Università degli Studi di Padova

Dipartimento di Ingegneria dell'Informazione

Master Thesis in Computer Engineering - Artificial Intelligence and Robotics

# A Lightweight CNN Architecture for Face Recognition on Embedded Devices

Master Candidate

**Vascellari Filippo**

**Student ID 2019155**

Supervisor

**Prof. Pretto Alberto**

**University of Padova**

Academic Year 2021/2022
December 12, 2022

# Abstract

In our daily life, faces are one of the most familiar and common biometric features that allow to identify a person through the use of artificial intelligence techniques. For this reason, in recent years the theme of Facial Recognition has had a huge impact on several fields such as forensic investigations but also such as access management to devices or entertainment applications. Although biometric facial information is not ideal compared to other biometric traits, due to its practicality and immediacy they are experiencing a very strong technological development. Current face recognition applications are based on deep neural networks that allow greater precision and adaptability to different uses, but require, at the same time, great resources for training and inference making their use inaccessible in those products that do not have large computing power. To compete in the video intercoms market by providing additional services to the final consumer, CAME S.p.a. commissioned the design and development of a lightweight neural network for facial recognition well suited for the embedded systems of their video intercoms. These embedded systems impose severe constraints on the available computational capabilities, further limited by privacy laws that forbid the use of delocalized infrastructures. In this thesis, we addressed the Facial Recognition problem by proposing a lightweight pipeline that will be executed online in CAME video intercoms. In particular, we will exploit a Multitask Cascaded Convolutional Network (MTCNN) for the Face Detection step to detect faces in images and to align the faces, and a Siamese Network based on the ResNet-18 architecture for the Face Identification step whose training is based on a custom-built dataset. The experiments showed reasonable results despite the lightness of the architecture.

# Contents

# List of Figures

# Listings

# Chapter 1

# Introduction

The deep evolution and transformation of technologies that we have witnessed in recent decades, has led to the emergence of new needs and requirements in many areas of daily life, not only for entertainment but also for the safety of people.

One of the most important and explored research areas is image analysis, a key area for multiple applications of relevant importance. Over the last few years, the analysis of images has undergone deep evolutions, observing a radical change in the processing methods that nowadays are based on Deep Learning Networks. Deep Learning is a concept of enormous importance that allows to obtain very accurate results at the expense of a considerable amount of resources necessary both for the training phase and for the application phase, also known as inference phase.

In particular, one of the most successful applications of image analysis is Face Recognition, including the Face Detection phase and the Face Verification/Identification phase. This success comes from the technologies available after several years of research and development, technologies suitable to meet the high resource requirements needed by the methods developed and used by this field of research, and the allurement that attracts researchers from different disciplines such as image processing, neural networks, computer graphics, pattern recognition, etc. Researchers aim to solve problems of common use, such as the identification of a person from an image or a video for control purposes, i.e. identification of a person following an illegal act through surveillance cameras, or as a method of secure access to applications and sites protected by credential access through facial biometric information, i.e. access to banking applications. Researchers also have to take into account the advantages, but above all, the disadvantages deriving from the methods and biometric information used.

From a methodological point of view the greatest challenges derive from the cost of physical resources to implement the training and inference phase, the amount of time required for training and the need of a training dataset that complies with the required specifications. Nevertheless, these methods are now used also in simple entertainment applications that people use every day unaware of the algorithms and the tools necessary to perform certain tasks. These methods are deeply rooted in everyday life so that the new products launched on the market, such as smartphones, are influenced by these needs and consequently are produced by reserving some specifications for the use of neural networks. All the tasks based on specific neural networks are processed and computed into processors built specifically to enhance the neural engine through hardware blocks capable of great computing power.

Facial recognition not only allows the entertainment of the people in various applications and ensures greater security in different areas, it allows the product that implements it to overcome the competition, giving to the manufacturer the capability to provide new services. From the need to compete in the already saturated video intercom systems' market by providing an additional feature, CAME S.p.a. has requested the design and development of a neural network for face recognition in video intercom based on embedded systems and developed on a platform based on a single core processor.

The purpose of this thesis is to describe and analyze the entire training course carried out during the Internship at CAME S.p.a, which allowed the design and development of a neural network for the Face Recognition task fitting the needs and addressing the problems that will be described in the next chapters. The goal is the development of two neural networks, one for the task of Face Detection and one for the task of Face Recognition.

The first task has found in the MTCNN (Multi-Task Cascaded Convolutional Networks) the perfect candidate to identify faces in an image thanks to its high accuracy, minimum demand for computational resources and ease of use because it is implemented as Python library and it is widely used by researchers in this context of applications.

The second task required the development from scratch of a Siamese network capable of comparing pairs of input images in order to verify if they represent the same person. This comparison is carried out based on the difference, computed as

Euclidean distance, of the images' embeddings, known as features vectors, extrapolated from the subnetwork on which this Siamese is based and on a discriminative threshold computed through experiments. The subnetwork was developed starting from a Residual Network, in particular a ResNet-18, modified in order to minimize a loss function that is better suited to that network and this purpose. To search the loss function that would allow the network to obtain the best accuracy, experiments were carried out with different loss functions, in particular:

- Contrastive Loss;
- Triplet Loss;
- Contrastive Loss with Classification Loss;
- Triplet Loss with Classification Loss.

The result is the creation of a software that allows Face Detection through the MTCNN neural network and Face Verification/Identification through the Siamese neural network, respecting all the requirements imposed by an embedded system and by the company itself, obtaining excellent results in Face Recognition task.

In the following paragraph, there will be a short description of the company that allowed the design and development of this neural network and where the Internship was carried out.

The second chapter will provide the notional background to understand the choices made in the next chapter but also to understand the evolutionary leap made by these systems over the last decades.

In the third chapter the implementation path will be presented, describing the the entire development process, constraints, tools and choices that have been made during development, providing implementation details and parts of code, as well as the obstacles and problems that arose during the Stage.

The fourth chapter will contain the results of the experiments, with graphs and values, obtained during the tests carried out on the various types of neural networks and selected loss functions.

In the fifth chapter, the conclusions derived from this implementation will be placed, there will be a description of all the future implementations and ideas that are related to this development.

## 1.1 CAME S.p.a.

CAME[1] is a leading brand and a global partner for integrated solutions, engineered for automating, controlling and securing residential, public and urban environments, resulting in intelligent and healthy living and working spaces for people. The Group designs and produces entrance automation, video entry systems, climate control systems, home automation, burglar alarm systems and sectional doors, for residential and industrial environments. It also offers solutions for large-scale projects and urban planning, systems for managing automatic parking facilities, access control and the security of public areas. With a history of almost 50 years behind it, the CAME Group was founded by Paolo Menuzzo. From its Headquarters in Dosson di Casier, in the province of Treviso, CAME coordinates 10 Research and Development centres and 11 production plants in Italy, France, Spain, the United Kingdom, Turkey and Brazil. The company, which has around 1,750 employees, is present in the market with branches in 20 countries and operates in 118 countries worldwide through business partners and distributors.

CAME

**Figure 1.1:** CAME's Logo.

One of CAME's main objectives, and the basis of the company's business methodology, is the continuous improvement of production processes, safety and relationships.

The Group invests in research and development, believing it to be an indispensable tool for continuing to increase the innovation level of its projects and global solutions: to date, the company has more than 50 patents. In addition, the commitment to R&D has a positive impact on competitiveness and guarantees the company's ability to meet market challenges, generating value for all the people in

[1]https://www.came.com/global/en

the Group and the business system.

CAME offers integrated solutions for the automation, control and security of residential, public and urban settings, which generate intelligent spaces for people's well-being. Safety, design and innovation are the main features of products designed to be integrated into any type of residential and industrial building.

CAME offers different products to cover the following areas:

- Gate Automation;
- Video Entry Systems;
- Home Climate Control;
- Automation for Awnings and Shutters;
- Smart Home;
- Garage Door and Industrial Doors;
- Turnstiles and Pedestrian Access Control;
- Road Barriers;
- Parking Facilities;
- Bollards and Vehicle Access Control;
- Roadblocker.

# Chapter 2

# Background

In this chapter we will present some preliminary material on the Face Recognition task, starting with a description of this task and its applications, focusing more on the theory on which the implementation choices of Chapter 3 are based.

## 2.1 Background Notions of Face Recognition

Every human can distinguish different identities by exploiting the face biometric characteristics that are different in each person. Humans have the ability to verify the identity of a person or identify him by comparing these biometric features of the face with images, videos, documents, and so on. The ability to replicate this capability within a computer has involved many researchers over the past three decades. This trend of growth of interest that the Face Recognition task has obtained in recent years is due to the great demand in the legal, commercial, security and entertainment sectors but also to the availability of technologies suitable for its use.

Although faces are the most familiar and common biometric features, they are not as ideal and precise as other biometric traits such as iris or fingerprints and can be highly influenced by multiple factors such as cosmetics, accessories like glasses or scarves and, finally, by light conditions or in general by atmospheric conditions if this task is carried out outdoors. At the same time, this kind of biometric feature is the least intrusive because it does not require physical contact or active involvement of the subject.

Only in 1988 the first results of considerable interest in this area were obtained with the combination of artificial intelligence and the weak theoretical tools of previous years making it, consequently, independent from any need of human intervention. In the last 30 years, multiple methods have been developed that exploit increasingly powerful tools to obtain Three-Dimensional Face Recognition Approaches. To date different methods have been developed regarding approaches for images in two dimensions:

- Holistic Approach (PCA [34], LDA [35],...): also called subspace-based algorithms exploit the assumption that any collection of images contains redundancies that can be removed with tensor's decomposition generating, consequently, a collection of basis vectors that represents a subspace that preserves the information of the original set of images, in this way each face can be reconstructed. Finally, the classification is carried out by computing the distance between the image vector and the classes described in that subspace;

- Local Approach (EBGM [40], EGM [23], ...): these methods exploit the positions, distances and angles of particular points, called landmarks, which represent the salient points of a person's face, geometric-based methods allow to compute a salience map to locate them in space and classify faces;

- Local-Texture Approach (LBP [2], LPQ [29], ...): based on the distinctive regions of the face such as mouth, nose, eyes, etc., these methods exploit the geometric relationships between these facial points that are compared through pattern recognition techniques and graph matching methods, they are also called feature extraction algorithms as they compute local descriptors for each pixel of the image;

- Deep Learning Approach (AlexNet [22], ResNet [15], ...): by exploiting the great computational power of some tools, the new learning methods of machine learning and employing subsequent Hidden-layers hierarchically organized to process information, these methods that exploit deep neural networks are able to classify a large number of unlabeled images in a robust and accurate way.

To date the fields of application of this task are many and companies such as Microsoft, Google, IBM compete each other to provide the best facial recognition technology. The main fields of current application are:

- Access Control: facial recognition has been adopted for access control mechanisms in human-machine interaction such as Secure Login, outclassing older biometric methods such as fingerprints and irises thanks to the less intrusive approach;

- Surveillance: defined as close observation and monitoring criminals, these systems are created to meet the security despite the challenging task to obtain a fully automatic system;
- Entertainment: face recognition has been included in many applications of daily use for mere entertainment for face modifications through filters or in the most complicated virtual realities and human-machine interactions;
- Law Enforcement: face recognition is deeply use to identify criminals or to find missing people for example investigating hours of video.

Despite its wide use, the Face Recognition task is subject to several challenges that reduce its accuracy, these challenges are attributable to:

- Pose Variation: when the subject is in an uncooperative environment, changes or rotation of that subject reduce the capability of the network to correctly identify the face, for this reason multiple poses of the same person are collected although this solution can be applied only in some real-world application;
- Illumination Variation: illumination with its random changes or lighting effects are uncontrolled for machine intelligence due to skin reflectance properties, multiple camera sensors, resolution effects and environmental conditions, these led to a reduction of the classification precision;
- Occlusion: the presence of objects such as sunglasses, mask, helmet, etc involves the occlusion of important facial features with the consequent and drastic reduction of face recognition performances because some features are not correctly identified;
- Aging: over the years, human faces undergo drastic changes in a nonlinear and inconsistent way, making the recognition complicated.

When we talk about the Face Recognition task, we are actually talking about an activity that is based on a pipeline of subfunctions with specific purposes:

- Face Detection: it is the first step that determines the presence of face(s) within the image, sometimes providing some important information such as bounding boxes or facial landmarks. These information are vital in order to align the face in the image;
- Feature Extraction: it is the second step of the pipeline and consists in the extraction of a feature vector called embedding or encode that best represents the face identified in the previous phase;

- Classification: the last step is divided into verification, i.e. the matching of one face to another to verify an identity, or identification, i.e. comparison of a face to several other faces in order to give an identity to the input one.

Although apparently they seem to be three distinct phases, sometimes they are not separable because the features are used both in the Feature Extraction phase and in the Face Detection phase, consequently many times they are performed simultaneously. The accuracy of these steps is reduced due to the challenges mentioned above but also due to the size of the database, the presence of noise or blur in the images and other small factors. Each step of this pipeline is considered a critical research issue because it needs continuous improvement and is essential in several applications. For this reason, the execution of these steps is mainly entrusted to particular deep learning techniques described below in this chapter with reference to the Face Detection and Feature Extraction phases.

## 2.2 Multitask Cascaded Convolutional Network

Computer vision algorithms for Face Detection such as the algorithm proposed by Viola and Jones that uses Haar-Like features and AdaBoost [36] to train cascaded classifiers can achieve good performance even though they degrade due to the great visual variation of human faces in real-world applications, better performances have been obtained through deformable part models that require a lot of computation resources. Recently Convolutional Neural Networks (CNNs) have been applied with great success to different computer vision tasks but due to their complex structure the approach through this tool is time costly (among others [24, 41]).

Another task of vital importance is the Face Alignment which involves an improvement in terms of precision for the network that implements Face Recognition, this task can be carried out through regression-based methods or template fitting approaches. While CNNs for Face Detection require the calibration of bounding boxes and the correlation between them and the localization of landmarks that involves a greater consumption of resources to be carried out, CNNs for Face Alignment require as many resources without exploiting the strong correlation that exists between the two tasks that requires the technique of hard sample mining to strengthen the power of detector.

The purpose of the Multitask Cascaded Convolutional Network [44] is to propose a framework to unify the two tasks and exploit a smaller number of resources by using a unified cascaded CNNs by multitask learning consisting of three stages: the first stage quickly produces a candidate through a shallow CNN that is refined by a second and more complex CNN rejecting all the non-faces windows in the image, finally it outputs the facial landmarks representing the positions of the mouth, nose and eyes through the last CNN whose structure is the most complex.

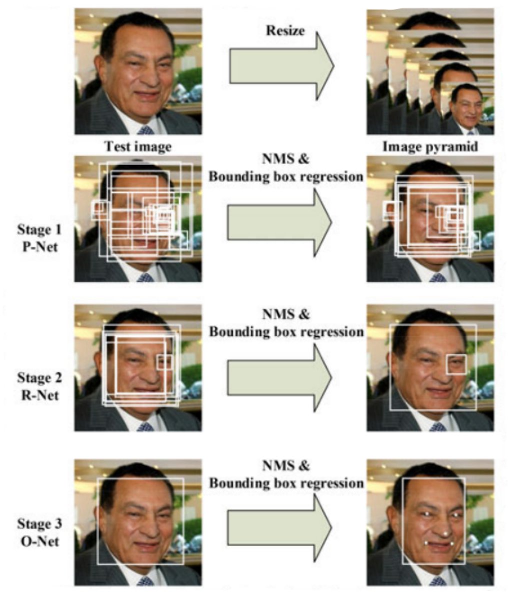


**Figure 2.1:** Stages of Face Detection and Landmarking of MTCNN [44].

The Face Detection and Landmarking pipeline exploits a cascade of 3 lightweight CNNs created specifically to require the least number of computational resources and simultaneously exploit the online hard sample mining to improve detector performance. A pre-processing is carried out on the input image to obtain a resize to different scales so as to provide an image pyramid as input to the CNNs cascade. The first stage uses the Proposal network (P-Net) that is a fully convolutional network to obtain the candidate faces and their bounding boxes regression that allow the final calibration of the candidates to whom are applied the non-maximum suppression to merge the overlapped candidates. In the second stage the candidates produced by the first network are provided as input to a second network, in particular a CNN, called Refine Network (R-Net) that performs a refinement by discarding false candidates or non-faces and performing again the calibration and the non-maximum suppression. Finally, in the Output Network (O-Net) the same refinement of the R-Net takes place but with more supervision in order to output the bounding box and the 5 facial landmarks.

The network structure uses 3x3 convolutional filters to obtain better performance with less runtime and applies a PReLU as a nonlinear activation function after the convolution and fully connected layers.

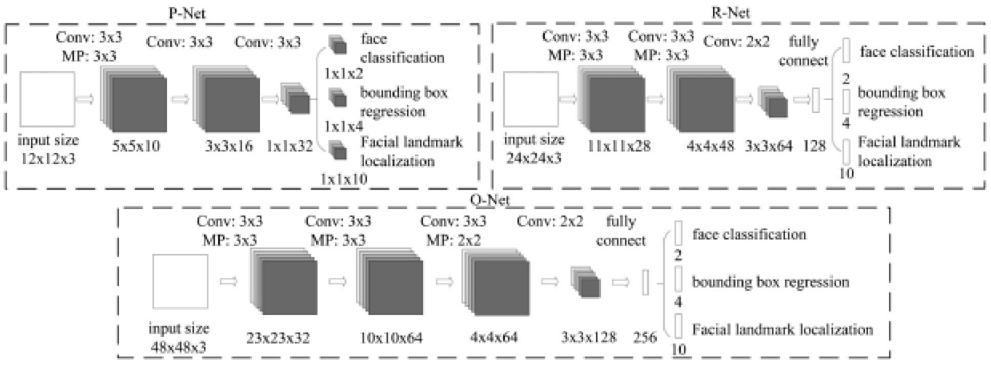


**Figure 2.2:** Architecture of CNNs in MTCNN[44].

## 2.3 Residual Network

Thanks to deep convolutional neural networks, the image classification task has managed to achieve excellent accuracy results, but many studies have highlighted the crucial importance of network depth which becomes an important challenge due to the vanishing gradient problem that hinders convergence from the beginning.

During the back-propagation phase, the error and gradient value are computed to update the weights of the next layer starting from the last layer up to the input layer. During this process the gradient becomes smaller at each layer, it is possible that the weights of the initial layer are updated slowly or remain the same making their update useless and preventing the convergence of the network with consequent degradation or saturation of accuracy.

This problem was addressed by normalizing initialization and intermediate layers that allow networks to converge with Stochastic Gradient Descent and back-propagation. Despite this resolution, as the depth of the networks increases the accuracy saturates and degrades rapidly and is not caused by overfitting but by the number of layers that makes complicated the optimization.

The aim is to achieve a deep neural network that performs well or at least similar to shallower networks. For this purpose there is a construction solution that consists in using copies of the layers of the shallower network while the additional layers are identity mapping, resulting in the existence of a network that should produce a training error less or equal to the same network with a smaller number of layers, but the current optimizers/solvers are not able to achieve this goal.

The problem is solved through residual learning that forces stacked layers to fit a residual mapping which is easier to optimize than unreferenced mapping because it is easier to push the residual to zero if an identity mapping is optimal instead of fit its by a stack of nonlinear layers. To make this happen, feedforward shortcuts are created to perform the identity mapping and their outputs are added to the outputs of the stacked layers without increasing the number of parameters and without requiring more computation, still allowing training with SGD and back-propagation.

The purpose of traditional networks like AlexNet [22] is to try to learn the output functions directly. By computing the error and gradient, through back-propagation the network learns to approximate these functions, but if "multiple nonlinear layers can asymptotically approximate complicated functions, then it is equivalent to hypothesize that they can asymptotically approximate the residual functions" [15]. In this way stacked layers can approximate a function that allows the reduction of vanishing gradient.

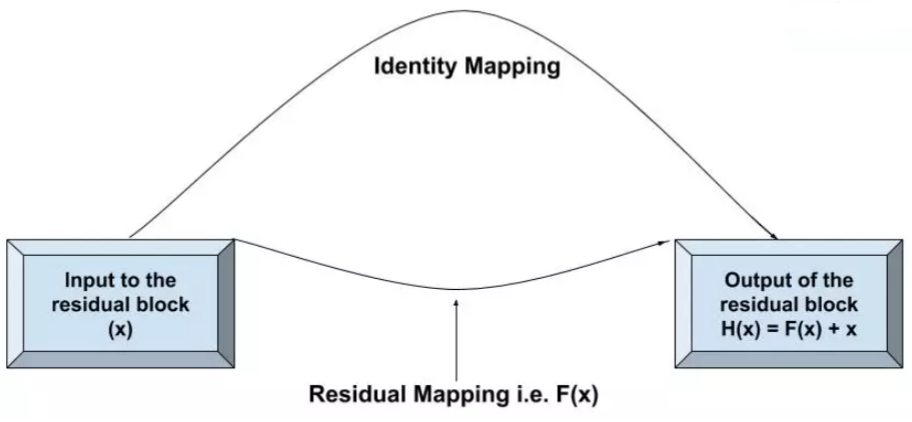

Identity Mapping
Input to the residual block (x)
Output of the residual block H(x) = F(x) + x
Residual Mapping i.e. F(x)

**Figure 2.3:** Example of Residual Mapping.

The residual function, also named residual mapping, is the value that will be added to the input of a specific function in order to best approximate the final function, i.e. the difference between input and output of a residual block[1]. In this

[1]Residual block is composed by a certain number of layers and their weights.

way stacked layers learn to approximate the residual function $H(x) - x$ instead how to approximate $H(x)$ that is the final function of each layer. Learning the residual value such that it approaches to zero means making the identity mapping optimal, in this way the overall accuracy can be increased because every layer can produce optimal feature maps.

