Biometric Systems Project

A Framework for face identification and verification, perceived for an automatic user management system in an airport

Lorenzo Papa 1699806 & Nicolas Zaccaria 1904366

September 8, 2020

1 Introduction:

The project takes inspiration from a video published on YouTube[1] by *HIA Qatar* about the biometric technology developed for the *Hamad International Airport, Doha, Qatar*.

As we can read in the description of the video, in the airport was launched the second phase of its innovative *Smart Airport program* based on facial biometric recognition across all key passengers. We can in fact see in the video how the face is used from the check-in until the gate operation to get into the airplane. They also develop an augmented reality to improve the user's time in the airport; those are only other aspects, probably also with more economic advantages for the shops, that can be improved combined with the improvement of those biometric systems.

For this reason, we find interesting to reproduce, in a simplified way and with the tools available, this process, that lead the user with his face from the check-in to the gate.

[1] https://www.youtube.com/watch?v=5glvuLUr06w

Obviously, the security of an airport can't be based only on this system, and for this reason, we classify it as a *framework* of a bigger and more robust multi-biometric system.

Doing some research, we have also discovered that at the Fiumicino airport, Rome, a facial recognition system for a specific flight segment will be experimented, as we can read in the article[2]. It should be noticed how this choice is justified: *"Facial recognition will offer passengers a smoother journey to the airport"*.

1.1 User steps in the airport:

In this sub-chapter, we will briefly describe the steps that the user must perform to facilitate the checks and make the system work at best.

The process we are going to describe has the common purpose of the article previously mentioned: making the airport system faster and more efficient by using only the face to have access to the different areas. Obviously, this procedure reports only the main steps inherent to the developed framework, in which it plays a key role:

- The first is the registration phase; the user will show the identity card or passport and through the acquisition of a series of images, the matching is performed to check the correspondence between the document and the user.

- The second is the classification of the user; in which the respective information of the face is attributed to the specific ID. Thanks to this operation, the user will be able to continue in the following steps by simply using his face. Moreover we have also simulated a boarding process, with 400 people; in fact, the system will know exactly which people to expect for each specific flight and their respective facial characteristics.

These are just two of the main operations carried out in the airport procedure. The information extracted from the system, for each person, can be used in many applications for economic or security purposes; often combined with other specific systems, as shown in the video above.

[2]http://www.adr.it/controllo-biometrico

1.2 Google Colaboratory:

In this long introduction, we also see why we chose to develop our code, in Python, using Google Collaboratory and its limitations.

The main reason is the availability of specialized hardware, as GPU and TPU, which the platform makes available, for free; those are really helpful for Machine Learning systems. In addition, a Google service facilitates the management of information that can be easily shared with Google Drive.

In our case, in fact, we saved the datasets used and the models trained directly on the drive, without having to use our personal computers.

Unfortunately, however, the platform also has limits of use which define the limits of development and training for the neural networks. In fact, each notebook has a maximum duration of 12 hours, after which the system deactivates them, and if we want to reload the model to continue the training, the system does not always allow us to reuse multiprocessing; this makes learning very slow[3].

This means that the models that we will present in the next chapters will be trained for a maximum of 12 hours, in multiprocessing, which corresponds to about 150 epochs.

1.3 Structure of the report:

To onclude this chapter we will se how the report is organized:

- Chapter 2: Data management
- Chapter 3: Feature extraction
- Chapter 4: Classification
- Chapter 5: Results

We chose this order of presentation of the different arguments to recall the classical organization of a biometric system: starting from the acquisition of the data to the decision module where, in the end, we will make our conclusion comparing the obtained results.

[3]In some cases even 1 hour at epoch

2 Data management:

In this chapter, we will illustrate all the operations that have been performed on the dataset and on the images, before they are given as input to the network. We will show the different approaches based on the organization of the dataset and the data augmentation techniques implemented by analyzing their pros and cons.

2.1 Dataset:

In a machine learning system, a fundamental requirement is the dataset; in fact, it is not just a source of large amounts of training data but also the first step of a process to improve results.

In this case, we wanted to train a convolutional neural network, given the image of a face as input, to extract a vector of features that perfectly represents that face in such a way it can be used as a sort of ID key for identification and recognition, as we'll show in the following chapters. For this reason, we have dedicated a lot of resources in tuning and testing our data, obtaining, in the end, three different versions of the same dataset that we are going to compare in the last chapter.

