastive loss	50	VGGNet-10	CelebFaces+ (0.2M,10K)	99.53 +/- 0.10
FaceNet (Schroff et al., 2015)	2015	triplet loss	1	GoogleNet-24	Google (500M,10M)	99.63 +/ -0.09
Baidu (J. Liu, Deng, Bai, Wei, & Huang, 2015)	2015	triplet loss	10	CNN-9	Baidu (1.2M,18K)	99.77
VGGface(Parkhi et al., 2015)	2015	triplet loss	1	VGGNet-16	VGGface (2.6M,2.6K)	98.95
light-CNN (Wu, He, Sun, & Tan, 2018)	2015	softmax	1	light CNN	MS-Celeb-1M (8.4M,100K)	98.8
Center Loss(Wen et al., n.d.)	2016	center loss	1	Lenet+-7	CASIA-Webface, CACD2000, Celebrity+ (0.7M,17K)	99.28
L-SoftMax (W. Liu et al., 2016)	2016	L-softmax	1	VGGNet-18	CASIA-Webface (0.49M.10K)	98.71
Range Loss (X. Zhang et al., 2017)	2016	range loss	1	VGGNet-16	MS-Celeb-1M, CASIAWebFace (5M,100K)	99.52
L2-softmax (Ranjan, Castillo, & Chellappa, 2017)	2017	L2-softmax	1	ResNet-101	MS-Celeb-1M (3.7M,58K)	99.78
Normface (Feng Wang, Xiang, Cheng, & Yuille, 2017)	2017	contrastive loss	1	ResNet-28	CASIA-Webface (0.49M.10K)	99.19
CO.oco loss (Y. Liu, Li, & Wang, 2017)	2017	CoCo loss	1	-	MS-Celeb-1M (3M,80K)	99.86
vMF loss (Hasnat, Bohné, Milgram, Gentric, & Chen, 2017)	2017	vMF loss	1	ResNet-27	MS-Celeb-1M (4.6M,60K)	99.58
Marginal Loss(Deng, Zhou, & Zafeiriou, 2017)	2017	marginal loss	1	ResNet-27	MS-Celeb-1M (4M,80K)	99.48
SphereFace(W. Liu et al., 2017)	2017	A-softmax	1	ResNet-64	CASIA-Webface (0.49M.10K)	99.42
CCL(Qi & Zhang, 2018)	2018	center invariant loss	1	ResNet-27	CASIA-Webface (0.49M.10K)	99.12
AMS loss(Feng Wang et al., 2018)	2018	AMS loss	1	ResNet-20	CASIA-Webface (0.49M.10K)	99.12
Cosface(H. Wang et al., 2018)	2018	cosface	1	ResNet-64	CASIA-Webface (0.49M.10K)	99.33
ArcFace (Deng et al., 2018)	2018	arcface	1	ResNet-100	MS-Celeb-1M (3.8M,85K)	99.83
Ring los(Zheng, Pal, & Savvides, 2018)	2018	Ring loss	1	ResNet-64	MS-Celeb-1M (3.5M,31K)	99.50

3. METHODS AND METHODOLOGIES

In this chapter, the system's core methodologies are presented. First, the tools and specifications are overviewed. After, the chapter describes the implementation of the end-to-end framework, displaying the schematic structure of the work, with every part being analyzed with the methods implemented and the respective theoretical fundaments that corroborate them.

3.1 System Tools and Specifications

The developed work was implemented on an Acer Predator Helios 3000. This computer contains 6 core Intel(R) Core (TM), an i7-8750H CPU processor @ 2.20GHz 2.21 GHz, and a NVIDIA GeForce GTX 1060 GPU. The operative system used was Ubuntu 18.0, given its accessibility to run data files of different sources and libraries, as well as easy and supported package installation.

The software's core code language used was Python3, since all the main libraries used are supported in that language. The code scripts were built via Notepad++ and executed via Ubuntu's command line.

All the data used was locally stored, i. e., allocated on the device, including images, checkpoints of the trained models, and data visualization graphics. To access those directories and files paths, the OS module was used. It provides a portable way of using operating system dependent functionality. ArgParse module made possible to write command-line interfaces into the code as parameter definitions.

The computer vision tasks, such as real time, video, or image processing and acquisition, were carried out via OpenCV library. Real time tasks were carried out with the built-in computer camera.

For matrix and array manipulation, the main library used was NumPy. It provides a multidimensional array objects, various derived objects (such as masked arrays and matrices), and an assortment of routines for fast operations on arrays, including mathematical, logical, shape manipulation. This library has an API for the main library used, TensorFlow, making it easy to communicate between modules.

For ML support, Scikit-learn was used. It is an open-source ML library that supports supervised and unsupervised learning. It also provides various tools for model fitting, data preprocessing, model selection and evaluation, and many other utilities.

The main developing library of this work was TensorFlow (Abadi et al., 2016), which is a ML system that operates at large scale and in heterogeneous environments. It uses dataflow graphs to represent all computation and state in ML algorithms, including the individual mathematical operations, the

parameters and their update rules, and the input preprocessing, focusing on training and inference (the testing part after training) of deep neural networks. The data is modeled as tensors, multi-dimensional arrays (Figure 25).

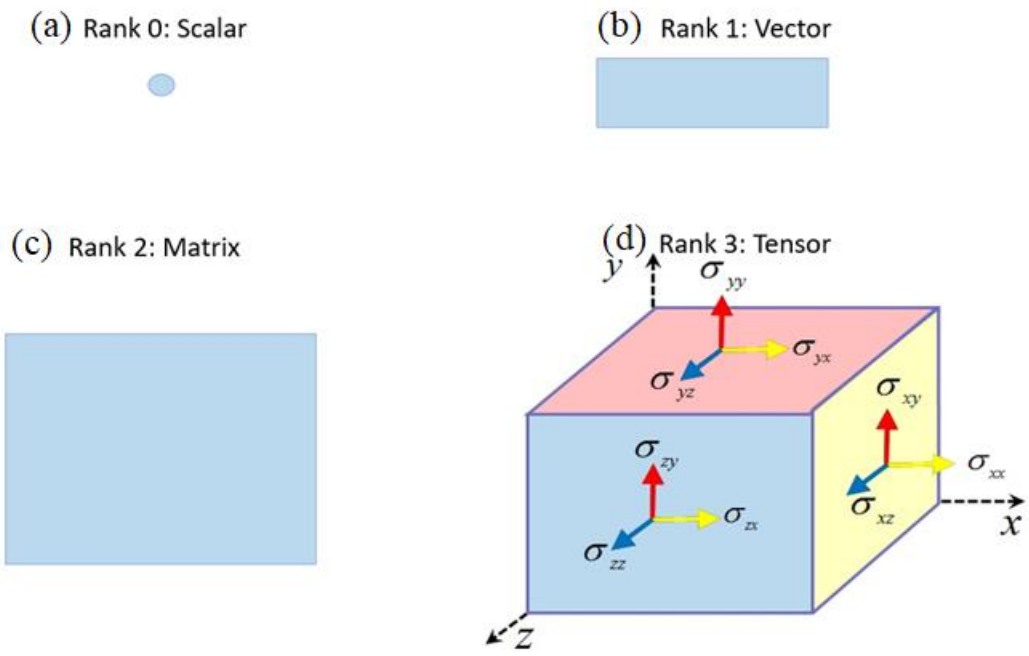


Figure 25 - Spatial representation of a Tensor multi-dimensional arrays; (a) - 0-D tensor- scalar; (b)- 1-D tensor- vector; (c) - 2-D tensor - matrix, (d) – 3-D tensor.

Since building and training neural network models was very computationally heavy, CUDA Toolkit API was installed and used with the TensorFlow framework (Figure 26). It is a software layer designed to give direct access to NVIDIA GPUs in order to promote parallel processing between cores while executing computer kernels. That way, using CUDA enabled TensorFlow, developers speed up computational-intensive applications by harnessing the power of 3D acceleration of the graphics card. To maximize performance, CUDA's library cuDNN was installed, a GPU-accelerated library of primitives for deep neural networks. It allows the framework to focus on training neural networks and developing software applications rather than spending time on low-level GPU performance tuning.

```
Successfully opened dynamic library libcudart.so.10.0
Successfully opened dynamic library libcublas.so.10.0
Successfully opened dynamic library libcufft.so.10.0
Successfully opened dynamic library libcurand.so.10.0
Successfully opened dynamic library libcusolver.so.10.0
Successfully opened dynamic library libcusparse.so.10.0
Successfully opened dynamic library libcudnn.so.7
```

Figure 26 - TensorFlow enabling the CUDA library for GPU accelerated processes.

Lastly, TensorBoard, TensorFlow's visualization toolkit, provided the visualization and tooling needed for ML experimentation. It allowed tracking and visualizing metrics, such as loss and accuracy, visualizing the model graph, viewing histograms of weights, biases, projecting embeddings to a lower dimensional space and displaying of dataset images. Table 8 presents all versions of the mentioned tools used. Some versions had to be downgraded for compatibility issues.

Table 8 - System tools and versions.

Tool	Version
Python3	3.6.9
OpenCV	4.4.0
OS	5.4.0-77-generic
NumPy	1.19.0
Scikit-learn	0.23.1
TensorFlow	1.15.0
CUDA Toolkit	10.0.130
cuDNN	7.6.5
TensorBoarD	1.15.0

3.2 Model Framework

As mentioned in Objectives of the Dissertation, the developed system was a real-time multi-stage deep learning pipeline for facial recognition. It analyzed frames from the camera, where it detected and aligned faces, searching for facial landmarks, such as eyes, nose, or mouth, predicting its bounding box coordinates, cropping the detected faces containing the least background possible, and drawing the bounding boxes on the proposals' original frame. Next, an array of the detected bounding boxes are encoded into another neural network, previously trained for that purpose, that extracted the features, creating a vector representing the faces, also known as embedding. With that, the classifier predicted the identity of all the elements of the embedding array, writing the name of the predicted person, as well as the certainty percentage. The proposed framework can be seen in Figure 27.

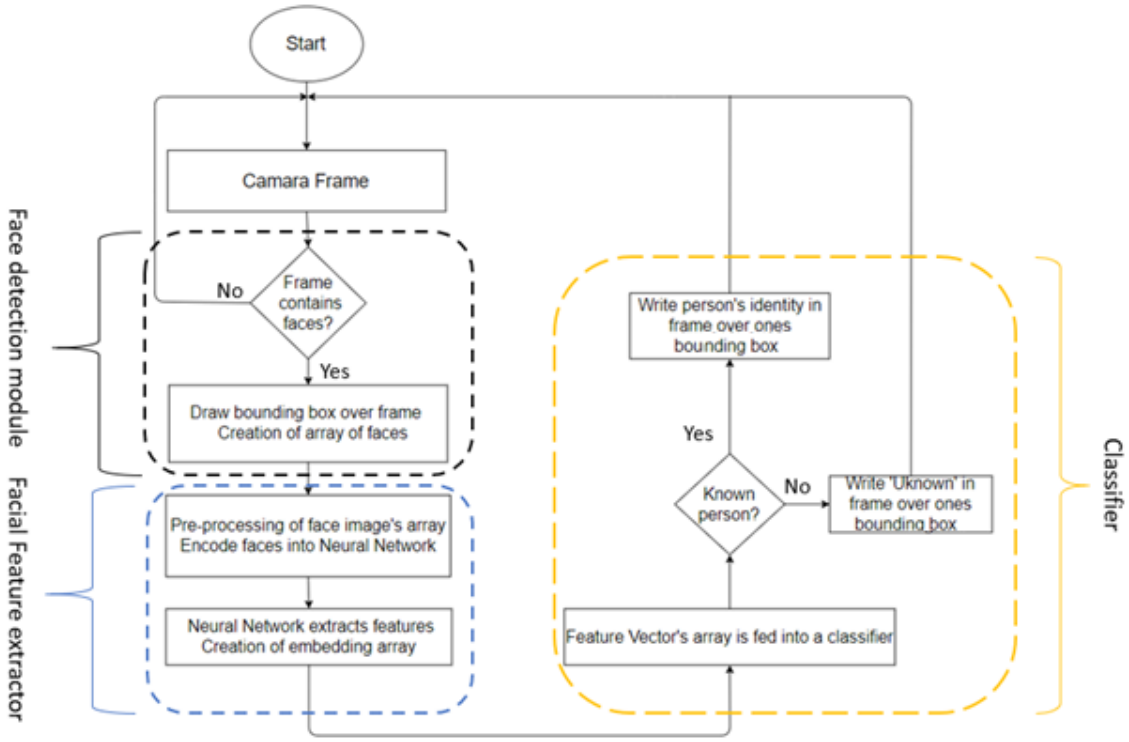


Figure 27 - Real time multi-stage deep learning pipeline for facial recognition.

The system is composed of three main modules: Face detection, Facial Feature Extractor, and a Classifier. Initially, to perform face detection and alignment, MTCNN method (K. Zhang et al., 2016), previously mentioned in Related Work, was used. The motive for this choice was that cascaded networks proved to be powerful multi-tasking models, accurately performing face regression as well as keypoint estimation, aligning the detected face proposals. It also proved to be a very robust system for real world situations, detecting faces in non-standardized situations. The ending results of the first module was an image array of crops of isolated faces in a camera frame. The second module was a feature extractor, mapping the detected faces into an n-dimensional array representing features extracted. The deep learning CNN used was the Inception-ResNet architecture (Szegedy, Ioffe, Vanhoucke, & Alemi, 2017). That network combined both Inception and ResNet algorithms, previously highlighted in Convolutional Network Architectures, providing an enhanced network that best suits the facial recognition purpose. This pipeline was chosen thinking of the tradeoff between performance and computational cost of the overall architecture. Given the owned resources, it was important to have a less expensive framework while not sacrificing performance. As previously underlined, Inception modules maintained state-of-the-art accuracy with a modest increase of the computational cost compared to deeper networks, whereas ResNet's residual blocks allowed for optimized deeper networks

while reducing training time with its skip connections. The cropped detected faces ultimately serve as inputs to the network to classify the faces, outputting them into an image representation vector allocated into a Euclidean space referred as embedding. For the classifier module, the analysis was made with SoftMax classifier and cross-entropy loss function when training the CNN of the facial recognition module, the standard practice, while the real time system uses a Support Vector Machine to classify the logits extracted. The reasoning of using the CNN as feature extractor, having a model classifying outside of the main network, was the motivation of this dissertation. As seen in previous sections, CHARMIE performs generic service tasks in non-standardized environment, focusing in providing healthcare and domestic support, performing collaborative and cooperative tasks whose communications revolve around interacting with specific workers/patients/users. When presented with an unknown person, the learning process of a new class needs to be efficient and fast. Training a CNN as a multi-class classifier, introducing a new class, means an end-to-end re-training and re-evaluation of the network, being an extended and expensive procedure. Using an external SMV classifier proved efficiency in adding new persons to the dataset without being time expensive, with great accuracy.

Two standard datasets were used in this dissertation. For training of the feature extractor, the CASIA-Webface (Yi et al., n.d.) dataset was applied, as well as the LFW (Huang et al., 2008) for validation and fine tuning. Additionally, the custom-made LAR dataset was created, with acquisition of images of researchers and professors associated with the laboratory.

3.3 Face Detection

MTCNN (K. Zhang et al., 2016) proposes a framework to integrate detection and alignment tasks using unified cascaded CNNs by multi-task learning. The cascaded CNN framework consists of 3 architectures, those being, by structural and increasing complexity order, Proposal-Network, Refinement-Network, and Output-Network, where the output of the previous convolutional model is the input to the next.

3.3.1 Image Pyramid

Before being fed into the pipeline, the images were resized into different scales in an image pyramid manner, as seen in Figure 28.

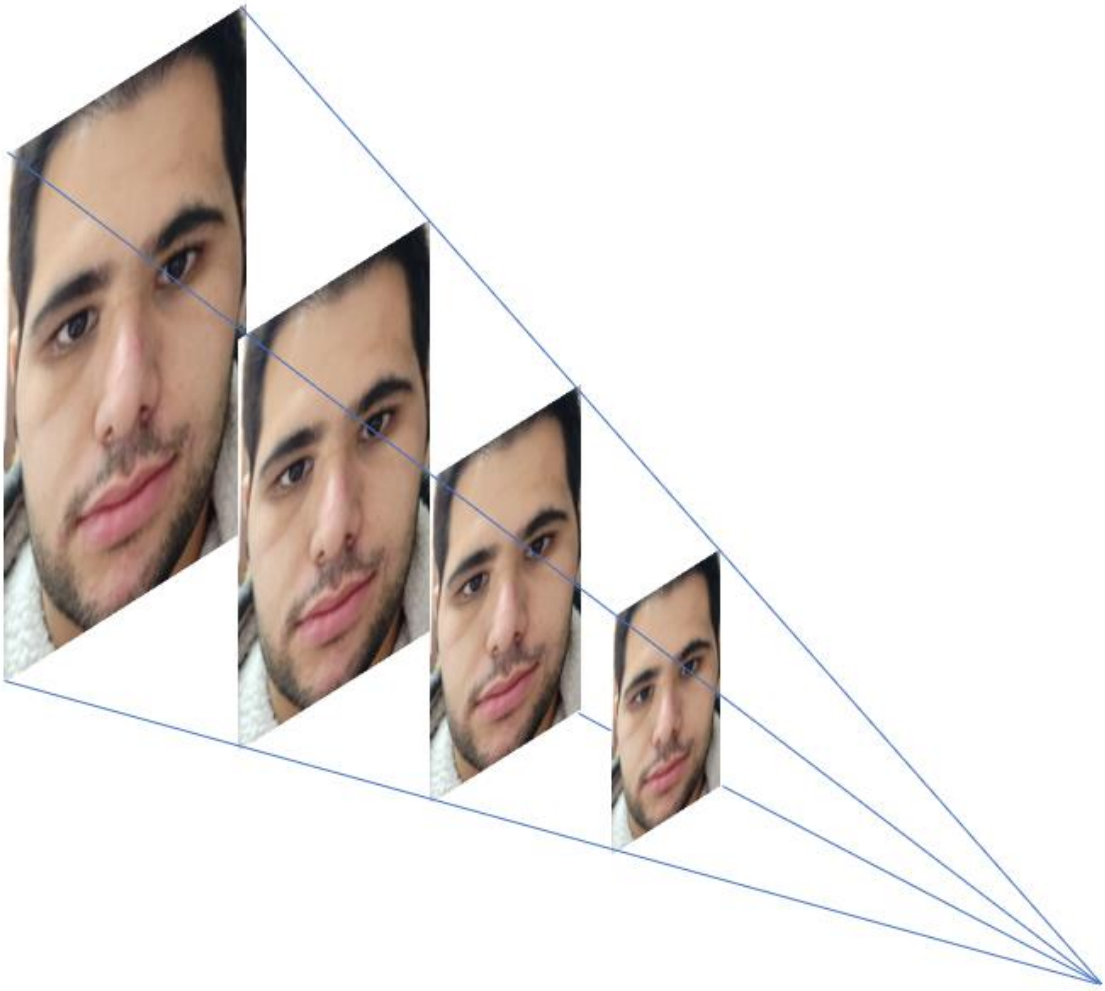


Figure 28 - Image pyramid.

This technique, widely embedded into cascaded frameworks, resized the original images using a scaled factor. That way the network's kernels could find different size faces in one sitting in the same image without the need to apply different spatial sizes.

3.3.2 Stage Cascaded Network and Non maximum suppression

The first network is known as P-Net, presented in Figure 29. Being a Shallow framework, it generates many candidates of where the face is, calibrated with their bounding box regression vectors.

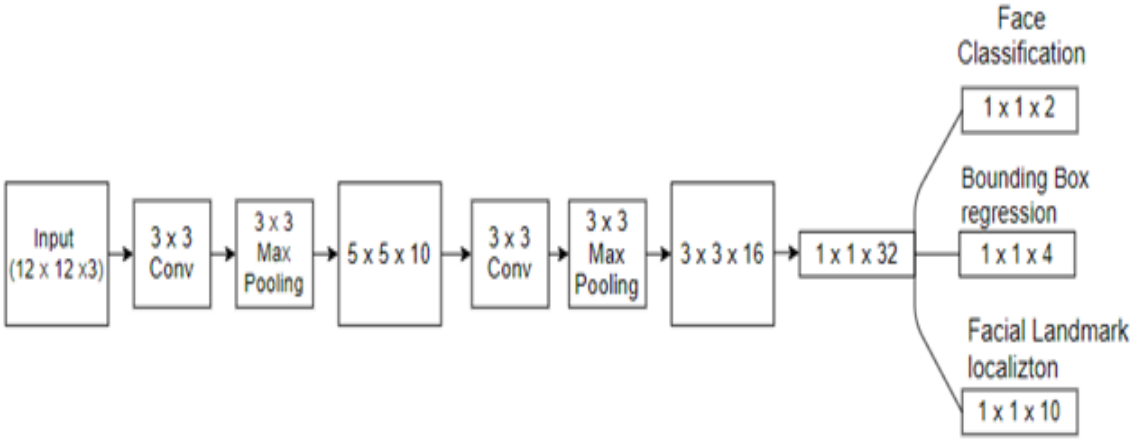


Figure 29 - Proposal Network architecture schematic.

After those proposals were found, the NMS algorithm was applied to merge highly overlapped candidates. This technique is used when various proposals of the same object, in this context a face, exist, choosing the candidate with the best prospect, filtering the rest. The score of the candidates is calculated by the IoU metric, used to quantify the percent overlap between two bounding box predictions. IoU can be determined by Equation (4):

$$IoU = \frac{Area\ of\ Overlap}{Area\ of\ Union} \tag{4}$$

Where $Area\ of\ Overlap$ is given by the Intersection (Figure 30.(a)) between two proposals and the $Area\ of\ Union$ is set by the Union (Figure 30.(b)) of the two. Figure 31 compares performance and with different IoU settings between 2 proposals.

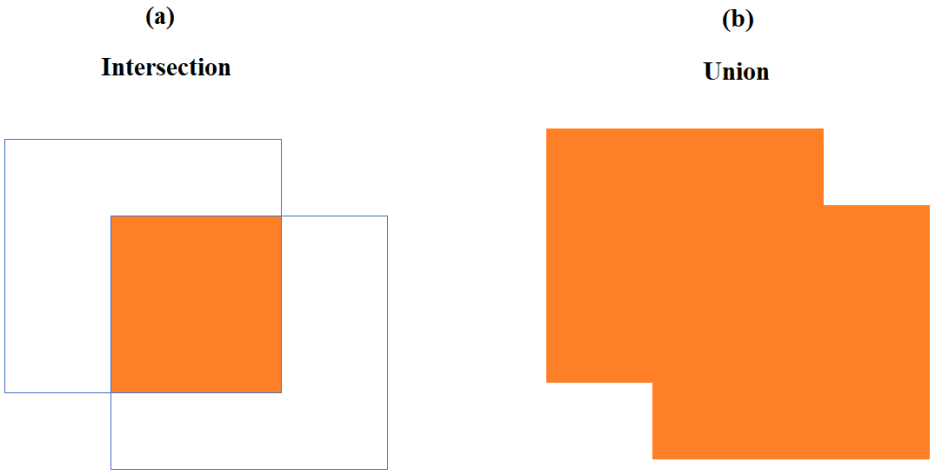


Figure 30 – Intersection (a) and Union (b) between two bounding boxes.

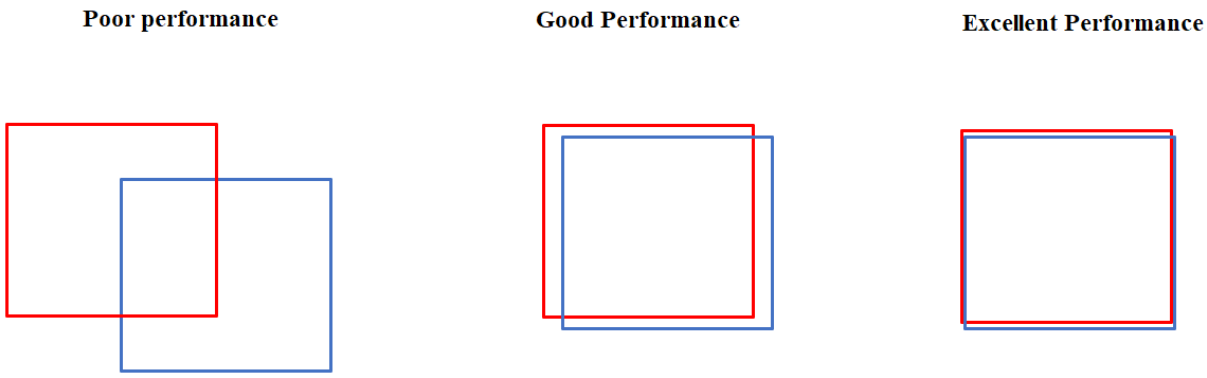


Figure 31 - IoU performance comparison.

NMS sets a threshold and calculates the IoU between the picked proposal with the best confidence and the rest of the proposals of the same candidate. If the IoU is higher than the threshold, meaning a big overlap between the two, that candidate is filtered. That way, a lot of saturated suggestions get merged to the previously chosen best candidate.

The R-Net (Figure 32) analyses the proposals given by the first CNN and filters false positives. Being more complex, while using a fully connected layer in the end of the model, it rejected many proposals given. Like the first stage, it calibrates the proposals with NMS merging and bounding box regression.

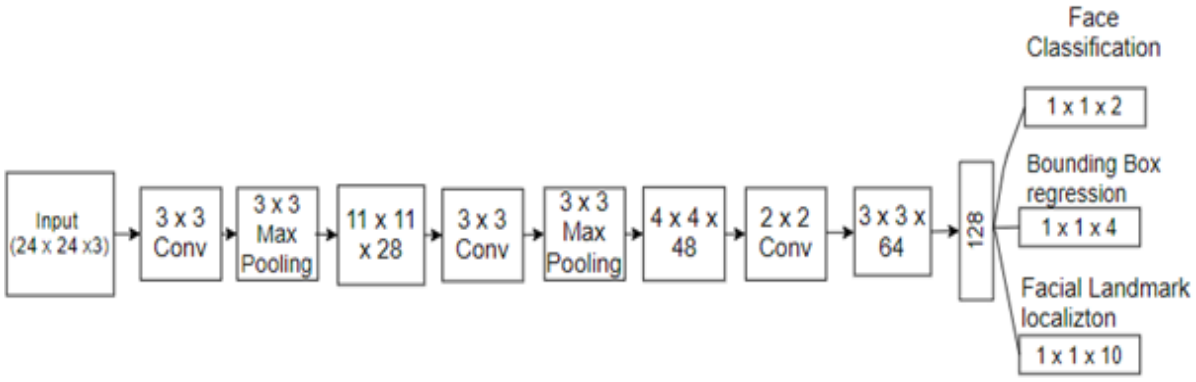


Figure 32 - Refinement Network architecture schematic.

The final network is the O-Net (Figure 33), which creates the final bounding box output of the detected face's image and 5 facial landmarks.

This framework produced 3 outputs: binary Face classification, 4 element Bounding Box Regression vector and 10-element Facial Landmark Localization vector.