During the propagation there are two ways for the transit of the gradient from the output layer up to the input layer while traversing the residual block. When the gradient pass from the path called Gradient-Pathway2 in Figure 2.4 there are some weighted layers to be updated, the weights of the kernel are updated and a new gradient is computed. Choosing this path until the input layer means there could be the vanishing gradient, to solve this problem the gradient could pass through the Gradient-Pathway1, i.e. the identity mapping, in this path it doesn't encounter any weight layer and it will not be updated, involving no computation of new gradient. Without update the gradient, it can reach the input layer without the possibility to vanish.

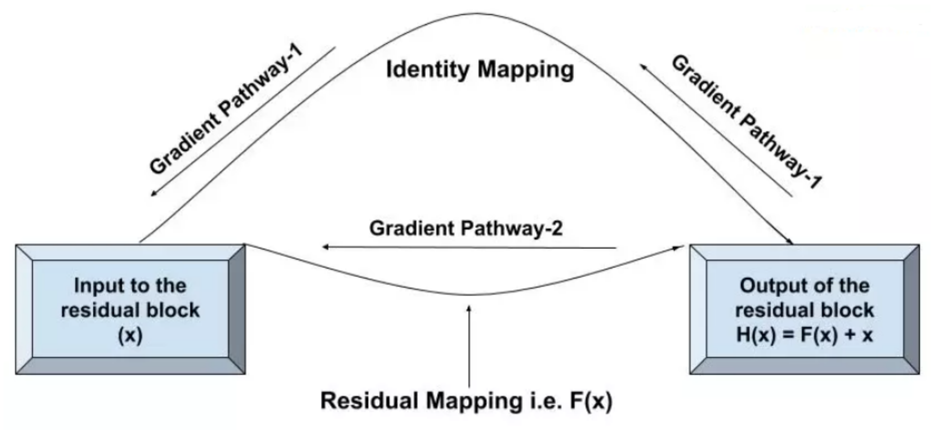

Identity Mapping
Gradient Pathway-1
Gradient Pathway-1
Gradient Pathway-2
Input to the residual block (x)
Output of the residual block H(x) = F(x) + x
Residual Mapping i.e. F(x)

**Figure 2.4:** Gradient Pathways in Residual Network.

The typical structure of a Residual Network takes inspiration from VGG [33], it is composed of residual blocks formed by at least two convolutional layers 3x3 following two rules: "(i) for the same output feature map size, the layers have the same number of filters; and (ii) if the feature map size is halved, the number of filters is doubled so as to preserve the time complexity per layer"[15]. Each network ends with an average pooling layer to which is connected a fully connected layer with output size 1000 to start the dimensionality reduction phase depending on the task to be performed. Identity shortcuts can be used between residual blocks with the same dimensions represented by solid line in Figure 2.5, or between layers with different dimension represented by dotted line. When the shortcut is between two layers in which the dimension increase, the shortcut can still performing the "identity mapping with extra zero entries padded for increasing dimensions"[15] without including extra parameter, or the shortcut became a projection shortcut to match dimensions by 1x1 convolutions. In both cases the stride performed is 2.

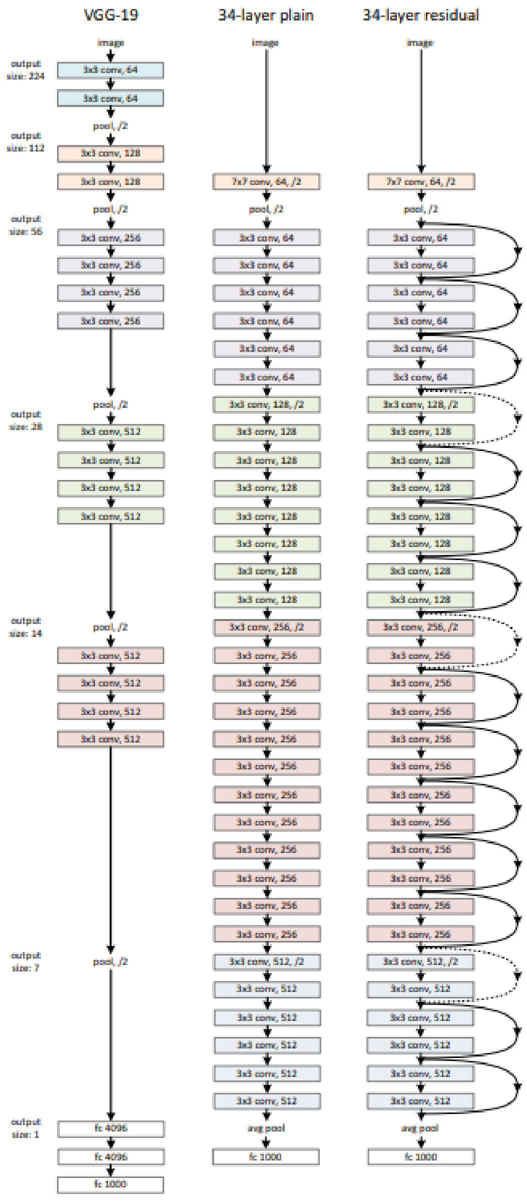


**Figure 2.5:** ResNet-34 Structure compared with VGG.

The network will be used in Chapter 3 is the ResNet-18 which is similar to the one in Figure 2.5 where for each couple of 3x3 convolutional layer there is a shortcut connection, in particular an identity mapping and zero-padding for increasing dimensions without adding extra parameters. Through the use of residual blocks, the ResNet-18 obtain an accuracy 3% smaller compared to the ResNet-34 despite is shallower, this indicates that the vanishing gradient has been accurately resolved despite the ResNet-18 allows faster convergence.

## 2.4 Siamese Network

The Siamese network [28, 6] was created to address the problems related to Classification and Regression neural networks that require a high amount of resources especially for the training phase. The classification problems typical of image processing require a trained model capable of classifying the images provided as input in specific classes to which they belong, to obtain these results the model needs labeled datasets that require considerable economic expenditure and as many computational resources to train the model on millions of images.

Most of the time these resources are not available, for this reason a further type of problem has been developed: the comparison problem. In this kind of problem the demand for computational resources can be small because the goal is to obtain a model trained on a specific similarity function that allows to measure how similar or related are two objects, greater is the similarity value greater is the probability that the two objects are similar.

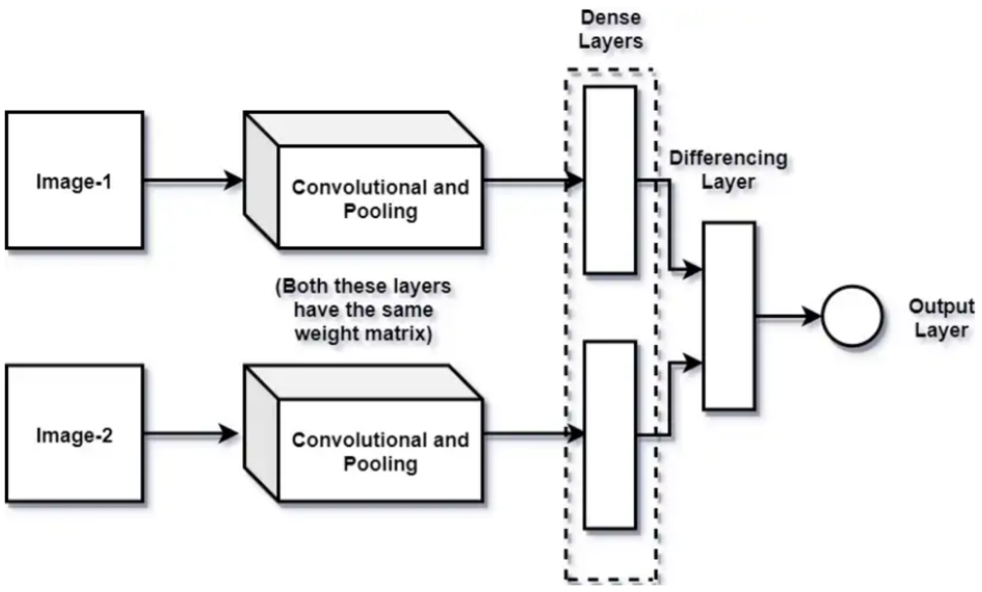

Dense Layers
Image-1
Convolutional and Pooling
Differencing Layer
(Both these layers have the same weight matrix)
Output Layer
Image-2
Convolutional and Pooling

**Figure 2.6:** Example of Siamese Network.

Tasks whose resolution is entrusted to classification models need new training with each registration of a new class otherwise it will never be recognized. Many classification tasks, however, can also be solved through comparison models that do not require new training at each insertion because they are models trained to extract a similarity value between pairs of images, without any need to compute probabilities of belonging to unknown classes. To carry out this specific task it is necessary a particular architecture that expects the analysis of multiple images in parallel, this structure is identified in the Siamese network.

Siamese Neural Network is a class of neural network architectures created for the task of Similarity Learning, their structure involves the use of two or more identical subnetworks, or neural networks that share the same configuration, the same parameters and the same weights. Each update of a parameter takes place identical in all the subnetworks that form the Siamese network. This kind of network are strongly used when we have to deal with One-Shot-Learning [10], or when there is only one image on which train the network, or when there is a need to verify similarity by comparing feature vectors for example in the comparison of biometric data such as fingerprints, faces, etc.

Based on the embeddings that the subnetworks are able to extrapolate from the images provided in input and exploiting comparison methods such as Euclidean distance or cosine similarity, these networks aim to minimize a loss function in order to obtain a refined parameter update for greater precision in comparison and similarity between images.

The main other advantages that these networks bring are:

- More robust to class imbalance: a small number of images for each class is sufficient for the network to correctly recognize and compare the images in the inference phase;
- Learning from Semantic Similarity: the focus is on learning embeddings in the deeper layer to gather together objects from the same class.

Although siamese network needs fewer resources because it is not subjected to continuous training, it may take longer time in the only training to which it is subjected with respect to a classification network because it involves quadratic pairs to learn from. In the inference phase it can require fewer resources if its behavior is modified: instead of comparing the embeddings produced by pairs of input images as can be seen in Figure 2.6, the network can be exploited a single time to extract the feature vector of the input image and compare it with the embeddings produced by other images in previous phases (Figure 2.7), in this way the expenditure of resources necessary for the computation of N pairs of images is greatly reduced, where N is the number of images into database with which to compare the current image.

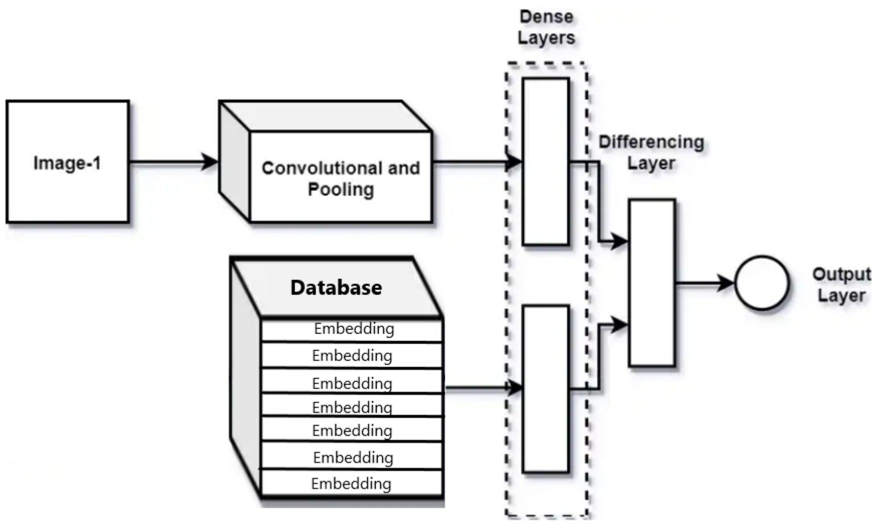


**Figure 2.7:** Different Behaviour of Siamese Network.

Finally, the loss function that the Siamese network learns to minimize must be different from the Binary Cross Entropy typical of classification networks because the type of this network does not produce probability vectors in output but feature vectors or similarity values, for this reason are used loss functions that exploit the comparison between images such as the Contrastive Loss, the Triplet Loss, N-pair Loss, NCE, InfoNCE, etc.

## 2.5 Contrastive Loss

When it comes to the Siamese network, we need loss functions to be minimized to train the weights of the layers of the network in order to perform optimally in the task for which it was created. In particular, in the Face Recognition task the most used loss functions are the Contrastive Loss [38] and the Triplet Loss [7] which are part of Contrastive Learning.

With contrastive learning we refer to a paradigm of Machine Learning where unlabeled data points are juxtaposed against each other to train the network model to be capable to distinguish if couple of data point belongs from the same class or not. Contrastive learning is based on d-dimensional feature vectors to compare data points pushing toward each other in the embedding space if they belong to the same distribution, on contrary they are pulled against each other.

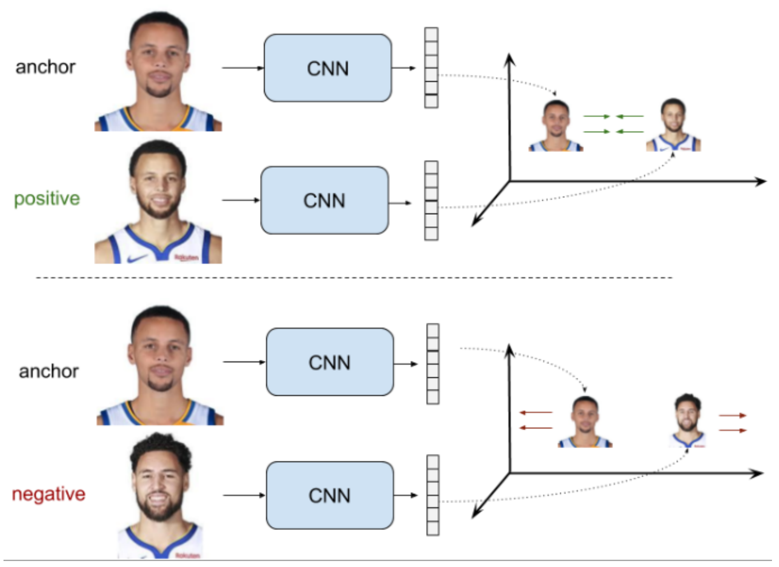


**Figure 2.8:** Contrastive Loss setup[13].

In this way, using the contrastive learning the network can learn to compare images based on the produced embeddings, its training is based on couple of images that can belong from the same class or from different classes trying to learn how to produce embedding that are very close each other when the images belong from the same object or that are very far away when the images represent different objects.

Contrastive Loss is the simplest and oldest loss function that exploits only couple of images. The purpose of this loss is to evaluates the bounty of a siamese network in distinguishing between the input image pair and minimizing its with a solver like Stochastic Gradient Descent in order to update the network to perform better and better. The formula to be minimized is:

$$Y * D^2 + (1 - Y) * max(margin - D, 0)^2$$

The Y value could be 0 or 1 if the input images belong respectively from different classes or from the same class, in this way only one the two terms could be different from zero depending also on the distance D between the embeddings and the margin to select couples of images that are harder o simpler to be compared. In this way the network can push toward embedding of the same distribution and pull other like in Figure 2.8.

## 2.6 Triplet Loss

One of the most popular loss function for metric learning or supervised similarity is the Triplet Loss. The purpose of this loss function is to give to network the capability to learn how to minimize L2-Distance between faces of the same person and enforces a margin between the distance of faces of different people.

Differently to other loss function, the Triplet Loss uses three images allowing to enforce the constraints. These three images are:

- Anchor: is the image used as the reference representing the main identity;
- Positive: is the image different from the anchor one but represent the same identity;
- Negative: is the image represents an identity completely different from the anchor one.

Using the embedding, that is a feature vector in a d-dimensional Euclidian space living on a d-dimensional hypersphere, representing the image the loss ensure that the anchor image of an identity is closer to all other images of the same person than it is to any image of any other person exploiting a margin alpha to select specific triplets. This is done over the set of all possible triplet and the mathematical formula to be minimized is:

$$\sum_{i}^{N} \left[ \|f(x_i^a) - f(x_i^p)\|_2^2 - \|f(x_i^a) - f(x_i^n)\|_2^2 + \alpha \right]_+$$

For every triplets in the batch, the network computes the distance between the embeddings extracted from the input images Anchor, Positive and Negative; the distance between couple of images can be computed using the Euclidian distance or the cosine similarity. In order to minimize the loss function the network has to compute embeddings where the L2-distance between Anchor and Negative must be greater than the L2-distance between Anchor and Positive up to the margin value.

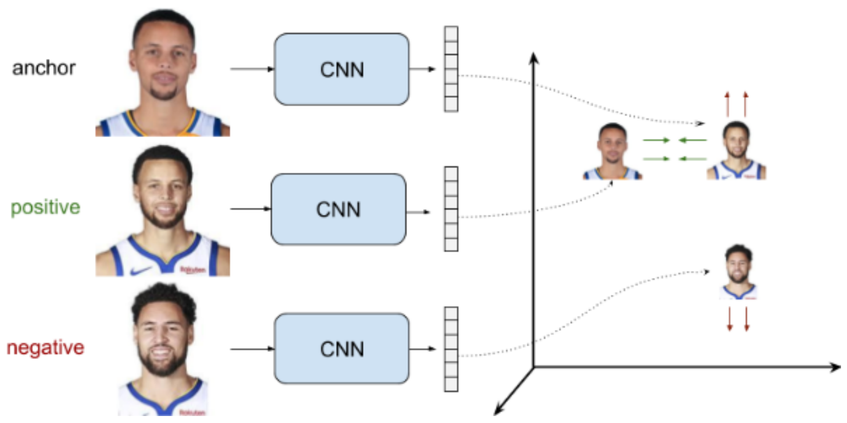


**Figure 2.9:** Representation of Triplet Loss behaviour[13].

In the set of all possible triplets there are some of them that are easily satisfied, i.e. the distance between Anchor-Positive and Anchor-Negative satisfy the constraint. Those triplets would not contribute to the training and result in slower convergence, for this reason is crucial to select hard triplets through an adequate selection method but also with an adequate margin. Depending on the margin is possible to select different types of triplets:

- Easy Triplets: $d(a, n) > d(a, p) + m$ where the negative sample is sufficiently distant to anchor sample with respect to the positive sample, in this case the loss is 0 and the parameters aren't updated;

- Hard Triplets: $d(a, n) < d(a, p)$ the negative sample is closer to anchor than the positive and the loss is positive;

- Semi-Hard Triplets: $d(a, p) < d(a, n) < d(a, p) + m$ negative sample is more distant to anchor with respect to positive sample but distance is no greater than the margin, in this case the loss is positive and smaller than margin.

To ensure fast convergence is better select the triplets that violate the constraint but it's infeasible computing all the the couples with high distance between Anchor and Positive or with low distance between Anchor and Negative, this also can lead to poor training. For these reasons there are two selection methods[32]:

- Offline mining: these triplets are generated offline every N training step by exploiting the latest newly updated weight values, i.e every N step the training is stopped to compute the distances between couple of images to select the hardest triplets;

- Online mining: using large batches, such as a few thousand images, the most difficult triplets can be computed in real time selecting them from this big batch.

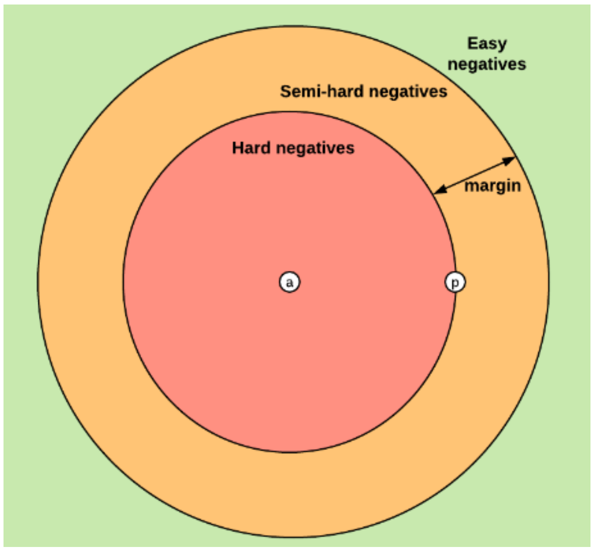


**Figure 2.10:** Triplet Selection based on margin.

Selecting the hardest triplets as the first inputs could lead to bad local minima, for this reason is better to mitigate the problem starting from the selection of semi-hard triplets.

# Chapter 3

# Experimental Setup and Code

In this chapter will be described all phases of the process to create the neural network, starting from the constraints imposed by the client company and the performance constraints of the processors used in order to understand the choices made. All the tools necessary for the development and the creation of the environment will be described, as well as the implementation choices due to the obstacles that have arisen in the development process and the solutions adopted. We will also give space to some code examples.

## 3.1 Constraints

As already discussed in Chapter 1, the purpose of this thesis is the development from scratch of a Siamese Neural Network based on ResNet-18 and on a custom dataset, but these choices derive from important implementation constraints imposed both by the client company and by the application environment.

The neural network for Face Recognition arises from CAME's need to be competitive in the video intercom market, a market stalled for what concerns new technologies, trying to providing to the customer additional services that will support additional features that will be implemented in the ecosystem of products in the future, also the compatibility with external devices such as Alexa or other home assistants could be integrated with the neural network in order to generate additional value.

To date, the video door phones produced by the company are based on two main products: the external video ring bell with the aim of recording and encoding the video captured by camera and the internal device called intercom to which the video is sent, whose purpose is not only to decode the video produced by the external terminal allowing it to be viewed on its screen, but it is also to implement all the functions of home automation and home control.