As suggested by the Professor, we decided to use as a base dataset the one called "CelebA in the wild". Given the huge amount of 202599 images, of 10177 different famous people, our classes, and thanks to the possibility, if needed, to extend each class just by searching for new images on the internet, it fits well our purpose.

All the images are grouped in a unique folder and they are organized by two text files where the label and the split set (train, valid, test) of each image are specified. This organization causes a too complicated path tree to be handled by Google drive, where the dataset is stored as just introduced, leading to a "Google Drive Timeout error" that stops the execution of the training process. Then, as a first step, we dealt with this problem by splitting the images in the dataset root folder into three sub-folders called: train, valid, and test. In each folder, we grouped the images of the same class in a sub-folder called with the name of that class, its representative ID. This subdivision drastically simplifies the path tree solving the timeout error and speeding the data loading.

About the speed of images loading, we have also tried to convert and save each image as a NumPy array (.npy) instead of the original format

(.jpg); this because the jpg-compression requires a decoding phase when read, differently from the npy format. Overall, this process doesn't really speed up the algorithm so we decided to work with the original file format. As we can see the obtained result for each group of images is reported below:

Load Time Comparison		
Trials	file.npy	file.jpg
n1: loaded in	0.78	0.02
n2: loaded in	1.26	0.01
n3: loaded in	0.01	0.02
n4: loaded in	0.96	0.81
n5: loaded in	0.01	0.79
n6: loaded in	1.31	0.02
n7: loaded in	0.53	0.02
n8: loaded in	0.32	0.70
n9: loaded in	0.65	1.21
n10: loaded in	0.01	1.05

We can notice that the average on ten trials it is equal to 0.58s for the npy and 0.46 for the other. This justifies our conclusions. Also, we tried to store every image, for each class, in a single npy-file but since in our project we have to load each time a single or couple of images, from the same class, this approach would have increased the loading time.

As we have introduced we worked with three different versions of the same dataset; those, with their pros and cons, are described below.

The *First* one is the pure *"CelebA in the wild"* where for each face we perform face detection and alignment. Also, in the beginning, we tried to apply some data augmentation methods that we then removed since the samples are already statically augmented; that we'll discuss in the next subchapter.

The *Second* one is based on *"CelebA align"* on which we performed, as for the first version, the face detection but not the alignment. This dataset, in fact, has been used to test the alignment process performed on the previous version. From the results, we observe that it seems to apply some face distortions, with a heavy change of proportions.

In both cases, when the face detection failed, we decided to still keep that image because every sample of these two datasets contains at least one face for sure.

However the dataset, presents some classical problems as high intrapersonal variations and some interpersonal similarities as tweens[4]. It also has a wide range of different resolution images, with some really low-resolution, and a lot of occlusions that cover some interesting regions of the faces as sunglasses, microphones, ... As we can see in the images below reported:

All these features are useful to build a solid and accurate face recognition system able to generalize in a wide range of different situations but it would need more training time to converge and, moreover, it doesn't fit well our case where people can't cover their face.

Then, we decided to tune the dataset to our work with a simple cleaning process where, starting from the "CelebA align", we removed all the images where the face detection failed, because of some extreme bad pose or some disturbing element or to low resolution, and also without performing the alignment. The result is the *Third* version, the final one, that preserves the dataset characteristics avoiding partial frontal faces and keeping the small A-PIE variance, typically of the dataset, with an overall higher quality of the samples. This pruning reduced the dataset of 5837 images, from 202599 samples down to 196762.

[4]As the presence of Cole and Dylan Sprouse

2.2 Face detection:

The first step in a face recognition system, given a new image, is the face detection phase. This operation is used to search a face structure inside an image and, if it is detected, the system computes a rectangle around it. It can be used to crop around the area of interest, removing the background of the input image that is unnecessary. Also, feeding an image containing only the face to the model used will force it to focus only on the face's features, free from disturbing elements, to extract the main embeddings.

There are a lot of different detection algorithms but in our project, we used the one offered by the python library called *Dlib*.