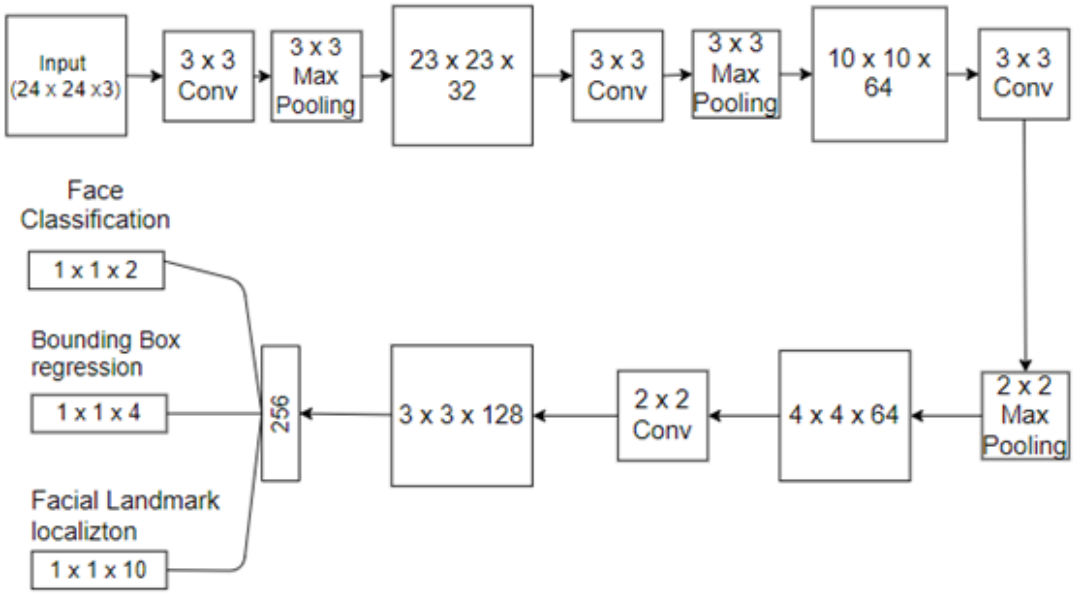


Figure 33 - Output-network architecture schematic.

3.3.3 MTCNN Training Methodology

The training was conducted on the WIDER FACE (Yang et al., 2016) dataset. Face classification problem uses cross-entropy loss to check if the predicted proposal is indeed a face, given by Equation (5):

$$L_i^{det} = \left(y_i^{det} \log(\rho_i) + \left(1 - y_i^{det}\right)(1 - \log(\rho_i))\right) \tag{5}$$

where ρ_i is the probability generated by the network that indicates a sample being a face. The notation y_i^{det} denotes the ground-truth label.

Bounding box regression vectors and NMS, the offset is predicted, and the ground truth employed with Euclidean loss, seen in Equation (6):

$$L_i^{box} = \left\|\hat{y}_i^{box} - y_i^{box}\right\|_2^2 \tag{6}$$

Where $\hat{y}_i^{box}$ is the target obtained from the network and y_i^{box} the ground truth coordinate.

Like the bounding box regression, facial landmark detection is a regression problem where Euclidean loss is minimized by Equation (7):

$$L_i^{landmark} = \left\| \hat{y}_i^{landmark} - y_i^{landmark} \right\|_2^2 \quad (7)$$

For this dissertation, only the Bounding Box regression output was considered. The trained weights and biases of this model were obtained via Transfer Learning. This is a ML method allowed pre-trained models to be reused, skipping the training part, focusing on the Inference part. Inference does not re-evaluate the NN. It only applies knowledge acquired from a trained NN model, inferring the result. So, when new unknown data is presented, it outputs a prediction based on predictive accuracy of the NN.

The implementation and parameters of this framework was used from GitHub ("GitHub - davidsandberg/facenet: Face recognition using Tensorflow," n.d.). Showed on Figure 34, the parameters were loaded from NumPy array file, replacing the TensorBoard placeholders used in the implementation

```
with tf.variable_scope('pnet'):
    data = tf.placeholder(tf.float32, (None,None,None,3), 'input')
    pnet = PNet({'data':data})
    pnet.load(os.path.join(model_path, 'det1.npy'), sess)
with tf.variable_scope('rnet'):
    data = tf.placeholder(tf.float32, (None,24,24,3), 'input')
    rnet = RNet({'data':data})
    rnet.load(os.path.join(model_path, 'det2.npy'), sess)
with tf.variable_scope('onet'):
    data = tf.placeholder(tf.float32, (None,48,48,3), 'input')
    onet = ONet({'data':data})
    onet.load(os.path.join(model_path, 'det3.npy'), sess)
```

Figure 34 - Parameter acquisition via Transfer Learning.

3.3.4 Implementation

The three initial variables declared had the same value at every framework. Those were the n x n minimum size of faces to be detected, the thresholds used in the NMS algorithm and the scalar factor for the image pyramid method. The values used are presented in Figure 35

```
minsize = 20 # minimum size of face
threshold = [ 0.6, 0.7, 0.7 ]  # three steps's threshold
factor = 0.709 # scale factor
```

Figure 35 - MTCNN variables for face detection.

The face detector can be used on individual images and videos by inputting the path file on Ubuntu terminal, or the real-time applications using the computers camera. These are executed with different scripts via Ubuntu Terminal, as showcased in Figure 36.

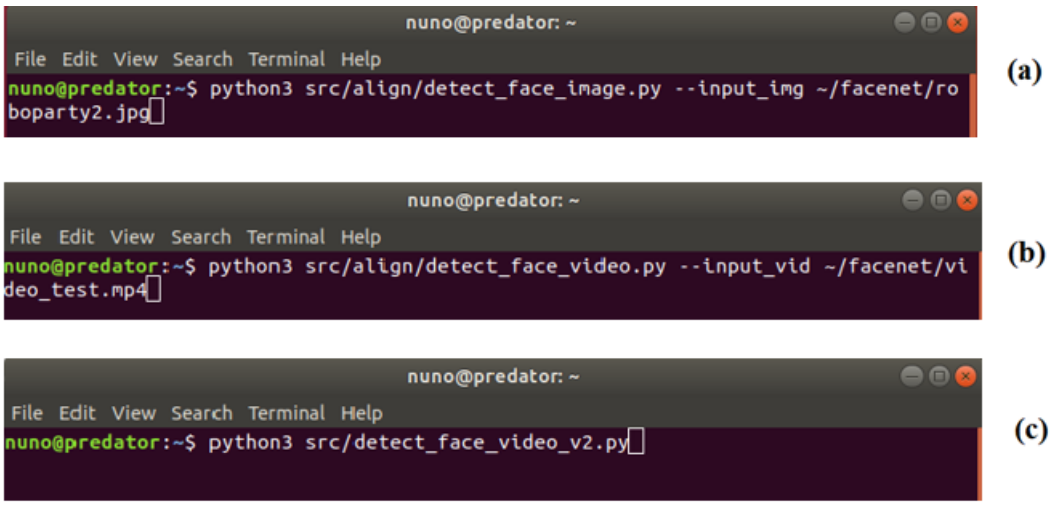


Figure 36 - Executing Python Scripts for face detection; (a) - Images; (b) - Videos; (c) - Real time.

As seen in the Figure 37 example of a dataset image, the align_dataset_mtcnn.py script automatically detected and aligned the faces in images in a dataset directory, creating a new dataset with only cropped faces with a specific configuration (160 x 160 image), the same configuration as the input values needed to feed CNN for feature extraction.

Figure 37 - Python Script used for detecting and aligning of faces in a dataset.

3.4 Datasets

To perform NN training and fine-tuning, two different datasets were used. Initially the CASIA-Webface (Yi et al., n.d.) dataset was used. It contained 494.414 images of 10.575 people for verification performance and it was further verified on the LFW (Huang et al., 2008) benchmark.

3.4.1 LAR dataset

Additionally, a personalized dataset was created and used for training and validation. It contains 19 classes of image data from students and teachers associated with the Laboratory of Automation and Robotics (LAR) of Minho University. The elements contained in the dataset are set in Table 9. All members consented the use of their identity and use of one's images.

Table 9 - Elements of the LAR dataset.

Name	Genre	Age
António Ribeiro	Male	20
Bruno Sousa	Male	25
Carlos Silva	Male	23
Carolina Pires	Female	24
Carolina Rua	Female	23
Dimitri Santos	Male	23
Fabio Cunha	Male	22
Fawad	Male	32
Fernando Ribeiro	Male	54
Francisco Ribeiro	Male	18
Gil Lopes	Male	49
Inês Garcia	Female	25
Inês Ribeiro	Female	23
Michael de Oliveira	Male	26
Nuno Andrade	Male	22
Nuno Pereira	Male	25
Rafael Marques	Male	25
Sérgio Baixo	Male	24
Tiago Ribeiro	Male	25

The individual images that made the dataset were fetched using the OpenCV library. It extracted frames from video of a sole identity using the computer's camera or from a personal device. The video content was simple: the person would start by looking directly at the camera, then rotating their head 180° from left to right. That way, the model would have different perspectives of its classes.

Then, the python script New_class.py, Figure 38, created a directory with the target's name inside the selected dataset. The script checks the video frames, turning them into ordered file images with the person's name, inserting them into the created directory. It had 4 arguments: the video for image extraction, dataset's directory path to be inserted, the class name, and the flag 'save' to verify the extraction.

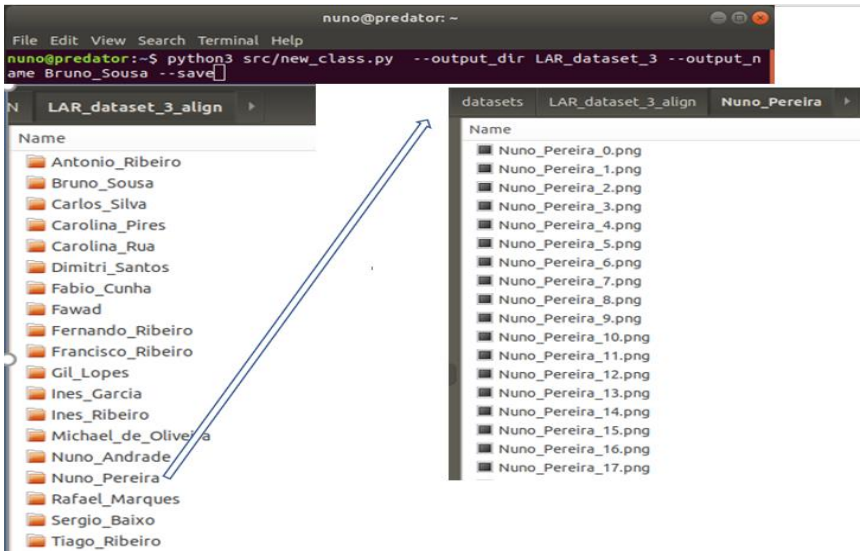


Figure 38 - Custom Dataset.

3.4.2 Image Transformations

In order to create a robust system, image transformations were performed in some of the frames extracted from the videos when the images were created. This included random rotations and horizontal flips of the image done via OpenCV library. The rotations were restricted to -30 to 30°, as seen in Figure 39.

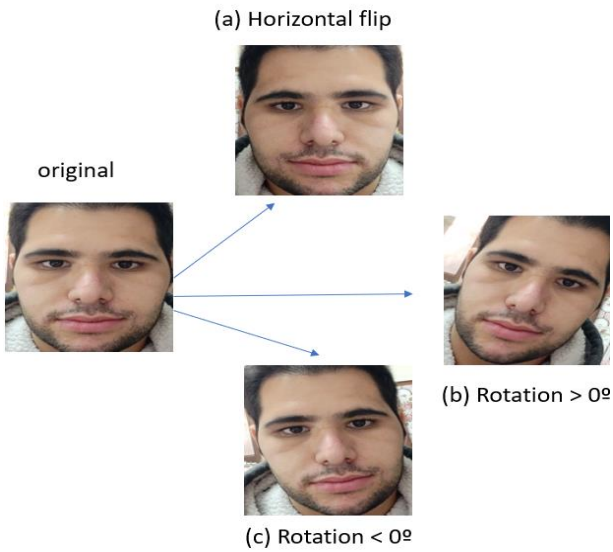


Figure 39 - Image transformations using OpenCV Library, (a) - horizontal flip; (b) - Random rotation (> 0°); (c) - Random rotation (<0°).

3.5 Inception-ResNet Network

The main deep NN pipeline used for faces feature extraction is the Inception-ResNet version 1 and 2 (Szegedy et al., 2017). This model results in a combination of two different deep learning methodologies, previously reviewed in Convolutional Network Architectures: residual connections introduced by ResNet (K. He et al., 2016) and the Inception modules introduced with GoogleNet (Szegedy et al., 2015). The proposed model was trained locally using supervised learning methods.

3.5.1 Approached Concepts

In traditional NNs, a layer feeds data to the following one, but with residual connections (Figure 21), the information is sent directly from a layer into a deeper one. These blocks improve two distinct areas. First, the training time, given that skip-connections can jump layers without training. Secondly, the lost information when making gradient descent, since more deep networks have a hard time being accurate without overfitting or working more straight forward tasks (Vanishing Gradient problem (Glorot & Bengio, 2010)). The residual Inception blocks used on this architecture were optimized with a filter-expansion layer, a 1 X 1 convolutions without activation (Figure 40). Those were implemented in order to compensate the dimensionality reduction induced by the Inception part, given that residual addictions only work if the input and output matched the same depth sizes.

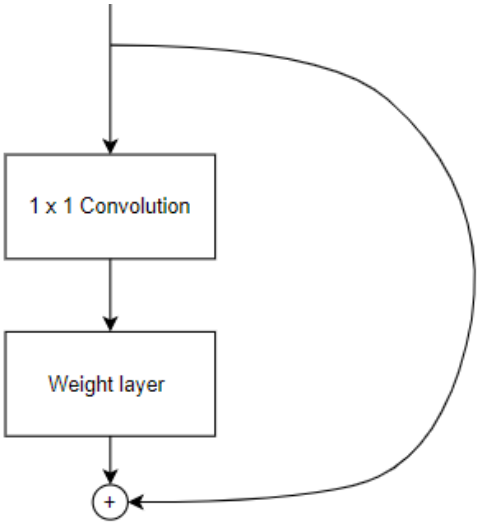


Figure 40 - Optimized residual block.

3.5.2 Network Architecture

The pipeline's architecture was the same for both versions, seen in Figure 41:

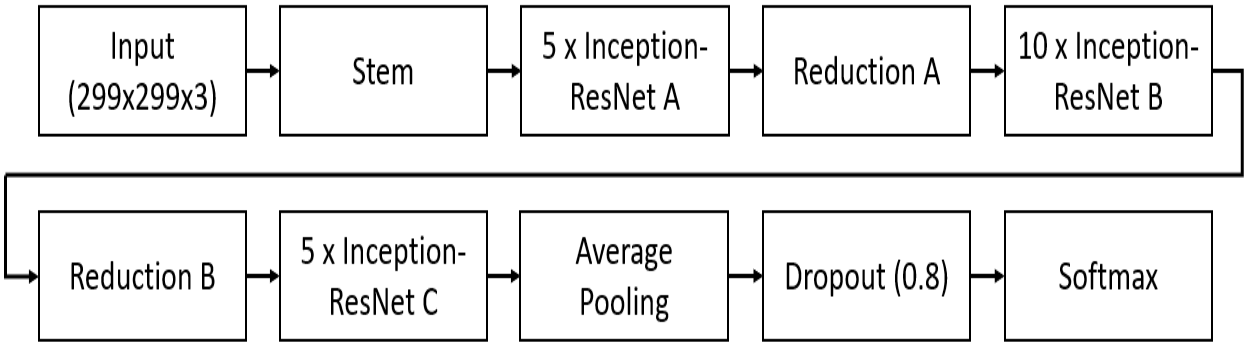


Figure 41 - Schematic of Inception-ResNet version 1 and 2.

Versions 1 and 2 of the models only differed from the stem (Annex I – Stem of Inception - ResNet architecture versions 1 and 2), which was the initial set of operations performed before introducing the Inception blocks and hyperparameter selection. Version 2 has the same stem as the Inception v4 (Szegedy et al., 2017) model, presented in the same paper, with filter concatenation approaches, while the version 1 stem is a standard CNN. Inception-ResNet v1 has a computational cost similar to that of Inception v3 (Szegedy et al., 2016), while Inception-ResNet v2 has a computational cost that is similar to that of Inception v4.

Both versions presented the same Inception modules, A, B and C, differing only on hyperparameter settings of this network. They were designed so that Pooling layers were replaced by the residual connections, adding the output of the previous Inception module to the final output.

Inception-ResNet-A block, version 1 (Figure 42.(a)) and version 2 (Figure 42.(b)), presented a 35 x 35 grid module. Inception-ResNet-B block, versions 1 Figure 43.(a)) and version 2 (Figure 43.(b)), presented a 17 X 17 module, and Inception-ResNet C, versions 1 (Figure 44.(a)) and version 2 (Figure 44.(b)) presented a 8 x 8 grid. The ideas presented in earlier Inception versions are also embedded, like asymmetric convolutions or replacement of 5 x 5 convolutions with two 3 x 3 convolutions or. Also, in Figure 42, Figure 43, and Figure 44 the number of activation maps in a convolution is shown inside parenthesis. One can notice that the version 2 has a more complex Stem and more filters per convolution.

.(a)) and 2 (Figure 44.(b)) presented a 8 x 8 grid. The ideas presented in earlier Inception versions are also embedded, like asymmetric convolutions or replacement of 5 x 5 convolutions with two 3 x 3

convolutions or. Also, in Figure 42, Figure 43, and Figure 44 the number of activation maps in a convolution is shown inside parenthesis. One can notice that the version 2 has a more complex Stem and more filters per convolution.

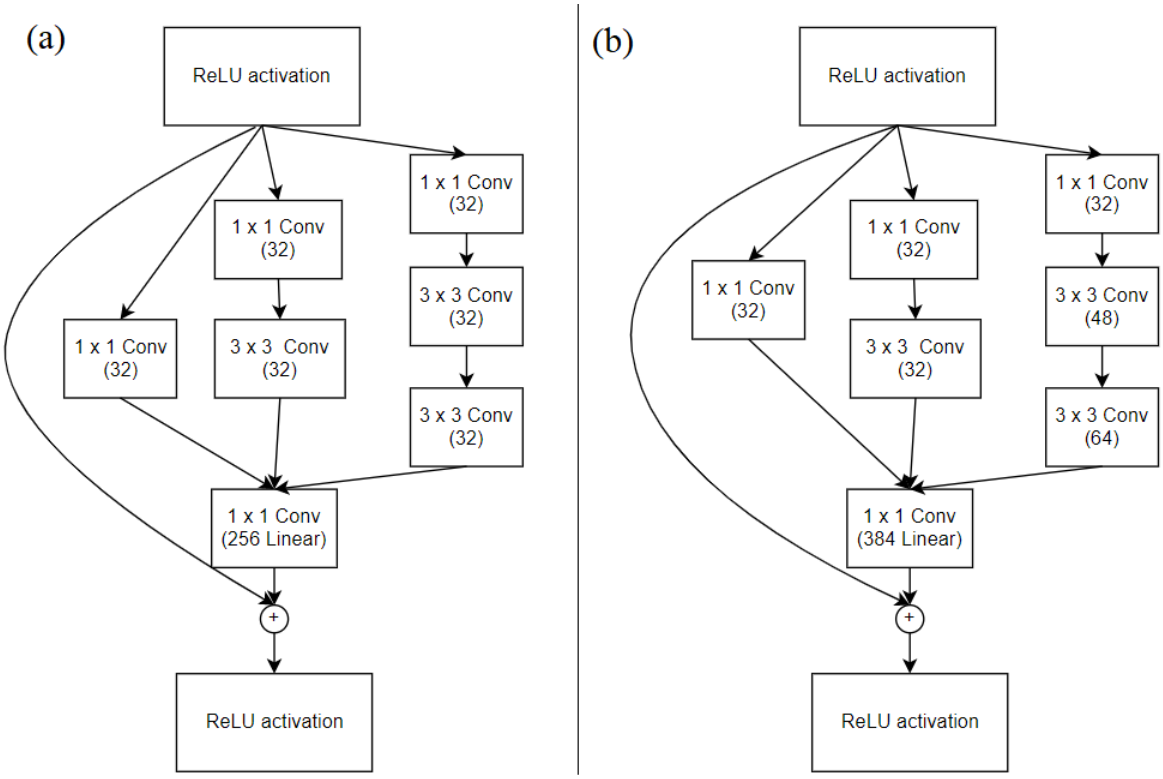


Figure 42 - Inception ResNet Block A; (a) version 1; (b) version 2.

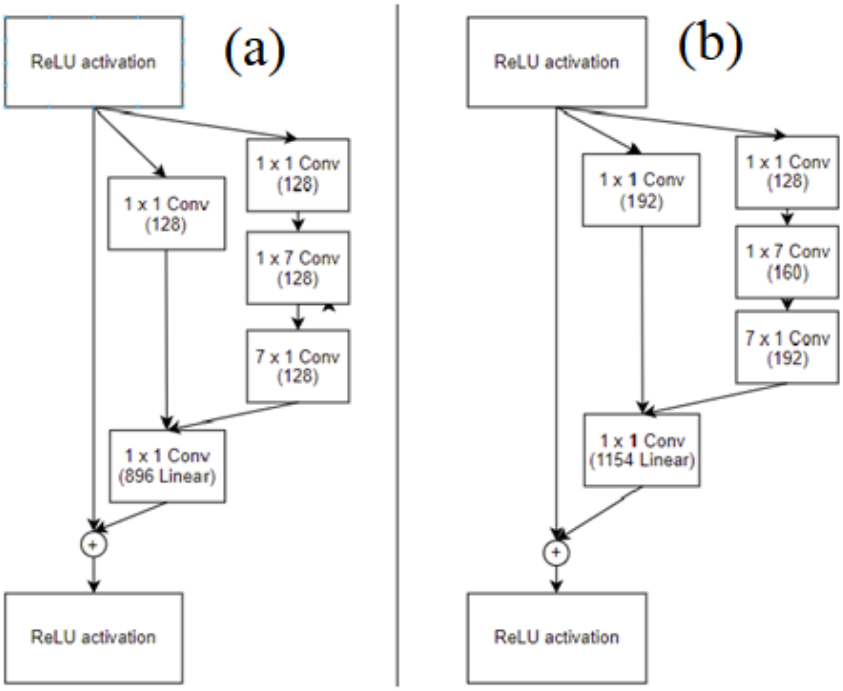


Figure 43 - Inception-ResNet Block B; (a) version 1; (b) version 2.

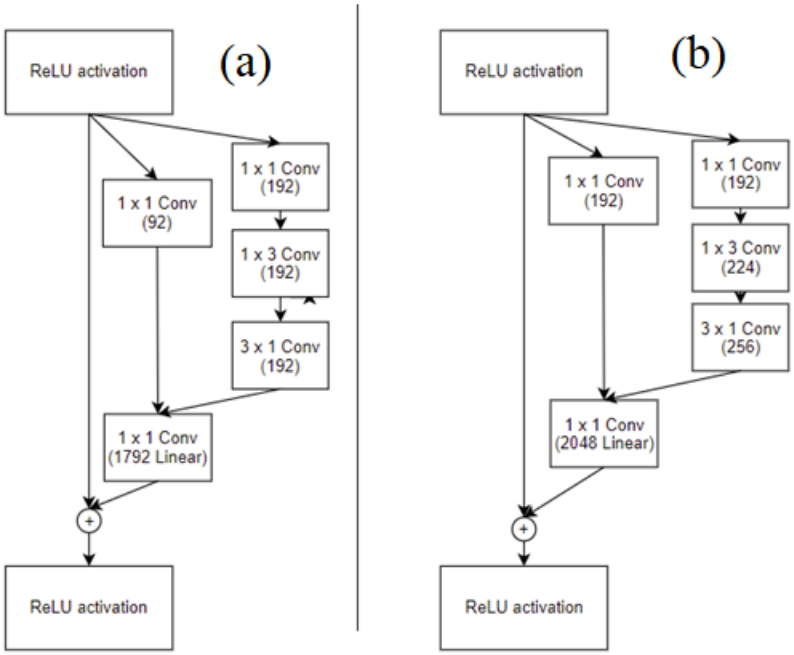


Figure 44 - Inception ResNet Block C; (a) version 1; (b) version 2.

Regarding dimension reduction, instead of pooling layers, two specific structures were conceived. Reduce-A block (Figure 45.(a)) reduces the feature maps from 35 x 35 grid to a 17 x 17, while the Reduce-B block (Figure 45.(b)) reduces the 17 x 17 ones to an 8 x 8 grid.

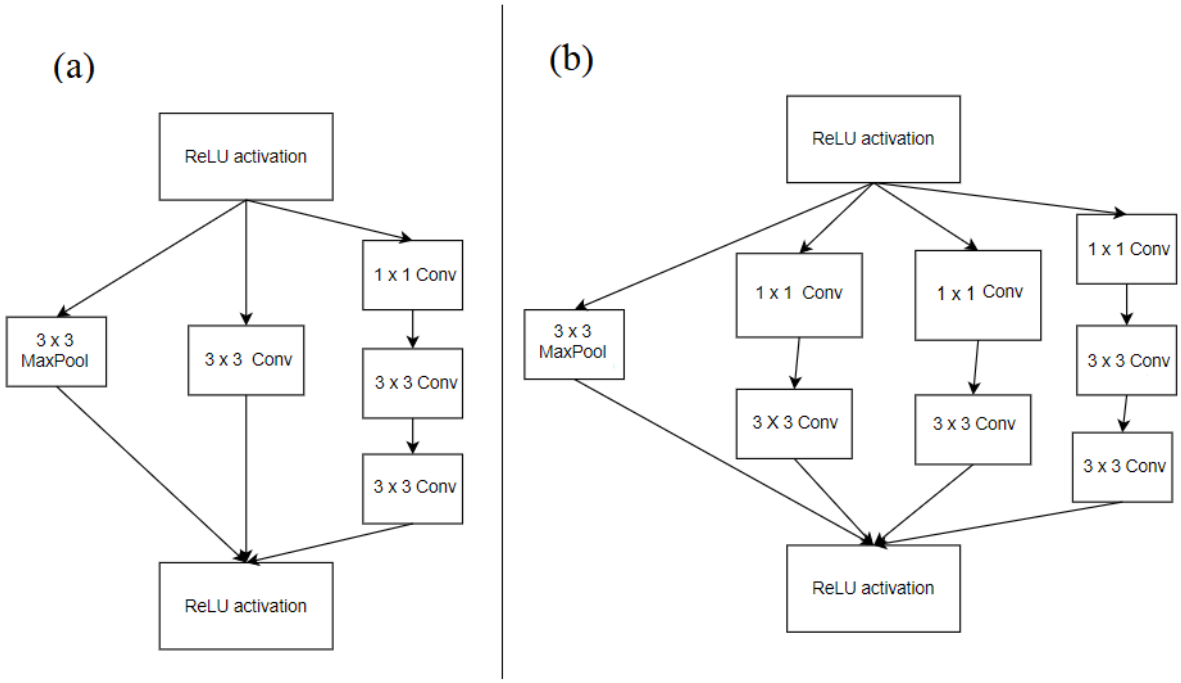


Figure 45 - Reduction blocks; (a) Reduction-A; (b) Reduction-B.

Batch Normalization, like the one presented by Szegedy (Szegedy et al., 2016), was applied on top of traditional layers, but not on filter concatenations, since the authors reported disproportionate amount of GPU memory. By omitting the batch-normalization on top of those layers, overall number of Inception blocks was increased substantially.