The video intercoms are based on embedded systems and develop their operation around the IMX6 single core processor. IMX6 is a processor developed eight years ago by the NXP company equipped with a Cortex - A7 CPU at 900MHz and 1GB of RAM. This processor is built around a SoM created ad hoc directly by the company, it allows the development of all the features described above, keeping the acquisition and production costs of this customized SoM low, while they don't allow the training of a neural network due to lack of adequate resources such as a dedicated graphic card or a tool created specifically for the neural engine, i.e. the Intel's Neural Computing Stick (NCS).

The lack of resources by the IMX6 processor that are necessary for the computation of a neural network, implies the need of a research to select the new generation of processors with adequate computation capacity for processing the inference phase of the neural network in well-defined timing. The choice of these processors is based on the lapel of the implementation of the Face Recognition task according to the computing needs, which provide a constraint on the choice of the network. This constraint arises from the need that the cost of supplying of the new processor from third-party companies must be below a threshold setted by the production budget in order to not increase the cost of the final product for the customer, but guaranteeing at the same time minimum performance for network inference and a surplus value thanks to the new services that could be implemented at the expense of a slight increase of the final price.

The low budget planned by the company limits the choice of possible processors, identifying a priori in the Raspberry Pi4 the solution with more computational power that could be within the maximum procurement costs, without taking into account the costs of transporting the software from the current platform to the new one. This processor, however, will limit the choice of the neural network to be implemented imposing a very important constraint: the computing resources will not be adequate to train the network when a new person to be classified will be registered, removing a priori any Neural Network for classification that requires a new training at each new registration, involving therefore the need to use a network that wants only one initial training but allows the classification of any person existing in its database even when it changes.

The need to discard a priori any deep neural network which requires continuous training, from the Convolutional Neural Network to the most recent Residual Network, is also dictated by the impossibility of using servers or clouds. Talking about Face Recognition, in order to correctly identify people through their faces, there is the need to have a database containing images of the face of those people that must be classified and to process videos containing faces of strangers protected by privacy rights.

While video intercoms encode and decode the video locally in the location site without using delocalized tools or without the need of a database containing faces of people, the classification neural network needs a database through which train himself to obtain the best weights in its nodes in order to increase accuracy in the inference phase. This could led to harm the privacy rights of the people involved if the database is saved in the cloud or, in any case, delocalized. To overcome this problem we can exploit a database locally created in the processor using its memory that will be, consequently, limited compared to the needs of the size of the dataset for network training, but this solution avoid the use of the cloud that harms people's privacy.

The use of delocalized tools for the phases of the network is always impossible due to the privacy laws, for example the use of a server with adequate computing capacity to allow continuous training will require to send from the local instrument an image both in training and inference phases and also to obtain the classification of the people. The image is extrapolated from the decoded video and contains the face of a person: this person could already be saved in the database and consequently he has provided the rights to use his image, but the image could contains the face of a stranger who will not have provided such rights. The lack of rights from a stranger consequently prevents the use of any delocalized tool necessary both for saving, such as the cloud for database, and for classification, such as the server containing the neural network for training and inference.

In order to design and implement an adequate neural network, the following issues and needs must be taken into account:

- A low-performance processor due to procurement costs and budget;
- Restrictive privacy laws that do not allow the use of servers or clouds.

- A tighten request on execution times that must be less than two seconds;
- An image from poor quality video door phones (320p or 480p);

The network will have to take into account not only the poor computing capacity of the instruments, but will have to face with a classification through frames extrapolated from a video of relatively low resolution and in any weather condition. The network will have also to process the image of a face cropped from frames derived from a poor quality video trying to obtain the best possible accuracy.

## 3.2 Environmental Setup

In this paragraph we will describe the choices regarding the tools necessary for the development of the neural network as well as the choice of programming languages and the environments used. The entire development took place on Windows platform installed on a Dell G3 15 laptop with i7-9750h CPU, 16GB of RAM and an NVidia GTX 1660Ti Mobile graphic card.

### 3.2.1 Python

In the first phase, the main programming language was identified in Python both for the creation of the neural network and for all the necessary scripts to support it. The choice of this language is not random but is focused on a series of advantages that it entails, starting from the fact that it is widely used in the embedded system of video intercom for different purposes and therefore easy to use and implement.

Python is a high-level object-oriented language used by almost 60% of developers, it is characterized by simplicity dictated by its almost didactic use, its easy learning and the consistency dictated by the concise and easily readable code. It is also characterized by great flexibility, independence from a specific platform and a large community for the support, allowing the programmer to focus only on the problem and not on the technical nuances of the language.

The choice of Python was dictated by the multiple system libraries and frameworks for Artificial Intelligence and Machine Learning that simplify the implementation of different features, the most important are:

- Keras, TensorFlow, and Scikit-learn for machine learning;
- PyTorch for deep learning;

- NumPy for high-performance scientific computing and data analysis;
- SciPy for advanced computing;
- Pandas for general-purpose data analysis.

TensorFlow is one of the most used libraries in Machine Learning because it allows the acceleration of algorithm computation exploiting all the resources of the available GPUs and using functions that allow the use of tensors, the creation and manipulation of Convolutional Neural Network models are carried out by a TensorFlow API called Keras. Keras is a library created to accelerate the development and use of Machine Learning, but it does not support the use of a wide variety of neural networks with respect to PyTorch.

PyTorch was developed as an optimized Deep Learning tensor library, it is favored with respect to the previously mentioned libraries because it uses dynamic computation graphs and it is completely Pythonic, it is able also to taking advantage of GPUs and CPUs. The main features are the computation of tensors through GPU acceleration and the creation and training of deep neural networks with automatic differentiation, thus allowing the creation of custom loops and allowing the choice of the optimizer that best suits the task. It also includes a much larger set of pre-trained neural networks on popular tasks and datasets, such as ImageNet, allowing easier and faster training through fine-tuning of the pre-trained network. It provides the DataLoader function that allows an optimized loading of the dataset: it combines the functions of the classic DataSet with those of a Sampler providing an iterable value and supporting both map-styles and iterable-style datasets with single or multi-processing loading.

In Python, moreover, there is the possibility of taking advantage of particular libraries and tools that allows to have maximum performance by exploiting the parallel calculation of GPUs. NVidia develops graphic cards with a particular architecture for parallel computing, called CUDA (Compute Unified Device Architecture), which exploits cores similar to those existing in CPUs allowing, through acceleration methods, the use of complex algorithms, very large datasets and, consequently, deep neural networks.

In the end, the choice of Python is linked to the existence of a library created ad hoc that contains one of the best and lightest neural networks for Face Detection with respect to the inference phase. This is the MTCNN implemented in the library called "facenet-pytorch" and easily imported as a function that allows the tracking of multiple faces with different levels of confidence both in videos and images. Here there is an example of how the network is imported:

```
from facenet_pytorch import MTCNN
```

**Listing 3.1:** Code to import MTCNN.

### 3.2.2 Anaconda

The development environment used for Python language has been identified in Anaconda. Anaconda is an open-source package and environment management system that runs on Windows, macOS, and Linux. Anaconda brings with it several benefits compared to other environments for Python: it is free and open-source with more than 1500 data science packages simplifying management and deployment, it easily collects data using machine learning and AI techniques but, in particular, it allows to easily create and manage the environments necessary for each project. Finally, it allows to manage libraries, dependencies and environments through "conda" from the command prompt or, much more easily, from the navigator that it provides to be more user-friendly at the expense of the speed that could have a normal "pip" call on Python.

The choice of Anaconda derives from the possibility of creating independent virtual environments, each one with its own packages and specifications. The environments can be easily selected from the GUI that Anaconda makes available. In particular, it is possible to create an environment in which to install two vital deep learning tools developed by NVidia: CUDA Toolkit and cuDNN. The first tool includes libraries to accelerate the GPU, for debugging and other optimization tools to develop applications with graphics acceleration of the graphic card, the second instead provides highly tuned implementations for standard routines in deep learning exploiting the GPU, in particular of the forward, backward, pooling, normalization and activation layers.

Thanks to the help of Anaconda Navigator, the development environment was created with all the libraries necessary for deep learning and graphic acceleration, exploitable by the different coding environments, among these JupyterNotebook was used for the possibility of dividing the code into blocks. This subdivision has allowed a greater order of the code but, above all, has provided the possibility of not executing some blocks allowing a better debugging and workflow in a first phase of learning the code, also the kernel has the ability to keep active and keep data and variables in memory even after the end of the execution of the last block.

### 3.2.3 MATLAB

At some point during the process, the development and the coding of the neural network has brought the need of a different coding environment, this environment was identified in MATLAB. Although the community of programmers and researchers is larger in terms of the Python programming language than MATLAB, this platform provides vital tools in creating and debugging a neural network.

MATLAB is a programming platform designed specifically for engineers and scientists to analyze and design systems and products that transform our world. The heart of MATLAB is the MATLAB language, a matrix-based language allowing the most natural expression of computational mathematics[1].

After Python, MATLAB is the most suitable language for programming neural networks thanks to the multitude of tools that are made available, due to this fact it is a paid platform and not open-source like Pyhton, this comes from the fact that it is used by engineers and scientists to develop, automate and integrate deep learning models into their domain-specific workflows. The main purpose of MATLAB is the development of workflows suitable for the creation and use of systems that are based entirely on deep learning, these workflows are applied to develop specific applications such as Image Processing using data preparation tools, pre-trained network models such as GoogleNet and ResNet-50, simulation and test environments such as Simulink, automatically generated and natively optimized codes for different platforms including NVidia's GPU's.

There are several tools used in the development described below, from parallel computing tools to machine learning tools. Among these there is a tool called Deep Learning Toolbox which led to the choice of this platform. Through models for neural networks, the converter for ONNX, etc. this tool provides a framework for design and implement deep neural networks with algorithms, pre-trained models and apps allowing to build CNN, LSTM, GAN and Siamese networks exploiting, in particular, automatic differentiation, custom loops and shared weights. Thanks to the Deep Network Designer, it is also possible to visualize the designed network in

[1]https://it.mathworks.com/discovery/what-is-matlab.html

graphic form and exploit the GPUs in parallel with the Parallel Computing Toolbox.

## 3.3 Development

This paragraph will describe the different phases of implementation and development of the neural network, starting from the first phase of analysis of the neural networks on the market until the implementation of the last script in the pipeline, passing through the various phases of testing and design with the problems deriving from lack of resources or lack of a small dataset specific for the purpose.

### 3.3.1 Analisys

As in any creation and development project, the first phase concerns the research on other researchers' studies on similar systems, applying to them the initial constraints born from the economic and computational needs of the company in order to adapt, if necessary, these systems to their needs.

The constraints (Section 3.1) previously mentioned were born in this phase of research, giving rise to the computational and, above all, economic limits to deal with in the production and development phase of a product to be launched on the market in order to produce the greatest possible revenue while keeping competitive in terms of price compared to other companies. These limits are dictated, as well as by the market, also by the privacy laws that forbid the use of cloud or server instruments, imposing the use of local tools that must be, at the same time, economic and performing to guarantee the computation of the neural network at least in the inference phase.

Neural networks, in general, require large amounts of resources in terms of processing, mainly in the machine that allows their training phase, requiring computational resources such as highly performing graphic cards and equally performing processors. The first constraint imposed was the limited computational capacity of the machine that was used as the development environment, whose performance depended on the NVidia GTX 1660Ti Mobile graphic card, the 16GB of RAM memory and the Intel i7-9750h processor. The computational limitation of this machine has set a unique direction of research, restricting the field of application of neural networks only to those that allow the training with a limited number of images with respect to the numbers that can be used on machines that exploit more powerful GPUs. At the same time these neural networks must allow the training with a considerable number of images in order to obtain results adapted to the needs and expectations of the company in an amount of time involving a maximum of 24 hours for the entire training phase.

Without initial budget impositions and being aware of the need for CAME to select a new generation of processors to be installed in its products, but above all having some test samples available, the first research was carried out around devices that would allow the continuous training of a neural network even after installation in a new product, those tools are Nvidia's Jetson Nano graphic card, in particular the BOXER-8221AI, and a Neural Computing Stick from Intel to be applied in a Raspberry Pi4 B.

**Figure 3.1:** BOXER-8221AI with NVidia Jetson Nano.

The BOXER-8221AI is a compact AI@Edge system powered by 4GB of RAM, 16 GB of eMMC storage and by the NVIDIA Jetson Nano which is a chip of the Jetson line dedicated to embedded and IoT products that want to take advantage of artificial intelligence. The design of this SoM allows, as stated by NVidia, to speed up processes up to 472 GFLOPs and process multiple neural networks in parallel.

**Figure 3.2:** Neural Computing Stick 2.

The Neural Computing Stick 2 (NCS2), created by Intel, speeds up the development of deep neural networks, artificial intelligence algorithms and computer vision prototypes thanks to the Movidius Myriad X Vision Processing Unit (VPU) flanked by the OpenVINO toolkit [2]. Being developed as a USB stick, the NCS2 requires no additional hardware and can be easily used via plug and play on the Raspberry Pi4.

Starting from the assumption that the number of people to be classified will be limited only to the family involving the need for a relatively small database and taking into account the great computing power declared, in particular for the Jetson Nano, the direction on which the research initially focused was a Convolutional Neural Network for the classification task. This kind of neural network is the most reliable in terms of accuracy but requires continuous training that is carried out whenever a new person is saved into the database, with its images, ready to be classified.

There are many lightweight neural networks developed in recent years by researchers such as LightQNet[4] and ShuffleFaceNet[27], it is a neural network deeply modified to reduce the necessary resources derived from the famous ShuffleNet V2[25]. These networks are built for the classification task exploiting computing resources that are easy for anyone to access, usually they are trained on laptops and gaming computers that take advantage of older internal GPUs, such as NVidia's 1000 or 2000 series, and Intel's fairly recent processors, such as the 7th and 9th generations.

Although the networks mentioned were developed by their researchers to best exploiting the low power available to them both in the training phase and in the inference phase, the power provided by the Jetson Nano and the Neural Computing Stick seemed not to be enough to allow continuous background training when a new person to be classified is registered. To confirm this statement there are multiple algorithms that do not exploit the final softmax layer of the classification neural network to compute the probability of belonging to a certain class or person, but there are neural networks trained to do Feature Extraction, or to extract an embedding of a certain size for example 128 and 512, which represents the characteristics of a face. Once an embedding is obtained for an input image into the network, this is compared to an embedding resulting from another input image into the network or to the embeddings of images saved into databases.

[2]An open-source toolkit for optimizing and deploying AI inference through processor acceleration

The behavior described above is the typical behavior of a Siamese network 2.4, it is a neural network that allows the parallel inference of one or more Convolutional Neural Networks used as a skeleton for feature extraction with the aim of comparing the vectors of extracted features and providing a similarity value between the two images provided as inputs, in other words network must be able to understand if two images represent the same person with a certain accuracy.

The algorithms developed by the researchers[11] to best exploiting the computing power of NVidia and Intel tools are based on the comparison of feature vectors using deep neural networks as backbones for the feature extraction phase, such as the ResNet-34[31] or the Inception ResNet V1 with 140 layers[18], finally the triplet loss is used as a loss function 2.6. Although these algorithms are designed to exploit the low computational power that is provided, the problem remains the training phase which, in the case of loss functions such as the triplet loss, requires that at each iteration, depending on the size of the batch, at least 3 images are provided as input to the network that uses, for example, the ResNet-34 to extract the features. In parallel, or sequentially depending on the computing possibilities, the features of the images are extracted from the reference image called Anchor, of the image containing the same person as the Anchor and called Positive and finally of the image containing a person different from the Anchor called Negative, these features are extrapolated in batches of 32, 64 or 128 triplets of images requiring a very expensive computation.

The strength of these networks exploits their ability not to learn how to diversify faces according to the people on which they are trained, but they learn how to extract features that characterize each face differently from that of a different person, thus allowing to add new classes without the worry of having to carry out further training, i.e. new people to be recognized in the database. In this case, the inference phase can be speeded up by comparing the embedding extracted from the input image with previously computed embeddings and saved into databases with appropriate search algorithms, in order to use the backbone network once to reduce the overall time and resources necessary for computation.

The need of high-power computational tools such as Jetson Nano and NCS2 derives mainly from the backbone networks that are exploited in these algorithms: greater is the depth of the network in terms of number of layers, longer it will takes to produce a response or an embedding. Greater is the depth of the network, greater will be the accuracy of the extracted feature vector, regardless of its dimensionality. In order to reduce the times that must be below a minimum threshold in a face recognition task or in any other field of application, these computational tools are exploited to obtain an inference time estimated around 3-4 seconds.

The Face Recognition task carried out by a neural network built for classification was impossible, i.e. the probabilistic selection of the class to which the input image belongs. It is impossible even by exploiting the acceleration and deployment tools of deep neural networks because it requires adequate tools that include, at least, an NVidia GPU of the 1000 series.

After a economic and market deep analysis with the top management of the Came Research & Development department concerning the possible procurement and production costs of the new generation of processors that would allow the implementation of neural networks in the inference phase, it appears necessary to further limit the procurement budget. NVidia and Intel's tools were too expensive to purchase, thereby they will increase the final price of video intercom above the threshold limit that would have made these products competitive in the market. The further reduction of the budget led to the need to select a tool that could simultaneously host the neural network and all the functions already present in the videos intercoms trying, as far as possible, to ensure backward compatibility with the products already installed at the customers.

As shown above, the final solution to the described constraints was a Raspberry Pi4 B. This processor, within its computational limits, allowed several researchers to implement pre-trained neural network models for the inference phase [30, 8, 19]. Aware of the capabilities and limitations of this tool, the research was directed towards the use of a network with a behavior similar to a Siamese network that exploited as a backbone a neural network with a limited number of layers but at the same time allowing to obtain excellent accuracy and very fast computation, finding in the ResNet-18 the ideal candidate.

Instead of using any CNN as the backbone of the network, it was decided to use the shallower version of the neural network which won the ILSVRC 2015: the Residual Network 2.3. This residual network tries to remove the vanishing gradient problem that appears when the number of layers increases, through the identity connection between layers, allowing greater accuracy than other convolutional networks such as GoogleNet, VGG, AlexNet, etc. Validated by the results obtained by several researchers in this area of application[5, 16], the ResNet-18 was the best compromise between accuracy and resources required for computation in both training and inference phases.

Once the network to be used as a backbone in the Siamese network to perform the feature extraction is selected, the research has shifted the focus on the loss function that the network must minimize during training in order to obtain final embeddings. The embedding must have a small distance from the embeddings of images containing the same person and a great distance from embeddings of images containing different people. These functions have been identified in the Contrastive Loss 2.5 and the Triplet Loss. In order to select the network with the loss function that best suits the future purpose of use, the goal was to verify the differences between these two loss functions by adding also the Classification Loss to make the final weights of the network more tighten after training.

Once the best network for the Face Recognition phase was found, a step back in the recognition pipeline was taken. Before the recognition of the face there is a need for a process to select the appropriate method to identify the presence of a person in the image, i.e. the method for the Face Detection phase. This phase was left for last because it was less constrained than the computational needs of the next steps.

For the Face Detection phase there are multiple methods that do not make use of neural networks but are based on algorithms from Computer Vision, the most famous is the Viola and Jones algorithm. These algorithms are based on features by looking for distinctive features in images such as nose and mouth, or on templates verifying if there is a match with a portion of the image, or on appearance through the use of patches and cascades of classifiers. These algorithms make their simplicity and low resource consumption their strength points, suffering of occlusions that could occur in images or videos.

In particular, the cascade face detector proposed by Viola and Jones degrades significantly in real-world applications due to the great variation of human faces, even with the most advanced features and classifiers. Recently these methods have been replaced by a Cascaded convolutional neural network known as MTCNN (Multitask Cascaded Convolutional Network)2.2 which uses three convolutional networks, namely P-Net (Proposal Network), R-Net (Refine Network) and O-Net (Output Network) to output a face/non-face classification, a bounding box regression and a facial landmark localization.

Through multi-task learning, the network seeks to integrate both face detection and face alignment by exploiting a three-stage convolutional network: the P-Net to produce candidates that are refined by the R-Net, and the O-Net to provide as output the bounding boxes and the positions of the face landmarks. The strength of this multi-tasking network is its ease of use due to the implementation as a library that can be easily imported into Python, but above all in its high accuracy and speed thanks to its ability to exploit all the resources available, identifying it as a perfect candidate in order to find the face with greater confidence in future videos and allow the alignment of the faces in the database to slightly increase the accuracy of the network for face recognition.

### 3.3.2 Dataset

Knowing the implementation constraints regarding the resources available for the network's training phase, the dataset had to comply with specific canons regarding the size in order not to overload the GPU and not make the training very long. To this end, it was decided to use a custom dataset derived from famous datasets online.

The creation of a customized dataset for specific objectives and resources from scratch requires a considerable expenditure of resources in terms of timing because it is necessary to collect a series of images with relative labels. Usually this collection is taken from the web using the images of famous people, appropriately cropped and resized in order to obtain a homogeneous dataset from a qualitative point of view. Searching for images with specific qualities and in specific positions of the face requires considerable effort as well as the phase of aligning the faces and labeling the pictures which are time-consuming and sometimes expensive.

To overcome this problem, were used famous datasets already created by some researchers and made available to the public. There are many datasets online such as Labeled Face in The Wild (LFW), WebFace260M, CelebA, IMDb-Face, CASIA-WebFace, etc, but each of them does not fully reflect the needs and constraints imposed. In particular the network needs not only to have a dataset already aligned but must be a compact dataset to avoid long training periods due to the need to use all the images for a fixed number of times, called epochs. At each epoch, the neural network uses the entire dataset to change its weights, usually the number of these epochs can vary from a few to several hundred, resulting in times of several hours to train depending on the size of dataset and computing power provided.