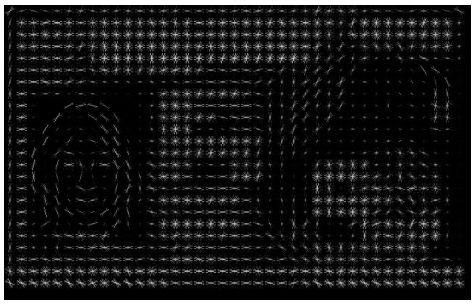
This method is based on four main components: the Histogram of Oriented Gradients, HOG, the features combined with a linear classifier, an image pyramid, and, finally, a sliding window detection scheme. The general process is the following; it, basically, reduces the image to a feature descriptor representation, this step is useful to simplify the data structure extracting useful information and discarding the rest. The features, in this case, are described by the distribution of directions of gradients which magnitude is known to be higher near edges and corners. At this point, a linear classifier, SVM, is used to detect, using this features representation, if there is a face like structure in the image. Then the image pyramid and the sliding window are also used to improve the accuracy by looking for a face in the image at different scales and locations.

As anticipated, in the dataset section, we also used the face alignment of the OpenFace[5] library, based on the same python library. In particular, a pre-trained model is used to compute 68 facial landmarks that describe the pose of the face; then, with those informations, it tries to align the face. But, as just explain, in our case, it did not improve the process and also makes the organization of the dataset slower.

To conclude this subchapter we report a graphical representation of the execution of the algorithm with the following four imagers.

As we can see, starting from the identity card, and applying the face recognition algorithm just described, the system identifies the face through a rectangle which then allows us to remove the information that is not of our interest, extracting only the face. The ID card was created appositely for simulating the enrolment phase in the airport.

[5]https://cmusatyalab.github.io/openface/



2.3 Preprocessing and Data Augmentation:

In this work, some pre-processing steps have been implemented to make the dataset more consistent with the project concept and to make data available for the feature extractor model we want to build. In particular, our system, as already said, must be able to verify if the person, that is using the terminal for the check-in, is the same person in the ID card/passport image; this last one can be a grayscale image.

For this reason, we try two different approaches; the first one, working only with grayscale images, and the second one, randomly convert RGB to 3-grayscale-channels since the network has a predefined static input shape. This to simulate a combination of ID grayscale images and color images captured by the terminal.

After some tests, we found that using only grayscale images leads to better results so we decided to keep the RGB to grayscale conversion as a pre-processing step.

Also, as anticipated above, the network accepts, as input, images with the same shape but the samples in the dataset have different resolutions. As we know a normal resize to a fixed shape do not preserve proportions in the image, altering the face's distances and features and, basically, leading to an

inconsistent features vector. Accordingly to this problem, we implemented a 'resize to square' function that resizes: the shorter side of the image to the desired length, and the longer side to the size that keeps the proportions intact. Then, we randomly crop the excess of the longer side to fit the final image shape and normalize the data.

To conclude the image processing phase, we implemented also some data augmentation techniques like random vertical flip, random contrast adjustment, random brightness adjustment, and random noise. These operations are very useful in deep learning to dynamically create new samples from the existing ones, during the training phase, helping the convolutional network to generalize better and converge in a more optimal minimum. Despite these advantages, after having a closer look at the original images, we removed the data augmentation process because images in the dataset are already statically augmented; as we can see in the images reported below.

This means that if, on one hand, we don't have to pre-process every single image every time they are loaded in a batch, reducing a lot the training time; on the other hand, we can't virtually extend the dataset at each epoch, changing randomly one or more parameters with the data augmentation method, because we would apply some adjustments on images already augmented, compromising too much the quality of the samples as for example adding random noise in an already noisy image or flipping an image that in the dataset is just flipped.

3 Feature extraction:

In this chapter, we will analyze how the feature vector will be obtained, our biometric key, which will be able to discriminate between different identities.

The net presented by us, as just introduced, has a Siamese structure; this Deep Learning architecture consists in feeding different inputs to the same net and thanks to the triplet loss will be able to maximize the distance between images belonging to different identities.

But, to clarify the key concepts of our development, we divide this process into the various sub-chapters; we will analyze specifically: the neural network developed, the loss function, and finally a brief analysis of FaceNet, the network with which, in the final chapter, we will compare the results obtained.

3.1 The Network:

The Siamese network, as we can find in literature, is made up of several inputs; each of one, in our case, will be summarized in a feature vector.