Both versions' implementations of this network architecture on TensorBoard were used from ("GitHub - davidsandberg/facenet: Face recognition using Tensorflow," n.d.).

3.5.3 Pipeline Implementation

The CNN implementation is imported into the training/inference via python script using the Importlib package, or via a trained network's metagraph and checkpoint, that contains the dataflow of the previous implementation and trained parameters of a previous network, using a TensorFlow saver, as seen in Figure 46.

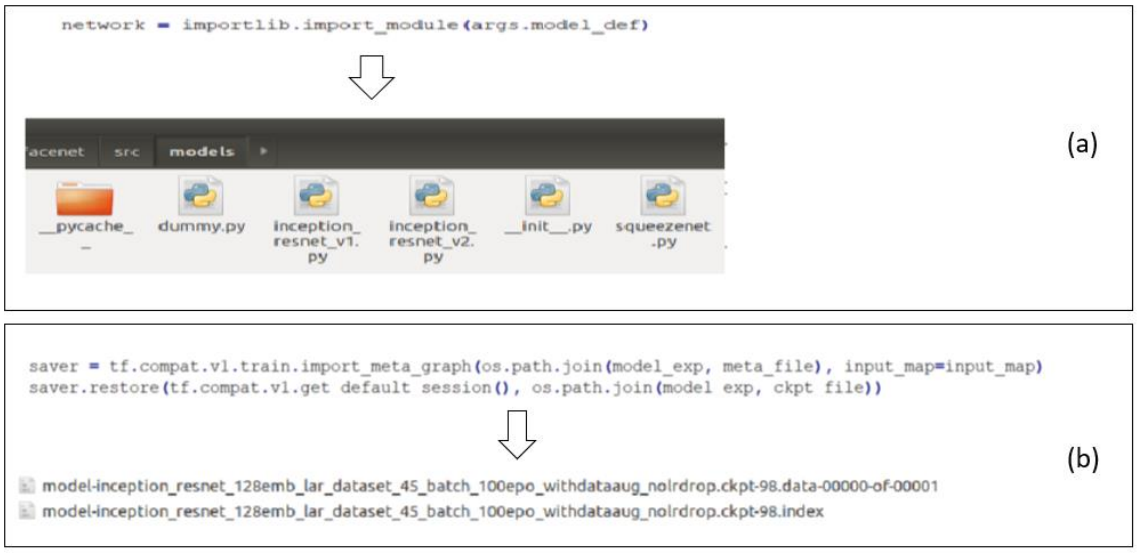


Figure 46 – Import of the Inception-ResNet model; (a) Import the python implementation file; (b) Reload a trained model metagraph and checkpoint.

The TensorFlow framework during training transforms the images in a batch into the input of the network. Those were a 4-D tensor of size [batch size, height, width, 3(the channels R G B)]. The height and width were 160 x 160, the same as the author's implementation of the network.

The network's inference function inputted the variables set in the training first, as well as initiate the weights and biases and its regularizer. When the training starts, it feeds the network with the batches of

images. It is also responsible for setting the batch normalization parameters and the dropout layer, as well as configurate the size of the prelogit, the output of the network.

The fully connected layer for classification, better known as logits layer, is built outside the main script, with its size being the number of classes to identify. After, the logits are normalized with an L2_Normalization, a technique where sum of the squares of the output will always be up to 1. These 3 configurations are presented bellow in Figure 47.

```
# Build the inference graph
prelogits, _ = network.inference(image_batch, args.keep_probability,
    phase_train=phase_train_placeholder, bottleneck_layer_size=args.embedding_size,
    weight_decay=args.weight_decay)
logits = slim.fully_connected(prelogits, len(train_set), activation_fn=None,
        weights_initializer=slim.initializers.xavier_initializer(),
        weights_regularizer=slim.l2_regularizer(args.weight_decay),
        scope='Logits', reuse=False)

embeddings = tf.nn.l2_normalize(prelogits, 1, 1e-10, name='embeddings')
```

Figure 47 - Inception-ResNet inference function initialization and classifier layers.

3.6 Training Methodology

For model training, the script uses various input arguments (example on Figure 48) for various purposes such as loading the dataset, modification of the hyperparameters to control the learning process, as well as defining the file information storage, saving the information obtained, such as parameters graphs, metafiles or training checkpoints.

```
nuno@predator:~/facenet$ python3 src/train_softmax_v3.py --logs_base_dir ~/facen
et/models/facenet/inception_model_CASIA --models_base_dir ~/facenet/models/facen
et/inception_model_CASIA/v1 --data_dir ~/datasets/Casia_maxpy_mtcnnpy_182 --imag
e_size 160 --model_def models.inception_resnet_v1 --optimizer ADAM --learning_ra
te -1 --learning_rate_schedule_file data/learning_rate_schedule_classifier_casia
.txt --keep_probability 0.4 --weight_decay 5e-4 --embedding_size 512  --prelogit
s_norm_loss_factor 5e-4 --model_name inception_resnet_512_emb_casia_dataset_500_
batch_100esize_500epo_nodataaug_lr_decay_0.1 --batch_size 50 --epoch_size 100 --
max_nrof_epochs 500
```

Figure 48 - Python script for training the Inception-ResNet CNN.

3.6.1 Hyperparameter Settings

Different hyperparameters input settings were employed to study the better model configuration. For batch configuration, the inputs included the batch size, the number of batches to use in an epoch and flags for data augmentation

For the training flow, the input parameters defined the number of max epochs for training and the number of epochs between another validation evaluation of the model.

For the network structure, the inputs replaced the pre-determined parameter values. Those included the dropout layer, the lr, the optimizer used for gradient descent and the weight decay used in the l2_regulizer.

3.6.2 Image manipulation

When importing the dataset directory, the images are grouped into 3 sub-sets: a training dataset containing 90% of a class images (with 10% of those being allocated for the cross-validation) and a test dataset containing the remaining 10%. The training dataset are the images used for training. The validation dataset is used during training to estimate the model skill while tuning model's hyperparameters. The test dataset is used in the end of training to verify its performance.

The images are fed into the network though batches using a FIFO queue containing 3 elements: The Image, the matching Label, and a Control number. The Control number manages Data Augmentation, a technique used to increase the number of images, and employed to simulate an additional pose variation, hence the diversity in the dataset. This is managed by 3 flags: flipped_images, cropped_images and rotated_images. If the flags were activated, via input argument on the terminal, a copy transformation (either flip, rotated or cropped copies per image or a combination of them) of the data were added to the original queue. As example, Figure 49 shows a batch input of the LFW dataset using data augmentation with rotated images.

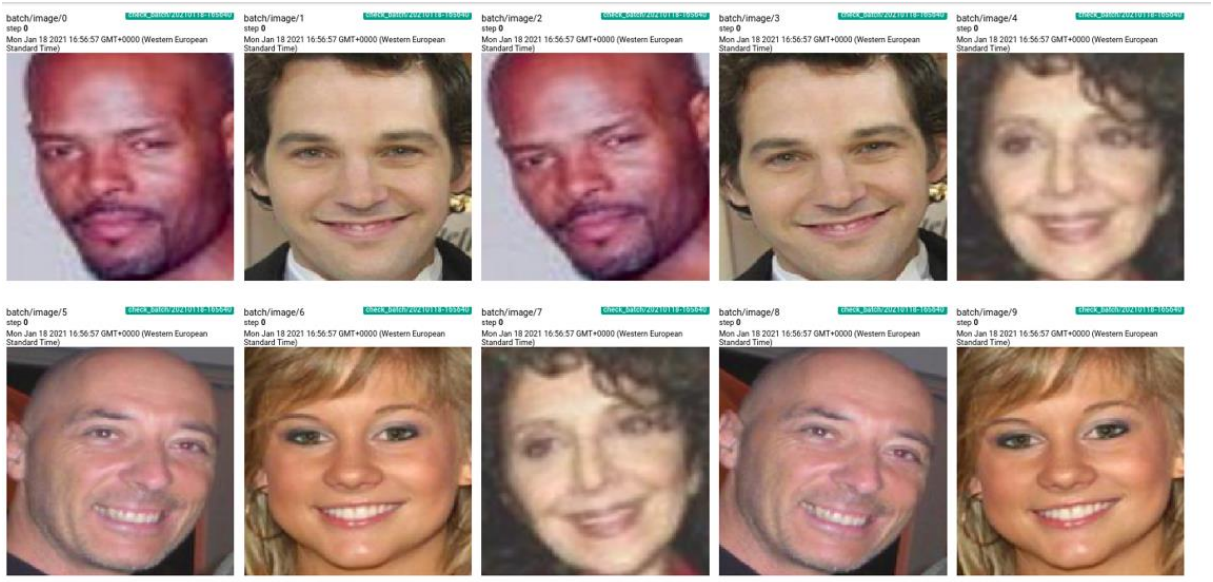


Figure 49 - Batch visualization using data augmentation with rotated images.

Before being inputted to the network, the images are prewhiten, which subtracts the average and normalizes the range of the pixel values of input images. It made the training process a lot easier by speeding up the training and inference.

3.6.3 Learning Methodology

The NN used Softmax with Cross Entropy as training loss function over an input batch. The Softmax function computes the output probabilities array of the classes in the dataset. The higher the probability on a class, the more the model thinks the input object represents that class. The sum of the class's probability is always 1. This is represented by the function:

$$S_i = \frac{e^{V_i}}{\sum_j e^{V_j}} \tag{8}$$

Where V_i represents the input elements and $\sum_j e^{V_j}$ the normalization term to range the probability in the 0-1 range.

The loss is then calculated with Cross-Entropy, defined by:

$$H(p,q) = -\sum_x p(x) \log \mathrm{q}(x) \tag{9}$$

Where $p(x)$ represents the conditional empirical distribution over class labels, being 1 to the observed class and 0 for all others, and $\mathrm{q}(x)$ is the conditional distribution, probability obtained from the model given an input.

3.6.4 Saving the Model

Saving the model and its variable logs and events was carried out with Summary FileWriters, providing a mechanism to create an event file in a directory and add summaries and events to its protocols. The tf.Summary module provides APIs for writing summary data, being visualized in TensorBoard.

Split into 3 files for training, validation, and testing, observed in Figure 50, those events were inserted into the directories trained model. That way, when using TensorBoard, one can check the 3 directories at the same time or import one isolated list.

```
train_summary_writer = tf.compat.v1.summary.FileWriter(log_dir + '/' + 'train', sess.graph)
validation_summary_writer = tf.compat.v1.summary.FileWriter(log_dir + '/' + 'validation', sess.graph)
test_summary_writer = tf.compat.v1.summary.FileWriter(log_dir + '/' + 'test', sess.graph) #visualize t
```

Figure 50 - Creation of the file writers for data exportation.

After an epoch training, the model's checkpoint and metagraph were also exported to the model's directory with a saver. Those can later be used for the classifier pipeline or training other models with transfer learning. The variables tuned from training epoch are then saved to the file writers, as seen in Figure 51.

```
start_time = time.time()
checkpoint_path = os.path.join(model_dir, 'model-%s.ckpt' % model_name)
saver.save(sess, checkpoint_path, global_step=epoch, write_meta_graph=False)
save_time_variables = time.time() - start_time
#print('Variables saved in %.2f seconds' % save_time_variables)
metagraph_filename = os.path.join(model_dir, 'model-%s.meta' % model_name)
save_time_metagraph = 0
if not os.path.exists(metagraph_filename):
  #  print('Saving metagraph')
    start_time = time.time()
    saver.export_meta_graph(metagraph_filename)
    save_time_metagraph = time.time() - start_time
    #print('Metagraph saved in %.2f seconds' % save_time_metagraph)
summary = tf.compat.v1.Summary()
train_summary_writer.add_summary(summary, epoch)
validate_summary_writer.add_summary(summary, epoch)
test_summary_writer.add_summary(summary, epoch)
```

Figure 51 - Saving and importing the model's metagraph and variables.

3.7 Classifier

With the model of Inception-ResNet, adding a new person to the dataset implied having to always restart the training with new classes. As a solution, using the model only as a feature extractor to map images into an embedding vector, the predictions can be done using outside classifiers with less computational cost.

Before building the classifier, the Inception-ResNet feature extractor graph is loaded, where the images used are encoded into an embedding array. Those are then used to train an SVM classifier.

Support Vector Machines produced significant accuracy with less computation power. The objective of the support vector machine algorithm is to find a hyperplane in an N-dimensional space (N the number of features) that distinctly classifies the data points, as observed in Figure 52.

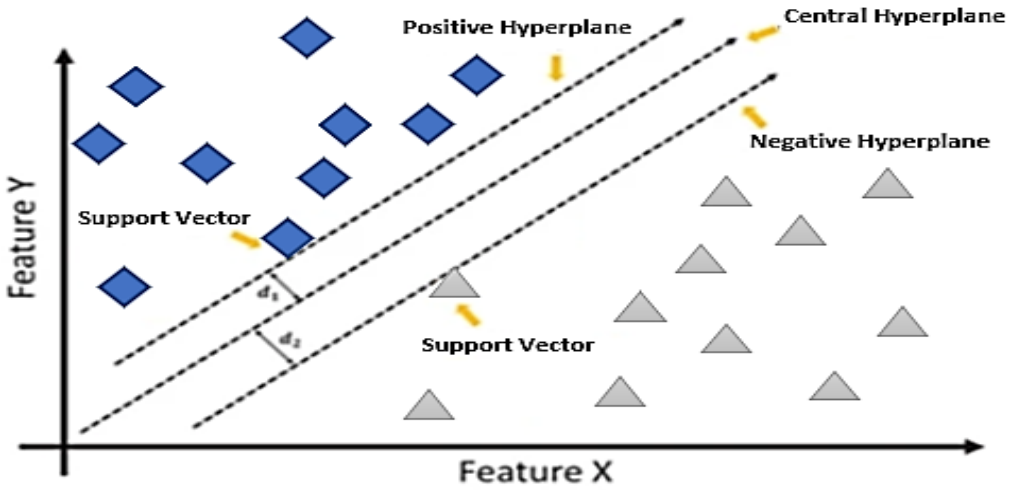


Figure 52 - Support Vector Machine representation.

Hyperplanes work as decision boundaries that classify the data points. Those allocated on either side of the hyperplane can be attributed to different classes. By finding the maximum margin the maximum distance using the support vectors (data points closer to the hyperplane that influence its position and orientation), future data points could be classified with more confidence. Some classification problems cannot be solved using a linear hyperplane. The SVM used a kernel to transform the input space to a higher dimensional space to be able to segregate the data points.

The model used was the SVM implementation of Scikit-learn library (Figure 53). It used linear kernel technique. It is used when the data can be separated using a single line, since the dataset had a large number of features.

```
print('Training classifier')
model = SVC(kernel='linear', probability=True)
model.fit(emb_array, labels)
```

Figure 53 - SVM implementation on Scikit-Learn library.

The python script used has 2 input modes, with its input initialization specified on

Figure 54: TRAIN and CLASSIFY.

Figure 54 - Classifier Script inputs; (a) Train the classifier; (b) Predict with classifier.

The TRAIN mode inserted the images and labels into the SVM model, where the final classifier is stored in a Pickle file, a Python module that enables objects to be serialized on a disk or deserialized into a program runtime classifier uses. On the other hand, the CLASSIFY loaded the classifier, where the input images classes are predicted by the previously trained classifiers.

3.8 Data Visualization

TensorFlow's data visualization toolkit, TensorBoard, provided the necessary mechanisms to evaluate the model's performance, plotting how the learnable parameters converged over time of training with different hyperparameters. It also allowed the display of different variables, like the dataflow of the computations, images, or projection of data used.

3.8.1 Computational graph

The computational graph allowed a schematic view of the TensorFlow model using the TensorBoard's Graph Dashboard, examining its structural design. As previously discussed, the data used by TensorFlow is stored in tensors. Hence, the computations done are schemed into the operations/nodes performed by the code, with the computational dataflow being represented as well.

In Annex II – TensorBoard Training Graph, the computational graph of the used model is presented. In order to have a simpler layout, all operations involving a module were designated with the same name scope. As an example, the millions of parameters of the network are hidden inside the Inception-ResNet name scope, not compromising the graphs design.

3.8.2 Plots

Key metrics, such as loss or accuracy, were plotted using the TensorBoard's Scalars Dashboard via Summary File Writers (Figure 55). The same way, images were exported using the TensorBoard's Image Dashboard (Figure 49).

```
summary =tf.compat.v1.Summary()
#pylint: disable=maybe-no-member
summary.value.add(tag='time/total', simple_value=train_time)
summary.value.add(tag='Accuracy', simple_value=accuracy_array_mean)
summary.value.add(tag='Loss', simple_value=loss_array_mean)
summary.value.add(tag='regularization_loss_train', simple_value=reg_loss_array_mean)
summary.value.add(tag='cross_entropy_loss_train', simple_value=xent_loss_array_mean)


summary_writer.add_summary(summary, global_step=epoch)
```

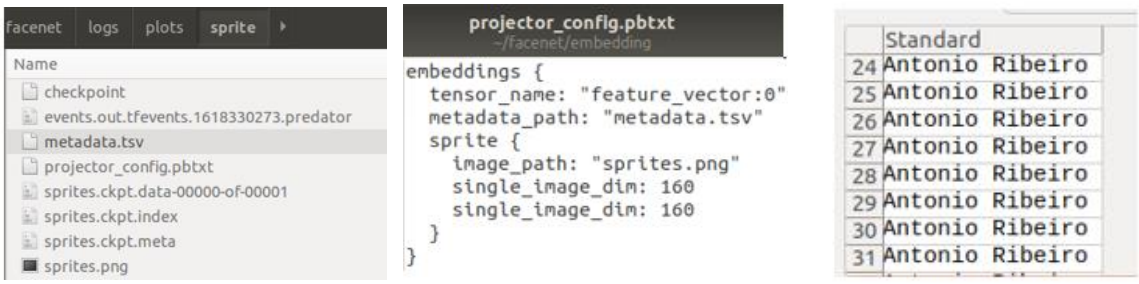
Figure 55 - Scalar call after an epoch train and exporting via Summary writer.

3.8.3 Embedding Projector

TensorBoard ‘s Embedding Projector graphically represents the high dimensional feature vectors into smaller dimensions using Dimension Reduction algorithms. That way, examining how those methods separate/cluster the data created by the extractor can validate the training process.

To encode the images into the projector, a file directory had to contain a sprite file to be uploaded, the configuration file and the neatly labels that identify the images (Figure 56.(a)).

The configuration file associated the sprite image with the metadata. It also set the single image size to then be isolated for each other (Figure 56.(b)). The metadata was a TSV file, a file extension for a tab-delimited file used with spreadsheet software, that contained the matching labels in the sprite images (Figure 56.(c)).



(a) **(b)** **(c)**

Figure 56 – Embedding Projector Setting (a)-Projector Directory; (b)- Projector configuration file; (c) – Metadata file containing the ordered labels

The sprite image was a single frame compilation of all images uploaded to the projector as seen in.

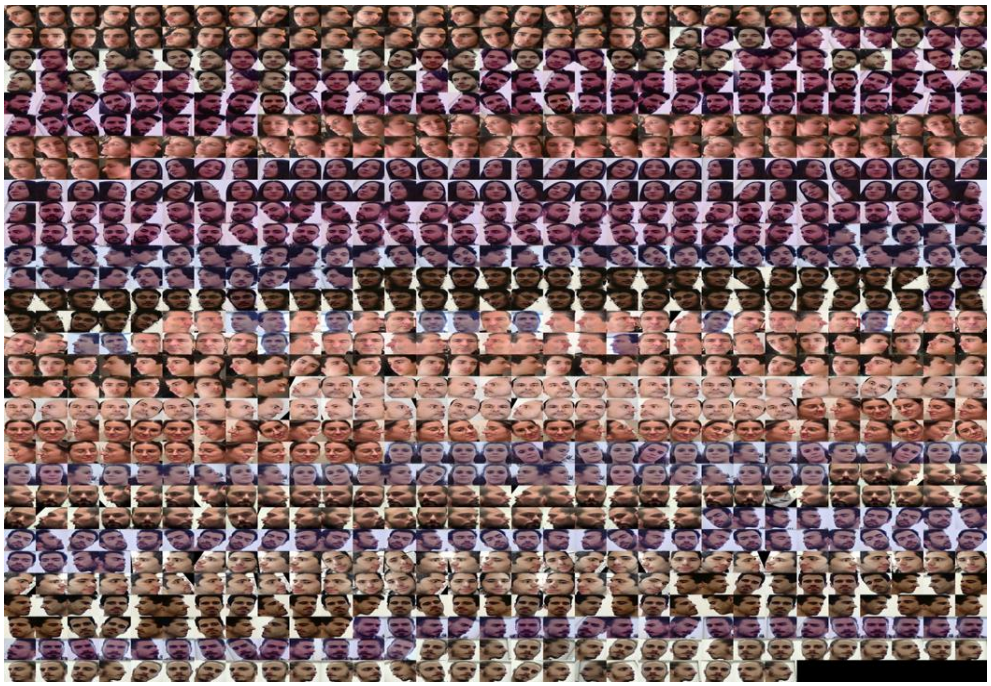


Figure 57 - Sprite of some images of the classes used in the LAR dataset.

3.8.4 Principal Component Analysis

Principal Component Analysis reduces the data dimensionality, making a linear projection of smaller components, with the premise that more straight forward samples are more accessible to interpret and visualize at the cost of very little information loss.

At a conceptual level, PCA determines the principal components of new variables that are a linear combination of the original data. It accomplishes this by creating new uncorrelated variables that consecutively maximize variance.

After normalizing the data, the covariance matrix is calculated by Equation (10):

$$cov(X,Y) = \frac{1}{n-1}\sum_{i=1}^{n}(X_i - \bar{x})(Y_i - \bar{y}) \quad (10)$$

To check how the variables are related to each other, the Eigenvectors, vectors whose direction remains unchanged when transformed, are calculated, as well as the Eigenvalues. The scalar from which the eigenvector is scaled from uses Equation (11):

$$Av = \lambda v \quad (11)$$

Where A is the correlation matrix, v is the eigenvector, and λ is the eigenvalue. The Principal Components are the top k-dimensional eigenvectors that give the direction of the axes, with the largest eigenvalues, meaning that more variance was stored, where k is the new dimension. It ranks these from highest to lowest K value returning the principal components in order of significance.

3.8.5 t-SNE

T-distributed stochastic neighbor embedding, presented by Van Der Maaten and Hinton (Van Der Maaten & Hinton, 2008b) is a non-linear method based on Stochastic Neighbor embedding that computes K-dimensional data into a lower dimensional mapping by performing several region-based transformations. t-SNE differs from PCA by preserving only small pairwise distances or local similarities whereas PCA is concerned with preserving large pairwise distances to maximize variance.

For that, it calculates the joint probability distributions (similarities between data, mapping them into a dataset of points in the new n dimension), applying gradient descent to tune similarities in the distribution.

The similarity of datapoint x_j to datapoint x_i is the conditional probability $p_{j|i}$, where x_i would pick x_j as its neighbor if neighbors were picked in proportion to their probability density under a Gaussian centered at x_i, and those probabilities are proportional to the similarities. That probability is given by Equation (12):

$$p_{j|i} = \frac{exp\left(-\|x_i - x_j\|^2 / 2\sigma_i^2\right)}{\sum_{k \neq i} exp(-\|x_i - x_k\|^2 / 2\sigma_i^2)} \quad (12)$$

Where σ_i represents the variance of the Gaussian that is centered on datapoint x_i. Joint probability in the high-dimensional of data is defined as p_{ji}, to be the symmetrized conditional probabilities, calculated by Equation (13):

$$p_{ij} = \frac{p_{j|i} + p_{i|j}}{2n} \tag{13}$$

To reduce the data, a random dataset of the same number of points of the high dimension dataset is created in the K-dimension desired. The distribution joint probability is created with t-distribution, given that it maximizes distances between points in the low dimension space compared to the same distance in the correspondent high-dimensional space.

Joint probability in the low dimensional space is defined as q_{ij}, calculated by Cauchy distribution using Equation (14):

$$q_{ij} = \frac{\left(1 + \|y_i - y_j\|^2\right)^{-1}}{\sum_{k \neq l}(1 + \|y_k - y_l\|^2)^{-1}} \tag{14}$$

Kullback-Leiber divergence, defined by Equation (15) measures the difference between P (joint probability distribution in the high dimension) and Q (joint probability distribution in the low dimension) distributions in X space, the result as smaller as similar they are. This makes the map's representation of joint probabilities (almost) invariant to changes in the scale of the map for map points that are far apart.

$$D_{KL}(P \parallel Q) = \sum_{x \in X} P(x) \log\left(\frac{P(x)}{Q(x)}\right) \tag{15}$$

The gradient of the Kullback-Leiber divergence between P and the Student-t based joint probability distribution Q is given by Equation (16):

$$\frac{\delta C}{\delta y_i} = 4 \sum_j (p_{ij} - q_{ij})(y_i - y_j) \left(1 + \|y_i - y_j\|^2\right)^{-1} \tag{16}$$

Where the cost function applies gradient reduction to the low dimensional dataset, so that its distribution resembles the one on the original.

4. RESULTS

4.1 MTCNN

4.1.1 Implementation Results

The first presented result refers to the output implementations of the MTCNN framework using the computers camera, presented in the Stage Cascaded Network and Non maximum suppression section. With an image-pyramid scale factor of 0.709, as well as a [0.6;0.7;0.7] detector threshold for the bounding box IoU for the 3 stages.

The P-Net flagged all face proposals with different kernel sizes, using the image pyramid, generating numerous bounding boxes. Afterwards, those proposals were filtered by non-maximum suppression, ready to be inputted into the next network. The P-Net final output with NMS is shown in

Figure 58.(a).

Next, the R-Net focuses on filtering a high number of false positives. It used the previous network's proposals data to generate more precise bounding boxes, with all the lower confident candidates being eliminated. The boxes are then calibrated to have a square-like shape. The R-Net output is presented in

Figure 58.(b). By using NMS again, the bounding boxes are reduced, merging overlapped proposals, being calibrated and padded to have a squarer like shape, as shown in Figure 58 - Output bounding boxes of MTCNN framework. (a) P-Net output with NMS. (b) R-Net output. (c) R-Net output after NMS (d) O-Net output (e) O-Net output after NMS (f) Final Cropped face.(c). Finally, the O-Net standardizes both the bounding box and facial landmarks coordinates,

Figure 58.(d). After the NMS, only the bounding box prediction with the highest confidence level is provided, as shown in

Figure 58.(e). The final output of the MTCNN is the cropped imaged from the final bounding box prediction represented in

Figure 58.(f). This image is a representation of the data used for training/inference the Inception-ResNet. The final output is a 160 x 160 image. After the implementation result, more tests were done on public images on the internet, as well as real-time tests regarding different variables that could risk the viability of the framework. The results signalized detected faces in the frames, with the bounding box coordinates serving as measurements. For a better visualization, the bounding box was outlined on the faces in the images. After, the MTCNN aligned and cropped faces in a dataset in an equal configuration to the necessary input settings for the Inception-ResNet CNN.