The datasets used in research areas with greater resources range in size from 42 million images of 2 million different people of the WebFace260M to 500,000 images of 10,600 people of the CASIA-WebFace, dimensions that the computing power of an NVidia 1660Ti Mobile GPU could not support without several weeks of training. The ideal dataset was comparable to the Labeled Face in the Wild (LFW)[17] as it presents about 13,000 images of 5750 different people of which 1680 have at least two images representing them, all the images were aligned using the Viola and Jones algorithm and manually refined at the end. Unfortunately LFW was not suitable for certain purposes as only 610 people present more than 4 images making it necessary a non-random division into train set and validation set for the training of the network that will involve an overfitting, also because the real purpose for which this dataset was created is the Pair Matching and the test of the performance of the networks, not for training.

**Figure 3.3:** LFW Pair Match Images.

Although the purpose of LFW is different from the one of the needed custom dataset, after appropriate changes in order to make it more suited to the needs of the network, was verified the poor quality of LFW as a training dataset for the pre-training of the ResNet-18 as a backbone network for feature extraction on the classification task, obtaining results below the threshold of 70% during validation. Not being able to exploit LFW for backbone network and Siamese network training, it was necessary to customize an existing and free-to-use dataset identified in IMDb-Face.

IMDb-Face comes from IMDb website from which the 1.7 million images of 59,000 different people were obtained. This famous dataset is one of the largest freely available and easy to use and download for the public, it is also the one that allowed the best customization. Since the images are directly downloaded from the

IMDb website, the images have not been refined and aligned, but they are very rough thus allowing better processing for training purposes. In particular, IMDb provides an Excel file containing the download URLs of the images, through a script people can download and divide all the images into their classes, this script was created in Python to allow a subsequent refinement with specific tools.

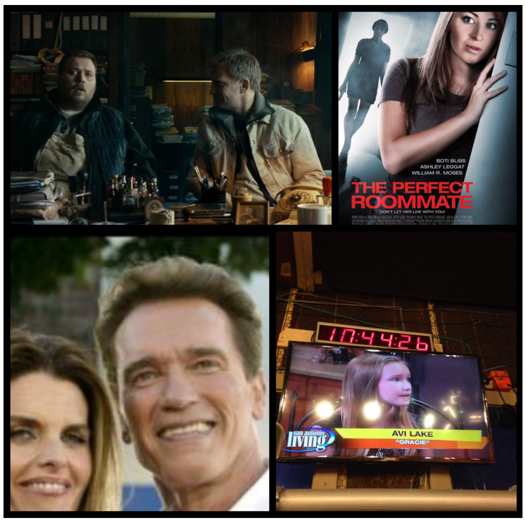


**Figure 3.4:** IMDb Wrong Images.

The images in the file provided by the creators of IMDb do not enjoy copyright and being download URLs many of these links have expired and no longer contain the downloadable images. Nevertheless, through the script it was possible to form a rough dataset containing, for the purposes and constraints previously described, about 55,000 images of 6000 people. These images are the result of a mere download, they have not been processed and contain images from frames derived from the characters' films, from movie posters and sometimes even include several people in the same image as can be seen from Figure 3.4 in which can be seen the presence of an image even coming from a television program.

The raw dataset must be refined by eliminating all classes that contain a low number of images that would not allow a good division into train and validation, before obtaining this skimming it is useful to refine the images and eliminate the unusable ones. Many of the images provided by the creators may have poor facial quality, i.e. the face is not in a position appropriate to its detection and recognition, they can also present different faces within the same image making them more difficult to train and test. In order to eliminate all those images, a Python script has been created ad hoc and it uses the MTCNN to select images that have only one recognizable face and with a recognition confidence close to the value 1 that is the maximum value, in this way the script can delete all the images in which more people appear and those images containing faces too complicated to identify and recognize.

By removing all classes that contain a limited number of images, the number of people and images drops dramatically, considering also that many photos of movie posters remain because there is a clearly visible face inside. Although the dataset remains partially raw, it allows to obtain excellent results in the training phase in the classification task of the backbone network with accuracy in the validation set around 85%.

Aware of the existence of a dataset with which the network would have obtained greater accuracy, the research shifted attention to a public dataset in particular: the CASIA-WebFace. Currently this dataset tends to be the best for Face Verification and Face Identification being able to take advantage of 494,414 images of 10,575 real identities deriving, always, from IMDb website but reworked and optimized better than the previous dataset. The creators of CASIA-WebFace have selected only celebrities born between 1940 and 2014, they have labeled each face in the photos to simplify the recognition phase and they selected celebrities with at least 15 images removing all identities present in the LFW dataset to make it compatible with this one in the test phase in order to avoid an overfitting due to a memory training by the network, in addition each image has been subjected to a phase of face detection, face landmarking and alignment to make the dataset as homogeneous as possible.

Being a large dataset, CASIA-WebFace still has a certain number of miss classified samples even if in a very small percentage, them do not greatly affect the accuracy of the training and testing phase, accuracy that can drop by 1-2% in the worst case. The dimensions do not match the constraints imposed by the available computing resources, exploiting the script in Python with the MTCNN network previously described, a refinement phase has been prepared to better define the dataset and select only the classes with a minimum and adequate number of images for a good random division into train set and validation set, obtaining a dataset comprising 212 different identities with a total of 61,500 images, despite the large number of images the training times are within the maximum terms imposed. Finally, for the training phases, the dataset will be randomly divided so that the train set has 80% of images and the remaining is divided equally between validation and test sets with the appropriate specific methods of Python and MATLAB.

Below there is a part of the script for the selection of images with specific canons without image alignment because is not necessary in CASIA-WebFace:

```
from facenet_pytorch import MTCNN

#KEEP ONLY THE IMAGES THAT CAN BE LOADED AND HAVE ONLY 1 FACE DETECTED
#BY MTCNN WITH A CONFIDENCE >= 0.999

for classPath in fileList:
    deleteFolders(classPath)
    fileRemoved = False
    torch.cuda.empty_cache()
    for imgPath in glob.glob(classPath + "/*.jpg"):
        torch.cuda.empty_cache()
        #If an image can be read by CV2 keep, otherwise remove it
        try:
            image = cv2.cvtColor(cv2.imread(imgPath), cv2.COLOR_BGR2RGB)
        except:
            removeFile(imgPath)
            fileRemoved=True

        #Keep only images with only one face detected by MTCNN and with a confidence >= 0.999
        boxes=[]
        try : boxes, confidence, landmarks = mtcnn.detect(image, landmarks=True)
        except: removeFile(imgPath)
        if boxes is None :
            removeFile(imgPath)
            fileRemoved=True
        elif len(boxes) == 1:
            if confidence <=0.99899999:
                removeFile(imgPath)
                fileRemoved=True
        elif len(boxes) > 1:
            removeFile(imgPath)
            fileRemoved=True
    if fileRemoved == True:
        deleteFolders(classPath)
```

**Listing 3.2:** Script for selecting images under some constraints

Pytorch has been used in Python through the functions of Dataset, which holds the real set of images, and Dataloader, which allows you to manage and simplify Machine Learning pipelines by iterating data, managing batches, etc. allowing rapid use of loaded data. In MATLAB, on the other hand, ImageDataStore has been used allowing a better management of the files, where each individual image fits in memory but the entire collection of images does not necessarily fit, describing them and specifying how to read them from the datastore that holds them in memory.

### 3.3.3 Face Detection

In anticipation of the application in video intercoms, the Face Detection phase will have to take place by processing the frames of the decoded video coming from the camera of the external system. The quality of these videos and, consequently, the number of frames to be processed will depend on the camera installed in the product depending on the price range: the video quality can range from 360p to 720p with a default number of fps (frames per second) of 25, although some product can be configured with only 10 fps.

Faster is the processing of each frame in order to identify the face with better confidence, sooner the identification phase with the Siamese network will be started. To this end, the selection of a fast algorithm that uses few computing resources is essential, discarding a priori any computer vision algorithm because it is liable to occlusions that would make vain any attempt to detect the face. The only plausible solution is a lightweight convolutional network that is able to verify the presence of a face with great repeatability and speed, the solution is identified in the aforementioned MTCNN.

This multitasking network allows not only to carry out face detection using a easily importing library and processing a considerable number of images with a fairly small amount of resources, but also allows to obtain as output at the same time the landmarks that identify the pivotal points of the face as well as the bounding box rectangle through which the face can be aligned, the same face will be provided as input to the network for Face Identification.

Ahead of the future implementation in a product intended for sale, the face detection phase must be active for the entire duration of the incoming encoded video coming from the external intercom camera in order to identify when the person will stand in front of the product and to detect his correct position and the presence of a single face to be classified. The work that this network will carry out will be parallelized as much as possible depending on the possibilities provided by the processor, processes in parallel will allow the processing of a series of consecutive frames for several reasons:

- to select the face only when the confidence is above a strong threshold, i.e. when the person is in an ideal position to be identified, this is carried out processing different images. In high-end quality products will be possible to ask to the person to place himself in a specific position through a voice or an image on the display;
- to improve the accuracy of the network by verifying the faces contained in several consecutive images in order to provide an answer only if most of the labels obtained as a response from the Face Identification network correspond to a single person. In other words the identification will takes place for a fixed number of images, if more than 50% of these identify a specific person, the person in the encoded video will be labeled with the label of the most present data called mode. This procedure, which can be defined as "Best Of", has

not yet received validation in terms of qualitative tests to verify the actual gain in accuracy, but the choice of the network was based on the ability to process the information that could be required by this technique.

Considering also that the video capturing device of the intercom encodes a video of fixed duration that goes from 15 seconds to 180 seconds, but in most cases is setted by default at 60 seconds, and knowing a priori the number of frames per second and the total duration of recording, it is possible to speed up the detection phase by avoiding providing a certain number of frames as input to the network. The application cases are the following:

- Unidentified face: in case of absence of the person in front of the intercom camera, even the subsequent frames coming from a fixed number of seconds may not contain the face of the person. In the case of a number of identified bounding boxes, or squared faces, equal to 0, will not be necessary to process through the MTCNN the next N frames because the average reaction time and movement in the desired position of the person is about 1-2 seconds. This method allows to remove some frames, where the raw N frames correspond to the average movement times multiplied by the frame rate of the video;

- Face identified with low confidence: if the face is identified but with a confidence below a minimum restrictive threshold, it will mean that the person is in an extreme profile position for the neural network or there could be an occlusion. The average time to remove an occlusion, such as sunlight, or for the repositioning/end of movement of the person is about 1 second, time within the processed frames would have the same result as the previous one or with a small difference, consequently N frames could be removed without any consequence in the process;

- Face correctly identified: in this case in order to diversify even slightly the position of the face or the facial expression of the person to obtain a different embedding from the previous one in the face identification phase, it is useful not to process the next series of frames in order not to obtain an identical response. The average time for the change of facial expression and all the key points of detection of the features is 0.5 seconds, these changes are due to grimaces, sighs, blinking etc. and the methods allow to remove from the process the frames in this time span;

- Other cases: they will be treated in the final implementation phase depending on the capabilities of the system location device.

The frames selected according to the described procedure will undergo a processing in order to identify the face in them and to be aligned according to the positions of the landmarks and the bounding box provided in output from the network, in this way the line joining the eyes will always be parallel to the ground and the landmarks will be placed around the same point for each image allowing an extraction of the features of greater accuracy.

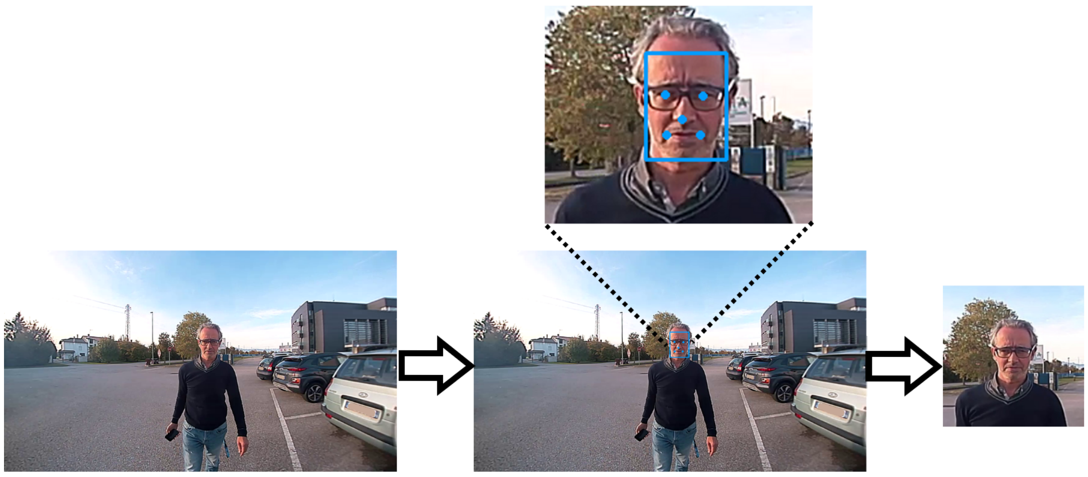


**Figure 3.5:** Example of Detection and Alignment.

The alignment of the faces recognized in the frames of the video through the multitasking network follows the specifications used by the creators of the training dataset used in the Siamese network for the face identification phase, allowing the extraction of the features in the surroundings of coordinates that are almost similar for all images, providing greater precision. In this case the alignment refers to the average measurements extrapolated from the images in the custom dataset derived from CASIA-WebFace, these measures will also be applied to align the images that will form the database of people to be recognized.

### 3.3.4 Backbone Network

The Siamese network that will perform the Face Identification task needs a backbone network, i.e. a basic network which performs feature extraction on the images provided as inputs. The accuracy of the Siamese network depends on this backbone network because from the goodness of the extracted feature vectors will depend the final accuracy of facial recognition.

This backbone network must respect the performance constraints imposed, avoiding being formed by an excessive number of layers, i.e. avoiding the use of deep neural networks of classification that count a large number of layers for the available resources, but at the same time it must be deep enough to ensure adequate embedding extraction. This implies the need to select an adequate and pre-existing convolutional network with good accuracy verified by researchers in similar areas of application and consequently removing the possibility of creating custom layer sequences that would make the backbone network even shallower and faster both in training and in inference at the expense of precision.

After a research of possible convolutional networks, the selected network was the ResNet-18, or a Residual Network composed of 18 layers, this is the perfect compromise between depth and accuracy. The initial goal was to create a ResNet-18 from scratch so that it would properly fit the needs. The problem of creating a network from scratch consists in the need of a large dataset on which to be trained to be able to obtain reasonable results, as well as the need of a very high number of epochs because the weights are updated very slowly, for example the ResNet-18 created from scratch, in a first phase, was able to obtain an accuracy of 35% on the LFW dataset during validation, results that were not adequate to the capabilities of the network.

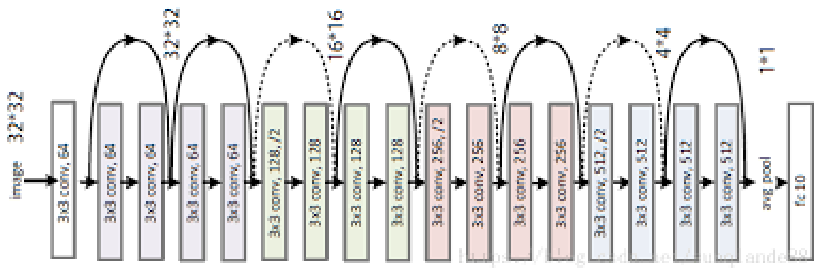


**Figure 3.6:** ResNet-18 common layers.

In order to limit the use of resources and training times and to improve the results previously obtained, the next choice was to apply Fine-Tuning to a ResNet-18 already trained that simply needs an "improvement". The fine-tuning technique consists in reusing the weights of a neural network trained on a similar task, modifying only the bare minimum of the architecture and updating the weights of some layers only. This technique is made possible by the fact that the first layers of any neural network cannot extrapolate a large amount of information, consequently the weights they use are not too binding for the accuracy in the new task, in this way the weights of the pre-trained network can be used. The so-called freezing of the layers is allowed, i.e. the parameters of a specific number of layers are made not learnable and not modifiable, usually this technique is applied on the initial layers as the final ones strongly affect the result of the network.

Using a neural network pre-trained on a similar task allow to modify only some layers and train the network with better results and with a smaller amount of resources. In this case the fine-tuning is carried out on the ResNet-18 trained on the object classification of 1 million images in the ImageNet database, those images are classified into 1000 object categories. Due to the quite similar Image Processing task and to the great accuracy of the network obtained by ImageNet, is required an architectural modification of only the final classification layers to obtain a result capable of classifying the 212 people in the custom dataset created previously. The need to use the pre-trained ResNet-18 on ImageNet led to the use of the Pytorch library being the only one to implement this pre-trained network, requiring a custom loop for the training phase.

```
model = models.resnet18(pretrained=True)

featuresNumber = model.fc.in_features

#Freeze layers' parameters
for module, parameter in zip(model.modules(), model.parameters()):
    if isinstance(module, nn.BatchNorm2d):
        parameter.requires_grad = False

#Modify ResNet-18 architecture
headModel = nn.Sequential(
    nn.Linear(featuresNumber, 512, bias = True),
    nn.ReLU(),
    nn.Dropout(0.6),
    nn.Linear(512, dataset.numberOfClasses(), bias=True)
)
model.fc = headModel
model = model.cuda()

#Select some training parameters
criterion = nn.CrossEntropyLoss()
optimizer= optim.SGD(model.parameters(), lr=0.01, momentum=0.9)
scheduler = optim.lr_scheduler.StepLR(optimizer, step_size=6, gamma=0.1)
```

**Listing 3.3:** Code to import and modify the ResNet-18

After importing the pre-trained network from Pytorch, architectural changes were made to make it appropriate to the classification of the people that form the custom dataset. The architectural infrastructure to be modified resided after the Average Pooling layer, i.e. the sequence of Fully Connected layers for Dimensionality Reduction, in particular for this structure it was decided to use a pair of Fully Connected Layers interspersed with a Dropout layer, that randomly sets input units to 0 with a frequency equal to te selected rate at each step during training time, and the ReLU activation function, in order to obtain a final embedding of size 212, which is a size equal to the number of classes in the dataset.

In order to measure the accuracy of the network, Cross Entropy was used as the loss function that measures the performance of a classification model whose output is a probability value between 0 and 1. The output embedding, which is a 1x212 vector, is not characterized by probabilistic values between 0 and 1 as required because an additional Softmax layer is usually applied whose goal is to transform a real vector into a vector in which the sum of its values is always 1. This Softmax layer would have come into conflict with the loss function implemented in Pythorch nn.CrossEntropyLoss() which provides, as implemented by its creators, to apply a mathematical formula equivalent to the LogSoftmax and to obtain the desired probabilistic result without applying any additional layers.

Since Pytorch does not provide a training function, after adapting the network to the required specifications the training loop was created allowing to train the network for a fixed number of epochs using the entire training dataset, also for each epoch the validation dataset is used to verify the accuracy of the network during test phase.

```python
for epoch in range(epochs):
    for inputs, labels in trainLoader:
#Set netwrok in training mode, compute the loss and update weights
        model.train()
        steps += 1
        inputs, labels = inputs.to(device), labels.to(device)
        if len(trainSampler) % 2 == 0:
            labels= labels.squeeze_()
        else :
            labels = labels.squeeze_(1)
        inputs= inputs.float()
        optimizer.zero_grad()
        logps = model.forward(inputs)
        loss = criterion(logps, labels)
        loss.backward()
        optimizer.step()
        actualLoss += loss.item()
#If the entire training dataset is used, compute accuracy on validation set
        if steps % validationStep == 0:
            validationLoss = 0
            accuracy = 0
            model.eval()
            with torch.no_grad():
                for inputs, labels in validationLoader:
                    inputs, labels = inputs.to(device), labels.to(device)
                    if len(valid_sampler) % 2 == 0:
                        labels= labels.squeeze_()
                    else :
                        labels = labels.squeeze_(1)
                    inputs= inputs.float()
                    logps = model.forward(inputs)
                    batchLoss = criterion(logps, labels)
                    validationLoss += batchLoss.item()

                    ps = torch.exp(logps)
                    _, class = ps.topk(1, dim=1)
                    equals = class == labels.view(*class.shape)
                    accuracy += torch.mean(equals.type(torch.FloatTensor)).item()
    scheduler.step()
```

**Listing 3.4:** Custom loop to train the ResNet-18 on Python

The code above is the code that develops the custom loop allowing to exploit all the images of the dataset collected as a dataloader to speed up and better manage their use. At each iteration, all the images of the trainSet are suitably adapted to the GPU's CUDA and to the necessary specifications of spatial dimensions and are used by the network. The image embedding is obtained through the forward function and its goodness is evaluated by the selected loss function. Based on the loss the backward function will update the weights using the selected optimizer: SGD or Adam. When the entire training dataset has been used, the accuracy of the network is evaluated on the validation dataset selecting the index with the highest value of the embedding produced by the Softmax function. The last step is to verify if the selected index corresponds to the index of the real class from which the image belongs.

In order to obtain the best result in terms of accuracy of the network during the classification phase, many combinations of parameters that Pytorch allows to modify have been tested:

- Fully Connected Layers: the depth of the network has been modified by adding FC Layers appropriately interspersed with ReLU and Dropout layers, in order to allow the network to learn the appropriate weights to obtain the best accuracy values;

- Learning Rate: the initial value of the Learning Rate has been appropriately modified at each training, starting from the value of 0.01 up to 0.0001 that are the most used values in deep learning, as well as the modification method of the Learning Rate in real time, i.e. the drop factor depending on the steps calculated by the optimizer;

- Optimizer: depending on the network could be used different optimization functions, the best at the moment is Adam Update that manages to solve the problems present in the other most famous optimizer like the Stochastic Gradient Descent. Both optimizers have a different scheduler that leads to the drop of the learning rate such as the StepLR that allows to drop the learning rate of each parameter by a gamma factor depending on the number of epochs;

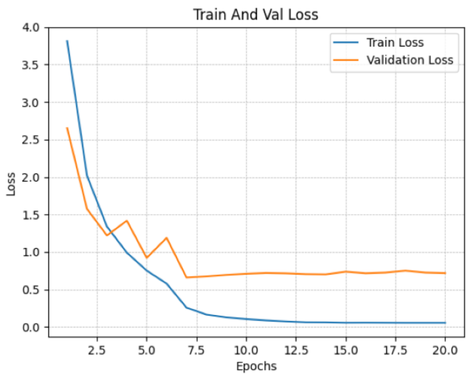
- Freeze Layers: depending on the depth of the Fine-Tuning that wants to be obtained, the layers were appropriately frozen starting from the output ones and going back to the input ones, depending on the number of frozen layers the network could train more specifically for the current task with respect to the one on which it was previously trained.