The concept behind this structure is the concatenation of outputs. A Siamese net is any network that takes different inputs, keeping the same weights and parameters, to concatenate them at the end of the process. To then converge, the values obtained will be managed by a specific loss function, in our case, the just mentioned triplet loss, to learn the parameters and obtain each time a better encoding. To better explain this structure, we report the function used to obtain this system:

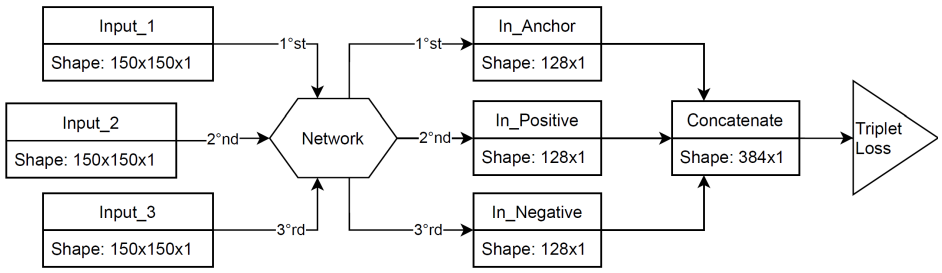
```
def build_siamese_network():
   input_1 = Input(img_shape)
   input_2 = Input(img_shape)
   input_3 = Input(img_shape)
   base_model = build_base_model()
   x1 = base_model(input_1)
   x2 = base_model(input_2)
   x3 = base_model(input_3)
   concat_vector = concatenate([x1, x2, x3], axis=-1, name='concat')
   model = Model([input_1, input_2, input_3], concat_vector)
   model.compile(loss=triplet_loss, optimizer=Adam(1e-3))
   model.summary()

return model, base_model
```

As we can see in the reported code, the model takes as input the three images, and as output, the concatenation of the three feature vectors computed by the network, *base_model*, for each input.

The model is then compiled, by defining the loss function and the solver; in our case, we opted for Adam with a learning rate of 10^{-3}. This because it is an adaptive method, with an adaptive learning rate, that tries to solve the problem of learning rate cancellation as RMSProp; it also implements Momentum to have fewer oscillations, accumulate "speed" to avoid local minima, for this reasons we expect that it will converge quickly.

Then we report a graph to summarize the concepts just introduced.



We can now proceed by analysing the main aspects of the network that we developed; because of this particular structure, it will not have a classification function at the end because, as we have just seen, we will be interested in these vectors for the learning procedure.

For the first thing, it is important to understand the number of embeddings, the size of our feature vector, in which the network will have to generalize the characteristics of the different faces.

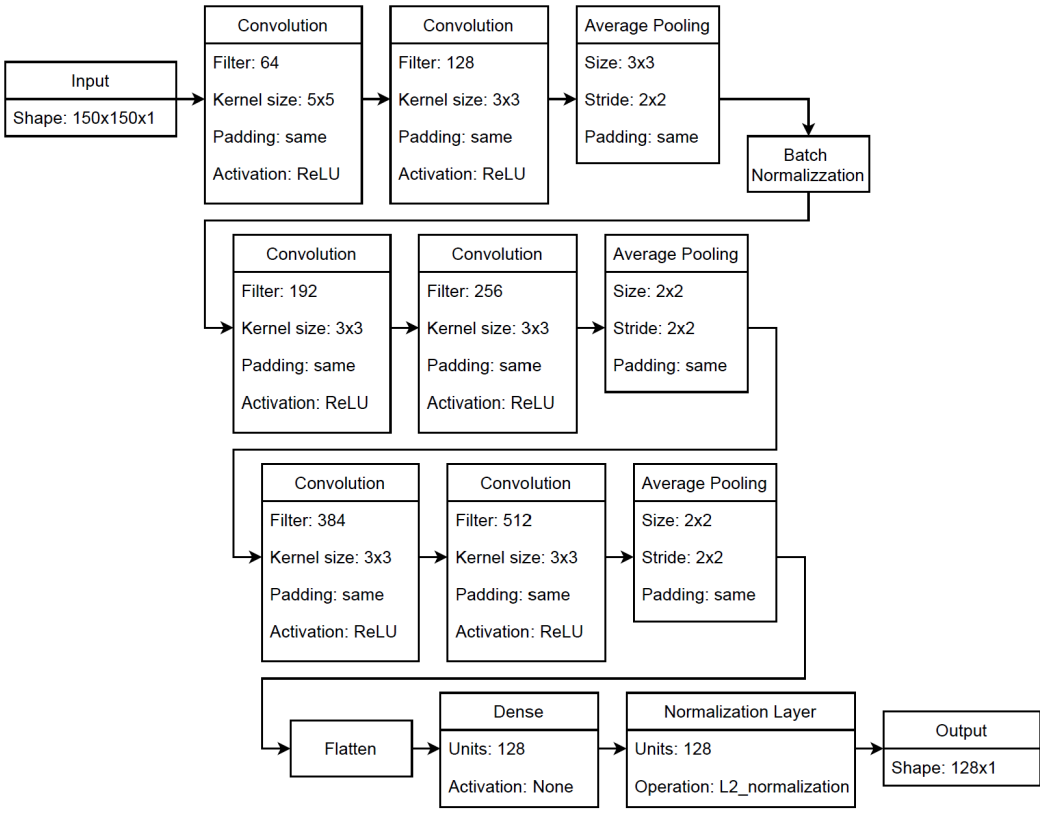
In the FaceNet paper[6], this test, called *embedding dimensionality*, is performed with different values from 64 to 512. The results obtained with their network show that the best results were obtained with a size equal to 128 and 256.