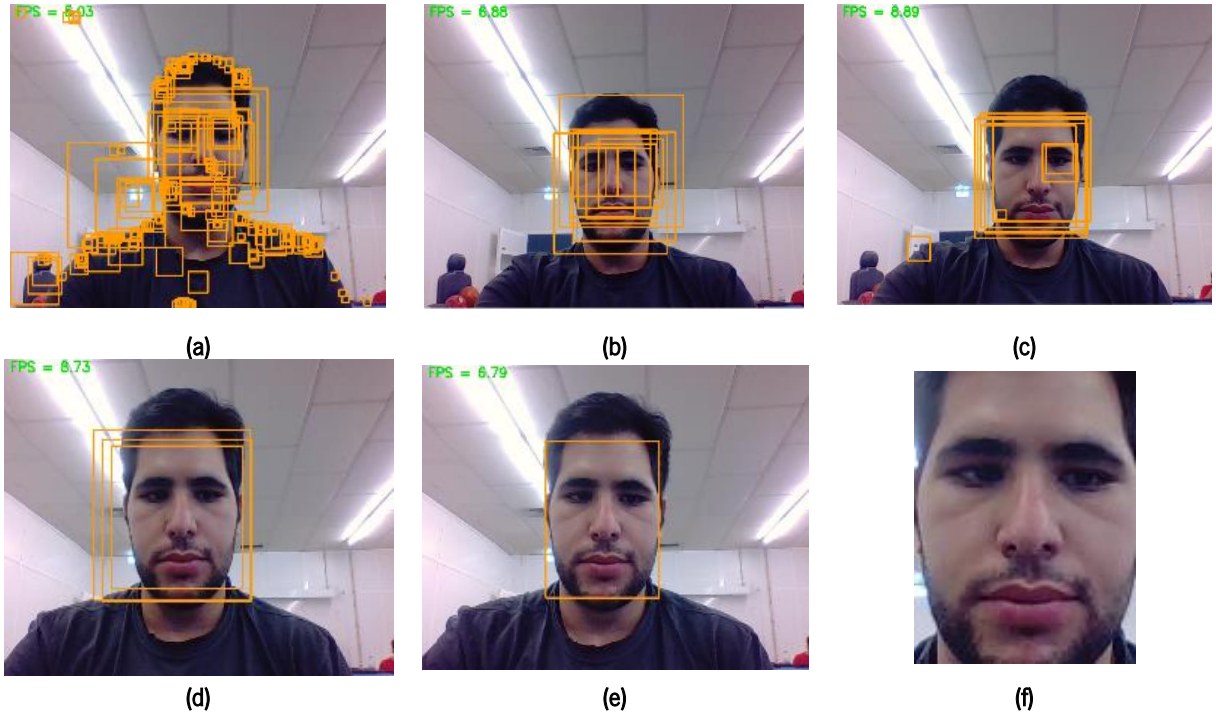


Figure 58 - Output bounding boxes of MTCNN framework. (a) P-Net output with NMS. (b) R-Net output. (c) R-Net output after NMS (d) O-Net output (e) O-Net output after NMS (f) Final Cropped face

4.1.2 Tests on Images and videos

The face detector was tested on different crowded scenarios, given that these configurations provided a large spectrum of faces for real world applications. This execution gave the model verification tools for many frameworks, with samples containing different face sizes, blurred faces, pose variation, lighting changes or localization on the frame on images and videos.

With an image-pyramid scale factor of 0.709, proposed by the authors (K. Zhang et al., 2016), as well as a [0.6;0.7;0.7] threshold array, different size faces are equally detected. Figure 59 showcases an image from RoboParty 2019 with a high people density with tiny face sizes found in different spatial locations, successfully detecting the faces with some minor exceptions where people are considerably not facing the camera.

Figure 59 - MTCNN output bounding boxes in a highly populated scenario with small size faces

Figure 60 showcases the same premise, where a highly populated scene is presented, but with considerably bigger targets. With the same scalar factor and threshold, MTCNN detected most of the faces. That image also presents several face pose variations detected. This included faced up faces, sideways perspectives, or different facial expressions. Although many of the faces are recognized, blurred faces in that frame challenged the detector. Hence, the 3-stage model thresholds were adjusted to compare the performance gain on difficult candidates. By tuning the threshold array to [0.5, 0.6; 0.6], Figure 61 shows the detection pipeline increased its performance on challenging cases on the same target. The drawback of that decision was the appearance of false positives, damaging the model's accuracy.

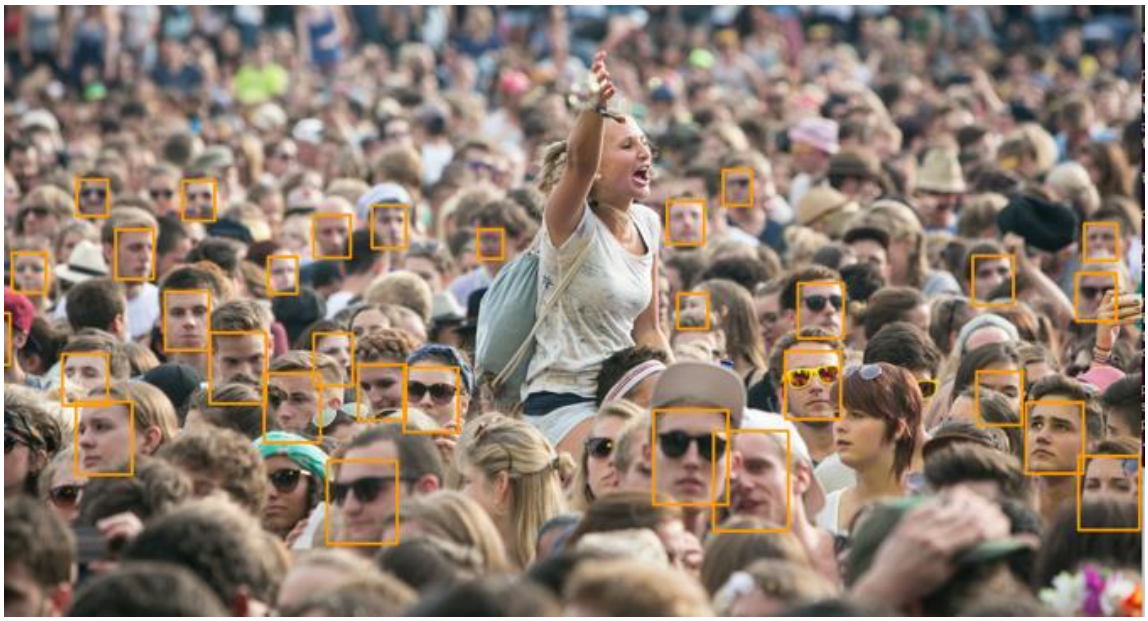


Figure 60 - MTCNN bounding box output Mob image showcasing different detected face pose variations with [0.6,0.7,0,7] threshold array. Photo by Michael Dornbierer covered by Creative Commons license

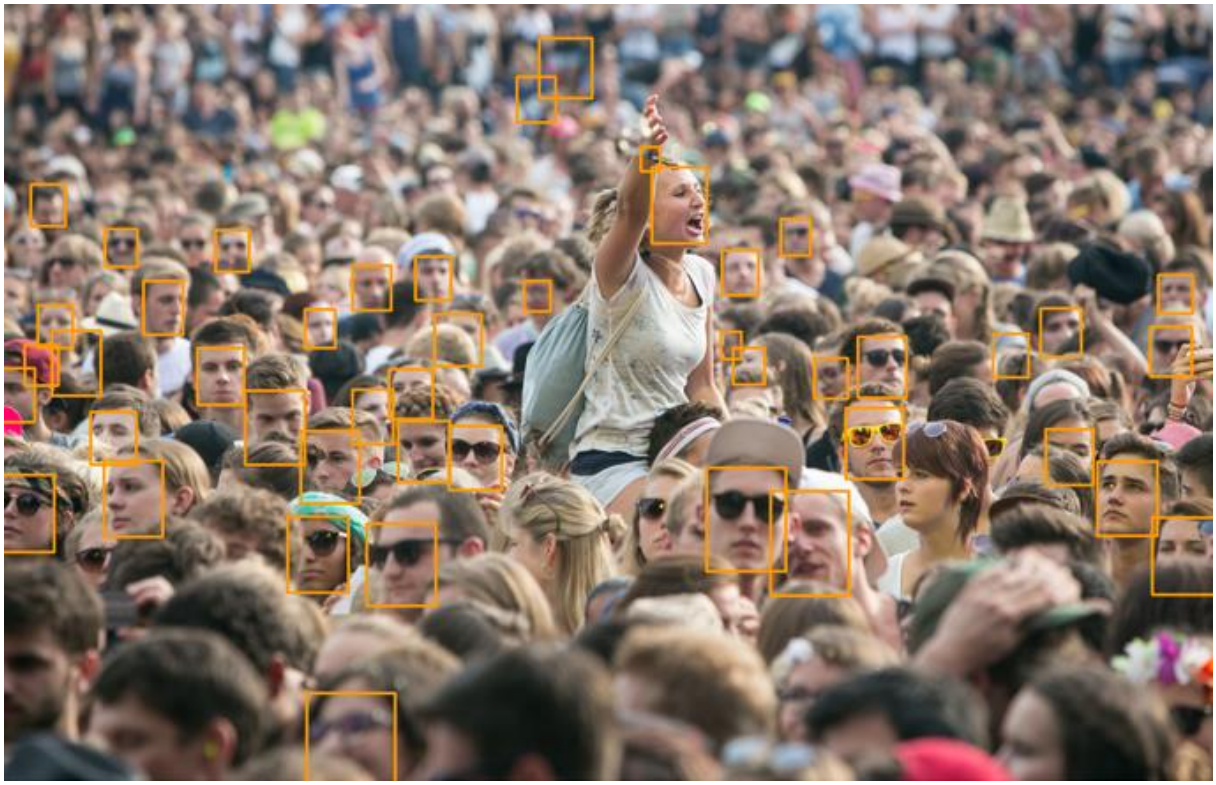


Figure 61 - – MTCNN bounding box output of the same frame (Figure 60)) using [0.5,0.6,0,6] threshold array. Photo by Michael Dornbierer covered by Creative Commons license

The same way, harder tests were also carried out regarding variables that possibly could affect the model's performance on real-world problems, such as blurred faces, lighting variations and 'disguise' or harder face characterization.

Figure 62 presents a situation where background people had their faces blurred. While some of the faces were detected, difficult samples were left unrecognized.

Regarding Illumination problems, the MTCNN was applied to images with low lighting backgrounds. Figure 63.(a) and Figure 64.(a) showcase bad performances regarding MTCNN bounding box output on a low illumination backgrounds in shrouded faces. By tunning the thresholds into the same configuration presented above, the threshold array [0.5, 0.6; 0.6] increased the framework's performance, as seen in Figure 63.(b) and Figure 64b). However, those results were still lackluster. By lowering the threshold array again to [0.4, 0.5; 0.5], the model's performance could not detect all the faces presented. Instead, it started to detect false positives on the same previously recognized targets (Figure 63.(c) Figure 64.(c)). The blurred and lighting tests proven to be the most demanding out of the model's performance. By lowering the thresholds necessary to detect the faces, the performance increased slightly, although some false positives were also being detected.

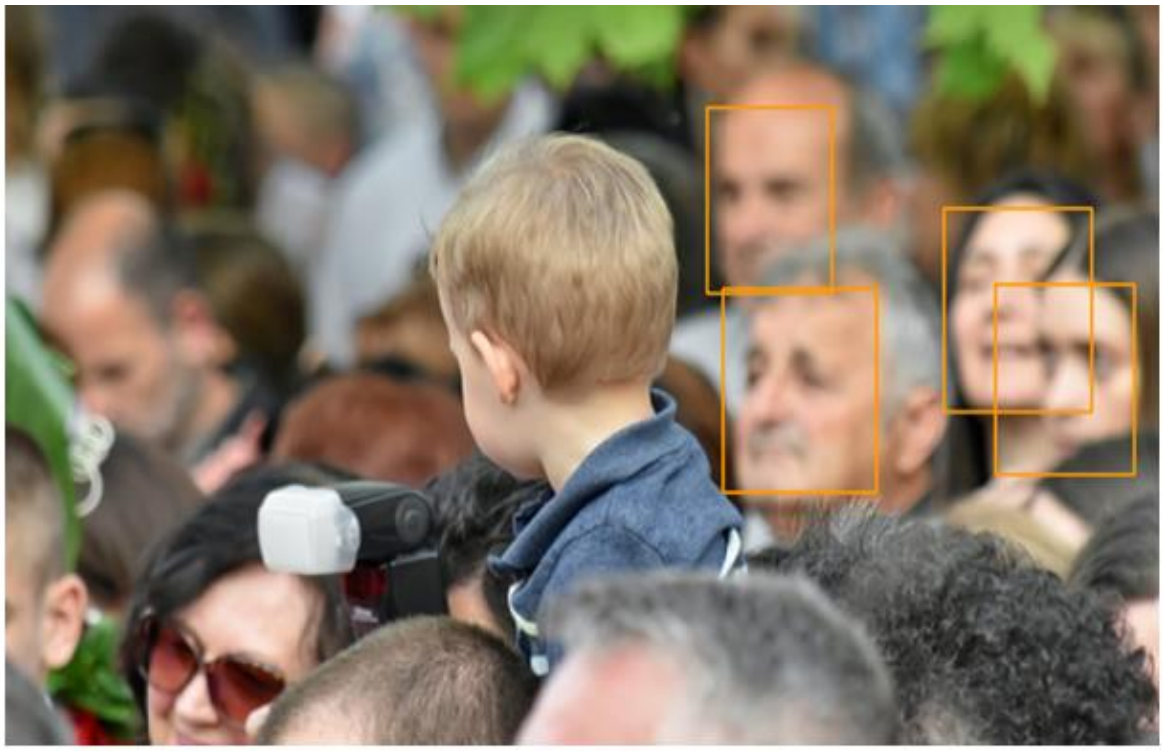


Figure 62 - Bounding box output on images containing blurred faces. Photo covered by Creative Commons license

Figure 63 -MTCNN bounding box output comparison on a low illumination background with different threshold arrays (1). (a) - [0.6, 0.7; 0.7]; (b) - [0.5, 0.6; 0.6]; (c) - [0.4, 0.5; 0.5]

Figure 64 - MTCNN bounding box output on comparison on a low illumination background with different threshold arrays (2). (a) - [0.6, 0.7; 0.7]; (b) - [0.5, 0.6; 0.6]; (c) - [0.4, 0.5; 0.5]. Photo under Creative Commons License

'Disguise' tests were also done on images. Those contained harder faces to characterize. On Figure 65, another real-world situation is presented, containing people wearing glasses and caps. Figure 65.(a) presents the [0.6, 0.7; 0.7] initial threshold array output, detecting the clearer candidates, with

Figure 65.(b) showcasing the [0.5, 0.6; 0.6] output having a better performance. Although neither output can detect all the targets, most of the explicit faces were recognized. Given the current world's pandemic, images with people wearing masks were also target tested (Figure 66). The model usually can easily detect faces in those settings, as long as the rest of the facial landmarks were not hidden.

Figure 65 – MTCNN output on harder faces to characterize with different threshold arrays. (a) - [0.6, 0.7; 0.7]; (b) - [0.5, 0.6; 0.6]. Photo covered by Creative Commons license

Figure 66 - MTCNN bounding box results on an image of people wearing masks. Royalty Free photo taken online

4.1.3 Camara tests

The same experiments carried out on Tests on Images and videos were applied to real time face detection, using the computer's camera.

The first test done was the number of frames possible using the MTCNN. Being computationally heavy, with the cascaded network, the fps can fluctuate between 10 to 18 fps, depending on other processes being used in parallel.

Lighting tests were done using the computer camera with different illumination settings. On Figure 67, different lighting deviations are presented on the same target environment. Having a target person presented in front of the camera, the model manages to identify the face on 3 illumination settings easily.

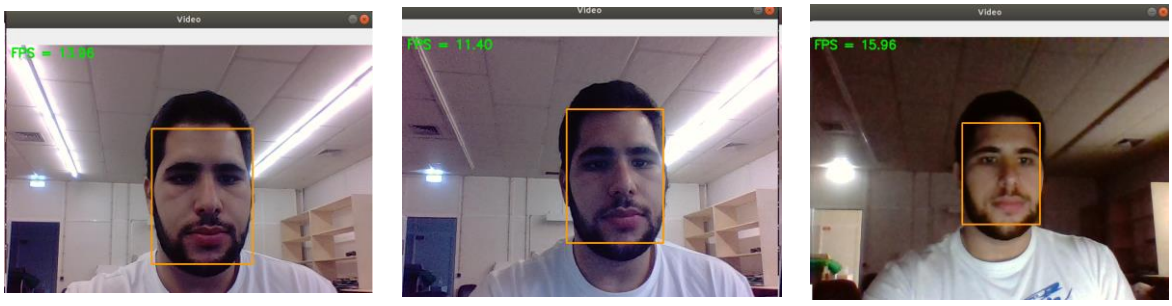


Figure 67 - MTCNN bounding box output on different face poses detected with the camara

Another important feature regarding face detection tested was the different face pose scenarios in real world applications. Various poses in front of the camera were carried, with Figure 68 example demonstrating different face-detection perspectives.

Some tests were also done on people wearing masks, since its use became a standard variable in day-to-day activities. The face detector managed to accurately identify the faces, failing in some frames, accusing some uncertainty. Two of those poses recorded are showcased in Figure 69.

Figure 68- Face detection using wiht computer's camara for diferent lightin settings. (a)- Bright enviroment; (b)- Intermediate lighting; (c) Sub-par ilumination

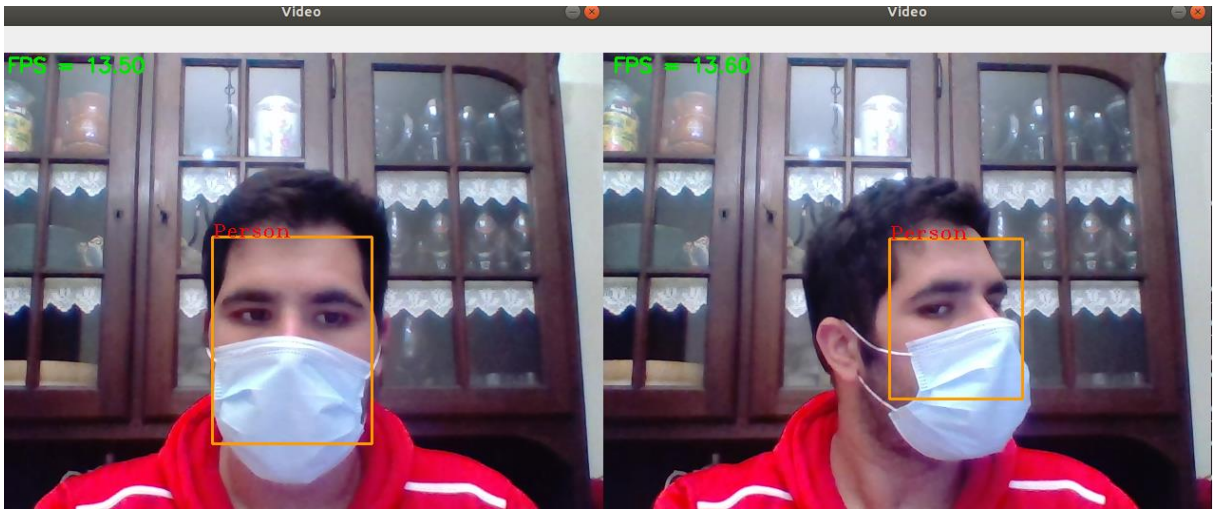
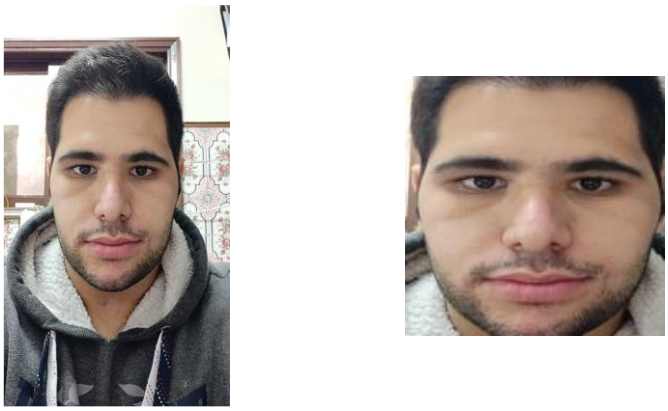


Figure 69 - Face detection using the computer's camara on masked faces

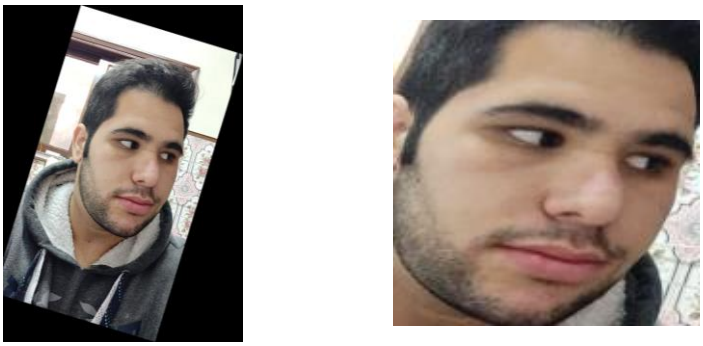
4.1.4 LAR dataset

The custom-built LAR dataset had 19 classes with 300 data points per identity. That summed up to 5700 images in total, with 4617 of them used for training, 513 for cross-validation and 570 for test at the end of training framework.

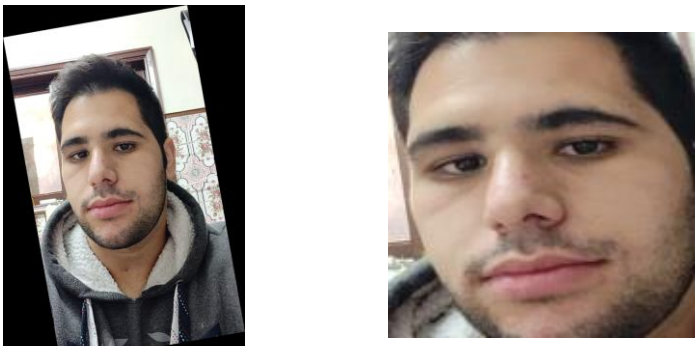
The images were gathered from videos of the individual identities, creating the pre-processed dataset. Before inputting the images into the face detector, random transformations were done on the images. Those allowed for more variety in the data intended to train the model. With that, the classifier was more prone to identifying new samples correctly in the inference part of the model. The methods used were random rotations and horizontal flips, with an example of those results being showcased in the left column of Figure 70.(b), Figure 70.(c) and Figure 70.(d). After the MTCNN output, the dataset images only contained the cropped and aligned faces in a 160 x 160 dimension setting. Those are showcased in the right column of Figure 70.



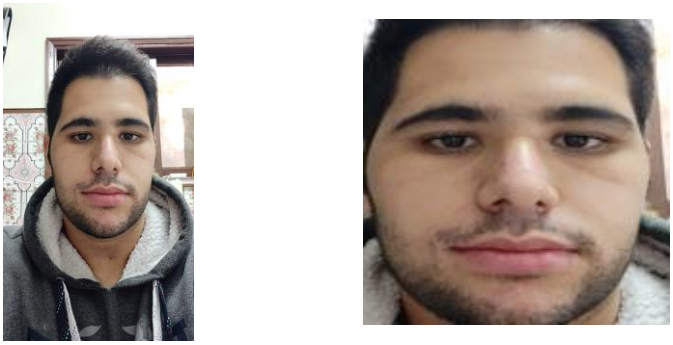
(a)



(b)



(c)



(d)

Figure 70 – Image transformations from frames gathered from videos form people in the dataset and respective MTCNN detector and alignment output; (a) - No transformation; (b) – Random rotation(>0°); (c) - Random rotation(<0°); (d) – Horizontal flip.

4.1.5 Face Detection Conclusion

Regarding the Implementation Results section, the MTCNN pipeline was easy to implement in real-time settings. By loading the weights and bias from transfer learning from the previously trained network by ("GitHub - davidsandberg/facenet: Face recognition using Tensorflow," n.d.), large part of the implementation was skipped. With those, the model was able to easily detect faces on images, recorded videos and in real-time video in standard conditions. The proposed variable settings displayed in Figure 35 were adequate pursuing a good framework accuracy when not exposed to extreme disadvantage situations.

On Tests on Images and videos section, tests with different number of targets, like the ones on Figure 59 and Figure 60, proved that the framework's performance was not affected by the number of faces to be detected. Those tests also demonstrated that different candidate's size faces, simulating how far/near the viewpoint was, were equally detected, proving the efficiency of the image pyramid algorithm.

MTCNN was also proven capable of identifying a face in various pose scenarios in real world environments, with different analysis being done such as the one in Figure 60 or Figure 65. Although the cascaded CNN was still not able to fully recognize all targets, those whom the model could not identify had a lot of noise in their surroundings or adverse conditions.

To better understand the model's flaws, more tests were done on blurred images, bad lighting or 'disguise'. Those proven very challenging for the detector framework when facial landmarks were not exposed or in bad detecting conditions. By lowering the threshold array of the 3 stages, more proposals marked as faces, passing through the NMS algorithm, as seen in Figure 61, Figure 63 or Figure 65. That allowed the MTCNN to point out more faces in harder situations. One drawback of this practice was that even when lowering the thresholds, still not all the faces were detected. The other drawback was that false positives were more frequent (Figure 63.(c) and Figure 64.(c)), costing the models accuracy.

When implementing the model with the camera for real-time settings, the model reached a constant frame count of 10 to 18 fps. That made the model reliable in real time configurations. The same tests were carried out using the camera in standard environments. Those verified good detection results in different distances from the camera. The model also easily detects different faces pose variations in different illumination settings.

When building a dataset for face recognition, the samples are usually the cropped face of the class identity, with the least background possible. The MTCNN pipeline did that by cropping and aligning the face of the people to be inserted in the custom dataset of users from LAR. Those data points were obtained in the frames directly fetched from videos done individually. Due to Covid restrictions, this approach was a simple method to build the dataset, without the acquisition of many images. To make the system more robust to new samples of data, image transformations were implemented on random frames (Figure 70). Those improved the classification of new samples in the inference part of the work. The reason for this framework decision was the motivation of the dissertation. Since the premise was creating a more suitable HRC, real world situations needed to be kept in mind. With more diverse data fed into the classifier, the model was able to recognize identities in various scenarios other than looking directly forward.

4.2 Inception ResNet Training

For the Inception-ResNet training, both versions v1 and v2, important hyperparameters were studied to tune the networks to achieve maximum accuracy and minimum loss. Those included the tunning of the batch size, number of mini batches epoch, lr and dropout. The results of training were uploaded into the TensorBoard feature, visualizing the parameters convergence as training went by. For every train done, 3 graphs were plotted and analyzed: train accuracy, loss scalars and cross-validation scalar. Both trained frameworks used ADAM optimizer for gradient descent. The prelogit normalization loss factor for batch normalization was set to 0.005, as well as L2 weight decay and L1 norm loss activation.