There are other important parameters that can be modified in Pytorch that do not have directly to deal with the architecture of the network, those are the number of training epochs that also strongly affect the timing of the training, but especially on the dataset used. All the tests were carried out using the different datasets of the paragraph 3.3.2, the greatest variation between the different tests results in the random division in terms of images provided to the train set and to the validation set which could vary from 25% to 10%, but also on the augmentation provided to diversify the dataset using translations, rotations, etc.

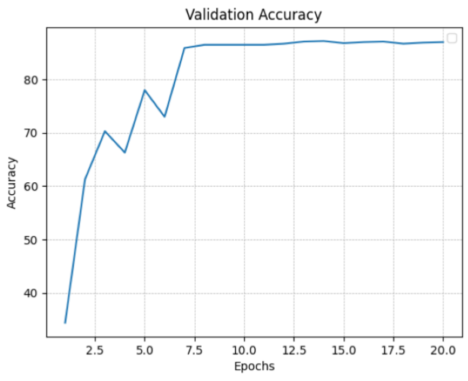
Below are the graphs that highlight the loss in the train set, the loss in the validation set and the accuracy always on the validation set. For ease of reading, the graphs show only the best training obtained with accuracy of 87% using the following parameters:

- Learning rate: 0.01;
- Optimizer: Stochastic Gradient Descent with momentum=0.9;
- Scheduler: StepLR with gamma factor = 0.1 and number of epochs = 20;
- Frozen Layers: only the first two layers were frozen;
- Additional Layers: FC, ReLU, Dropout, FC;
- Accuracy: Top 1 Accuracy;
- Batch size: 64;
- Train and Test Split: 90%-10%;
- Dataset: custom dataset with 212 classes belonging to CASIA-WebFAce with only RandomHorizontalFlip as augmentation parameter.

These results could be considered satisfactory given the shallow depth of the network and the very small dataset due to the low computing capacity of the laptop in which the training phase took place. The technique called "Top N Accuracy" also used on ImageNet involves the selection of N indices of the highest values and the identification if among these indices there is the index of the real class to which the image belongs, this method is also used by the network to obtain, in the case of Top 3 Accuracy, results over 93%, reaching 99% in the case of Top 5. Due to the fact that Python, together with Pytorch, is an open-source code easily modifiable as can be seen in the customization of the training loop, a series of errors could occur.



**(a)** Plot of Train and Validation Loss.



**(b)** Plot of Validation Accuracy.

**Figure 3.7:** Training results of ResNet-18 on Pyhton.

In order to verify the real behavior of a classification network during Fine-Tuning, we decided to use a code that could be defined "closed-source" and is identified in MATLAB. Unlike Python, MATLAB provides specific functions that cannot be customized for network training and fine-tuning classification. Once the dataset and the pre-trained ResNet-18 on ImageNet are properly imported, the final architecture was modified freezing the same number of layers as in Python in order to perform very similar tests.

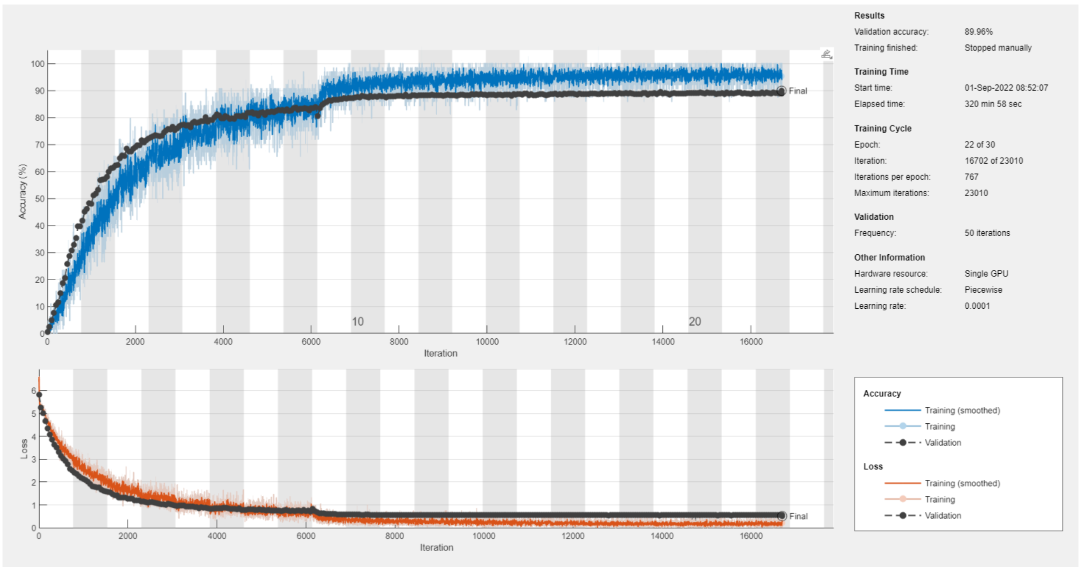
As for the training in Python, also in this case different tests were carried out by modifying the available optimizers, the learning rate drop factor period and its initial value, the layers following the average pooling were also modified in order to obtain the best result which, thanks to MATLAB, is 90% in the validation set.

```
%Set the new network's architecture
newLearnableLayer = [fullyConnectedLayer(512,'Name','in_fc'),...
    reluLayer('Name','reluFinal'),...
    dropoutLayer(0.6),...
    fullyConnectedLayer(numClasses,'Name','new_fc')];

%Set the train parameters
options = trainingOptions('sgdm', ...
    InitialLearnRate =0.01, ...
    LearnRateSchedule='piecewise', ...
    LearnRateDropFactor=0.1, ...
    LearnRateDropPeriod=10, ...
    ValidationData = augmentedValidation, ...
    Shuffle= 'every-epoch', ...
    MaxEpochs=20, ...
    MiniBatchSize=64, ...
    Verbose = true, ...
    Plots='training-progress');

net = trainNetwork(augmentedTrain,lgraph,options);
```

**Listing 3.5:** Code to train ResNet-18 on MATLAB



```
Results
Validation accuracy:          89.96%
Training finished:            Stopped manually

Training Time
Start time:                   01-Sep-2022 08:52:07
Elapsed time:                 320 min 58 sec

Training Cycle
Epoch:                        22 of 30
Iteration:                    16702 of 23010
Iterations per epoch:         767
Maximum iterations:           23010

Validation
Frequency:                    50 iterations

Other Information
Hardware resource:            Single GPU
Learning rate schedule:       Piecewise
Learning rate:                0.0001
```

**Figure 3.8:** Fine Tuning results on MATLAB.

This small difference with Python is mainly due to the slight structural difference of the network which, in MATLAB, expects the presence of the Softmax layer and the Classification layer that allow to obtain the probabilistic embedding and to select the highest probability index for the class matching . Another factor is that MATLAB handles better the train and validation dataset augmentation, which differs from augmentation parameters applied on Python both in precision and applied modifications.

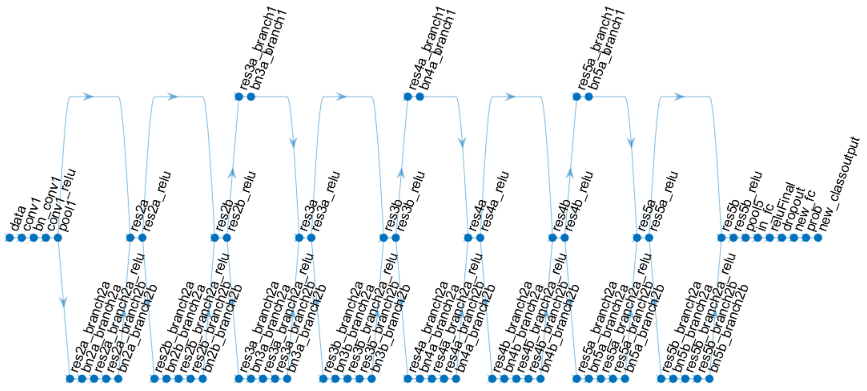


**Figure 3.9:** ResNet-18 Architecture on Matlab.

An excellent result of accuracy with a network in the classification task means having a backbone network that is able to carry out the feature extraction phase with excellent precision and it is able to obtain very accurate embeddings for each image, these led to a simplification in vector comparison in the next phase of face identification.

### 3.3.5 Face Identification

The entire Fine-Tuning procedure of the ResNet-18 served as a fundamental step to obtain a backbone network with high precision in the classification task, involving obtaining a neural network that carries out very well the job of the feature extraction. The goal of the previous phase 3.3.4 was to obtain the weights of all the layers between the input layer and the average pooling layer that are needed to obtain excellent quality embeddings, these weights will therefore be made unchangeable through the freezing of their parameters because they are already optimized.

The purpose of this last phase is to make the network adequate for face identification, i.e. to adjust the weights of the new layers that will be added so that the network will be able to verify the similarity or dissimilarity between two images provided as input, the method is based on what is the Siamese network behavior. To this end, in this paragraph, we will describe the procedure for implementing and training this neural network that bases its learning capacity on the minimization of two main loss functions: the Triplet Loss and the Contrastive Loss. These loss functions will also be flanked, for some tests, by the loss function obtained during the classification phase starting from an untrained backbone network in order to understand if the most restrictive constraint can benefit the accuracy of the network.

The development environment chosen to design such network is MATLAB. This choice was imposed by the lack of a framework that allows communication with Python and modification of the neural network. There is a unidirectional communication from Python to Matlab, but not vice versa if not exploiting the ONNX framework, this is a very powerful tool that allows communication between the two languages allowing Python to execute a neural network only in the inference phase as long as this is in a specific format. The final network for face identification needs some architectural changes in order to complete the task for which it was designed and the lack of the ability to make changes to the MATLAB network even with ONNX has forced the use of the latter platform in which no researcher has ever developed a Siamese network with triplet loss.

The goal is the creation of a network that is generally able to process two images an to provide as output the encodes of a certain dimensionality for both, and based on the similarity of these encodes, i.e. on the Euclidean distance or on the cosine similarity, and on a threshold the network must be able to understand if the images provided in input belong to the same person. This network is known as Siamese and is based on the use of two subnetworks that share the same structure, the same parameters and the same weights.

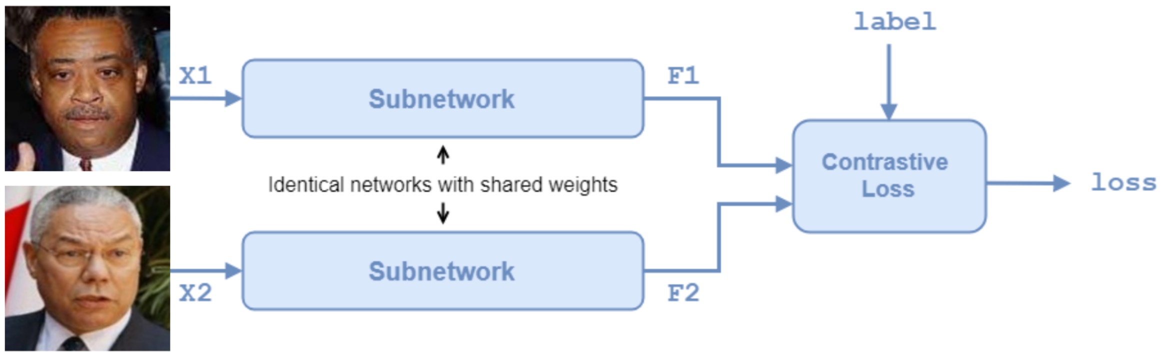


**Figure 3.10:** Structure for Siamese with Contrastive Loss.

The use of a Siamese network therefore requires a subnetwork, also called backbone network, which has the ability to perform features extraction with excellent precision. Suitably modified and retrained this network is identifiable in the pre-trained ResNet-18 on the custom dataset that was previously trained for the classification task. In order to complete the classification task, a deep neural network must be able, upstream of the problem, to extract the encode of the input image with excellent precision, this is done through the weights provided to the network before the average pooling layer, all subsequent layers are simply fully connected layers for the dimensionality reduction task, that is, layers that are needed to resize the embedding produced by the pooling layer to the required size, but in the mean time they provide further refinement by making the network deeper and with more weights to train to be more accurate.

This subnetwork will be shared in the different input branches of the Siamese network, that is, in the different rectangles called Subnetworks visible in Figure 3.10. The number of branches depends on the number of images to be processed in parallel, these subnetworks share weights, parameters and structure that will be updated equally according to the loss function that want to be minimized.

Although MATLAB is a more closed-source code than Python, in the case of training a Siamese network that uses batches formed by pairs or triplets of images, custom training loops are required both for the technical code specifications of MATLAB but also to lighten the computational load of the machine in which the execution takes place. The main purpose is therefore the implementation of a custom loop to train the network on pairs or triplets of images that are provided in input to two or three subnetworks that perform their task in parallel.

Unlike Siamese neural networks that exploit Binary Cross Entropy as a loss function, which need additional layers after the comparison of the encodes provided by the subnetwork as can be seen from Figure 3.11, the only parameters that can be trained in Siamese networks with triplet loss or contrastive loss are the parameters in the layers of the subnetwork, in particular the layers that follow the average pooling in the case of ResNet-18 pre-trained to classify faces.

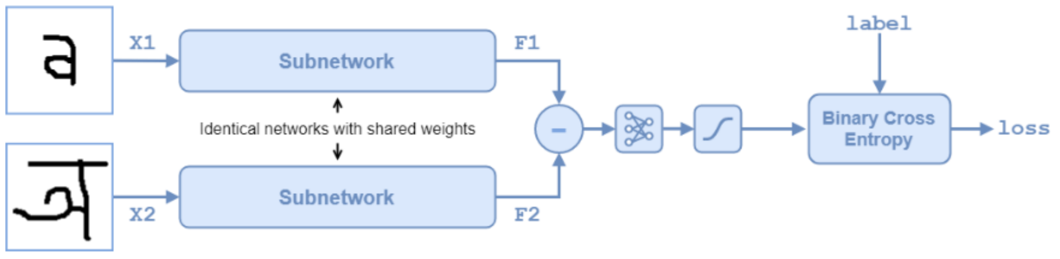


**Figure 3.11:** Structure for Siamese with Binary Cross Entropy.

The choice on the architectural changes of the subnetwork depends on the complexity of the comparison between the size of the extracted features vectors and on the depth of the backbone network. These parameters will affect the processing times but also the precision: larger is the size of the embedding, greater will be the comparison accuracy but will involve a greater expenditure of resources and time, vice versa smaller is the size, lower will be the precision but the resources required will be less as well as the timing to complete the task. After various tests on the trade-off between timing and precision, the final choice is to select the extraction of a 512-dimensionality encodes favoring the already low accuracy due to the shallow depth of the network. Following this choice, all layers subsequent to the average pooling layer have been deleted from ResNet-18 except for the only Fully Connected that will perform the Extraction feature with output size at 512.

When the structure of the subnetwork was reworked and all the layers before to the pooling layer were frozen because they were already provided with weights suitable for the extraction of the features, a method was needed to train the network based on pairs or triplets of images. The custom training loop involves the choice of the number of images equal to the size of the batch and the method to select them based on the type of loss function used:

- Contrastive Loss: since contrastive loss is a greedy loss function, it randomly selects a pair of images with 50% probability that belong to the same person or to different people;

- Triplet Loss: this loss function requires 3 images named Anchor, Positive and Negative whose choice is random, except for the Positive which must be part of the Anchor class.

```
for iteration = 1:numIterations
%Select couple of images
    [X1,X2,pairLabels] = GetSiameseBatch(IMGS, miniBatchSize);
    dlX1 = dlarray(single(X1),'SSCB');
    dlX2 = dlarray(single(X2),'SSCB');

    if (executionEnvironment == "auto" && canUseGPU) || executionEnvironment == "gpu"
        dlX1 = gpuArray(dlX1);
        dlX2 = gpuArray(dlX2);
    end

    % Evaluate the model gradients and the generator state using
    % dlfeval and the modelGradients functions
    [loss,gradientsSubnet] = dlfeval(@modelLoss,dlnet,dlX1,dlX2,pairLabels);

    % Update the Siamese subnetwork parameters using the chosen optimizer
    [dlnet,trailingAvgSubnet,trailingAvgSqSubnet] = ...
        adamupdate(dlnet,gradientsSubnet, ...
        trailingAvgSubnet,trailingAvgSqSubnet,iteration,learningRate,gradDecay,gradDecaySq);
end

function [loss,gradientsSubnet] = modelLoss(net,X1,X2,pairLabels)

%Compute the loss
[F1,F2] = ForwardSiamese(net,X1,X2);

margin = 1;
loss = ContrastiveLoss(F1,F2,pairLabels, margin);

% Calculate gradients of the loss with respect to the network learnable parameters.
gradientsSubnet = dlgradient(loss,net.Learnables);
end
```

**Listing 3.6:** Custom loop to train the Siamese Network

Once the batch of images for the current iteration of the custom loop has been selected and graphics acceleration is applied to the arrays that contain these images, the relative loss function is computed through the mathematical function assigned to it:

- Contrastive Loss: for each pair of images in the batch, the embeddings are computed using the forward function of MATLAB, which is necessary because there are Batch Normalization layers in the subnetwork that behave differently depending on the training or inference phase. The embeddings thus found are then compared with each other according to the mathematical formula, the value of the margin used is 1 which allows to give greater emphasis to the pairs of images that are more complicated to compare. Finally, the loss is normalized according to the batch size and used in the dlgradient function to compute the gradient of the function necessary for the optimizer to update the network weights. The optimizer which gave the best results during the tests is adamupdate;

```
Y1 = forward(dlnet,dlX1);
Y2 = forward(dlnet,dlX2);
delta = 1e-6;
Y = abs(F1-F2);
distances = sqrt(sum((Y).^2,1) + delta);

lossSimilar = pairLabel.*(distances.^2);

lossDissimilar = (1 - pairLabel).*(max(margin - distances, 0).^2);

loss = 0.5*sum(lossSimilar + lossDissimilar,"all");
loss=sum(loss)/numel(pairLabel);
```

**Listing 3.7:** Contrastive loss implementation

- Triplet Loss: similarly to what happens for the previous loss, for each triplet of Anchor, Positive and Negative images the embeddings are computed through the forward function that can be flanked by a sigmoid function to obtain values between 0 and 1 because there will be high values due to the square distance that is needed in the triplet loss formula. For each pairs of the Anchor-Positive and Anchor-Negative embeddings the Euclidean distance is computed, even if it is possible to use the cosine similarity to check the distances of the embeddings in order to minimize the distance between positive couples and maximize the distance between negative couples. The loss is then normalized and computed through its mathematical formula using a margin of 0.8, and then it is used to compute the gradient for updating the network weights with the adamupdate optimizer.

```
[F1,state] = forward(dlnet,dlX1);
F1 = sigmoid(F1);

F2 = forward(dlnet,dlX2);
F2 = sigmoid(F2);

F3 = forward(dlnet,dlX3);
F3 = sigmoid(F3);

Y = abs(F1 - F2);
Z = abs(F1 -  F3);

Y=sqrt(sum((Y).^2,1) );
Z=sqrt(sum((Z).^2,1) );

    A=Y.^2-Z.^2;

loss=max(Y+margin,0);
z=0;
for i = 1:length(loss)
  z=z+(loss(i));
end

loss=z/numel(Y);
```

**Listing 3.8:** Triplet Loss implementation

In this way the custom training loop, which trains the network and the non-frozen layers for a fixed number of iterations, has been defined to obtain the best result in accuracy. The tests were carried out by modifying the main parameters such as:

- Optimizer: the optimizers used for the training phase were AdamUpdate and SGDMupdate which requires a few more steps to be computed;
- Learning rate: the learning rate has been appropriately modified between values 0.01 and 0.0001 in order to test the most suitable value for the purpose of the network;
- Batch Size: although with great limitations due to the computing power available, depending on the loss used that could requires a greater number of images, the batch size could reach up to 64 for the contrastive loss and 32 for the triplet loss;
- Dataset: the main datset is the custom made dataset from CASIA-WebFace, but an additional dataset from CASIA-WebFace has been used that contains different people;
- Network architecture: the layers after the pooling layer have been modified at each training to understand the possible performance increase with respect to a deeper network;

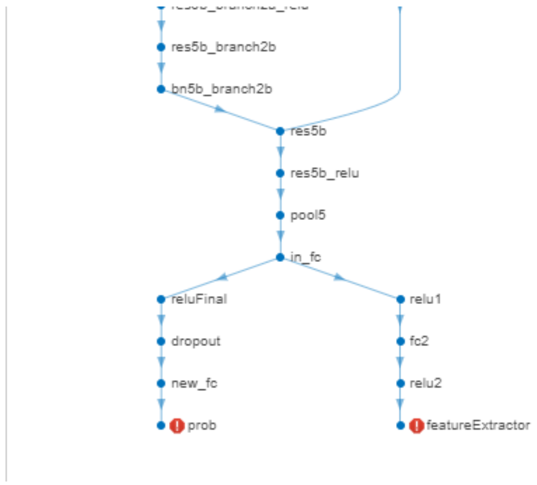


**Figure 3.12:** Branched Subnetwork.

Finally, a further constraint has been added during training: the Classification Loss. With the aim of making the training of the network more complicated to obtain more refined weights, the Classification Loss has also been added to the loss functions previously used requiring a rather complex architectural change. In order to use the classification loss all the layers of the subnetwork must have their own parameters with the possibility of being modified. In addition, having to perform at the same time both the classification and extraction of features to minimize the loss function, the subnetwork needs to be branched, doing so each branch is specific for a given task.