For our project, having limited computing power, we, therefore, decided to test the same network with 64 and 128 embeddings, to also limit the number of network parameters. To avoid a day of training we based on the values obtained by the two networks on the validation performed at each epoch. Each one is composed of 100 steps each of which with a batch of 32, so as we will see later from 32 triples of images.

[6]https://arxiv.org/pdf/1503.03832.pdf

The validation has found a faster convergence, lower loss values, for the network with 128 embeddings; value that we have then chosen for our model.

After having defined the main feature of our model and the expected output of the network, we can then proceed to observe its structure. First of all, we report in the image below its structure with the relative characterizing parameters.



As we can see, the network is organized into four levels, with a classical shape; it was created to maximize the number of convolutional layers while keeping the size of the filters as small as possible. This is because in general, smaller kernel sizes are preferred over larger for better performance in extracting useful features.

Also, the goal was not to increase the number of trainable parameters in our network due to the time and power limits of the system used; as already described in the introduction. Obviously, a deeper network will in general

obtain better results but it requires more time to converge. We have to notice that the results of our network are obtained with only $27M$ of parameters with respect to the $140M$ of FaceNet.

For this reason, it was necessary to use Pooling Layers which aim is to take the best output of the convolutional layer collapsing all the values contained in the applied window in a single value. As we know, however, we are going to lose the spatial correlation, an important factor in our application.

Otherwise, we are going to reduce a lot the resolution while keeping only the most important values in which we are interested. For those pros and cons, we decided to apply filters of a size no greater than 2x2 or 3x3 to try to not discard too much the data. As we can see in the net graph the 3x3 filter is used only in the first level of our net because the receptive field size is bigger.

Focussing on the convolutional layers, we can notice that: these have a filter size that tends to gradually increase from 64 to 512, with a classic incremental structure.

Also, we have to notice the presence of only 3x3 kernels with the exception of the first layer where the input information is greater. The choice of this dimension, 3x3, was made because it is the smallest possible odd dimension; since a 1x1 kernel is not used to extract features but to decrease or increase the number of filters in the network while keeping the number of parameters limited. Furthermore, the application of 2x2 filters is usually not recommended. Another common feature to all convolutional layers is the use of the same activation function, in our case the ReLU, which in general, and also in our project, works very well with images. To conclude we can notice that in the network each convolutional layer is never alone but they are always in a couple; this was chosen for each level of the net, in this way the next convolutional will focus better on the features of interest extracted by the precedent one before reducing the size of the input through the Pooling layer. A similar sequence of convolutional layers, but longer, will be applied in the FaceNet, as we will see in the next chapter.

To conclude this sub-chapter, let's look at the two normalization layers applied. The first, at the end of the first level, Batch Normalization, is used to normalize the data extracted from the first pair of convolutional layers.

While the other, in the last level, Normalization layer, takes advantage of the L2 normalization, which is essential to bring all the values contained in the embeddings into a determined range.

It is important to notice that the result of this normalization will then be

the output of our network; this is because otherwise, it would not be possible to establish a consistent and univocal threshold value, nor to calculate the Euclidean distance[7].

3.2 Loss Function:

As we can read in the literature, the Loss function is a fundamental component of a network; this because it is used to compute the gradients. Those are then used to update the weights of the network, making the learning process.

In our project, we read in many articles, that one of the preferred loss function for a face detection algorithm is the triplet loss. The loss function that was also implemented for the first time in the FaceNet paper.

We, therefore, decided to try to reimplement it or in any case to exploit its concepts to improve the convergence of our network.

The function needs four elements: three inputs and a slack variable α;

- The Anchor
- The Positive sample, belonging to the same class of the anchor
- The Negative sample belonging to a class different from the anchor and the positive
- The Slack Variable, α, representing the margin between positive and negative pairs.