The batches were divided into a set of mini-batches, for 2 reasons: computer resources and mini-batch gradient descent. Since the computer's resources were limited, the GPU had difficulty allocating enough memory to train big batches of data, necessary when training a deep neural network such as Inception-ResNet model. Mini-batch gradient descent reduced the variance of the gradients when calculating the error and updating the coefficients. Figure 71 demonstrates how the model's training was supervised using the Ubuntu's terminal.

```
Epoch:[378/500]        Time 116.204    Loss 3.127      Xent 1.435      Accuracy 0.823  Lr 0.00500
Train time : 116
Epoch:[379/500]        Time 116.911    Loss 3.076      Xent 1.385      Accuracy 0.838  Lr 0.00500
Train time : 116
Epoch:[380/500]        Time 116.082    Loss 3.128      Xent 1.437      Accuracy 0.825  Lr 0.00500
Train time : 116
Running forward pass on validation set
Validation Epoch: 380  Time 868.190    Loss 3.632      Xent 1.943      Accuracy 0.734
Epoch:[381/500]        Time 115.785    Loss 3.170      Xent 1.481      Accuracy 0.823  Lr 0.00500
Train time : 115
```

Figure 71 - Terminal output example for supervision when training Inception-ResNet models

The results presented in this document on developed work in the CASIA-Webface dataset were made using only version 1 on the Inception-ResNet. Since the version 2 presented a much more expensive architecture topology, the computer was not able to allocate enough memory to present good results (Figure 72). Since the custom dataset presented a much smaller number of dataset classes, a of both version results were used and compared.

```
2021-07-29 11:12:19.586639: I tensorflow/stream_executor/platform/default/dso_loader.cc:44] Successfully opened dynamic library libcudnn.so.7
2021-07-29 11:12:20.723316: W tensorflow/core/common_runtime/bfc_allocator.cc:239] Allocator (GPU_0_bfc) ran out of memory trying to allocate 2.04GiB with freed_by_count=0. The caller indicates that this
is not a failure, but may mean that there could be performance gains if more memory were available.
2021-07-29 11:12:20.818584: W tensorflow/core/common_runtime/bfc_allocator.cc:239] Allocator (GPU_0_bfc) ran out of memory trying to allocate 2.11GiB with freed_by_count=0. The caller indicates that this
is not a failure, but may mean that there could be performance gains if more memory were available.
2021-07-29 11:12:20.826780: W tensorflow/core/common_runtime/bfc_allocator.cc:239] Allocator (GPU_0_bfc) ran out of memory trying to allocate 2.02GiB with freed_by_count=0. The caller indicates that this
is not a failure, but may mean that there could be performance gains if more memory were available.
 Epoch:[1/700]  Time 60.283     Loss 24.601     Xent 9.279      Accuracy 0.000  Lr 0.05000
 Train time : 60
Running forward pass on validation set
2021-07-29 11:12:30.617662: W tensorflow/core/common_runtime/bfc_allocator.cc:305] Garbage collection: deallocate free memory regions (i.e., allocations) so that we can re-allocate a larger region to avoi
d OOM due to memory fragmentation. If you see this message frequently, you are running near the threshold of the available device memory and re-allocation may incur great performance overhead. You may try
 smaller batch sizes to observe the performance impact. Set TF_ENABLE_GPU_GARBAGE_COLLECTION=false if you'd like to disable this feature.
2021-07-29 11:12:30.676898: I tensorflow/stream_executor/cuda/cuda_driver.cc:831] failed to allocate 3.94G (4227137536 bytes) from device: CUDA_ERROR_OUT_OF_MEMORY: out of memory
2021-07-29 11:12:30.678295: I tensorflow/stream_executor/cuda/cuda_driver.cc:831] failed to allocate 3.54G (3804423680 bytes) from device: CUDA_ERROR_OUT_OF_MEMORY: out of memory
Validation Epoch: 1     Time 500.917    Loss 24.403     Xent 9.266      Accuracy 0.001
```

Figure 72 - CUDA error when trying to allocate large sets of data

4.2.1 CASIA-Webface dataset

The first hyperparameter test and tunning on the Inception-ResNet v1 CNN models trained with CASIA-Webface dataset was the training lr. For that, the same CNN was trained, with different learning rates. That network had a 5000-batch size (50 batch in 100 mini batches), trained during 300 epochs with a 40% dropout. The framework results are showcased in Figure 73.

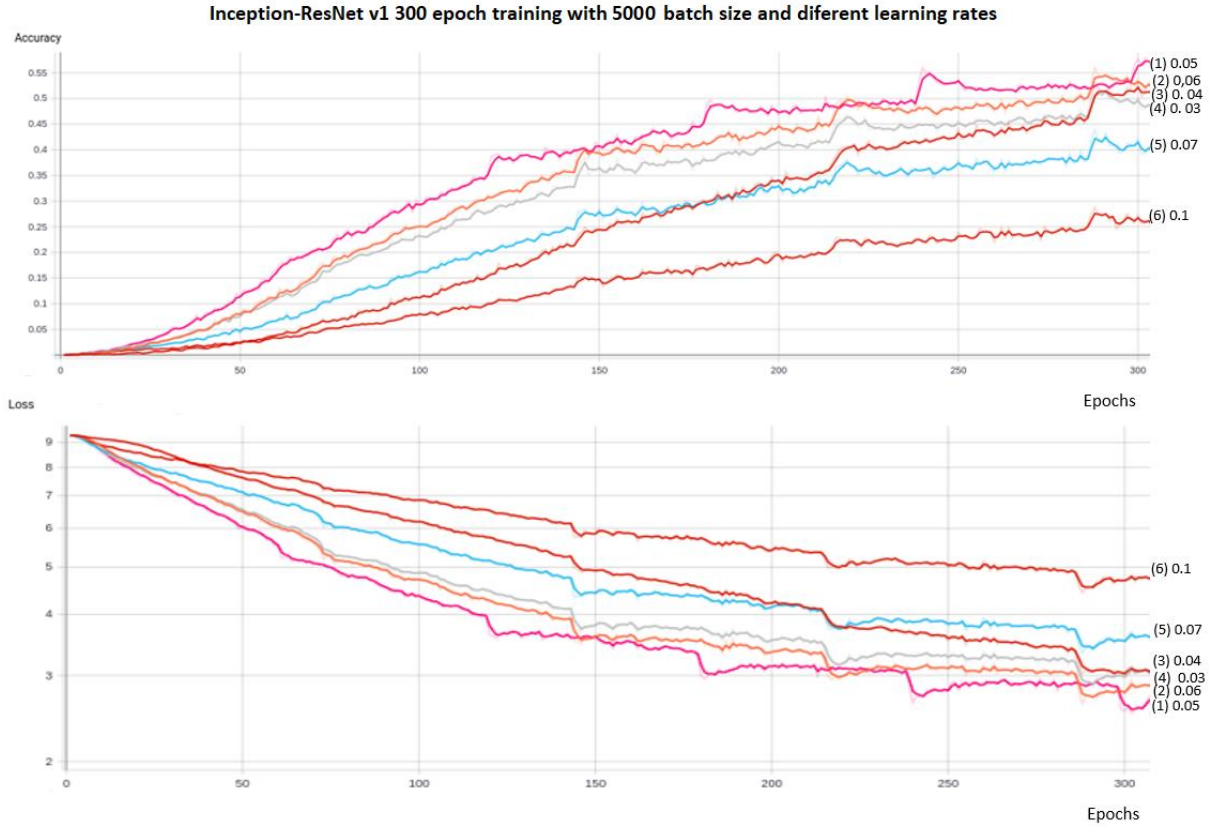


Figure 73 - Train accuracy and loss plots using different Learning Rate on the same CNN configuration

The 0.1 and 0.07 learning rates, although bigger, presented the worst accuracy results and bigger loss. Between 0.03 and 0.06 the CNN presented a good convergence as time went by, with the 0.05 lr having the highest train accuracy at the time of the 300 epochs. This Inception ResNet v1 framework had a training time of around 9 hours. Although the Inception-ResNet models on Figure 73 trained during 300 epochs, its accuracy was still underfitted. By increasing the epoch training, the accuracy on the models plateau at around 65% accuracy during the 500 epochs, taking 14 hours of training.

The next test was understanding how increasing the batch size would influence the learning process. Figure 74 presents two Inceptions-ResNet v1 models with a 0.04 lr, 40% dropout, trained in 500 epochs using a 6000-image batch.(1) and 5000 image batch.(2). The 6000 batch was the maximum batch size the networks could train without crashing the whole framework.

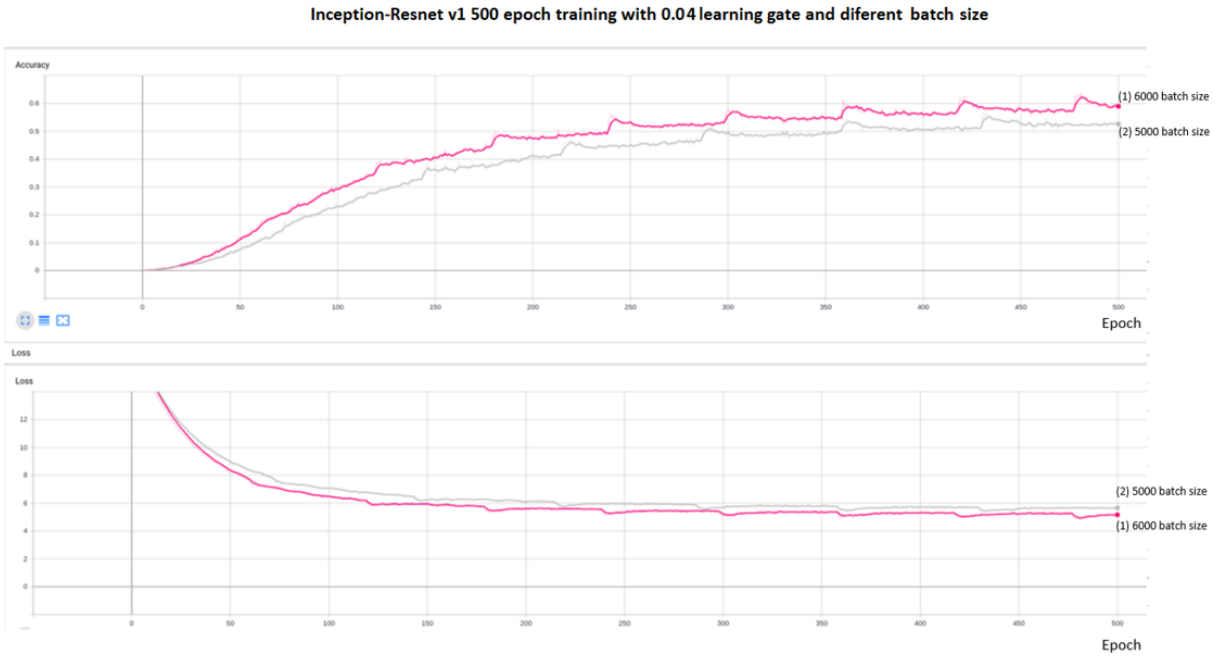


Figure 74 - Accuracy and Loss comparison between the same network with different batch sizes; 1- 6000 train batch size; 2 5000 train batch size

With more 1000 images per batch, the CNNs had the same convergence pattern, with a slightly accuracy gain in the 6000-batch framework. As tradeoff, the 6000-batch network trained for 18 hours while the 5000-batch one only trained for 15 hours and 9 minutes. Even increasing the batch size on different learning rates, the model's accuracy was having difficulties converging.

Since these difficulties involved bad converging status passing a certain number of epochs with a stable lr, learning rate decay technique was applied. The model starts with a large lr, decaying into smaller numbers in a pre-set number of epochs. The technique was added to the model (6) pipeline on Figure

73. Starting at a 0.1, the learning rate decrease to 0.009, decaying 0.01 at 30 epochs. It then continue to drop the same 0.01 amount every 50 epochs, substantially increasing the model's accuracy. The comparison between the models is showcased in Figure 75. After 400 epochs the model went from 29% to 67% accuracy using only the lr decay approach.

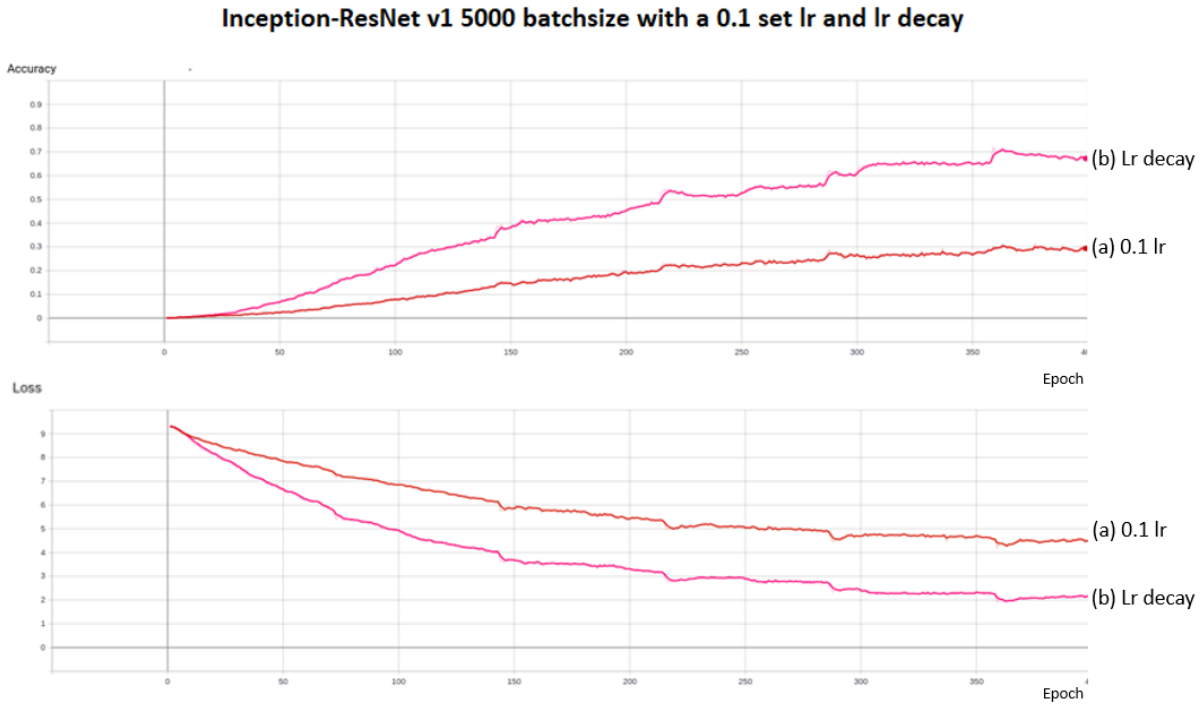


Figure 75 - Accuracy and Loss gain between the same network using learning rate decay on an undefeated network with high learning rate; (a)- Static 0.1 learning rate; (b)- Learning Rate decay starting at 0.1

Regarding the dropout parameter, different instances were tested to verify overfitting problems. The way the overfitting could be verified was to plot the validation subset accuracy, comparing it to the matching train accuracy.

Three models with the same learning rate decay and batch size using different dropout settings were trained and plotted (Figure 76). The network's dropouts were set to 80%, 70% and 40% dropout. The way the overfitting could be verified was to plot the validation subset accuracy when cross-validation is computed, comparing it to the train accuracy showcases the plots.

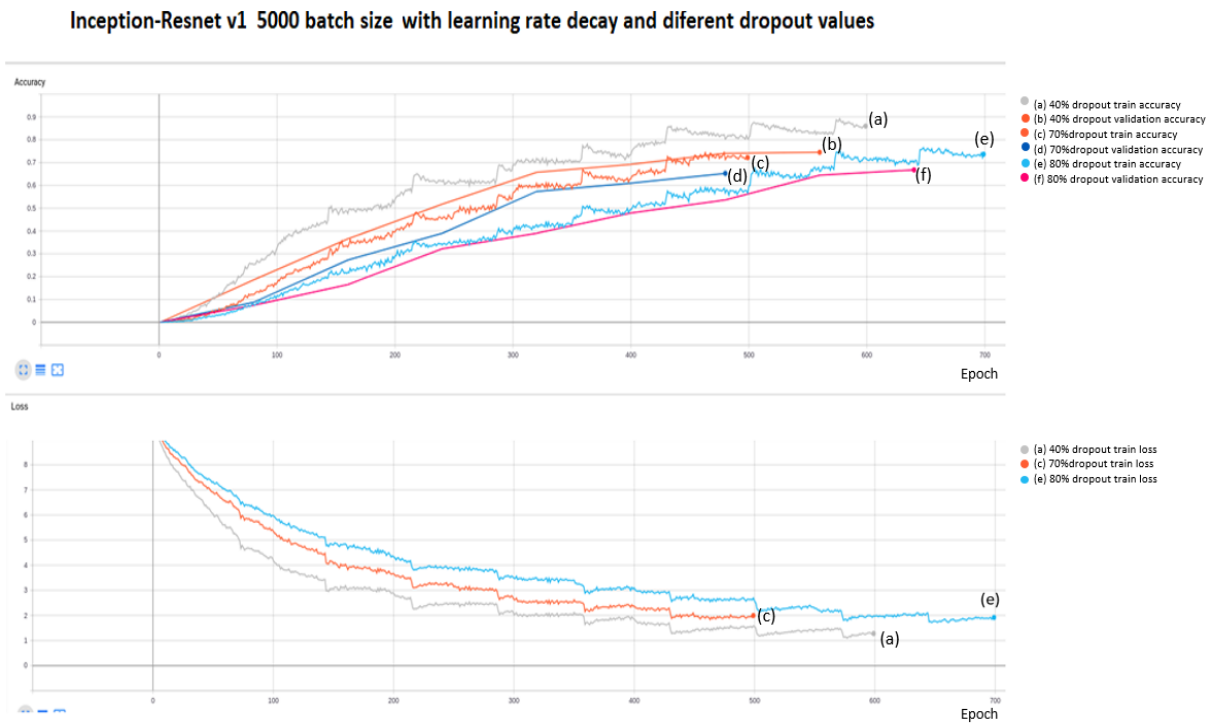


Figure 76 – Train and validation accuracy and loss plots on the same Inception-ResNet model using different dropout settings.

The model with the least amount of dropout had the higher performance Figure 76.(a), as well as the better performing validation performance (Figure 76.(b)). The least accurate model was also the one with the biggest dropout percentage (Figure 76.(e)). Since the 3 modules validation accuracy proved to converge accordingly, lowering the dropout below 40% would only damage its performance.

The best performing Inception-ResNet V1 model had the following topology: 5000 batch size, with a starting lr of 0.6, decaying 5e-2 every 100 epochs, finishing with 0.02 lr at 500 epochs. The dropout rate was set at 20%. It reached 90% train accuracy and 77.5% validation accuracy after 500 epochs, taking approximately 14 hours. The model's convergence over time can be seen in Figure 77.

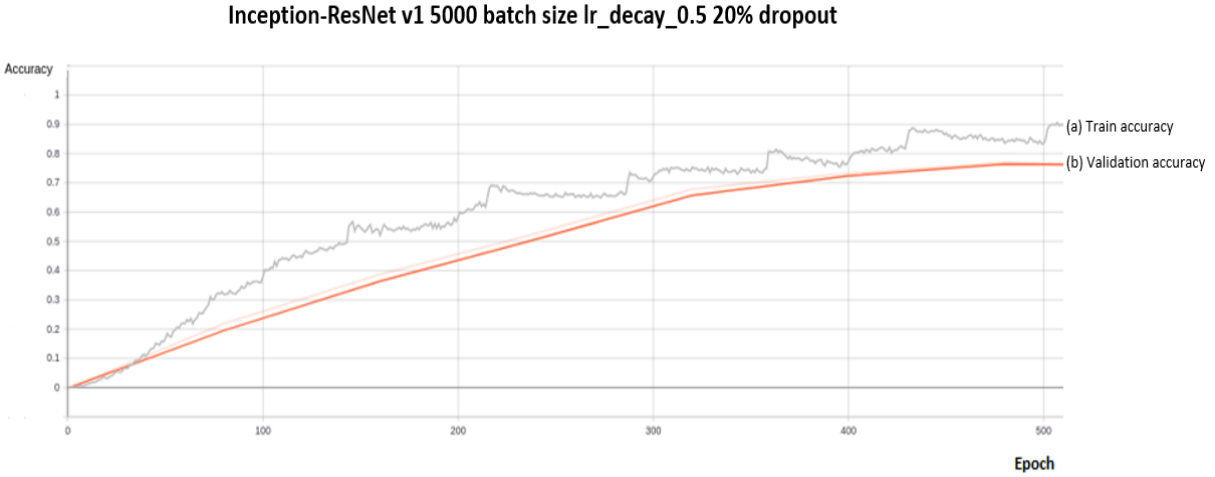


Figure 77 - Best Performing Model using the CASIA-Webface dataset

4.2.2 Custom Dataset

With the custom dataset, CNN was trained with both version v1 and v2. Being such a complex, the biggest problem surrounding the network's train was the overfitting when training with smaller sets of data. As previously stated, that happens when the train accuracy is very high, while not able to correctly classify the outside samples, in other words, the validation subset.

The first test trained the model with a batch size of 45 throughout 100 epochs with a learning rate of 0.01 (Figure 78). It managed to reach 99% train accuracy and only 40% validation accuracy on the v1 version, and 99.4% train accuracy and 60% validation accuracy on v2. Checking the converge over time from the models, both versions demonstrated overfit features, with a high train accuracy and low validation accuracy. Other conclusion was that the v2 version had better accuracy, as stated in the model's literature.

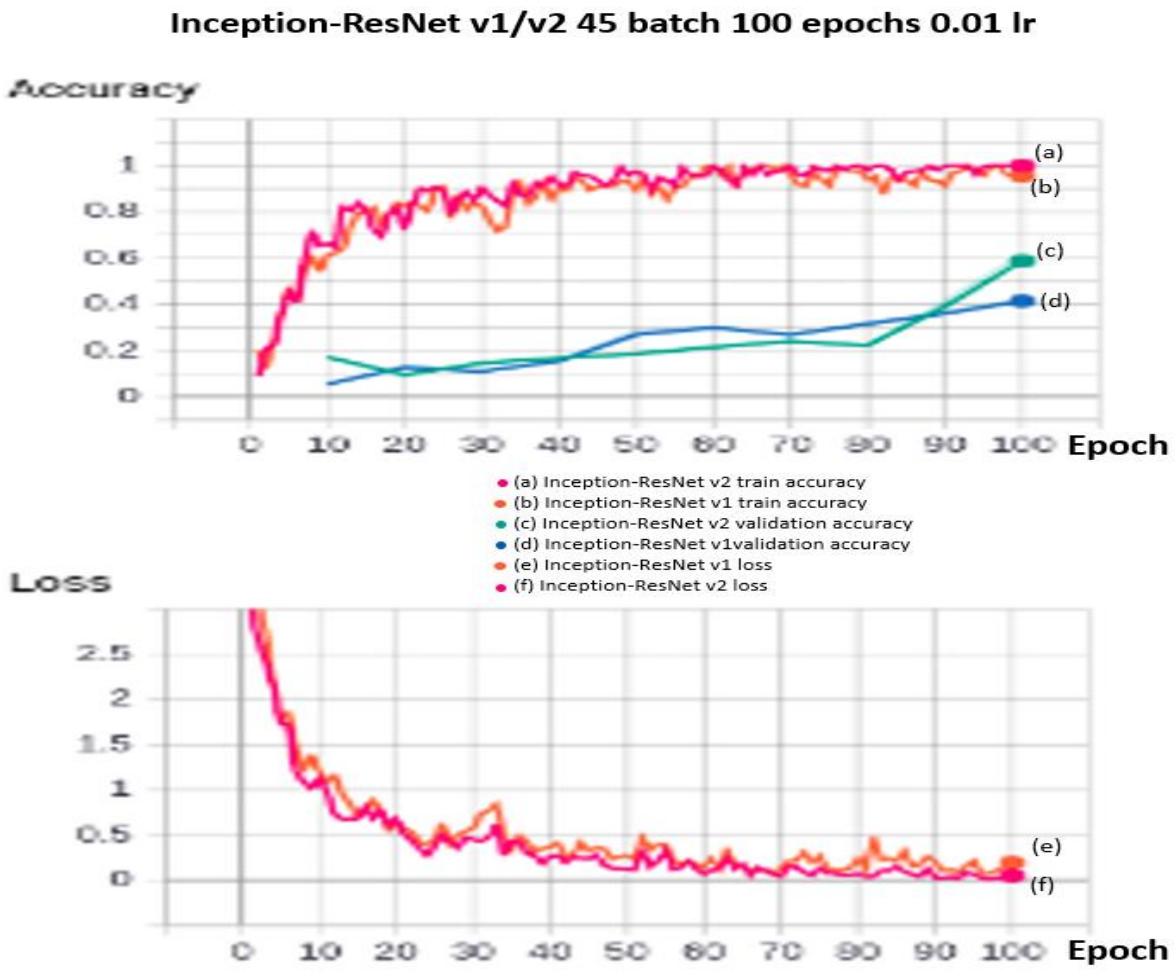


Figure 78 – Same hyperparameters Inception-ResNet v1 and v2 network comparison

Reviewing the data augmentation effect on smaller datasets, the topology on Figure 79 introduced random crops, flips and rotations to the dataset when inputted into the FIFO pipeline. A data augmentation model was plotted in the same graph as the non-augmented counterpart, with a batch size of 40 throughout 150 epochs with a learning rate of 0.05. The dropout was also lowered to 40%. The introduction of data augmentation lowered the accuracy during training but increased during validation of the model. It also did nothing regarding the overfitting problem, since the validations subsets still had problems converging, doing it latter in training when the model already learned the dataset's patterns.

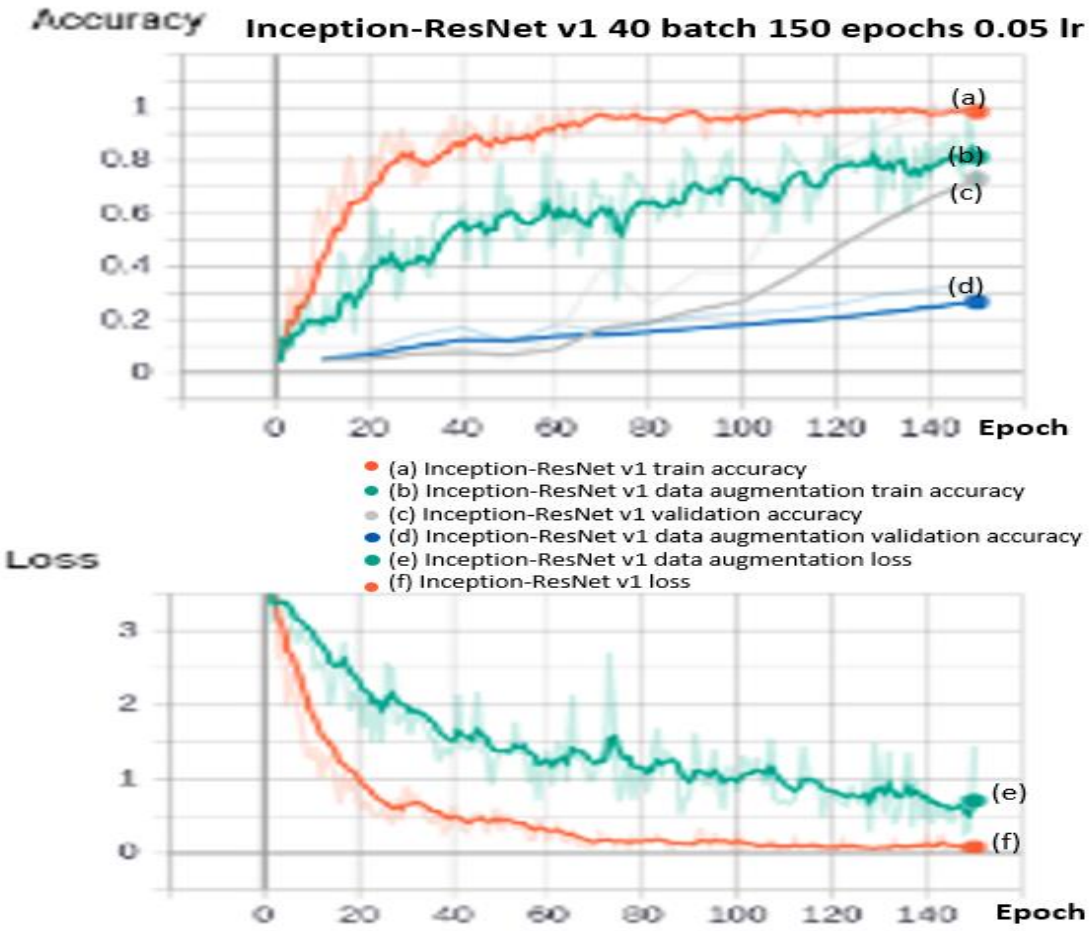


Figure 79 - Data Augmentation effect on an overfitted topology

Learning Rate decay was also studied, but its effect on the different topologies did not bring a consensual conclusion. Due to the high fluctuation introduced by small changes in hyperparameters, the decaying learning rate proved to be an advantageous solution in some cases but in some others could not help the model to converge.