As can be seen from the Figure 3.12, the subnetwork's architecture has deeply changed obtaining a branch immediately after the Fully Connected layer "in_fc". From this layer the left branch keep the classification part of the ResNet-18 up to the Softmax layer because the MATLAB dlnetwork (Deep Learning Network) does not allow the use of output layers such as the classification layer, while in the other branch the layers necessary for the feature extraction of the network have been added. The same architecture has been used for both Contrastive Loss and Triplet Loss.

While the training loop keeps almost identical to what was implemented for the subnetwork without Classification Loss, in this case the mathematical formula for computing the final loss changes. In both loss functions their mathematical computation remains unchanged, to them will be added the contribution of the classification loss that varies depending on the number of images that are used:

- Triplet Loss: in this case we continue to use batches but with smaller size due to a greater demand for resources, these batches formed by triplets of images must be classified using the left branch of the network. Through the forward function of MATLAB it is possible to select the layer from which obtain as output the desired encode for each image of the triplet, if these encodes are provided by the feature extraction layer they will be computed with the mathematical formula of the triplet loss, if they are provided by the softmax layer through the help of the onehotencode function [3] they will be computed by the binary cross entropy loss one for each of the three images in i-th triplet in the batch. The sum of the classification loss of the three images will be added to the triplet loss, triplet loss will give a contribution of 50% because its importance in updating the parameters of the entire subnetwork is less than the loss of classification[43]. The final loss given by the sum of

[3]OneHotEncode function replaces each element of a vector containing the labels with a numeric vector of length equal to the number of unique classes in the label vector. The vector contains a 1 in the position corresponding to the class of the label and a 0 in every other position.

the previous losses will be used to compute the gradient necessary for the optimizer to update the parameters;

- Contrastive Loss: the mechanism of operation is similar to the Triplet Loss, except for the use of pairs of images, while the mathematical formulation remains the same and the Contrastive Loss contribution is weighted only 50% in the training phase.

```
[C1,state1] = forward(dlnet, dlX1,'Outputs','prob');
labels= onehotencode(label1,1);
loss1 = crossentropy(C1,labels);
```

**Listing 3.9:** Classification Loss implementation

One of the biggest problems of using a custom loop in MATLAB with adamupdate or sgdmupdate as optimizers is the presence of Batch Normalization layers. Unfortunately, the optimizers developed by MATLAB are not able to update the state, i.e. the TrainedMean and the TrainedVariance, of the Batch Normalization layers whose behavior differs between the training phase and the inference phase. Updating them is vital for good accuracy because when MATLAB's predict function is used to obtain the embedding in the inference phase, the last state in which the Batch Normalization layers were updated is used, if this update was done in an unbalanced way such as using only one image in the couple or triplet, the results of the inference phase will be wrong.

Without using the classification loss this problem does not arise because the state of the Batch Normalization layers remains the one obtained during the training phase for the subnetwork classification task that is optimal for the prediction phase, this is due to the fact that all the layers upstream the average polling are frozen. Being that the state of the Batch Normalization layers derives from the forward function and depending on the loss that is used, the need was to understand how to compute the correct state and which forwards depended on because there will be six forward employed in the worst case.

Since the state of the entire network upstream the pooling layer depends on the classification of the images, to update the states of these layers a script has been created that allows the computation of an average state determined by the forwards of the classification during training leaving out the forwards of the Contrastive Loss or Triplet Loss that have the purpose of updating only the right branch of the network.

An additional MATLAB problem arises when the network is branched as one of the branches cannot take advantage of GPU acceleration, requiring several days to complete the necessary update iterations.

Despite the problems described above, different tests were carried out by modifying the dataset, the learning rates and the subnetwork.

### 3.3.6 Face Recognition

The last step of the face recognition pipeline is to label the input image comparing it with the images in the database. For this purpose, in the first instance, there is the need to create the reference database with images of the people that must be recognized.

Considering the computing power available and the tight timing of the pipeline that must be under 2 second, it was decided to limit the number of images representing each person to a minimum of 5 and a maximum of 10. The imposed limit is necessary to have an adequate number of images for comparison but allowing, at the same time, to cover all the positions of the face in the various profiles also allowing the use of accessories such as glasses. Each person will have at least one image in front and one for each side of the profile, if the person needs the aid of glasses the three main templates will be duplicated to address the presence and absence of these accessories, as can be seen from the Figure 3.13 in which there is one of the person to be recognized in the various positions with the presence and absence of glasses.

During production phase, this database will be created directly by the customer following the guidelines that will be imposed on him. The database will be created using either the smartphone camera that will allow the final user to connect, through the application, to the system where the neural network will be implemented and upload the images or through the photo taken directly from the video door phone to make the image resolutions equal to the one in the future recognition. The image that will be used for the database will go, in the background, through a script that uses the MTCNN to align the image and make it similar to those used in the dataset coming from CASIA-WebFace during training.

The script expects to have an input image from which to extrapolate the information of the facial landmarks and bounding boxes coming from the MTCNN, based on the landmarks of the eyes a rotation matrix is computed and through an affine transform the rotation of the image is performed to have the line joining the landmarks of the eyes parallel to the horizontal axis. Once the rotation is obtained, the image is cropped according to the dimensions of the bounding box suitably readjusted to obtain an image with the size and position of the face very similar to the images in the training dataset, finally the image undergoes to a resize action to obtain a common size suitable for the input dimensions of the networks, i.e. 224x224, keeping all the channels for RGB colors.

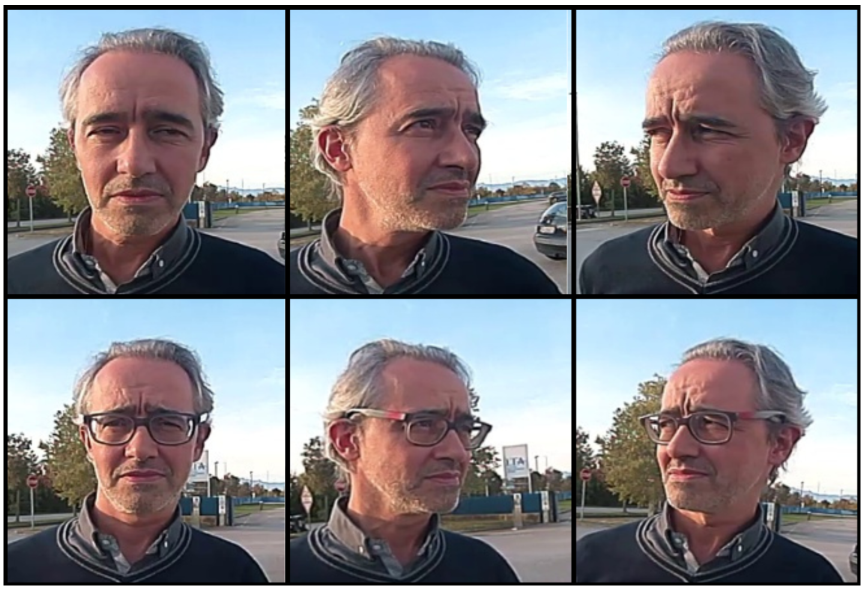


**Figure 3.13:** Example of one class in the final database.

In addition, the final database in use at the consumer will never include the presence of more than 10/15 people to be identified imposing a maximum limit on the number of images equal to 100, the images must be appropriately aligned. During the test phase, the database was derived through the script described above from the videos recorded directly from one of the video door phones available in the company, each frame of the video was provided as input to the script and appropriately modeled in case of presence of a face, the images in the final database were manually selected according to the constraints emulating a user who manually takes the correct pictures.

As known, the typical behavior of a Siamese network requires two input images from which the embeddings are extrapolated, thus requires a considerable amount of resources for the computation of each possible pair of images, in the case of a production database containing 100 images will be inserted in input for 100 times every possible pair "input image - image from database", requiring comparison times and resources infeasible for the constraints described.

To overcome this problem, the final behavior of the network will emulate the behavior of a Siamese network: the comparison between embedding pairs will always be provided, but only one image will be provided as input to the network, or the image coming from the encoded video, while the comparison embeddings will derive from the database, in this way, for each input frame the network will have to produce a single encode instead of 100 pairs. To this end, each image that is used for the database is provided, in the background, to the final network that will produce the embedding that will be saved, together with its label, in the database resulting in a considerable saving of space as strings and vectors will be saved instead of images.

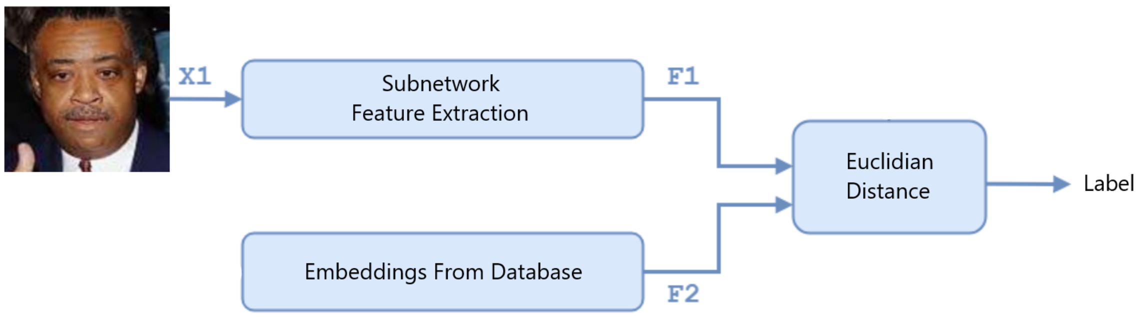


**Figure 3.14:** Structure of Network during Inference.

With this method a great speed up to the entire process is allowed, requiring a single inference phase for each selected frame. To speed up this phase even more an excellent search algorithm could be used. In a first phase of testing a search algorithm was used that aimed to reduce the comparison times, this algorithm compares the embedding produced by the network with the first two embeddings of each class in the database, if the distance between them was greater than a similarity threshold appropriately identified by different tests based on the embeddings produced by each type of backbone network, the remaining images would never belong to that class and would therefore be discarded from comparison. In this way a large number of images would have been discarded a priori making it faster to select the most similar image if there is one, but reducing the accuracy of the network in terms of False Positive and False Negative that vary depending on the selected threshold.

Since the selected threshold can be restrictive, it may happen that embeddings that have a distance greater than the threshold close to 0.01 are discarded, i.e. with threshold set to 0.24 if the distance of the first two database embeddings is 0.241 these are discarded while the third may have a distance lower than the threshold but will never computed because it is discarded a priori. Considering therefore that there will be a maximum of 100 images to be compared in the worst case and that, through the CPU, the entire execution including the extraction of features and the comparison with 62 images requires an average time of 0.04 seconds, it was decided to eliminate any search algorithm that limits the number of comparisons in favor of greater accuracy.

The final script compares all the embeddings present in the database with the one extrapolated from the input image, selecting and saving in a variable that contains the label and the numerical value of the distance all the embeddings that have a distance less than the setted threshold. In a subsequent step, from this variable, the label with the lowest distance is selected which will correspond to the person identified in the process comparing the label with the selected one in order to compute the number of False Positive and False Negative to understand the accuracy of the network as the threshold changes.

```
for i=1:row
%Extract the emebedding from the input image
    X1(:,:,:) = testImages.readimage(i);
    X =  dlarray(single(X1),'SSCB');
    imageFeatures = predict(siamese,X);
    for j=1:rowTest
%Compare with all the embedding in the database
        datasetEmbedding = embedding{j, 2};
        Y = imageFeatures - datasetEmbedding;
        distance = sqrt(sum((Y).^2,1) );
        if distance < threshold
        %If distance < threshold keep the db embedding
            spCell=spCell+1;
            searchPerson{spCell,1} = cellstr(dbLabels(j));
            searchPerson{spCell,2} = extractdata(distance);
        end
    end
%If there isn't matches but the input image label is Contained
%in the labels saved in database, there is
    %a False Negative
    if searchPerson{1,1}==""
        if ismember(testLabels(i),dbLabels)
            FNMR=FNMR+1;
        end
    else
%Compute the min distance to find the label
        mat = [searchPerson{:,2}];
        [minimum, index] = min(mat);
        person=string(searchPerson(index,1));
        boh=string(testLabels(i));
%If labels with min distance is different from the one
%in the image there is a False Positive
        if string(testLabels(i))~=string(searchPerson(index,1))
            FMR=FMR+1;
        end
%Compute the mode to select the label, if there is only one
%element for each label, take the label with min distance
```

```
        matrix=[searchPerson{:,1}];
        matrix=string(matrix);
        matrix=categorical(matrix);
        [moda,F,C]=mode(matrix,'all');
        if F == 1
            mat = [searchPerson{:,2}];
            [minimum, index] = min(mat);
            person=string(searchPerson(index,1));
            moda=string(testLabels(i));
        end
%Control the False Positive
        if string(testLabels(i))~=string(moda(1))
            FMRModa=FMRModa+1;
        end
    end
```

**Listing 3.10:** Code for Face Recognition inference phase

A problem that can arise by selecting the minimum distance is the selection of a wrong label due to an encode extracted incorrectly. Since the depth of the network is reduced to comply with the constraints imposed, sometimes the precision in the extraction of the features is not high, providing in very few cases, embeddings that are more similar to different people than to the person we want to classify. This error can occur with a frequency of 2-4% and can be avoided by exploiting the mode: having saved in a previous phase all the embedding with distance less than the similarity threshold, instead of selecting the label of the embedding with minimum distance can be selected the most present label as it would be the one with the greatest probability of matching the right identity. In the case that there is only one element for each label the script selects the minimum distance, with this technique it is possible to reduce the number of False Positives by over 50 % without involving any additional cost.

With this script the Face Recognition pipeline ends, also in this case different tests were carried out by calculating the curves representing the False Positive and False Negative to determine the accuracy of the network when the threshold varies for all networks with the different loss functions trained.

# Chapter 4

# Experiments Results

In this chapter we will describe, thanks to some graphs, the results obtained in the various phases of training and testing of the neural network, divided according to the selected loss functions. For each paragraph the results in the training phase and in the test phase will be shown with the values assigned to the various parameters. The result of subnetwork training on the classification task visible in Figure 3.7 and Figure 3.8 of the paragraph 3.3.4 will be omitted.

## 4.1 Siamese Network with Triplet Loss

The first tests were carried out using the Triplet Loss as a loss function because in the literature it is generally better performing than the Contrastive Loss. Since the ResNet-18 used as backbone is pre-trained in majority of the tests, almost all the trainings of the Siamese network for face identification were carried out with frozen layer parameters up to the average pooling layer because they were assumed to be optimal for feature extraction. In addition, the values of the loss functions are computed through the use of a sigmoid that is required by the squared distance between the couple Anchor-Positive and Anchor-Negative in the mathematical formula.

The different tests were carried out by modifying the following parameters:

- Learning rate: the learning rate values used were 0.01, 0.001 and 0.0001;
- Dataset: as the main training dataset was used the custom dataset described in the paragraph 3.3.2, for ease of use this dataset will be called Webface200, for further verification a new dataset was created from CASIA-WebFace, this dataset was called WebfaceSiamese and contains 56.515 images of 428 different identities, these identities are completely different from the one contained in the Webface200;

- Network Architecture: the sequence of layers after to the average pooling layer has been adapted to the different tests that were carried out based on the dimensionality of the feature vector that must be extracted;
- Other Parameters: in some tests the ResNet-18 was used without fine-tuning, for this purpose the layers were not frozen.

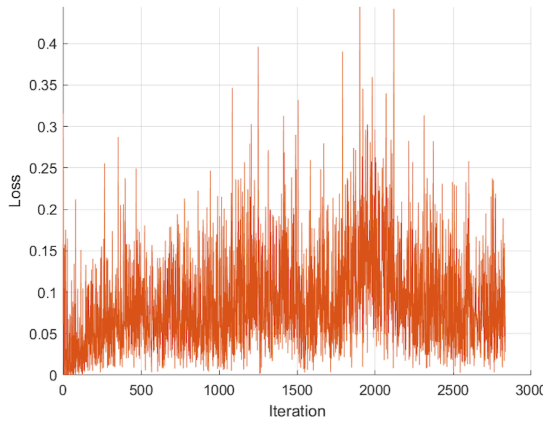
During the entire training phase, the margin to compute the Triplet Loss was set at 0.8 which was the best in a series of tests carried out but they are not reported in this section. The results of all possible combinations will not be reported for practical reasons as each training takes from 4 to 8 hours to be carried out in the best case, only some combinations of the training parameters will be corroborated giving more space to those that have obtained the best results.

Below will be reported the graphs of the training deriving from the different combinations of the parameters that have been selected using the Siamese network with triplet loss, the descriptions of the architectural changes will only concern the structure following the average pooling layer, the descriptions upstream of this layer will be deliberately omitted.

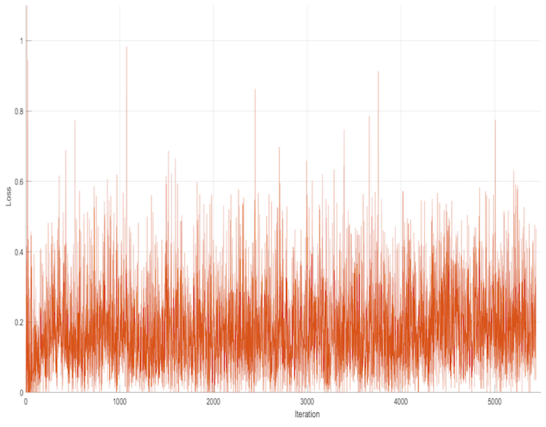
The training trends visible in Figure 4.1 derive from a Siamese network that uses the same architecture, i.e. only one Fully Connected Layer after the Average Pooling to extract features of dimension 512, but the learning rate parameter varies. The figure shows an oscillating and non-parabolic trend of the computed loss function value, which therefore represents the worst cases of training with the Triplet Loss.

In these graphs the loss function has a growth trend that is the opposite of the expectations because the expectation is a decreasing or a constant trend. This behavior is due to the fact that the single FC layer for feature extraction during the various iterations uses triplets of images that are increasingly difficult to compare and it fails to update its weights correctly, this led to increase the loss function because the layer weights remain about the same with each update.

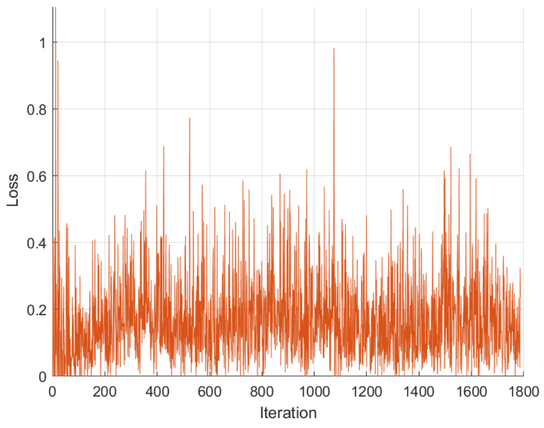
To solve this problem, the network structure has been deepened by adding some Fully Connected layers to obtain a greater number of trainable parameters allowing a greater refinement of the embeddings produced by the network. The network whose training is visible in Figure 4.2(a) has excellent results in terms of the trend of the loss function that tends to decrease in a smaller number of iterations than the network whose training is visible in Figure 4.2(b). Both results of these networks



**(a)** Learning Rate = 0.01 and one FC=512.



**(b)** Learning Rate = 0.001 and one FC=512.



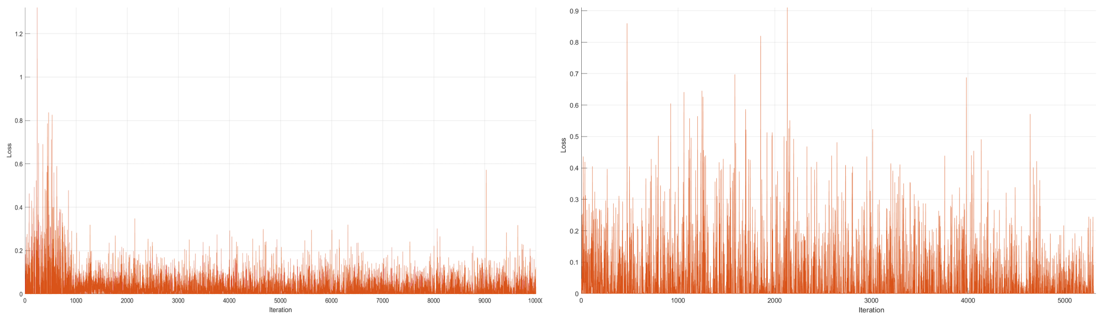
**(c)** Learning Rate = 0.0001 and one FC=512.

**Figure 4.1:** Training Siamese Network.

are excellent as the loss mean value is between 0 and 0.2 due to the random pick up of triplets, these results make the Siamese network with learning rate = 0.0001 the optimal candidate for the next test phase as it tends less to overfitting thanks to a longer and linear training to reduce the loss function.

In order to understand the lack of a parabolic trend typical of each loss function, other tests have been performed using the ResNet-18 not pre-trained on the Webface200 dataset but with the weights deriving from ImageNet, another purpose of the next series of trainings is to understand if the pre-training in a more specific task applied to the backbone network can really increase the accuracy of the network.