These four components are then combined as shown in the formula below:

$$Loss = max(dist(Anchor, Pos) - dist(Anchor, Neg) + \alpha, 0)$$

The aim is to minimize the loss by ensuring that the sum of the distance between the anchor and the negative sample, plus the margin, is never greater than the distance between the anchor and the positive sample and also that this last distance tends to zero.

How can we read in the FaceNet paper:

The triplet loss, however, tries to enforce a margin between each pair of faces from one person to all other faces. This allows the faces for one

[7]Each features vector could have values over very far ranges.

identity to live on a manifold, while still enforcing the distance and thus discriminability to other identities.

As we can read, this loss function should improve and make the system convergence very fast. However, there are several variants of this function, those also reported in the paper; and they are called: easy triplets, semi-hard triplets, and hard triplets. The main difference is that the first one generates triplets with random images that cannot always bring an improvement to the network, the second one generates triplets where the negative is not closer to the anchor than the positive, and the last one, are triplets where the negative is closer to the anchor than the positive.

We tried to implement each of these by setting the margin equal to 0.4, as default, but the main problems were that: implementing the semi-hard, using greyscale images, the loss was fixed at 0.4 and the model collapsed. While in the use of hard triplets the time for the execution of a single epoch was more than an hour and a half; since for each batch, the network was looking for the triplets that best satisfied the distance requirements to minimize the loss. Also, to obtain better triplet samples we should had to use very large batches; FaceNet developers report in fact that:

In most experiments, we use a batch size of around 1,800 exemplars

With our instruments we have decided to use random triplets with a batch size of 32; approximately, the maximum supported, in order to not exceed the available space of the GPU. To conclude this chapter we report below the code of the used function, to better underline the development and the concepts just described.

```
def triplet_loss(y_true, y_pred, alpha = 0.4):
   anchor = y_pred[:,0:emb_size]
   positive = y_pred[:,emb_size:emb_size*2]
   negative = y_pred[:,emb_size*2:embg_size*3]
   pos_dist = K.sum(K.square(anchor-positive), axis=1)
   neg_dist = K.sum(K.square(anchor-negative), axis=1)
   loss = K.maximum(pos_dist-neg_dist+alpha, 0.0)

   return loss
```

3.3 FaceNet:

In this section, we are going to talk about the source of inspiration for our work: *"FaceNet: A Unified Embedding for Face Recognition and Clustering"*.

FaceNet is a deep neural network developed by Google; it is one of the most powerful tools for Face Recognition, claiming different accuracy records. It represents an innovative approach, in this field, since it aims to directly optimize the embeddings that are a map between face images and Euclidean space. In earlier models, the features extracted from the input image were the output of an intermediate layer rather than the final output, thus they were not optimized to represent an immediate similarity measure.

The key aspects of this new approach are: the model architecture, the triplet loss function, and the batch strategy; thanks to these three elements, FaceNet achieved the state-of-art in face recognition in Neural Networks.

The architecture is a straight deep convolutional neural network where the convolutional layers are characterized by an increasing number of filters and they are alternated by an average global pooling layer. In detail, only the first convolution block is unpaired and it has a greater kernel size of 7x7 while all the other convolutional blocks are made by, at least, two layers with an alternating kernel size of 1x1 and 3x3.

Comparing this structure with the network implemented in this work, we can notice how they share the structure of alternating pairs of convolutional followed by a pooling operation, with the first convolutional kernel greater than the others. Differently from our model, the last block of convolutional layers, in FaceNet, is made by three consecutive pairs and a final pooling. At this point, the output is flattened to feed the Dense layer, which output size is the desired embeddings size, 128. A least-square normalization is, then, applied to the final output. This last part of the architecture is the same we decided to implement for our model. This kind of network counts about 140M parameters.

The loss function, used for FaceNet, is the same we have already discussed in the triplet loss chapter but what makes it more effective is the batch strategy. The one used in the Google work is the semi-hard triplet that pushes the model to increase a lot the distance between the anchor and negative and to reduce the anchor-positive one. This kind of batch must be computed, at each step, using the current model's weights to predict the embeddings of a batch of random triplets and then extract only those that satisfied the semi-hard constraint. As they reported in their paper, they

picked for a mini-batch, every time, 40 images per identity, and some random images of different identity to extract the triplets for a final batch size of 1800 exemplars.

Due to this batch processing and to the architecture of the network, the training process took from 1000 to 2000 hours, plus 500 hours of fine-tuning, on a CPU cluster.