The best performance (Figure 80) was an Inception-ResNet v2 model, with a 60-batch size segregated into 6 mini-batches of 10 images, proven beneficial as already stated and proved in the last section. Also, the lr was dropped to 0.001, extending the training epochs to 400, while the dropout was set at 80%. The model reached 85% train accuracy and 75% validation accuracy.

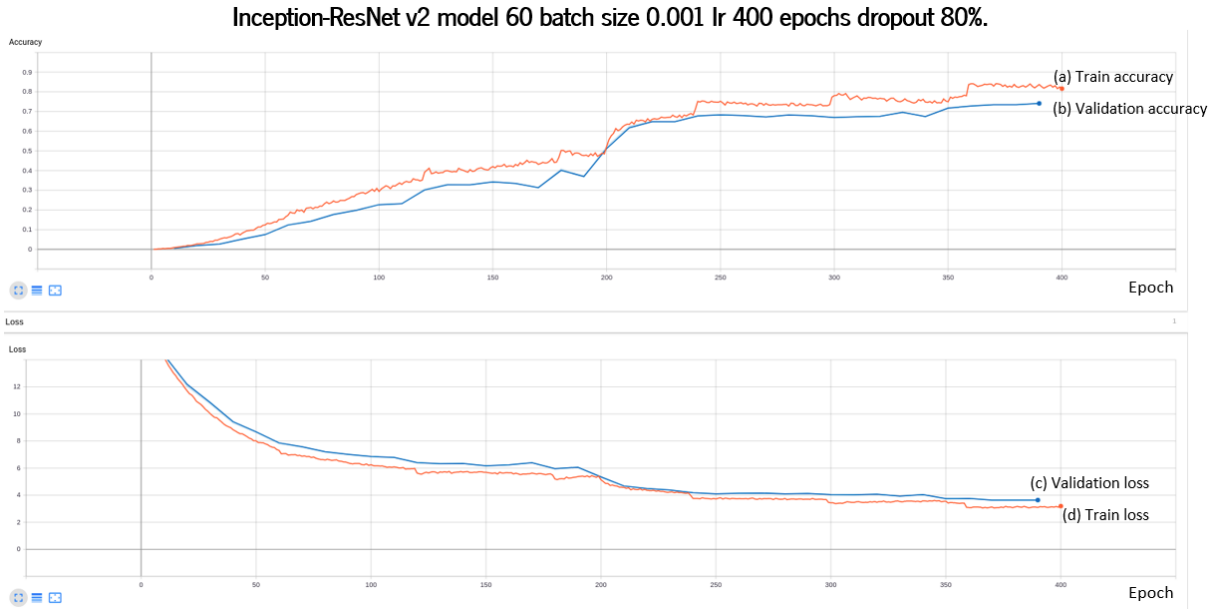


Figure 80 - Best Performing Inception-ResNet model on the LAR dataset

4.2.3 Inception-ResNet Conclusions

In this chapter, the Inception-ResNet CNN was trained and validated with a large dataset and a small dataset. A study to optimize the network's hyperparameters was conducted with the help of the TensorBoard scalar plots, visualizing how the topologies would converge over time. This CNN proved challenging to optimize, given the number of gradients and parameters presented in its architecture.

Regarding the CASIA-Webface dataset, the batch size had to be segregated into mini-batches. That way, more images could be fed into the CNN in one epoch. That was of immense importance, allowing the model to continuously variate the gradients into one epoch. When trying to train the model without mini-batches, the accuracy performance proved lackluster. The maximum number of images the computer manage to process in a batch was 6000. However, when compared to smaller batches, like the standard 5000-image batch, proven efficient, the accuracy performance would slightly increase, but producing a less rewarding time-consuming framework (Figure 74).

The ideal lr to initiate the training turned out to be between 0.03 and 0.06. Those could start the model's convergence whereas bigger values could not start the training process. Smaller values would work but would produce worse performance results (Figure 73).

Learning rate decay was essential for the model's execution (Figure 75). By slowly decaying the learn metric, the model was slowly adjusting by its patterns , whereas with a stable lr the Inception-ResNet topologies would have a very difficult time generalizing the learning process.

Overfitting of the model turned out to not be an issue over the large dataset. By tuning down the dropout percentage, the model would have a lower accuracy and validation (Figure 76).

The best performing Inception ResNet v1 model reached almost 90% train accuracy and 77.5% validation accuracy after 500 epochs. Since the dataset produced such a challenging classification framework, reaching that accuracy performance proved that a robust facial classifier to be used as feature extractor was accomplished.

Regarding the LAR dataset, the biggest problem when training was the constant overfitting, even with low dropout. Inception- ResNet v2 was clearly better performing topology in an accuracy standpoint but would be much more expensive to train (Figure 78).

Data augmentation managed to create more samples of each class, substantially increasing validation accuracy (Figure 79). For models that would converge naturally, the method t proved to be efficient on overfitted models but not so meaningful on models that were already predicting with high accuracy

When training with such small dataset in a bigger CNN, the best performing models had to divide the batch into mini batches, with ideal image batch size rounded between 45 and 60. The best performing topologies also had a lower lr, between 0.005 a 0. 001, trained for more epochs than the first approaches. (Figure 80). Overall, this topology proved very challenging when training with smaller datasets, since its performance was optimized for bigger datasets regarding classification.

4.3 SVM Classifier

The SVM machine learning method was trained and verified with the LFW and LAR dataset using a pretrained. After, the samples were loaded into the TensorBoard Projector, for the PCA and t-SNE analysis visualization. Also, the test Confusion Matrix was projected, checking where the model misclassified the samples.

First, the dataset was split into 2 subsets, depending on the mode input used in the script (Figure 54). The train subset was used to train the classifier and the test subset was used to verify the accuracy on new samples of data, as seen below on Figure 81. It showcases the total number of classes and images the dataset had. It also gives two class examples with an 80%/20% for train/validation of the LAR dataset.

Figure 81 – SVM dataset split example for training and validation using LAR dataset classes

4.3.1 Train

To train the SVM, the train dataset images had to be encoded into a feature vector. For that, a metafile and checkpoint of a previously trained Inception-ResNet CNN was needed, as seen in Figure 82.

```
Loading feature extraction model
Model directory: /home/nuno/facenet/models/facenet/inception_model_CASIA/v1/ince
ption_resnet_512_emb_casia_dataset_500_batch_100esize_500epo_nodataaug_0.1
Metagraph file: model-inception_resnet_512_emb_casia_dataset_500_batch_100esize_
500epo_nodataaug_0.1.meta
Checkpoint file: model-inception_resnet_512_emb_casia_dataset_500_batch_100esize
_500epo_nodataaug_0.1.ckpt-400
```

Figure 82 - - Loading of the pre-trained Inception-ResNet model to encode the images into embedding arrays

For a metric comparison, 4 pretrained models presented on Inception ResNet Training section were used, described in Table 10 Those were strategically chosen to verify how a bad feature extractor could affect the classifier performance in comparison with better counterparts.

Also, different sizes of the train dataset were tried out. That way, a study regarding the number of images necessary to obtain a good classifier performance was accomplished.

Table 10 - Description of the pre-trained models to use as feature extractor

	Inception-ResNet model	*Dataset used for training*	*Description*
Model 1	Figure 79.(b)	LAR dataset	Overfitted model
Model 2	Figure 80	LAR dataset	Best performing model using LAR dataset
Model 3	Figure 76.(b)	CASIA-Webface	Underfitted model
Model 4	Figure 77	CASIA-Webface	Best performing model using CASIA - Webface dataset

The embedding vectors are inputted in batches into the classifier. Varying from the number of images used, the train time was neglectable, always under 2 minutes. After it is completed, those classifiers were stored into pickle file directory, to latter be loaded to categorize new samples (Figure 83).

```
Training classifier
Saved classifier model to file "/home/nuno/facenet/models/facenet/Classifiers/ne
w_lfw_test.pkl"
```

Figure 83 - Saving of the Classifier model in a Pickle serialization file into a directory

4.3.2 SVM Results

Three train/ test splits were performed on the 4 selected models. These were carried out to find the minimum number of images necessary for the model to have a good performance. After training, the results were done by loading the previously trained classifier file from the pickle directory (Figure 84) and the matching feature extractor model used in train.

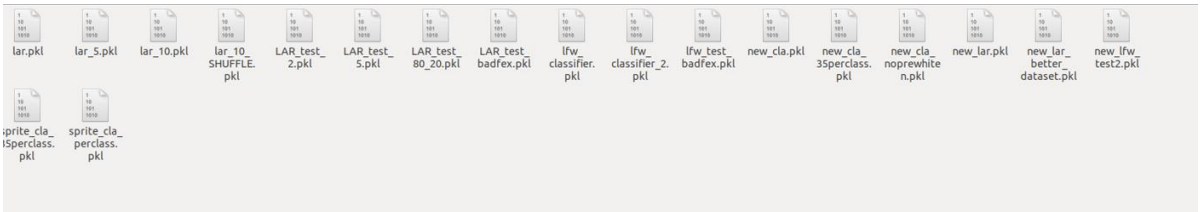


Figure 84 - Classifier storage directory

The model classified new samples of data by outputting the class probability array, similar to the SoftMax function. The Predicted class was the Argmax position of the array, with the Probability being the confidence percentage. Those are then printed in the terminal, as showcased in the example of predicted LFW classes in Figure 85.

```
522 Predicted:Tony Blair || Probability:0.843 || Label: Tony Blair
523 Predicted:Tony Blair || Probability:0.737 || Label: Tony Blair
524 Predicted:Tony Blair || Probability:0.843 || Label: Tony Blair
525 Predicted:Tony Blair || Probability:0.838 || Label: Tony Blair
526 Predicted:Tony Blair || Probability:0.593 || Label: Tony Blair
527 Predicted:Tony Blair || Probability:0.687 || Label: Tony Blair
528 Predicted:Tony Blair || Probability:0.812 || Label: Tony Blair
529 Predicted:Tony Blair || Probability:0.889 || Label: Tony Blair
530 Predicted:Tony Blair || Probability:0.729 || Label: Tony Blair
531 Predicted:Tony Blair || Probability:0.829 || Label: Tony Blair
```

Figure 85 - Predicted class, confidence, and ground truth of some of the LFW test dataset samples

The first train setting had 5 images per class for training using the 4 feature vector models, with the results on the test subsets over the 4 trained models being presented on Table 11:

Table 11 - Test Accuracy on the 4 Models trained with 5 images per class

5 images for test subset	*Accuracy*	
Dataset	LFW	LAR
Model 1	0.2%	36.6%
Model 2	0.1%	18.1
Model 3	71.8%	63.1%
Model 4	98.5%	90%

This framework had bad accuracy results on the models trained with the LAR dataset. Another thing to notice was that the overfitted model (Model 1) produced better results than Model 2. For the models trained with the CASIA-Webface, the accuracy was lackluster on the underfitted Model 3, but was very high on Model 4. Although Model 4 presented good results on accuracy, the confidence of the samples was low. That was checked when new proposals outside the train/test were misclassified regularly.

The next framework setting had 10 images per class for training. The results on the test subsets over the 4 trained models are presented on Table 12:

Table 12 - Test Accuracy on the 4 Models trained with 10 images per class

10 images for test subset	*Accuracy*	
Dataset	LFW	LAR
Model 1	0.3%	81.9%
Model 2	1.7%	67.5
Model 3	83.4%	97.3%
Model 4	99.7%	99.3%

With 10 images per class, the Inception-ResNet LAR trained models had a decent performance on classifiers trained with the same dataset, but still were not able to produce good results on the LFW dataset classifiers.

The CASIA-Webface trained models produced good accuracy results. When using independent samples from classes of the dataset, some were still prone to misclassification, especially with the LFW dataset.

The final classifiers allocated 80% of the dataset for train, with the remaining 20% used for testing. The obtained result on the test subsets over the 4 trained models are presented on Table 13:

Table 13 - Test Accuracy on the 4 Models trained with 80% of the dataset images used for train

80% of original dataset	*Accuracy*	
Dataset	LFW	LAR
Model 1	23%	98.8%
Model 2	23.6%	98.4%
Model 3	84.6%	99.6%
Model 4	99.6%	99.9%

The Inception-ResNet models trained with the LAR dataset presented subpar classification with other datasets. That meant that the feature extractor using the CNN trained with a small amount of data could not generalize outside of the dataset.

The CASIA-Webface models had a significant improvement classifying new samples of data. Model 4 using the Table 13 settings showcased great accuracy and confidence to be used as classifier on the real time face detection and recognition pipeline.

Annex III – Confusion Matrix displays the Matrix Confusion, plotted after every test subset prediction execution. It showcased a visual representation of the predictions percentages of the LAR dataset compared to the ground truths of the data points. This visualization tool was important to understand where the SVM was misclassifying the data when checking the minimal number of images for train.

To ensure the model's performance outside the LFW datasets, more tests were carried out on available images on the internet over celebrities on the LFW dataset. On Figure 86 a compilation of the results is showcased. The images presented the classifier prediction and confidence over the bounding boxes drawn with the help of the MTCNN face detector.

Figure 86 - Compilation of the Classifier prediction and confidence on images of LFW classes outside the dataset

4.3.3 SVM Conclusions

The classifiers training from feature vectors using a linear kernel support vector machine proved to be a fast solution with very high classification results for smaller datasets.

The train of the datasets was never over 2 minutes, even when thousands of images were inputted. The pickle serialization file managed to save the progress of the classifier, as well as the classes used for training.

When testing the classifier, the feature extractor model always had to be the same as the train framework. Four of the trained models with the Inception-ResNet were used for comparison. As demonstrated in SVM Results, feature extractors trained with the LAR dataset could not generalize outside of that dataset, given the lack of data necessary to make a robust system. It also proved the Model 1 overfitting, since this model had very good results classifying the same data it trained on. It meant that instead of learning how to classify the data, it memorized its patterns. It even outperforming Model 2, the better model on the same dataset.

Models 3 and 4, trained on the large CASIA-Webface dataset, proved to be strong feature extractors. Since the produced samples arrays were more distinguishable, the SVM managed to have high accuracy values.

Two datasets were used for classification. The LFW dataset presented a bigger challenge, since most of the classes had minimal images, while other presented a larger number. That would bias the classifier to predict those classes with a bigger number of samples when the result was uncertain. The LAR dataset had the same number of images per class, 300, so the classifier results were always balanced.

The minimal number of images per class to have a good accuracy result was studied. With only 5 images (Table 11), Model 4 had good accuracy, but the confidence metric was low, leading to misclassification problems outside the dataset. The minimum number of images per class that produced reasonable results on both datasets was 10 images (Table 12). By setting the train dataset to 80% of the images (Table 13), Model 4 obtained great accuracy on datasets, predicting samples outside the model with high confidence, as seen in Figure 86 and Annex III – Confusion Matrix.

4.4 Embedding Projector

The embedding arrays extracted in the SVM Classifier and matching sprite images were loaded into TensorBoard (Figure 57). With the projector plugin, the PCA and t-SNE analysis was executed into the Euclidean dimensional space. Those Dimension Reduction Unsupervised Learning methods then clustered the data points into the Euclidean Space. The grouped samples were observed and analyzed.

4.4.1 PCA Analysis

The PCA output is presented in Figure 87. The analysis was done with the 3 major principal components for max segregation. Overall, samples of the same class/identity were clustered together into different spatial locations. Some classes presented a more spreader cluster. Analyzing those samples, that happened because of the face pose, very similar to others in their surroundings. Hence, the PCA unsupervised method furthermore proved that the feature extractor Inception-ResNet CNN was able to differentiate classes into a lower-dimensional Euclidean space. One helpful setting on the TensorBoard plugin was the coloring of different classes for visualization pouposes, as seen in Figure 88. That way, a better understanding can be experienced from an observer standpoint.

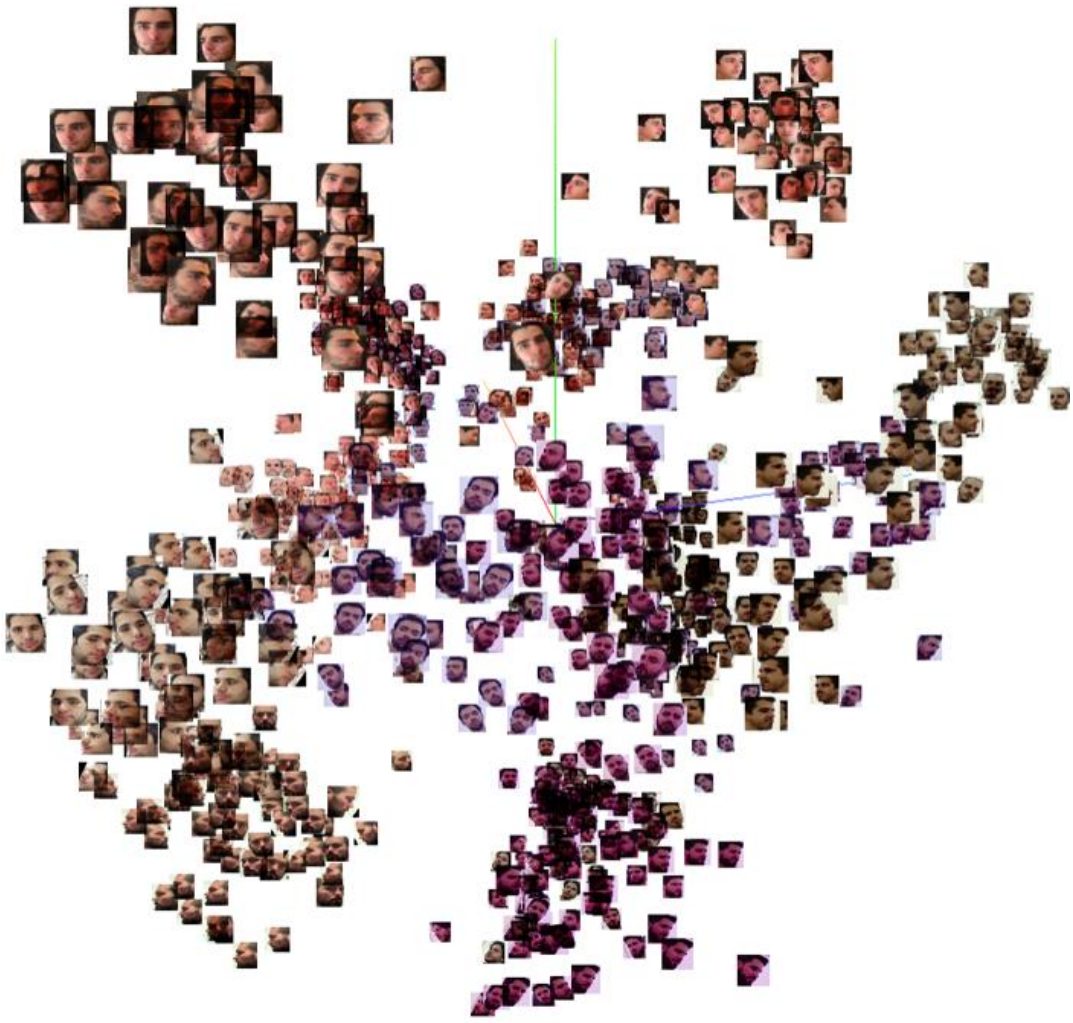


Figure 87 - Principal Component Analysis visualizer with three first PCs using the TensorBoard Embedding Plugin

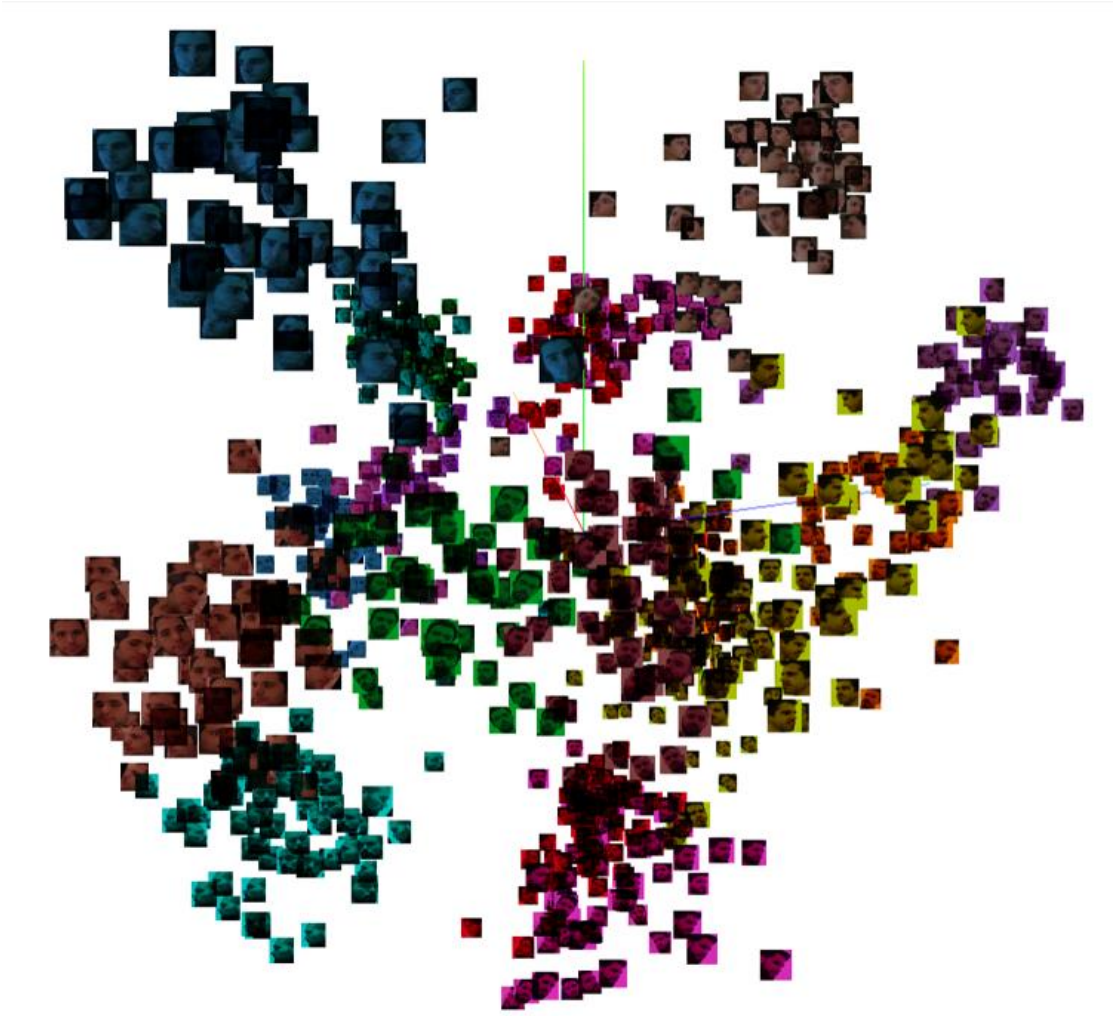


Figure 88 – Label coloring of the PCA analysis presented in Figure 87

4.4.2 t-SNE Analysis

As stated in t-SNE section, the algorithm starts by calculating the probability of similarity of points in high-dimensional space and calculating the probability of similarity of points in the corresponding low-dimensional space. In TensorBoard, the projector started at interaction 0 (Figure 89). It then tries to minimize the difference between these conditional probability's interaction by interaction. This process was done automatically by TensorBoard with a pre-set lr. After enough calculating interactions, the images were clustered according to similar neighbors. By analyzing Figure 90, the images in every spatial location were of the same identity, proving a good segregation between the labels.