In this case from the network were removed the unnecessary layers belonging from the ResNet-18 coming from ImageNet such as the Softmax layer and the classification layer, reusing the FC 1000 layer and adding the layers necessary for



**(a)** Learning Rate = 0.01 and sgdmupdate as optimizer. **(b)** Learning Rate = 0.0001 and adamupdate as optimizer.

**Figure 4.2:** Training Siamese Network with FC 512, ReLU, FC 1024, ReLU, FC 512.

dimensionality reduction, these layers are the ReLU layer and the FC 512[1] layer to obtain the desired embedding. In this case the training dataset is Webface200, the learning rate was set to 0.0001 and the optimizer was the adamupdate and the parameters of the layers up to average pooling were not frozen because the network does not have the specific weights for the task of classification of a face and it must learn them. The combination of parameters comes from an analysis of the best results obtained in previous trainings.

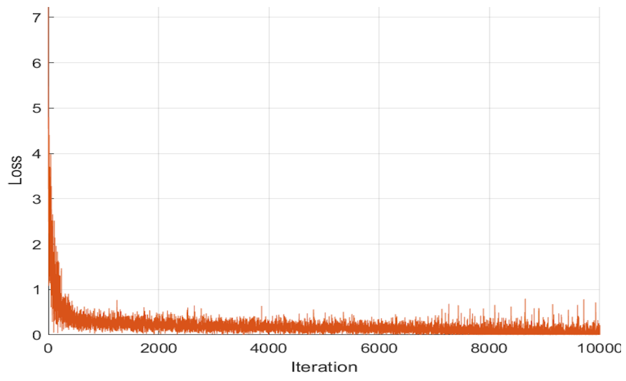
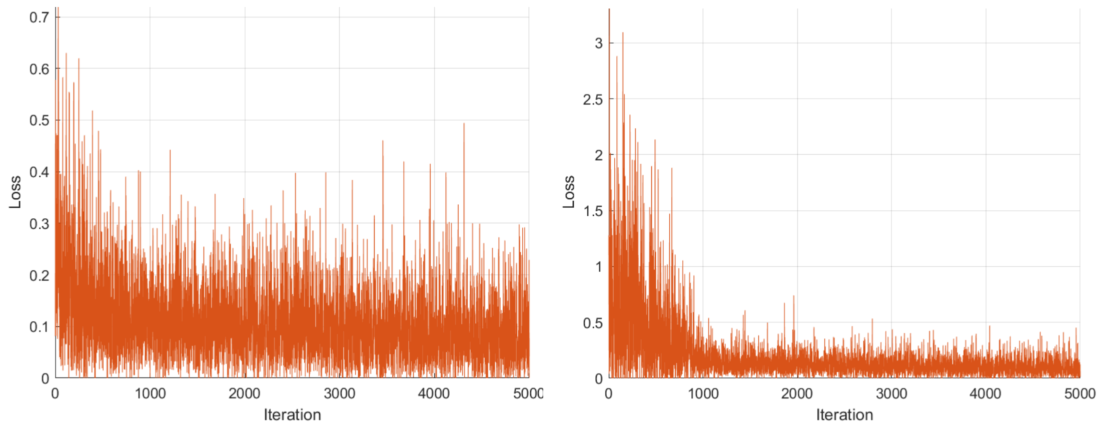


**Figure 4.3:** Training Siamese Network with ResNet-18 derived from ImageNet.

Through this subsequent series of training it was possible to understand that the lack of the parabolic trend of the loss function is due to the pre-training phase of the backbone network that allows an optimal extraction of the features. The Siamese network that uses the pre-trained ResNet-18 succeeds with fewer training iterations to learn the optimal weights for the next test phase.

[1]FC Dim means a Fully Connected Layer with output dimension equal to the number expressed by Dim.

A last series of trainings were carried out using the WebfaceSiamese dataset in order to understand some specifics of the trends of the previous trainings. In this case the network keeps the same parameters of the network visible in Figure 4.2(b), keeping the learning rate equal to 0.0001 and the added layers or exploiting only the FC 512. The results of these trainings show a network that manages to obtain good performance even on an additional dataset thanks to the greater number of layers that allows to obtain discrete weights and to the backbone network that performs optimally.



**(a)** FC 512, ReLU, FC 1024, ReLU, FC 512. **(b)** Only on FC 512.

**Figure 4.4:** Training Siamese Network with ResNet-18 on WebfaceSiamese.

All trends are characterized by a particular vertical oscillation of the loss function. This oscillation arises from the random selection of the triplets that compose the batches used at each training iteration. The composition of triplet in every batch can be different at each iteration and their random choice can led to batches without complicated triplets or batches with a considerable number of complicated triplets. Greatest is the number of the complicated triplets in the batch, greatest will be the final loss for the training in that iteration. The method to remove these oscillations is hard mining, which is the technique to select first the most complicated triplets to compare.

The most difficult triplets are those that have a distance between Anchor and Negative less than the distance between Anchor and Positive, these triplets will be the first to be used for training in order to learn heavier weights to reduce the loss. The heavier weights allows the network to obtain the correct embedding for the complicated triplets of images, i.e. the network has the capability to find embeddings where the distance of the embeddings belonging from the same class is small and the distance of embeddings belonging from different classes is high with a small percentage of error. To overcame the problem there are two solutions: Online mining and Offline mining 2.6.

While the online mining technique is impossible to use because the computing capabilities of the machine where the training takes place allow to exploit batches of maximum size of 32, the offline mining technique is more feasible. By means of a suitably created script, each possible pair of images is supplied as input to the network at the state of the i-th iteration to compute the matrix of distances from which to select the possible Anchor-Positive and Anchor-Negative pairs that represent triplets difficult to compare. The script was produced in MATLAB and involves the computation of a matrix array of size 45.000x45.000 due to the limited RAM of the machine and the need of an empty workspace to carry out the entire computation for this phase. After the matrix computational phase, the triplets for the hard mining technique would be selected in order to train the network for a certain number of iterations beyond which the matrix must be recomputed using the new update state of the neural network.

Unfortunately, the creation of the distance matrix took 48 hours to compute the distances between the first 20 images and all the available combinations with other 45.000 images despite the fact that the matrix was triangular and allowed a faster computation. The lack of computation capacity has not allowed to exploit the hard mining technique and consequently it does not allow the resolution of vertical oscillations problem.

Finally, the best network in the training phase was the network in Figure 4.2(b) thanks to its accuracy, training times and the fact that it provides excellent results even on WebfaceSiamese dataset.

## 4.2 Siamese Network with Contrastive Loss

Similarly to what was done for the Siamese that implements the Triplet Loss, different tests were carried out for the Siamese with Contrastive Loss exploiting the different values of the training parameters. In the case of Contrastive Loss, the sigmoid function was not used because the computed values remain within an acceptable threshold for graphic display and the tests were carried out both using the ResNet-18 pre-trained on Webface200 and the ResNet-18 pre-trained without the Fine-Tuning.

The different tests were carried out by modifying the following parameters:

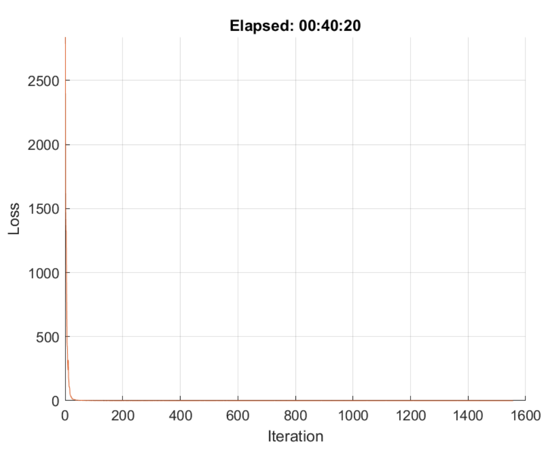
- Learning rate: the learning rate values used were 0.001 and 0.0001;
- Dataset: Webface200 and WebfaceSiamese were used as datasets;
- Network Architecture: the sequence of layers after to the average pooling layer has been adapted to the different tests that were carried out based on the dimensionality of the feature vector that must be extracted;
- Other Parameters: in some tests the ResNet-18 was used without fine-tuning, for this purpose the layers were not frozen.

During the entire training phase, the margin to compute the Contrastive Loss was set to 1 which was the best in a series of tests carried out but they are not reported in this section. The results of all possible combinations will not be reported for practical reasons because some trainings require multiple hours to be completed, only some combinations of the training parameters will be corroborated giving more space to those that have obtained the best results.

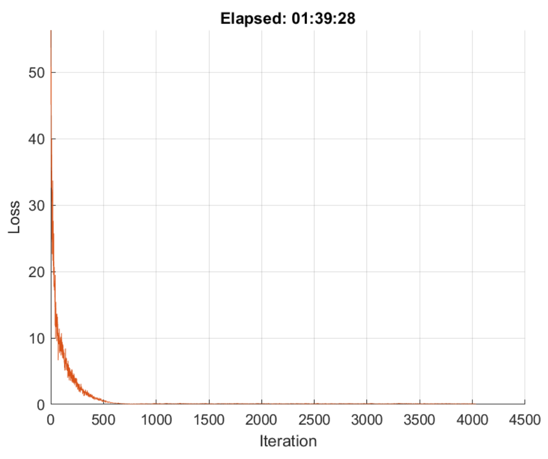
Below we will be report the graphs of the training deriving from the different combinations of the parameters that have been selected using the Siamese network with contrastive loss, the descriptions of the architectural changes will concern only the structure after the average pooling layer.

The trends visible in Figure 4.5 derive from trainings that use the same structure based on a single fully connected layer while the value of the learning rate changes to produce an embedding of size 512. Unlike what happened for the Triplet Loss, there is the typical parabolic trend of a loss function that decreases when the iterations increase, starting from very high values due to the greedy nature of this loss function caused by the random choice of the positive or negative pair of images.

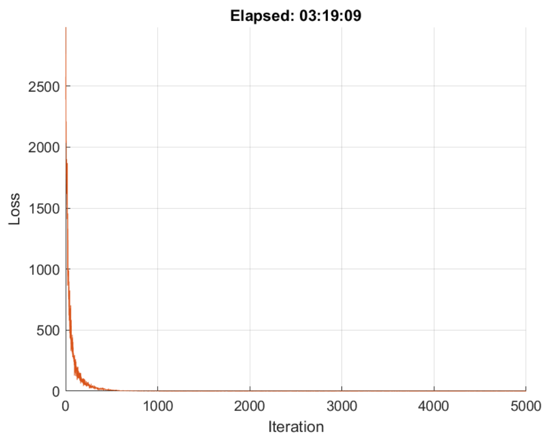
The right trade-off between decreasing the value of the loss and the necessary iterations make these trainings the optimal candidates to obtain the appropriate weights of the Siamese network for face identification. Despite the rapid learning resulting in a reduction in the value of the loss function, the training requires a minimum number of 4000 iterations to refine the weights in order to obtain for all results a final value of the loss close to 0.15. The different trend of this value, even



**(a)** Learning Rate = 0.01 and one FC=512.



**(b)** Learning Rate = 0.001 and one FC=512.



**(c)** Learning Rate = 0.0001 and one FC=512.

**Figure 4.5:** Training Siamese Network.

if it is imperceptible, plays a decisive role in the selection of the best candidate that has been identified in the training network visible in Figure 4.5(b).

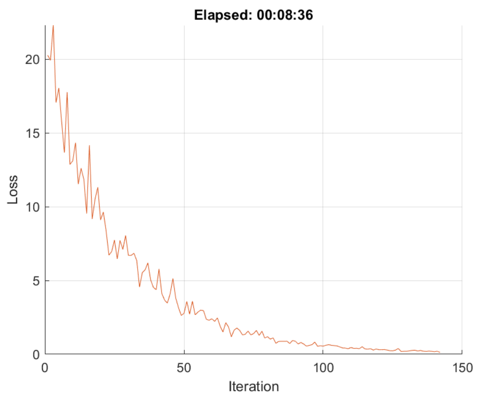


**Figure 4.6:** Train Siamese Network with FC 512, ReLU, FC 1024, ReLU, FC 512.

In order to understand if the deeper architecture of the network could bring an advantage in the training in terms of final loss value a new test was carried out using a new and deeper structure, but as we can see in Figure 4.6 the network goes into overfitting after only 100 iterations. This behavior is determined by the rapid learning due to the random selection of image pairs without knowing a priori if they represent the same person, in this way the network quickly learns how to produce the correct emebedding but learns by heart with consequent low accuracy performance in the inference phase with the test database.

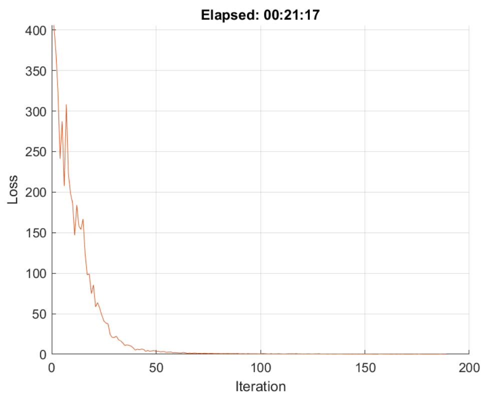
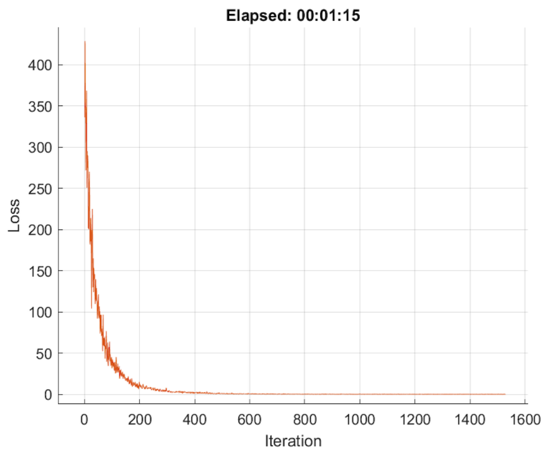


**Figure 4.7:** Train Siamese Network with ResNet-18 trained on ImageNet.

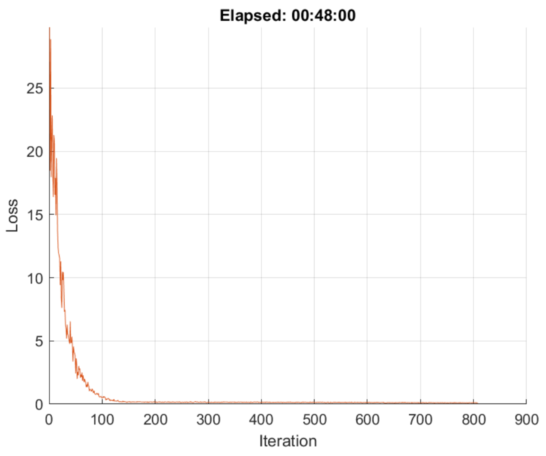
A similar overfitting behavior has also been identified in training using ResNet-18 from ImageNet and without fine-tuning. As can be seen in Figure 4.7, the drop of the loss function is almost instantaneous thanks to the learning rate = 0.0001 but above all thanks to all the non-frozen layers of the backbone network, in this way the network is completely free to update its weights adapting them too quickly for the purpose of training with consequences of overfitting and poor performance in a future inference phase. This network as well as the previous one has been deleted from the pool of possible choices.

As for what happened for the triplet loss in order to have a complete comparison, the result of the training of the Siamese network with contrastive loss on the WebfaceSiamese dataset was verified both using only the FC 512 layer and the additional structure composed of FC 512, ReLU, FC 1024, ReLU, FC 512. In both cases visible in Figure 4.8 it is possible to observe a discrete trend of descent of the value of the loss function, but it is still too rapid with respect to the trend of Figure 4.5(b) resulting in discrete results but not excellent in the test phase.

The oscillation visible during the training with the triplet loss is not present,



(a) Learning Rate = 0.0001 and one FC=512.



(b) Learning Rate = 0.001 with FC 512, ReLU, FC 1024, ReLU, FC 512.

**Figure 4.8:** Training Siamese Network on WebfaceSiamese.

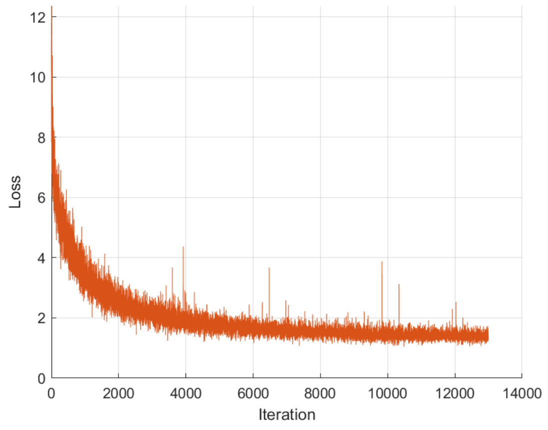
except in some cases, in the training with contrastive loss because the choice of image pairs is random.

## 4.3 Siamese Network with Triplet Loss and Classification Loss

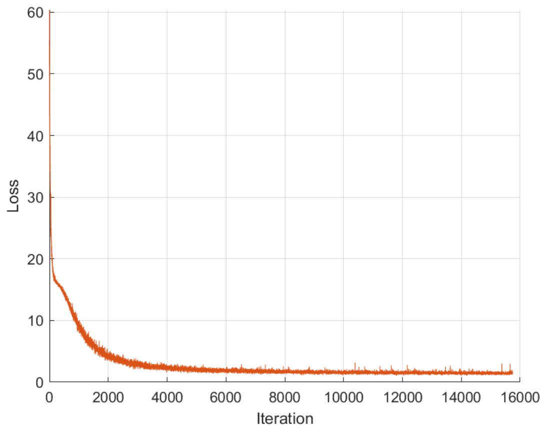
As described in Chapter 3, an additional constraint was imposed in the form of classification loss in order to understand whether more restrictive training could benefit the network in terms of accuracy.

The addition of the classification loss involves the possibility of learning and modifying all the parameters of all the layers in the network, including the parameters of the Batch Normalization layers requiring, in this way, the computation of an average state exploiting the values present at each iteration. Classification loss also involves the need to have a network with two branches to perform two tasks at the same time with the consequent problem that, due to an internal programming error of MATLAB, one of the two branches cannot exploit GPU acceleration.

The problems mentioned above resulted in training times that takes 1 hour every 100 iterations, requiring a total of 130 hours per training. For this reason, the number of trainings of the network has been reduced to the strictly necessary by exploiting specific values of the training parameters. To this end, the tests carried out involved the ResNet-18 pre-trained on Webface200 and on ImageNet without fine-tuning, the value of the learning rate was set to 0.0001 and adamupdate was used as optimizer. In addition, the number of images in each batch has been reduced to 16 due to lack of computation resources capable of using training batches with larger dimensions as happened in the previous phases.



**(a)** Learning Rate = 0.0001 and Resnet-18 fine tuned.



**(b)** Learning Rate = 0.001 and ResNet-18 without fine tuning.

**Figure 4.9:** Training Siamese Network with Classification Loss.

During the trainings the triplet loss contributed only 50% in the weight update making the classification loss the main function of the weight changes of the network itself. The need to classify Anchor, Positive and Negative simultaneously has made the total loss values of the network high, not allowing training to go beyond the loss value of 1.3 both in the case of ResNet-18 with fine tuning and without.

Although the final loss value does not result in an improvement in the possible results of the network due to its inability to correctly classify triplets of input images at the same time, the constant decrease trend could suggest an ability to learn more correctly if a dataset of adequate size was used to train a network of this type.

## 4.4 Siamese Network with Contrastive Loss and Classification Loss

Emulating what happened in the paragraph 4.3 with very similar problems, even the network including the classification loss with the contrastive loss has been trained.

Due to the lack of acceleration due to the network branch, the training required a training time of 40 hours due to smaller computations thanks to the use of only pairs of images instead of triplets. Using a network that uses only the FC 512, the learning rate = 0.0001 and adamupdate as an optimizer, the training obtained excellent results that are in line with the Siamese network equipped with only contrastive loss.

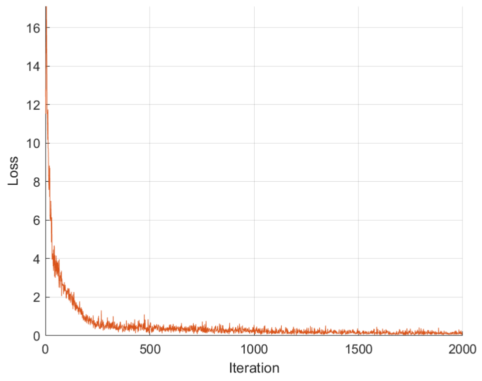
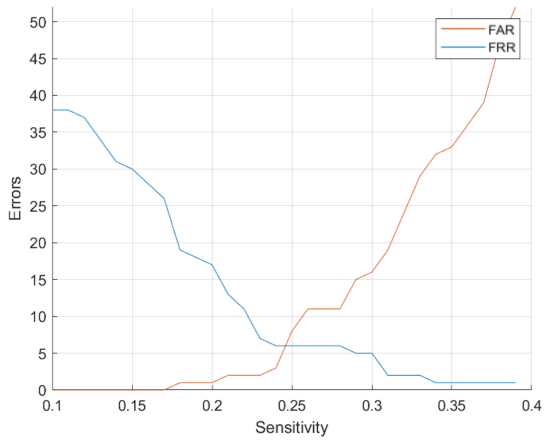


**Figure 4.10:** Train Siamese Network with Contrastive and Classification Loss.

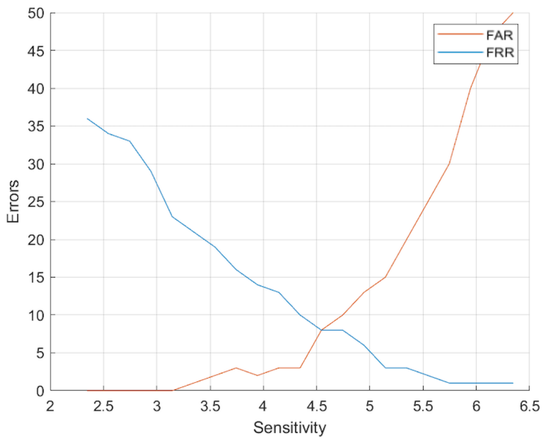
In Figure 4.10 the graph was deliberately stopped at iteration number 2000 for visual purposes. In this case the network has the ability to learn very quickly despite the high value of the Classification Loss that is well rooted in the first iterations and thanks also to the fact that the Contrastive Loss contributes only 50%. Despite this, with a small number of iterations the network is able to drastically reduce the value of the loss, proceeding very slowly in subsequent iterations until obtaining a final value that oscillates between 0.03 and 0.08.