It is interesting to highlight some numeric differences between the State-of-art, FaceNet, and our implemented model. Starting from the number of parameters, respectively $140M$ and $27M$, the Google network is 5 times larger than the presented model, and also the training process took 250 times the time of our training.

Before analysing the effective comparison of this net and the developed one, we would like to conclude this subchapter with an analysis of the results obtained by FaceNet.

As we know, the classic evaluation techniques, FAR, FRR, ROC, ... , which we will see in the final chapter, are only the basis for measuring the quality of the developed algorithm but they are not sufficient for a complete evaluation; therefore we can opt to use specific protocols or datasets as benchmarks. For this reason, we have discovered on a website[8] that reports the state of the art tables for the face recognition tasks, it reports the results obtained after the application of a series of different benchmarks performing face recognition and authentication, that FaceNet, with an adaptive threshold, achieved the highest results with the following accuracy values:

- 84.3% on Adience
- 83.8% on FERET

[8]https://paperswithcode.com/task/face-recognition/codeless

4 Classification:

Once the features extractor model completed its training, we can compute the embeddings of any face and associated each of them to the specific person by labeling it with a unique value[9].

At this point, every time an enrolled person wants to identify himself, a new face sample is acquired, after the computation of its embeddings, it can be classified among all the processed identities, our classes; this process represents the face recognition. As we can see, this operation is a typical multi-class classification where the values are the embeddings and their class is the identity of the person.

To solve this problem, we decide to use a linear classifier model called Support Vector Machine, SVM. This algorithm tries to find a hyperplane, in an n-dimensions space, that separates samples of two different classes (binary classification). This is done with an optimization process that tries to maximize the distance, called margin, of the closest points (support vectors) to the hyperplane. Higher is the distance between the closest points and the hyperplane, higher is the accuracy of the model. In our case, the multi-class classification can be reduced to multiple binary classifications.

We built, also, a simple neural network model to be trained over the embeddings samples in the dataset and which output is the respective predicted class. This model shows an accuracy similar to the SVM but with some disadvantages like the uncertainty of when the network should stop its training, to avoid overfitting, fine-tuning of different parameters, also in function of the number of classes, ...

Due to these reasons, we decided to keep the SVM that is more reliable and easy to train; as we can see in the sequent chapter.

[9]This value can be: an ID, name and surname, postal code, etc...

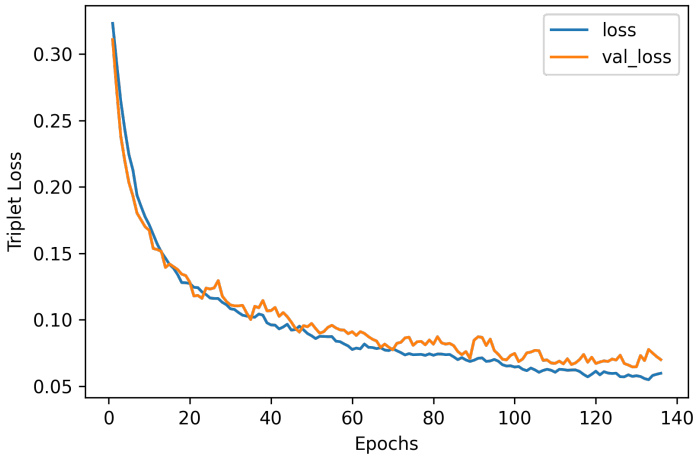
5 Results:

In this final chapter, we report the results obtained through the classical methods studied, used to measure the performances of biometric systems. To give greater value to the results, and contextualize the project, we will compare the graphs obtained with the pre-trained model of FaceNet; obviously, on the same test dataset.

This last one consists of 400 classes, this value was chosen because it is approximately equal to the average number of people present on medium and short flights, and was specially made by us. In fact, for the test, we decided to create a dataset in which each image for each class did not have evident variations of expression or evident occlusions[10]. On the other hand, we have maintained images of faces with different lighting conditions and poses.

Our test dataset was then used for two different tasks: the verification, based in checking if given two images those belong to the same person or not, and the identification, where given the face and relative extracted embeddings, we attribute the respective identity.

However, before carrying out this comparison, as just introduced, in this chapter we show the results obtained; for this reason, we want to report the graph representing the convergence of the network.