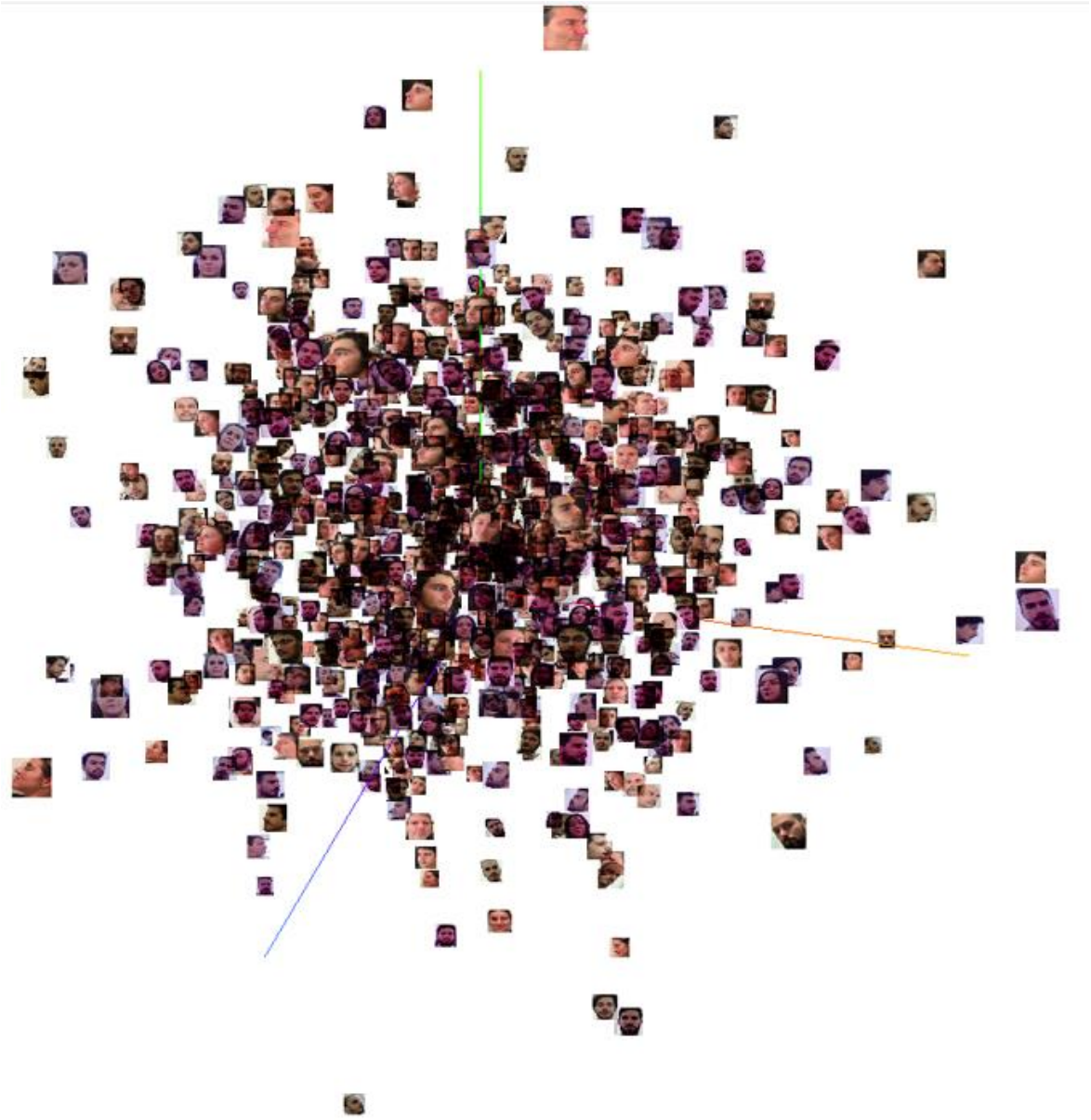


Figure 89 - t-SNE plot on TensorBoard embedding plugin at interaction 0

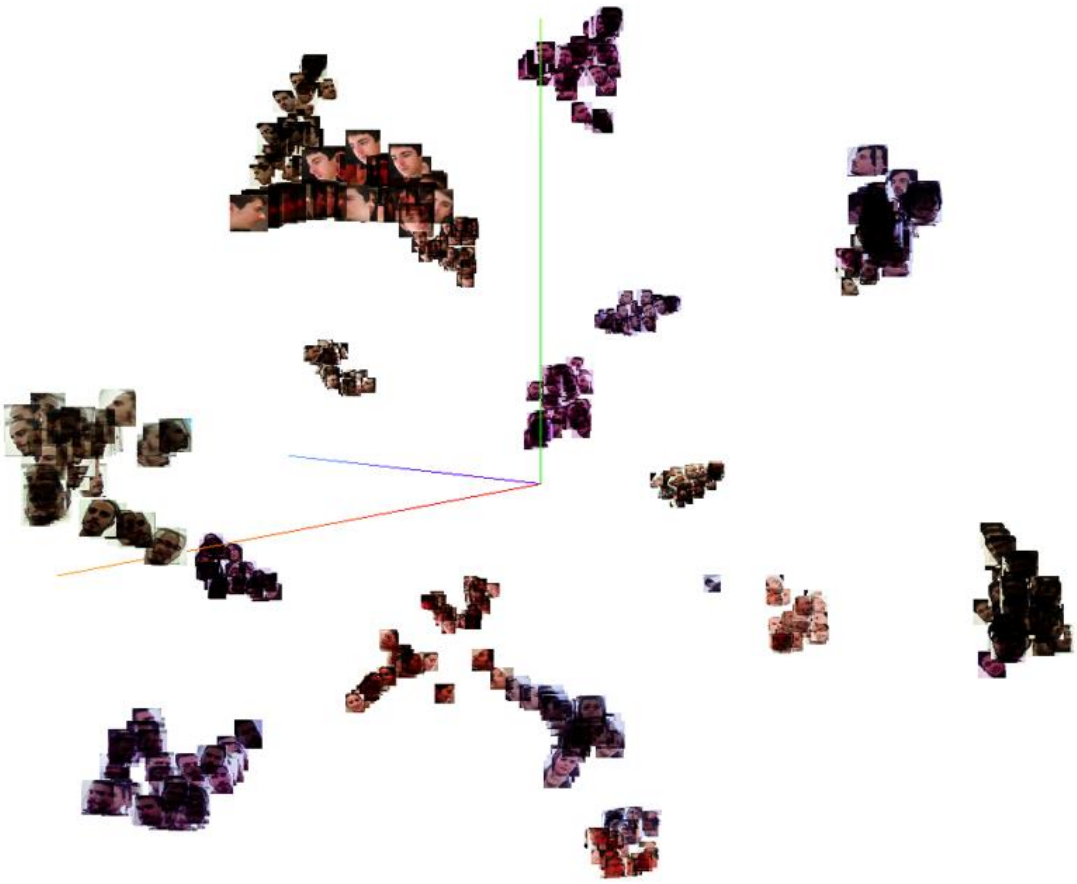


Figure 90 - t-SNE plot on TensorBoard embedding plugin after the necessary calculation interaction to cluster the labeled images

4.5 Real-Time Results

The real time framework used the multi-network methodology built by importing the 3 previously implemented modules. With the camera, the MTCNN isolated the faces and generated the bounding boxes. Afterwards, the array of the cropped faces goes through the Inception-ResNet feature extractor, returning the embedding array. That array was inputted one by one by the trained SVM classifier, predicting the input's identity and confidence. Should the confidence be higher than a certain threshold prediction value, the predicted person information was plotted in the original frame above the matching proposals. Figure 91 presents an example of the final pipeline result in a single person.

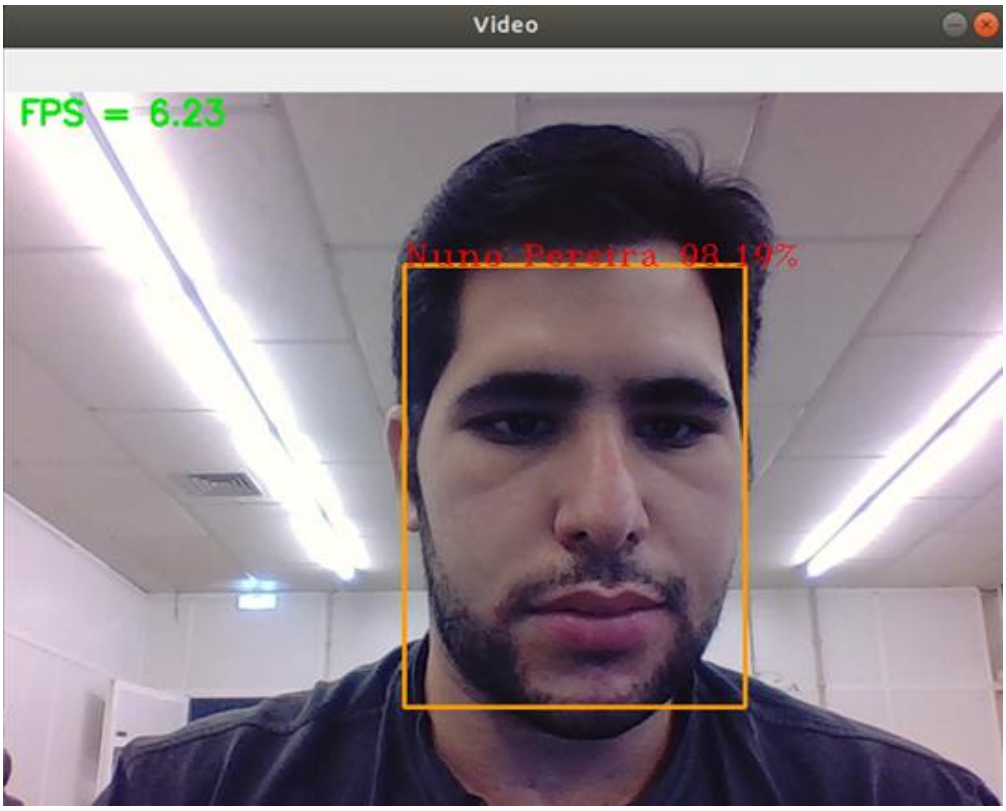


Figure 91 - Real-time user-based facial detection and recognition

The facial detection and recognition system maintained between 4 to 9 fps, depending on the amount of people presented in the camera's frame. That was a good real-time baseline, given that the model had to run with 2 CNNs and a SVM classifier at the same time.

Some tests regarding pose variations and distance to the camera were performed. In Figure 92 the model predicted correctly a slightly tilted face, although the confidence dropped slightly.

Figure 93 presents the prediction confidence on a test regarding different distances from the camera. That meant the size of the detected face would vary depending how far the person is from the camera.

Figure 94 showcases the identification problem using a mask on the person detected. The pipeline still was able to correctly predict the person's identity, but with a much smaller accuracy confidence. When using the threshold to discard less reliant proposals, normally candidates with less than 75%, these samples would be filtered.

Figure 95 and Figure 96 present two users being detected and classified at the same time. In Figure 95 two users part of the pre-trained classifier were both correctly detected with high confidence. Figure 96 presents 2 more candidates to be classified. Since the right user was not part of the pre-trained users, it was labelled as 'Unknown'.

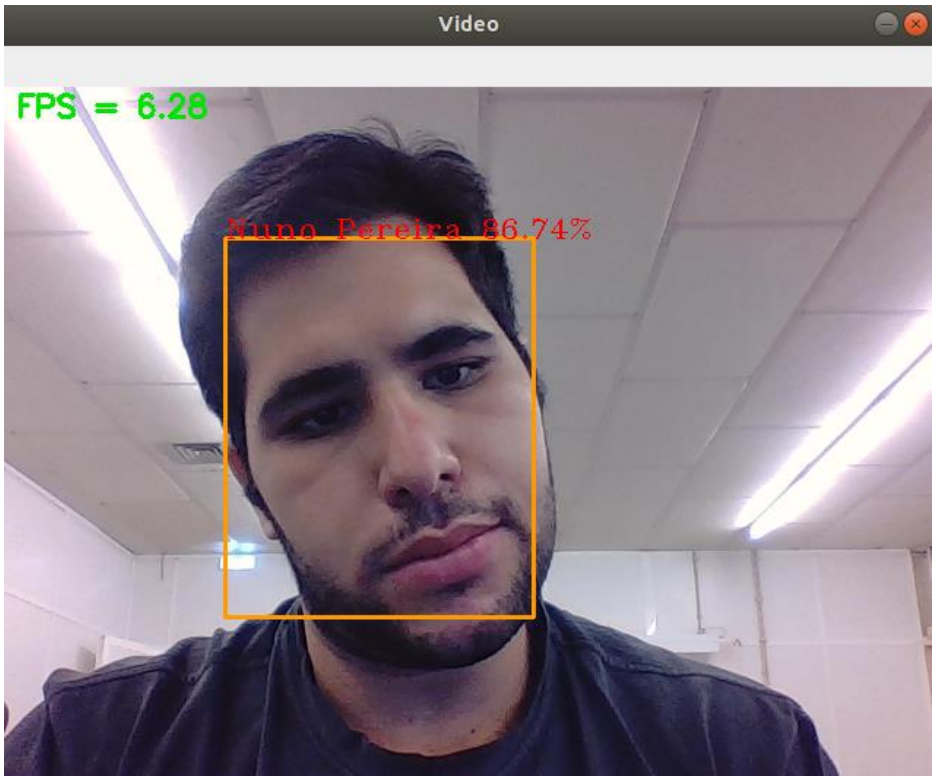


Figure 92 - Pose variation prediction example using real time face recognition

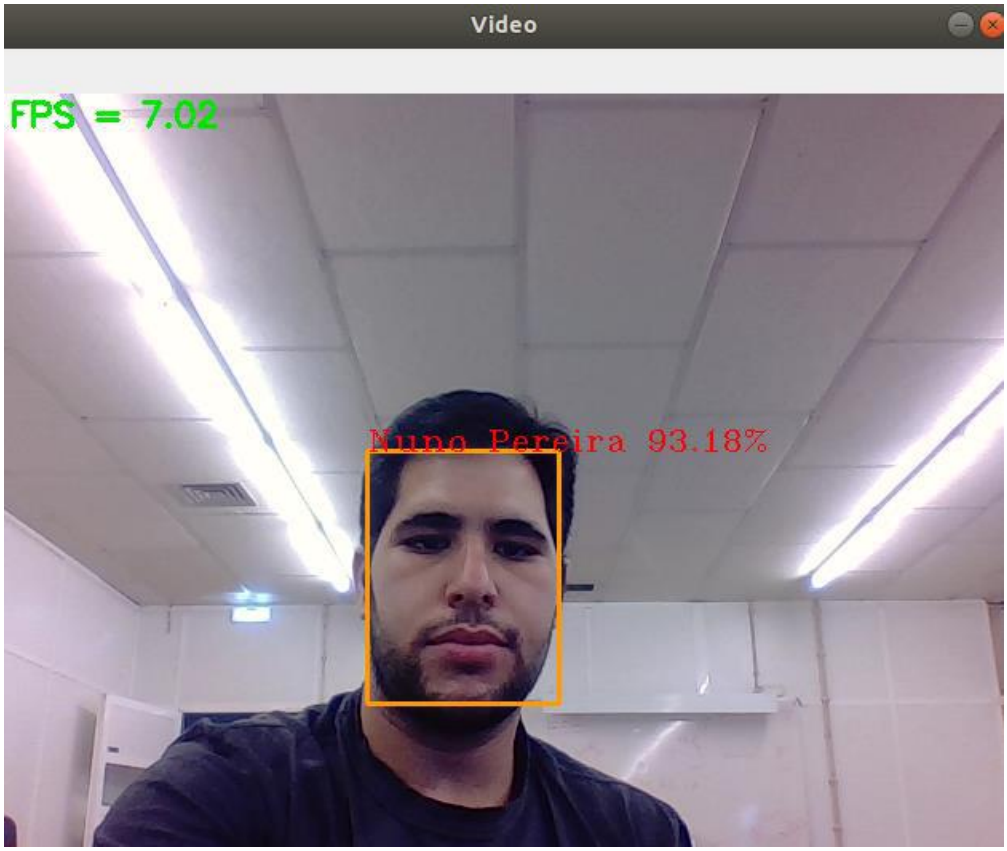


Figure 93 – Prediction on a further distance to the camera

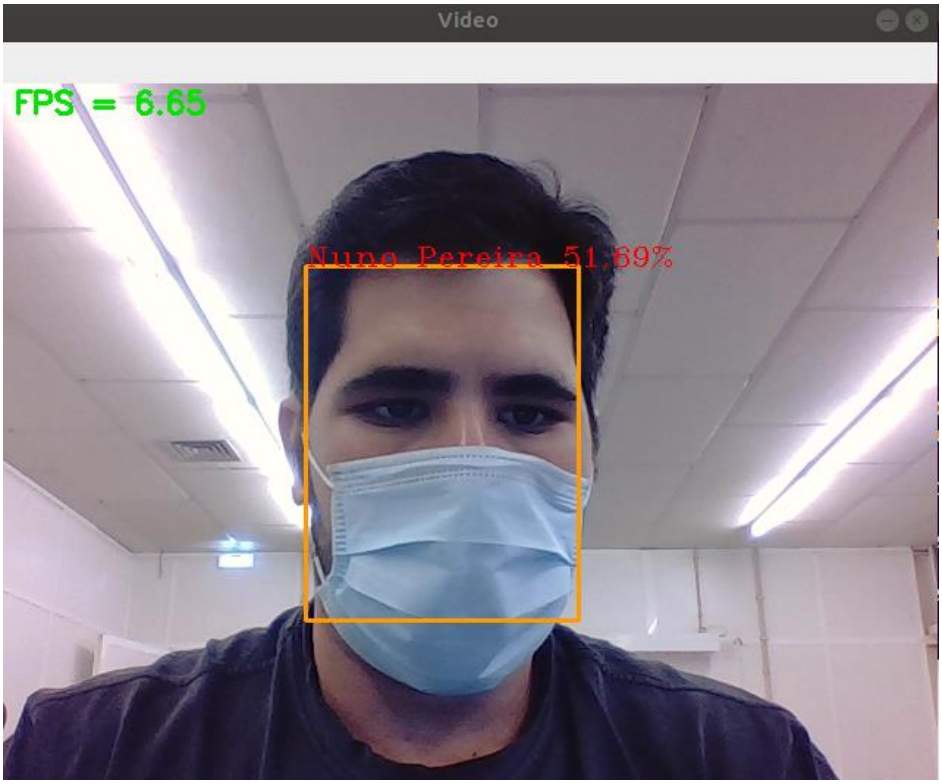


Figure 94 - Prediction using mask

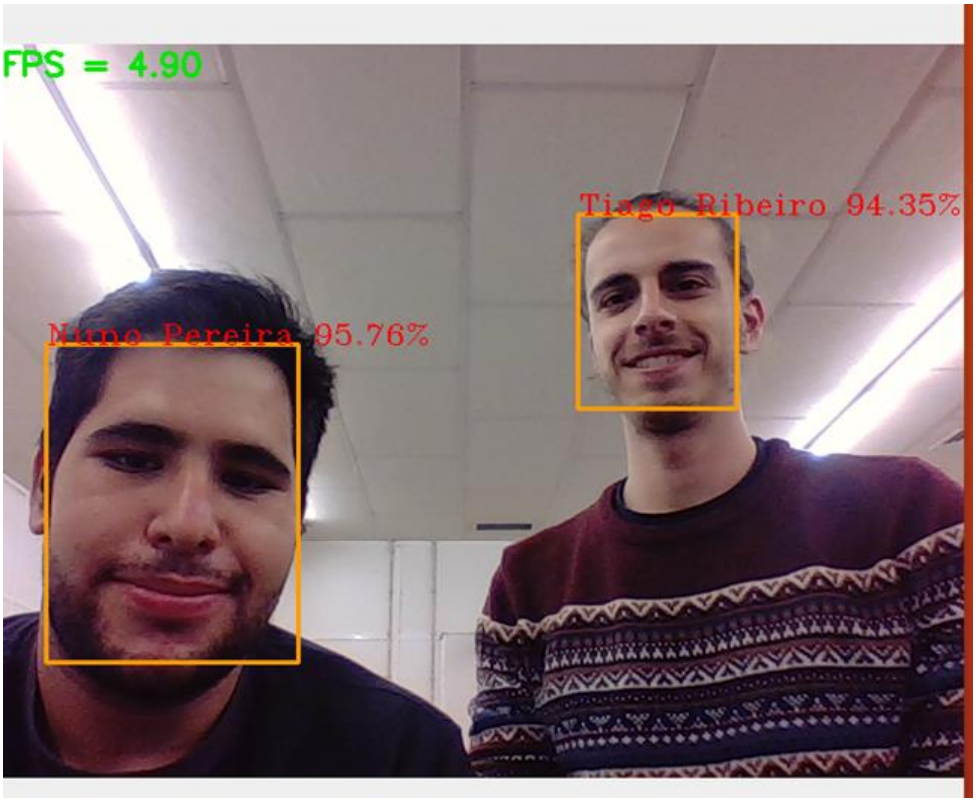


Figure 95 -. Real-time user-based facial detection and recognition with two different users part of the pre-trained classifier, both correctly detected and classified

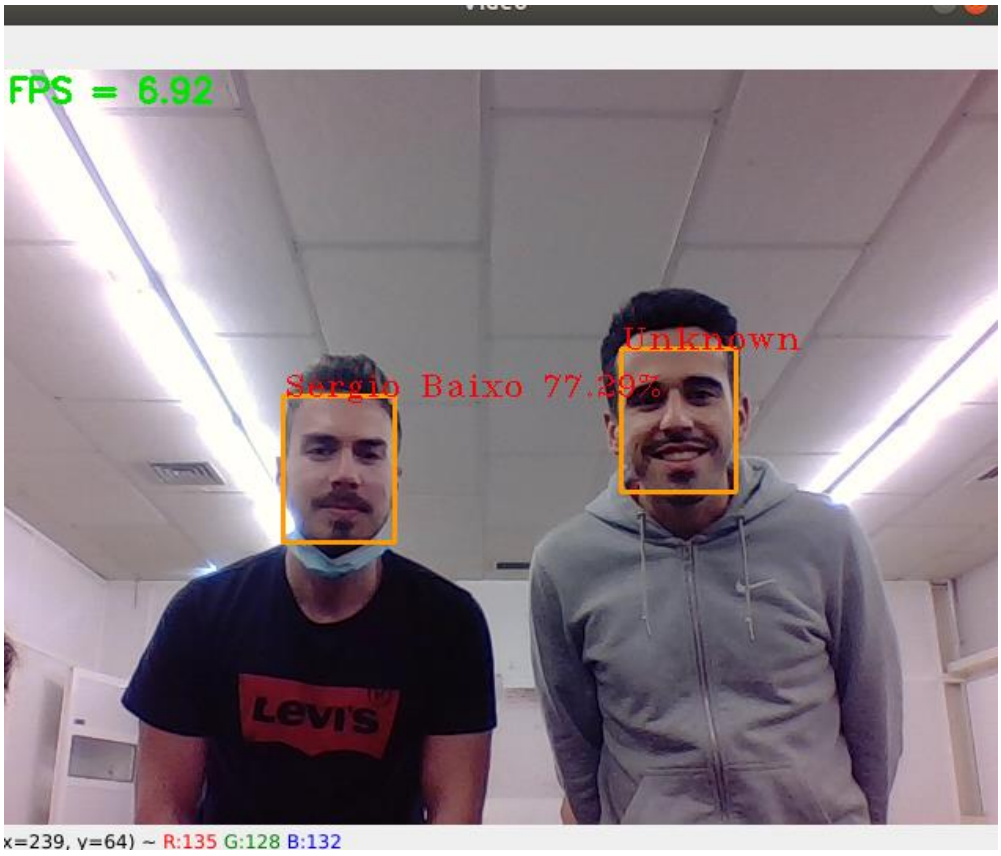


Figure 96 - Real-time user-based facial detection on 2 subjects. The left user is part of the pre-trained users and is classified correctly. The right user was not part of the pre-trained users, and so was labelled as 'Unknown'.

4.5.1 Real time Conclusions

The real time framework was successfully implemented when combining the previously implemented modules.

The recognition system maintained a minimum of 6 to 9 fps when a single person is detected, quickly dropping to 4 when more candidates appeared. Hence, even with 2 large CNNs working in the background of the framework, the system worked consistently without frame drops.

The showcased test results concluded that all the previously implemented modules could successfully work in conjunction with new data instances. Using the pre-trained members of LAR as test labels, the system correctly detect faces and predicted the user's identities on different configurations with very high accuracy and confidence. The system flaws came from the lack of diversity when building the custom dataset. Some pose variation settings, such as looking deeply left or right, would show a lower accuracy than the standard straight forward pose. That could successfully be tackled with a more diverse custom dataset, if not for the Covid restrictions at the time of the LAR dataset creation.

5. CONCLUSIONS

This dissertation presented a real-time user-based face recognition system using multi-network deep learning methods, with the capability to detect and identify people. The final pipeline was formed by 3 modules: the face detector, the face feature extractor, and the classifier. A custom dataset of users from the Laboratory of Automation and Robotics (LAR) from the University of Minho was built to support the execution and test of all the implemented algorithms. All LAR elements gave permission for their data to be used.

The intent of the facial detection and recognition framework was to be implemented on a domestic and healthcare service robot, CHARMIE. For a service and assistive robot such as CHARMIE, face detection and user recognition are essential for a more user-oriented interaction. Therefore, CHARMIE can adapt its approach as well as how it performs different tasks depending on who the interacting user is. Additional information regarding its users can be associated with a specific user, for example, if it is a child, an adult or a senior, whether that person has any mobility issues or if it is allowed or not to be in a specific area of an environment. All this information helps create a more personalized experience for all user interactions since CHARMIE can use this technology to directly adapt its behaviors to positively influence whom it is interacting with.

Initially, a detailed introduction to Deep Learning using Convolution Neural Networks was presented, from the earlier and basic concepts to the state-of-the-art architectures. It is concluded that CNNs benchmarked computer vision algorithms by relying on the spatial structure during image processing. CNN methods allowed for more robust detection and classification systems by training on large sets of data. These datasets were essential for AI methods to generalize specific classes problems, only available after extensive work by researchers and institutions. The state-of-the-art methods regarding Face detection and Facial Recognition were analyzed, allowing to conclude that using CNNs surpassed human performance accuracy using Euclidean-distance-based loss and Angular/Cosine-margin based loss methods.

As face detector, a multi-task cascaded framework, MTCNN, performed the detection and alignment module that isolated the detected faces, predicting the bounding box coordinates. It also processed the faces into 160 x 160-dimension settings, serving as input to the extraction model. The parameters were loaded via Transfer Learning, taking advantage of a previously trained model's inference. The framework was tested using the computers camera, achieving 10 to 18 fps consistently, making it reliable in real-time settings. It detected faces easily in standard scenarios, such as pose variance, lighting settings or 'disguise' conditions. Tests regarding face detection on real-world environments were

carried out. Some of those tests proved challenging for the model to have full accuracy on, even when lowering the model's thresholds. The custom LAR dataset aligned and cropped faces with MTCNN. OpenCV library successfully rotated and flipped random samples of data, to provide more face pose variations. This was later very important for the model's prediction in real time contexts.

Both v1 and v2 versions of the Inception-ResNet CNNs modules were trained with the custom dataset, while the Webface-CASIA dataset only trained with version 1 due to lacking computational resources. The study of different hyperparameters using the two datasets on the CNN is presented. The dissimilarities between large and small sets of data on a deep CNN were exposed.

Using an SMV classifier proved efficient in adding new persons to the dataset without being time expensive, with great accuracy. The SVM classifier was trained with the laboratory members using the built dataset. The samples used as input into the classifier were successfully encoded by the best performing Inception-ResNet module. Those modules as feature extractors were compared, selecting the best performing topology. Also, different experiments regarding the number of images necessary for an accurate classifier were carried out. Furthermore, the classifier using SVM can accurately predict new data instances of the labels.

The TensorBoard Embedding Projector plugin allowed for a PCA and t-SNE plot analysis into the Euclidean space, segregating the data produced by the best performing Inception-ResNet topology. Those dimension reduction algorithms provided an unsupervised learning metric, managing to successfully cluster samples of the same identity. proving the precision Inception-ResNet when extracting facial features.

All the networks end-to-end proved to be able to work on real-time applications and managed to detect users with different face poses and external variations such as illumination or even 'disguise'. The system flaws came from lack of diversity when building the custom dataset. Some pose variation settings, such as looking deeply left or right, would show a lower accuracy than the standard straight forward pose. That could successfully be tackled with a more diverse custom dataset. Overall, the system was able to detect and predict every student with an above 90% confidence accuracy.

5.1 Further Work

Regarding the further work, the system's framework can be upgraded by increasing the models complexity and performance. It can be used as standpoint for facial recognition comparisons, as well as a start point for more robust systems.