## 4.5 Results on Test Database

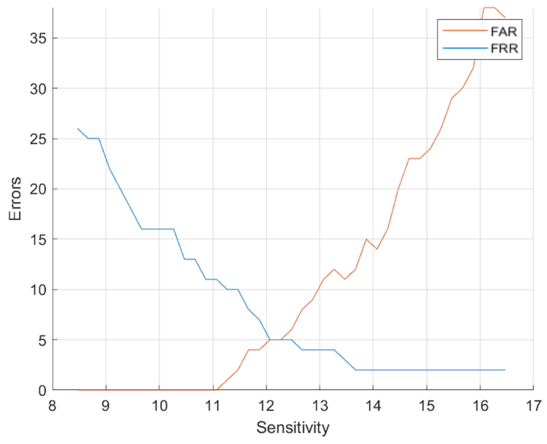
Using the face identification script, the capabilities of previously trained neural networks were verified following the guidelines of the ISO/IEC 19795 standard which establishes common measures for the evaluation of algorithms that exploit biometric data. In particular, to understand the recognition accuracy of networks developed around specific loss functions, the curve that compares the False Accept Rate (FAR) and the False Reject Rate (FRR) will be exploited.



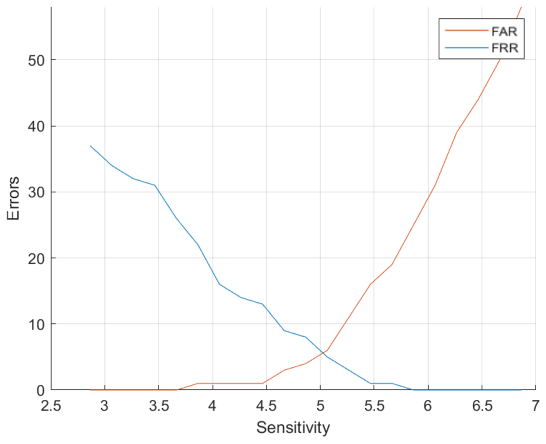
**(a)** Siamese with Contrastive Loss.



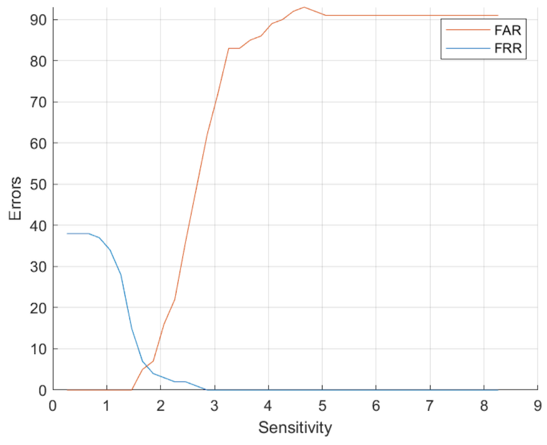
**(b)** Siamese with Contrastive Loss and Classification Loss.



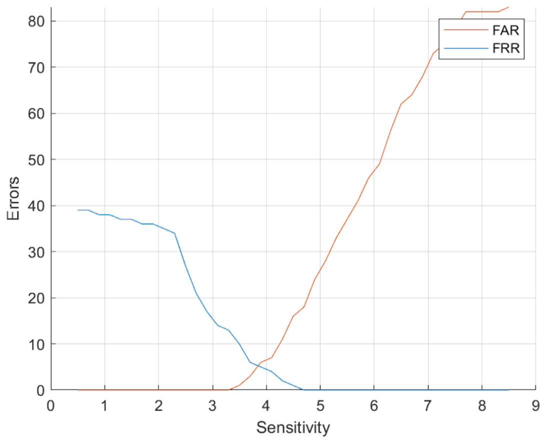
**(c)** Siamese with Triplet Loss.



**(d)** Siamese with Triplet Loss and Classification Loss.



**(e)** Siamese with Contrastive Loss trained on WebfaceSiamese.



**(f)** Siamese with Triplet Loss trained on WebfaceSiamese.

**Figure 4.11:** Report of the trade-off between FAR and FRR, the sc.

The scale of the graphs is not constant due to the different values of the thresholds, the comparison of the best results can be seen in Table 4.1.

| **Loss Type** | **Threshold** | **FRR** | **FAR** |
|---|---|---|---|
| Contrastive | 0.24 | 6 | 3 |
| Contrastive+Classification | 4.1 | 10 | 3 |
| Triplet | 12.2 | 10 | 1 |
| Triplet+Classification | 5.1 | 9 | 3 |
| Contrastive WebfaceSiamese | 1.7 | 7 | 5 |
| Triplet WebfaceSiamese | 3.85 | 6 | 3 |

**Table 4.1:** Results with selected threshold

The purpose of this graph is to highlight the capabilities of neural networks through False Match Rate (FMR)[2] and False Non-Match Rate (FNMR)[3]. Considering that there are no image acquisition failures thanks to the upstream MTCNN, the number of Failure-To-Acquire Rate (FTA) indicated by the standard is always considered to be equal to 0 matching the FAR with the FMR and the FRR with the FNMR.

The choice of the best threshold in the FAR-FRR ratio is based on the trade-off due to the increase of FRR when the threshold becomes very discriminative and the increase of FAR when the threshold becomes less discriminative. The choice of the optimal threshold is made based on the possibility of implementing the "Best of" technique aimed at reducing the number of errors committed and on the reduction of False Positives which are dangerous in the application context and therefore must be minimized at the expense of False Negatives.

The graphs visible in Figures 4.11 represent the trend of FRR and FAR, or False Negative and False Positive, as the threshold changes in the networks that were better in training. Based on the distance that each network can produce between pairs of positive and negative images, the threshold is selected and is verified within a possible range of values, identifying the right trade-off between FAR and FRR.

Since the context of application requires a discriminative ability in which it is preferable that the face is not recognized rather than that the face of an "impostor"

[2]Proportion of impostor attempts that are falsely declared to match a template of another object.

[3]Proportion of genuine attempts that are falsely declared not to match a template of the same object.

is matched with a database image, in choosing the optimal threshold it is preferable not to select the one that allows to obtain the lowest number of False Positive and Negative but the one that produces the least number of False Matches around the point of incidence of the two curves.

The tests performed exploit a database containing images derived from the videos produced by a video door phone in use at the company, the images are appropriately aligned using the generated scripts exploiting the MTCNN. This database contains 62 images of 8 different identities that will be compared with 100 input images extrapolated from different videos produced by the same video door phone and aligned with the MTCNN or from another dataset in order to be used as "impostors". In particular, the best networks deriving from each of the various types of training with loss functions were selected during the test phase in order to compare the accuracy of each Siamese to the variation of the loss used, the accuracies are visible in Table 4.1 where are reported the best trained network of each database and loss function. In particular in the table are reported the selected values of FRR and FAR that best fit with the scope of application.

After verifying the accuracy of the best network for each loss function and for each training dataset, the results obtained by the Siamese with Contrastive Loss and with Triplet Loss trained on WebfaceSiamese dataset make them the best networks in the face identification phase as we can see in Table 4.1, both with the ability to correctly identify 91% of images and with an adequate threshold in order to decrease as much as possible the number of False Positives that can be further reduced using the aforementioned technique of the "Best Of".

# Chapter 5

# Conclusions and future works

## 5.1 Conclusions

In this work, we addressed one of the most current issues namely Face Recognition and its implementation process in an embedded system that supports all the functions of home automation, control and video door entry that CAME provides to the customer. In particular, the entire process of analysis, design and implementation of the neural networks to carry out the two tasks of Face Detection and Face Identification was described.

The project started with an analysis on the Face Recognition pipelines and the tools necessary for their development, then we focused on the two tasks separately studying and implementing in detail the neural networks and all the tools and data necessary for their operation.

Our pipeline starts with a neural network for Face Detection: for this task, we have chosen the MTCNN architecture that allows not only a quick identification of a face using three cascade CNNs, but also allows to process the image to align the face with ones in the database through the help of facial landmarks and bounding boxes provided by the network. This network is vital for creating the production database and detecting an appropriate face for the identification phase.

All faces identified and accurately aligned by the Face Detection network are supplied as input to the Siamese network for Face Identification which uses the ResNet-18 as a backbone network for feature extraction. The network is trained to compare the features vectors extracted from the input image to find, if it exists, the most similar image in the database in order to find the face identity. This ability comes from the training through the loss functions used: Triplet Loss, Contrastive Loss and their combination with Classification Loss.

All the choices made in this development process derive from the constraints imposed such as the privacy law that limits the use of delocalized tools,the low computing power of the processors of the embedded system and of the machine in which the training of the neural network was carried out. These constraints led to the choice of a backbone network for the Siamese with a limited number of layers, thus limiting the generality that the network can have in feature extraction; the overall accuracy of the network is also limited by the quality of the images acquired by the video intercom.

All the limitations that were imposed during the development process led to the choice of a very small neural network for the most critical phase, i.e. for Face Identification, and also required a very small dataset compared to the one used in other approaches. The result of using this small dataset is the reduction of accuracy performance compared to networks that exploit datasets containing millions of images for training.

Despite a very small feature extraction network, the small training dataset, and the comparison database with a limited number of images, the Face Recognition task is executed with an accuracy of more than 90%. The accuracy of the network grows as the number of comparison images in the test database increases, with a growth trend of 0.5% for every 15 additional images representing the same number of people.

With this project, we wanted to demonstrate the possibility of implementing a Face Recognition pipeline with good accuracy and relatively low resource consumption that can be implemented in an adequate processor such as the Raspberry Pi4.

The results obtained in this development process provide a solid basis for a possible improvement of neural networks for embedded systems in terms of accuracy and speed; they also provide a comparison of the accuracy obtained from the various tested loss functions and a custom-built code that implements Triplet Loss in MATLAB in an alternative way. Furthermore, the results obtained allow the company to have a clearer idea about the future directions in terms of new processors for new products and for the services they will be able to provide in the future.

## 5.2 Future Works

Thanks to the promising results obtained, the design of this neural network will allow the company to develop new products and new functions to be integrated. Before being able to proceed with the implementation of the neural network within a prototype, a preliminary phase of adaptation of the entire ecosystem will be needed, in particular, the next few months will be dedicated to a total redesign of the Rest API for the configuration of video intercom and a re-work of the functions to adapt them to the new processor and new performance.

Once the entire system has been adapted inside the Raspberry Pi4 B, we will continue with the creation of the first working prototype using their high-end product that allows the help of a touch screen and a high-resolution camera. Through this prototype, it will be possible to carry out real field tests as well as a market survey in order to understand the real potential of this development and the real interest from the final customer.

In the prototyping phase, further details that cannot be known a priori will be analyzed, such as implementation details, assumptions on people's behavior, etc. These measures will be used to give greater accuracy and speed to the network that will be continuously analyzed and, if possible, improved in terms of performance and precision.

If we proceed with the creation of a new line of products that implement this "technology", a research and development phase will follow on how to guarantee the backward compatibility of Face Recognition even with older products that are based on the old IMX6 processors, backward compatibility that to date can only be guaranteed through the creation of an external product to be added to the actual system in order to intercept the video in the transition from video ring bell and intercom, encode it and give an identity to the possible face of the person in that video. To date, this backward compatibility is impossible because the inference of the entire facial recognition pipeline within the IMX6 processor would require an estimated time of over 10 seconds to obtain the label from a single frame of the video.

If the market response is proactive to the development of this new generation of products or tools necessary for backward compatibility, new functions will be designed for integration with all the systems that CAME develops and in particular with home automation also through external home assistants.

There are many ideas born around this project, but for the near future the focus is on prototyping a new product that natively implements the neural network and verifies the progress of the market survey to understand if the mass production of this product will be possible.

# Acknowledgements

I want to spend a few words to thank all the people who have supported me during this important journey.

First of all, I would like to thank my supervisor Alberto Pretto for the trust placed in me and in this work, but above all for having guided, supported, helped and motivated me in this path, and also for the enthusiasm he transmits to me.

Thanks to Francesco De Marco and Antonio Milici for following me in this project, in particular for the great opportunity given to me and for the trust placed in me for this important project. Furthermore I would like to thank the entire CAME company for the support and for believing in me for a project that could represent a technological step ahead applicable to a wide range of products, also to my colleagues who took part in the tests and allowed me to consistently evaluate the results.

Finally, I would like to express my deepest appreciation to my family who believed in me since the first moment, allowing me economically and emotionally to face this journey, for the continuous encouragement and support.

# Bibliography

[1] Insaf Adjabi, Abdeldjalil Ouahabi, Amir Benzaoui, and Abdelmalik Taleb-Ahmed. Past, present, and future of face recognition: A review. *Electronics*, 9(8):1188, 2020.

[2] Timo Ahonen, Abdenour Hadid, and Matti Pietikäinen. Face recognition with local binary patterns. In *European conference on computer vision*, pages 469–481. Springer, 2004.

[3] Waqar Ali, Wenhong Tian, Salah Ud Din, Desire Iradukunda, and Abdullah Aman Khan. Classical and modern face recognition approaches: a complete review. *Multimedia Tools and Applications*, 80(3):4825–4880, 2021.

[4] Kai Chen, Taihe Yi, and Qi Lv. Lightqnet: Lightweight deep face quality assessment for risk-controlled face recognition. *IEEE Signal Processing Letters*, 28:1878–1882, 2021.

[5] Yixian Cheng and Haiyang Wang. A modified contrastive loss method for face recognition. *Pattern Recognition Letters*, 125:785–790, 2019.

[6] Davide Chicco. Siamese neural networks: An overview. *Artificial Neural Networks*, pages 73–94, 2021.

[7] Xingping Dong and Jianbing Shen. Triplet loss in siamese network for object tracking. In *Proceedings of the European conference on computer vision (ECCV)*, pages 459–474, 2018.

[8] Oliver Dürr, Yves Pauchard, Diego Browarnik, Rebekka Axthelm, and Martin Loeser. Deep learning on a raspberry pi for real time face recognition. In *Eurographics (Posters)*, pages 11–12, 2015.

[9] Mohamad El-Abed, Christophe Charrier, and Christophe Rosenberger. Evaluation of biometric systems. *New Trends and Developments in Biometrics*, 11 2012.

[10] Li Fei-Fei, Robert Fergus, and Pietro Perona. One-shot learning of object categories. *IEEE transactions on pattern analysis and machine intelligence*, 28(4):594–611, 2006.

[11] Mohammad Basman Gh et al. A novel face recognition system based on jetson nano developer kit. In *IOP Conference Series: Materials Science and Engineering*, volume 928, page 032051. IOP Publishing, 2020.

[12] Hosseinali Ghiassirad and Mohammad Teshnehlab. Similarity measurement in convolutional space. In *2012 6th IEEE International Conference Intelligent Systems*, pages 250–255. IEEE, 2012.

[13] Raúl Gómez. Understanding ranking loss, contrastive loss, margin loss, triplet loss, hinge loss and all those confusing names. `https://gombru.github.io/2019/04/03/ranking_loss/`. Accessed: 2019-04-03.

[14] Raia Hadsell, Sumit Chopra, and Yann LeCun. Dimensionality reduction by learning an invariant mapping. In *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*, volume 2, pages 1735–1742. IEEE, 2006.

[15] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[16] Javier Hernandez-Ortega, Javier Galbally, Julian Fierrez, Rudolf Haraksim, and Laurent Beslay. Faceqnet: Quality assessment for face recognition based on deep learning. In *2019 International Conference on Biometrics (ICB)*, pages 1–8. IEEE, 2019.

[17] Gary B Huang, Marwan Mattar, Tamara Berg, and Eric Learned-Miller. Labeled faces in the wild: A database forstudying face recognition in unconstrained environments. In *Workshop on faces in'Real-Life'Images: detection, alignment, and recognition*, 2008.

[18] Lianfen Huang, Jia Guo, and Zhibin Gao. A face recognition system on embedded device. *J. Comput*, 31:176–83, 2020.

[19] Nourman S Irjanto and Nico Surantha. Home security system with face recognition based on convolutional neural network. *International Journal of Advanced Computer Science and Applications*, 11(11), 2020.

[20] ISO ISO. Iec 19795-1: Information technology–biometric performance testing and reporting-part 1: Principles and framework. *ISO/IEC, Editor*, 1(3):5, 2006.

[21] Gregory Koch, Richard Zemel, Ruslan Salakhutdinov, et al. Siamese neural networks for one-shot image recognition. In *ICML deep learning workshop*, volume 2, page 0. Lille, 2015.

[22] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Communications of the ACM*, 60(6):84–90, 2017.

[23] Martin Lades, Jan C Vorbruggen, Joachim Buhmann, Jörg Lange, Christoph Von Der Malsburg, Rolf P Wurtz, and Wolfgang Konen. Distortion invariant object recognition in the dynamic link architecture. *IEEE Transactions on computers*, 42(3):300–311, 1993.

[24] Haoxiang Li, Zhe Lin, Xiaohui Shen, Jonathan Brandt, and Gang Hua. A convolutional neural network cascade for face detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 5325–5334, 2015.

[25] Ningning Ma, Xiangyu Zhang, Hai-Tao Zheng, and Jian Sun. Shufflenet v2: Practical guidelines for efficient cnn architecture design. In *Proceedings of the European conference on computer vision (ECCV)*, pages 116–131, 2018.

[26] Bishwas Mandal, Adaeze Okeukwu, and Yihong Theis. Masked face recognition using resnet-50. *arXiv preprint arXiv:2104.08997*, 2021.

[27] Yoanna Martindez-Diaz, Luis S Luevano, Heydi Mendez-Vazquez, Miguel Nicolas-Diaz, Leonardo Chang, and Miguel Gonzalez-Mendoza. Shufflefacenet: A lightweight face architecture for efficient and highly-accurate face recognition. In *Proceedings of the IEEE/CVF International Conference on Computer Vision Workshops*, pages 0–0, 2019.

[28] Iaroslav Melekhov, Juho Kannala, and Esa Rahtu. Siamese network features for image matching. In *2016 23rd international conference on pattern recognition (ICPR)*, pages 378–383. IEEE, 2016.

[29] Ville Ojansivu and Janne Heikkilä. Blur insensitive texture classification using local phase quantization. In *International conference on image and signal processing*, pages 236–243. Springer, 2008.

[30] Muhammad Owais, Aireen Amir Jalal, Muhammad Moiz Hassan, and Ammara Shaikh. Facial recognition based attendance system using cnn and raspberry pi. In *2020 4th International Symposium on Multidisciplinary Studies and Innovative Technologies (ISMSIT)*, pages 1–5. IEEE, 2020.

[31] Vishwani Sati, Sergio Márquez Sánchez, Niloufar Shoeibi, Ashish Arora, and Juan M Corchado. Face detection and recognition, face emotion recognition through nvidia jetson nano. In *International Symposium on Ambient Intelligence*, pages 177–185. Springer, 2020.

[32] Florian Schroff, Dmitry Kalenichenko, and James Philbin. Facenet: A unified embedding for face recognition and clustering. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 815–823, 2015.

[33] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[34] Lawrence Sirovich and Michael Kirby. Low-dimensional procedure for the characterization of human faces. *Josa a*, 4(3):519–524, 1987.

[35] Matthew Turk and Alex Pentland. Eigenfaces for recognition. *Journal of cognitive neuroscience*, 3(1):71–86, 1991.

[36] Paul Viola and Michael J Jones. Robust real-time face detection. *International journal of computer vision*, 57(2):137–154, 2004.

[37] Fei Wang, Liren Chen, Cheng Li, Shiyao Huang, Yanjie Chen, Chen Qian, and Chen Change Loy. The devil of face recognition is in the noise. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 765–780, 2018.

[38] Feng Wang and Huaping Liu. Understanding the behaviour of contrastive loss. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 2495–2504, 2021.

[39] Yi-Qing Wang. An analysis of the viola-jones face detection algorithm. *Image Processing On Line*, 4:128–148, 2014.

[40] Laurenz Wiskott, Norbert Krüger, N Kuiger, and Christoph Von Der Malsburg. Face recognition by elastic bunch graph matching. *IEEE Transactions on pattern analysis and machine intelligence*, 19(7):775–779, 1997.

[41] Shuo Yang, Ping Luo, Chen-Change Loy, and Xiaoou Tang. From facial parts responses to face detection: A deep learning approach. In *Proceedings of the IEEE international conference on computer vision*, pages 3676–3684, 2015.

[42] Dong Yi, Zhen Lei, Shengcai Liao, and Stan Z Li. Learning face representation from scratch. *arXiv preprint arXiv:1411.7923*, 2014.

[43] Jianming Zhang, Chaoquan Lu, Jin Wang, Xiao-Guang Yue, Se-Jung Lim, Zafer Al-Makhadmeh, and Amr Tolba. Training convolutional neural networks with multi-size images and triplet loss for remote sensing scene classification. *Sensors*, 20(4):1188, 2020.

[44] Kaipeng Zhang, Zhanpeng Zhang, Zhifeng Li, and Yu Qiao. Joint face detection and alignment using multitask cascaded convolutional networks. *IEEE signal processing letters*, 23(10):1499–1503, 2016.