The model's performance and accuracy future upgrades can be summarized as follows:

- Creation of a more diverse dataset, improving the overall image acquisition of the person, as well as increasing the number of classes, comparing the model's performance
- Implement the models using 3D settings, comparing the models performance on a more real-world configuration
- Final implementation on the CHARMIE robot

REFERENCES

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Zheng, X. (2016). *TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems*. Retrieved from http://arxiv.org/abs/1603.04467

Ahonen, T., Hadid, A., & Pietikäinen, M. (2006). Face description with local binary patterns: Application to face recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *28*(12), 2037–2041. https://doi.org/10.1109/TPAMI.2006.244

Basiri, M., Piazza, E., Matteucci, M., & Lima, P. (2019). Benchmarking Functionalities of Domestic Service Robots Through Scientific Competitions. *KI - Kunstliche Intelligenz*. https://doi.org/10.1007/s13218-019-00619-9

Bauer, A., Wollherr, D., & Buss, M. (2008). Human-robot collaboration: A survey. *International Journal of Humanoid Robotics*, *5*(1), 47–66. https://doi.org/10.1142/S0219843608001303

Belhumeur, P. N., Hespanha, J. P., & Kriegman, D. J. (1997). Eigenfaces vs. fisherfaces: Recognition using class specific linear projection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*. https://doi.org/10.1109/34.598228

Bottou, L. (2010). Large-scale machine learning with stochastic gradient descent. *Proceedings of COMPSTAT 2010 - 19th International Conference on Computational Statistics, Keynote, Invited and Contributed Papers*, 177–186. https://doi.org/10.1007/978-3-7908-2604-3_16

Cao, Q., Shen, L., Xie, W., Parkhi, O. M., & Zisserman, A. (n.d.). *VGGFace2: A dataset for recognising faces across pose and age*. Retrieved from http://www.robots.ox.ac.uk/

Cao, Q., Shen, L., Xie, W., Parkhi, O. M., & Zisserman, A. (2018). VGGFace2: A dataset for recognising faces across pose and age. *Proceedings - 13th IEEE International Conference on Automatic Face and Gesture Recognition, FG 2018*, 67–74. https://doi.org/10.1109/FG.2018.00020

Capaldi, E. J. (1992). the Organization of Behavior. *Journal of Applied Behavior Analysis*, *25*(3), 575–577. https://doi.org/10.1901/jaba.1992.25-575

Chan, T.-H., Jia, K., Gao, S., Lu, J., Zeng, Z., & Ma, Y. (n.d.). *PCANet: A Simple Deep Learning Baseline for Image Classification?*

Chen, D., Hua, G., Wen, F., & Sun, J. (n.d.). *Supervised Transformer Network for Efficient Face Detection*.

Dalal, N., & Triggs, B. (2005). Histograms of oriented gradients for human detection. *Proceedings - 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, CVPR 2005*, *I*, 886–893. https://doi.org/10.1109/CVPR.2005.177

Deng, J., Guo, J., Xue, N., & Zafeiriou, S. (2018). *ArcFace: Additive Angular Margin Loss for Deep Face Recognition*. Retrieved from http://arxiv.org/abs/1801.07698

Deng, J., Guo, J., Zhou, Y., Yu, J., Kotsia, I., & Zafeiriou, S. (n.d.). *RetinaFace: Single-stage Dense Face Localisation in the Wild*. Retrieved from https://github.com/deepinsight/

Deng, J., Zhou, Y., & Zafeiriou, S. (2017). Marginal Loss for Deep Face Recognition. *IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*, *2017-July*, 2006–2014. https://doi.org/10.1109/CVPRW.2017.251

GitHub - davidsandberg/facenet: Face recognition using Tensorflow. (n.d.). Retrieved July 6, 2021, from https://github.com/davidsandberg/facenet

Glorot, X., & Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. *Journal of Machine Learning Research*, *9*, 249–256. Retrieved from http://www.iro.umontreal.

Glorot, X., Bordes, A., & Bengio, Y. (2011). Deep sparse rectifier neural networks. *Journal of Machine Learning Research*, *15*, 315–323.

Gonçalves, F., Ribeiro, T., Garcia, I., Ribeiro, F. A., Monteiro, C., & Lopes, G. (2019). Development of an anthropomorphic mobile manipulator with human, machine and environment interaction. *FME Transactions*, *47*(4), 790–801. https://doi.org/10.5937/fmet1904790F

Guo, Y., Zhang, L., Hu, Y., He, X., & Gao, J. (n.d.). *MS-Celeb-1M: A Dataset and Benchmark for Large-Scale Face Recognition*. Retrieved from http://www.hdwallpapers.in/anne

Hasnat, M. A., Bohné, J., Milgram, J., Gentric, S., & Chen, L. (2017). *von Mises-Fisher Mixture Model-based Deep learning: Application to Face Verification*. 1–16. Retrieved from http://arxiv.org/abs/1706.04264

He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, *2016-Decem*, 770–778. https://doi.org/10.1109/CVPR.2016.90

He, X., Yan, S., Hu, Y., Niyogi, P., & Zhang, H. J. (2005). Face recognition using Laplacianfaces. *IEEE Transactions on Pattern Analysis and Machine Intelligence*. https://doi.org/10.1109/TPAMI.2005.55

Hinton, G. E., Srivastava, N., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. R. (2012). *Improving neural networks by preventing co-adaptation of feature detectors*. 1–18. Retrieved from http://arxiv.org/abs/1207.0580

Holland, J., Kingston, L., McCarthy, C., Armstrong, E., O'dwyer, P., Merz, F., & McConnell, M. (2021). Service robots in the healthcare sector. *Robotics*, *10*(1), 1–47. https://doi.org/10.3390/robotics10010047

Holz, D., & Iocchi, L. (2013). Benchmarking Intelligent Service Robots through Scientific Competitions : The RoboCup @ Home Approach. *AAAI Spring Symposium - Designing Intelligent Robots: Reintegrating AI II*, 27–32.

Huang, G. B., Mattar, M., Berg, T., Labeled, E. L., Images, R., & Learned-miller, E. (2008). Labeled Faces in the Wild : A Database for Studying Face Recognition in Unconstrained Environments. *Workshop on Faces in'Real-Life'Images: Detection, Alignment, and Recognition*, *07–49*.

Hubel, D. H., & Wiesel, T. N. (1959). Receptive fields of single neurones in the cat's striate cortex. *The Journal of Physiology*, *148*(3), 574–591. https://doi.org/10.1113/jphysiol.1959.sp006308

Hubel, D. H., & Wiesel, T. N. (1968). Receptive fields and functional architecture of monkey striate cortex. *The Journal of Physiology*, *195*(1), 215–243. https://doi.org/10.1113/jphysiol.1968.sp008455

"ImageNet Large Scale Visual Recognition Competition 2012 (ILSVRC2012)." [Online]. Available:image-net.org/challenges/LSVRC/2012/. (n.d.).

ImageNet Large Scale Visual Recognition Competition 2014 (ILSVRC2014)." [Online]. Available:image-net.org/challenges/LSVRC/2014/. (n.d.).

ImageNet Large Scale Visual Recognition Competition 2015 (ILSVRC2015)." [Online]. Available: image-net.org/challenges/LSVRC/2015/. (n.d.).

Iocchi, L., Holz, D., Ruiz-Del-Solar, J., Sugiura, K., & Van Der Zant, T. (2015). RoboCup@Home: Analysis and results of evolving competitions for domestic and service robots. *Artificial Intelligence*. https://doi.org/10.1016/j.artint.2015.08.002

Ioffe, S., & Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. *32nd International Conference on Machine Learning, ICML 2015*, *1*, 448–456.

Jollife, I. T., & Cadima, J. (2016). Principal component analysis: A review and recent developments. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, Vol. 374. https://doi.org/10.1098/rsta.2015.0202

Kalal, Z., Mikolajczyk, K., & Matas, J. (2010). FACE-TLD : TRACKING-LEARNING-DETECTION APPLIED TO FACES Centre for Vision , Speech and Signal Processing , University of Surrey , UK Center for Machine Perception , Czech Technical University , Czech Republic. *2010 IEEE 17th International Conference on Image Processing*, (i), 3789–3792.

Kemelmacher-Shlizerman, I., Seitz, S., Miller, D., & Brossard, E. (n.d.). *The MegaFace Benchmark: 1 Million*

Faces for Recognition at Scale.

Khaligh-Razavi, S.-M. (2014). *What you need to know about the state-of-the-art computational models of object-vision: A tour through the models*. Retrieved from http://arxiv.org/abs/1407.2776

Kragic, D., & Vincze, M. (2009). Vision for Robotics. *Foundations and Trends in Robotics*, *1*(1), 1–78. https://doi.org/10.1561/2300000001

Krizhevsky, A., & Hinton, G. (2009). *Learning multiple layers of features from tiny images.(2009)*. Retrieved from https://www.cs.toronto.edu/~kriz/cifar.html

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2017). ImageNet Classification with Deep Convolutional Neural Networks. *COMMUNICATIONS OF THE ACM*, *60*(6). https://doi.org/10.1145/3065386

Lecun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. *Nature*, *521*(7553), 436–444. https://doi.org/10.1038/nature14539

Lecun, Y., Bottou, L., Bengio, Y., & Ha, P. (1998). Gradient-Based Learning Applied to Document Recognition. *Proceedings of the IEEE*, (November), 1–46. https://doi.org/10.1109/5.726791

Lecun, Y., Cortes, C., & C.J.C., B. (1998). *THE MNIST DATABASE of handwritten digits*.

Li, H., Lin, Z., Shen, X., Brandt, J., & Hua, G. (n.d.). *A Convolutional Neural Network Cascade for Face Detection*.

Lin, M., Chen, Q., & Yan, S. (2014). Network in network. *2nd International Conference on Learning Representations, ICLR 2014 - Conference Track Proceedings*.

Lin, T.-Y., Dollár, P., Girshick, R., He, K., Hariharan, B., & Belongie, S. (n.d.). *Feature Pyramid Networks for Object Detection*.

Liu, C., & Wechsler, H. (2002). Gabor feature based classification using the enhanced Fisher linear discriminant model for face recognition. *IEEE Transactions on Image Processing*, *11*(4), 467–476. https://doi.org/10.1109/TIP.2002.999679

Liu, J., Deng, Y., Bai, T., Wei, Z., & Huang, C. (2015). *Targeting Ultimate Accuracy: Face Recognition via Deep Embedding*. 1–5. Retrieved from http://arxiv.org/abs/1506.07310

Liu, W., Wen, Y., Yu, Z., Li, M., Raj, B., & Song, L. (2017). SphereFace: Deep hypersphere embedding for face recognition. *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017*, *2017-Janua*, 6738–6746. https://doi.org/10.1109/CVPR.2017.713

Liu, W., Wen, Y., Yu, Z., & Yang, M. (2016). *Large-Margin Softmax Loss for Convolutional Neural Networks*. Retrieved from http://arxiv.org/abs/1612.02295

Liu, Y., Li, H., & Wang, X. (2017). *Rethinking Feature Discrimination and Polymerization for Large-scale Recognition*. (Nips). Retrieved from http://arxiv.org/abs/1710.00870

Low, C.-Y. (n.d.). *Learning Compact Discriminant Local Face Descriptor with VLAD*.

Masi, I., Wu, Y., Hassner, T., & Natarajan, P. (2019). Deep Face Recognition: A Survey. *Proceedings - 31st Conference on Graphics, Patterns and Images, SIBGRAPI 2018*, 471–478. https://doi.org/10.1109/SIBGRAPI.2018.00067

Min, Y., & Chung, H. W. (2019). Shallow Neural Network can Perfectly Classify an Object following Separable Probability Distribution. *IEEE International Symposium on Information Theory - Proceedings*, *2019-July*(1), 1812–1816. https://doi.org/10.1109/ISIT.2019.8849497

Minaee, S., Luo, P., Lin, Z., & Bowyer, K. (2021). *Going Deeper Into Face Detection: A Survey*. 1–17. Retrieved from http://arxiv.org/abs/2103.14983

Nada, H., Sindagi, V. A., Zhang, H., & Patel, V. M. (n.d.). *Pushing the Limits of Unconstrained Face Detection: a Challenge Dataset and Baseline Results*. Retrieved from www.ufdd.info/

Nair, V., & Hinton, G. E. (n.d.). *Rectified Linear Units Improve Restricted Boltzmann Machines*.

Najibi, M., Samangouei, P., Chellappa, R., & Davis, L. S. (n.d.). *SSH: Single Stage Headless Face Detector*. Retrieved from https://github.com/

Nielsen, M. (2018). Neural Networks and Deep Learning. *Artificial Intelligence*, 389–411.

https://doi.org/10.1201/b22400-15

O'Shea, K., & Nash, R. (2015). *An Introduction to Convolutional Neural Networks*. 1–11. Retrieved from http://arxiv.org/abs/1511.08458

Parkhi, O. M., Vedaldi, A., & Zisserman, A. (2015). *Deep Face Recognition*. (Section 3), 41.1-41.12. https://doi.org/10.5244/c.29.41

Qi, X., & Zhang, L. (2018). *Face Recognition via Centralized Coordinate Learning*. 1–14. Retrieved from http://arxiv.org/abs/1801.05678

Ranjan, R., Castillo, C. D., & Chellappa, R. (2017). *L2-constrained Softmax Loss for Discriminative Face Verification*. Retrieved from http://arxiv.org/abs/1703.09507

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature*, *323*(6088), 533–536. https://doi.org/10.1038/323533a0

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., ... Ma, S. (n.d.). *ImageNet Large Scale Visual Recognition Challenge*.

Schmidhuber, J. (2015). Deep Learning in neural networks: An overview. *Neural Networks*, Vol. 61, pp. 85–117. https://doi.org/10.1016/j.neunet.2014.09.003

Schroff, F., Kalenichenko, D., & Philbin, J. (2015). FaceNet: A unified embedding for face recognition and clustering. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, *07-12-June*, 815–823. https://doi.org/10.1109/CVPR.2015.7298682

Simonyan, K., & Zisserman, A. (2014). *Very Deep Convolutional Networks for Large-Scale Image Recognition*. 1–14. Retrieved from http://arxiv.org/abs/1409.1556

Sun, Y., Chen, Y., Wang, X., & Tang, X. (2014). Deep learning face representation by joint identification-verification. *Advances in Neural Information Processing Systems*, *3*(January), 1988–1996.

Sun, Y., Liang, D., Wang, X., & Tang, X. (2015). *DeepID3: Face Recognition with Very Deep Neural Networks*. 2–6. Retrieved from http://arxiv.org/abs/1502.00873

Sun, Y., Wang, X., & Tang, X. (2015). Deeply learned face representations are sparse, selective, and robust. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. https://doi.org/10.1109/CVPR.2015.7298907

Szegedy, C., Ioffe, S., Vanhoucke, V., & Alemi, A. A. (2017). Inception-v4, Inception-ResNet and the impact of residual connections on learning. *31st AAAI Conference on Artificial Intelligence, AAAI 2017*, 4278–4284.

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., ... Rabinovich, A. (2015). Going deeper with convolutions. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, *07-12-June*, 1–9. https://doi.org/10.1109/CVPR.2015.7298594

Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., & Wojna, Z. (2016). Rethinking the Inception Architecture for Computer Vision. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, *2016-Decem*, 2818–2826. https://doi.org/10.1109/CVPR.2016.308

Taigman, Y., Yang, M., Ranzato, M., & Wolf, L. (2014). DeepFace: Closing the gap to human-level performance in face verification. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 1701–1708. https://doi.org/10.1109/CVPR.2014.220

Turk, M., & Pentland, A. (1991). Eigenfaces for recognition. *Journal of Cognitive Neuroscience*. https://doi.org/10.1162/jocn.1991.3.1.71

Van Der Maaten, L., & Hinton, G. (2008a). Visualizing Data using t-SNE. In *Journal of Machine Learning Research* (Vol. 9).

Van Der Maaten, L., & Hinton, G. (2008b). Visualizing Data using t-SNE. *Journal of Machine Learning Research*, *9*, 2579–2605.

Villani, V., Pini, F., Leali, F., & Secchi, C. (2018). Survey on human–robot collaboration in industrial settings: Safety, intuitive interfaces and applications. *Mechatronics*, *55*(March), 248–266.

https://doi.org/10.1016/j.mechatronics.2018.02.009

Viola, P., & Jones, M. (2001). Rapid object detection using a boosted cascade of simple features. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, *1*. https://doi.org/10.1109/cvpr.2001.990517

Wang, Fei. (n.d.). *The Devil of Face Recognition is in the Noise*. Retrieved from www.IMDb.com

Wang, Feng, Cheng, J., Liu, W., & Liu, H. (2018). Additive Margin Softmax for Face Verification. *IEEE Signal Processing Letters*, *25*(7), 926–930. https://doi.org/10.1109/LSP.2018.2822810

Wang, Feng, Xiang, X., Cheng, J., & Yuille, A. L. (2017). NormFace: L2 hypersphere embedding for face verification. *MM 2017 - Proceedings of the 2017 ACM Multimedia Conference*, 1041–1049. https://doi.org/10.1145/3123266.3123359

Wang, H., Wang, Y., Zhou, Z., Ji, X., Gong, D., Zhou, J., … Liu, W. (2018). CosFace: Large Margin Cosine Loss for Deep Face Recognition. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 5265–5274. https://doi.org/10.1109/CVPR.2018.00552

Wang, M., & Deng, W. (2021). Deep face recognition: A survey. *Neurocomputing*. https://doi.org/10.1016/j.neucom.2020.10.081

Wang, Y., Ji, X., Zhou, Z., Wang, H., & Li, Z. (2017). *Detecting Faces Using Region-based Fully Convolutional Networks*. Retrieved from http://arxiv.org/abs/1709.05256

Wen, Y., Zhang, K., Li, Z., & Qiao, Y. (n.d.). *A Discriminative Feature Learning Approach for Deep Face Recognition*. https://doi.org/10.1007/978-3-319-46478-7

Wu, X., He, R., Sun, Z., & Tan, T. (2018). A light CNN for deep face representation with noisy labels. *IEEE Transactions on Information Forensics and Security*, *13*(11), 2884–2896. https://doi.org/10.1109/TIFS.2018.2833032

Yan, J., Zhang, X., Lei, Z., & Li, S. Z. (n.d.). *Face detection by structural models ☆*. https://doi.org/10.1016/j.imavis.2013.12.004

Yang, S., Luo, P., Loy, C. C., & Tang, X. (2016). WIDER FACE: A face detection benchmark. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, *2016-Decem*, 5525–5533. https://doi.org/10.1109/CVPR.2016.596

Yi, D., Lei, Z., Liao, S., & Li, S. Z. (n.d.). *Learning Face Representation from Scratch*. Retrieved from http://www.imdb.com

Zafeiriou, S., Zhang, C., & Zhang, Z. (2015). A survey on face detection in the wild: Past, present and future. *Computer Vision and Image Understanding*, *138*, 1–24. https://doi.org/10.1016/j.cviu.2015.03.015

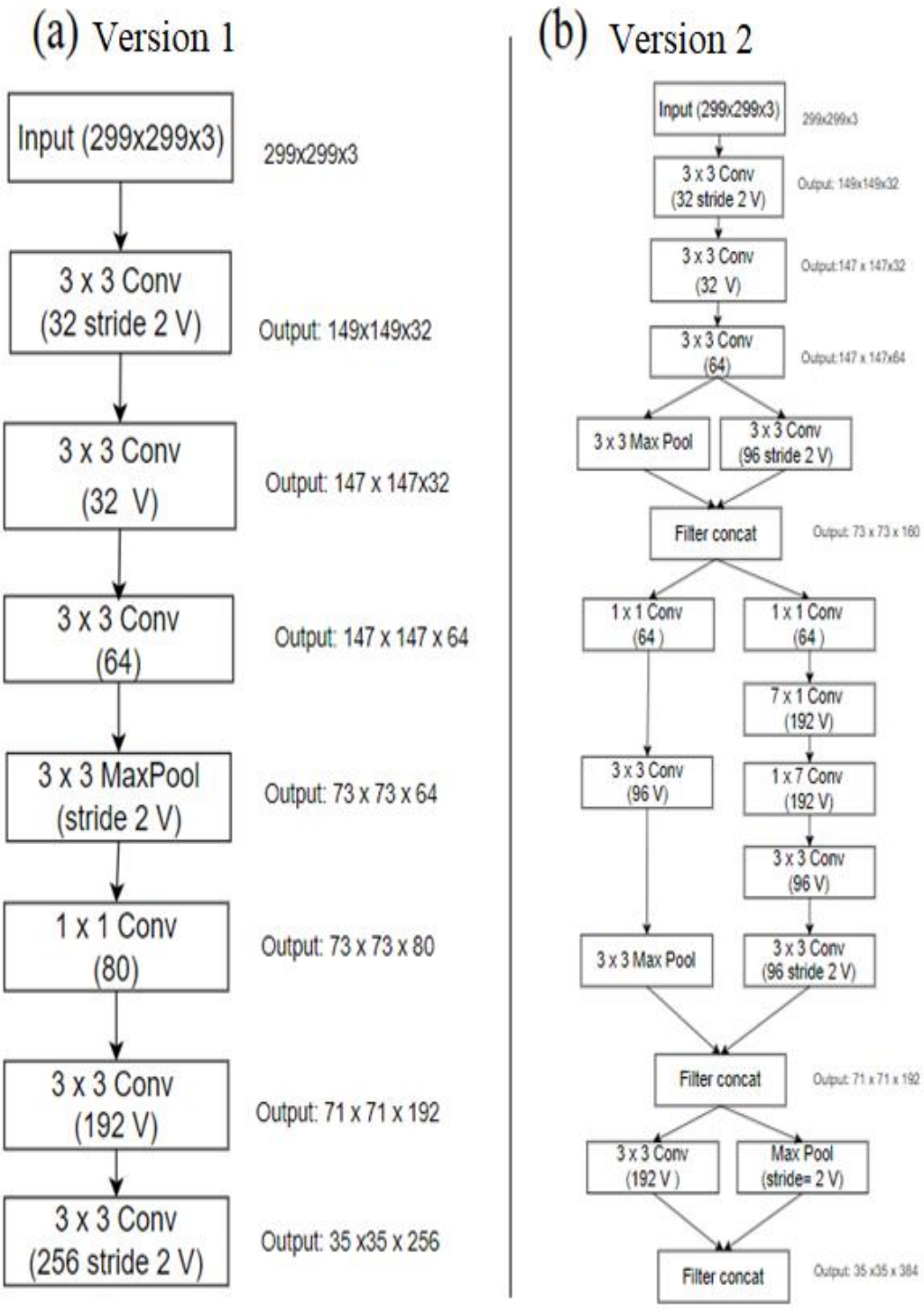
Zhang, J., Wu, X., Hoi, S. C. H., Zhu, J., Zhang, J. ;, Wu, X. ;, & Hoi, S. C. H. ; (n.d.). *Feature agglomeration networks for single stage face detection Feature agglomeration networks for single stage face detection Part of the Databases and Information Systems Commons, and the Data Storage Systems Commons Citation Citation Feature Agglomeration Networks for Single Stage Face Detection*. Retrieved from https://ink.library.smu.edu.sg/sis_research

Zhang, K., Zhang, Z., Li, Z., & Qiao, Y. (2016). Joint Face Detection and Alignment Using Multitask Cascaded Convolutional Networks. *IEEE Signal Processing Letters*, *23*(10), 1499–1503. https://doi.org/10.1109/LSP.2016.2603342

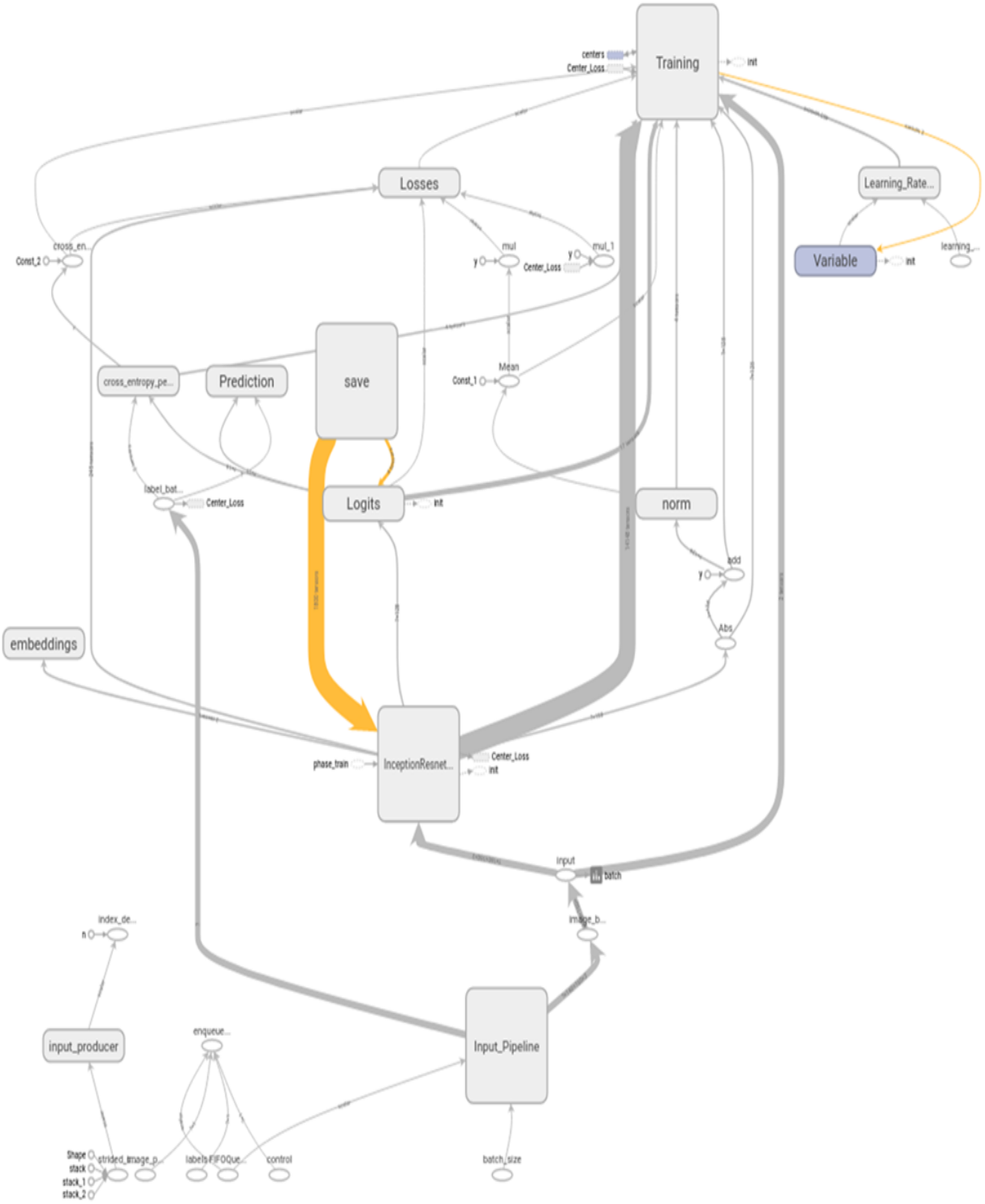
Zhang, X., Fang, Z., Wen, Y., Li, Z., & Qiao, Y. (2017). Range Loss for Deep Face Recognition with Long-Tailed Training Data. *Proceedings of the IEEE International Conference on Computer Vision*, *2017-Octob*, 5419–5428. https://doi.org/10.1109/ICCV.2017.578

Zheng, Y., Pal, D. K., & Savvides, M. (2018). Ring Loss: Convex Feature Normalization for Face Recognition. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 5089–5097. https://doi.org/10.1109/CVPR.2018.00534

ANNEX I – STEM OF INCEPTION - RESNET ARCHITECTURE VERSIONS 1 AND 2



ANNEX II – TENSORBOARD TRAINING GRAPH



ANNEX III – CONFUSION MATRIX