In the graph are reported: the loss values, in blue, and of the validation loss, in orange, the term used to understand the convergence of the network

[10]Leaving only pictures where glasses or hats were present

and the learning process. As we can see, on the y-axis, the train starts with a loss equal to the margin, α, fixed in the triplet loss, and it decreases until it tends to zero. It can also be noted that due to the randomicity of the triplets, the validation is not always convergent with the convergence of the loss.

Anyway, with a total of about 10 hours of training, 138 epochs, we are still satisfied with the results obtained and those that we are going to compare.

For the first task, the first comparison we are going to show is between the graphs representing the Equal Error Rate, the point of intersection between FAR and FRR, the value usually chosen as threshold of the system.

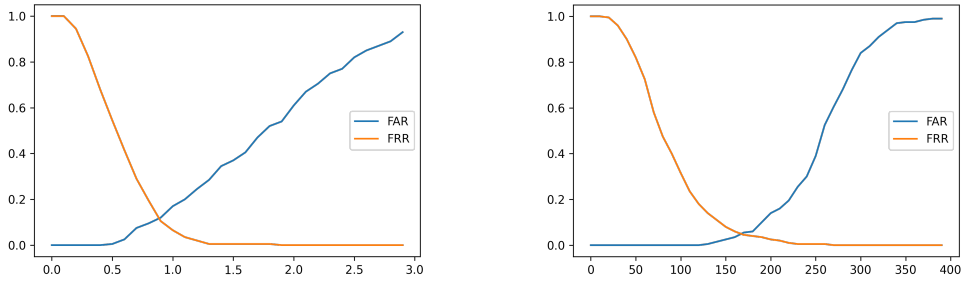


Figure 1: OurNet, on the left, and FaceNet on the right

As we can see, the main difference between the two graphs lies in the point of intersection between the two curves, where on average there is the same number of false alarms and missed false alarms.

We can notice how the y-coord of the EER point, the optimal point where the number of error of the system is minimized, is lower in the right graph with respect to the other; this means that the best that the models can do, FaceNet will be always better than the our.

Testing our network we observed that on average the distance between two samples of the same class is about 0.74 while between two different classes 1.67.

Then to conclude this task, we report each graph for the GAR, and the overlapping of the two models for the ROC, and DET curves. The first method, GAR, represents the number of genuine acceptance when increasing the threshold, the second one, ROC, is computed as FAR on GAR, this means the ratio of impostors acceptance on the range of genuine acceptance,

and the last one, DET, is computed as FAR on FRR, this means the ratio between impostor acceptance on genuine rejected with different threshold.

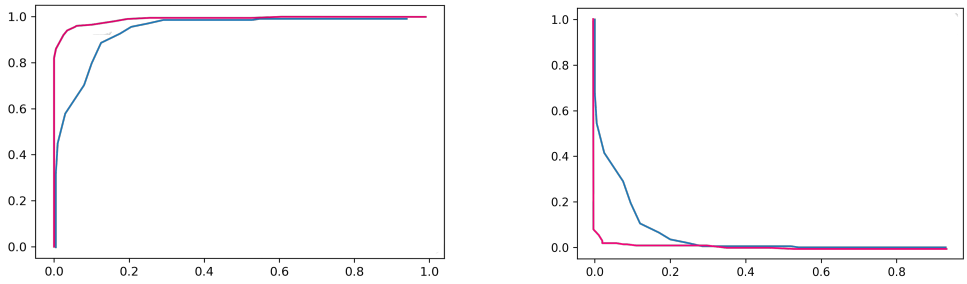


Figure 2: OurNet, in blue, and FaceNet in purple, ROC and DET

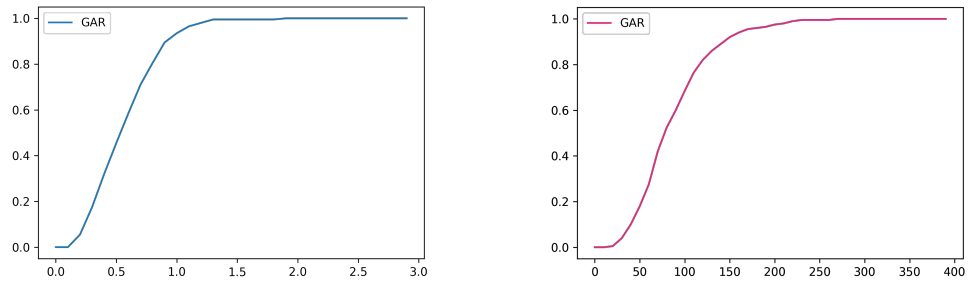
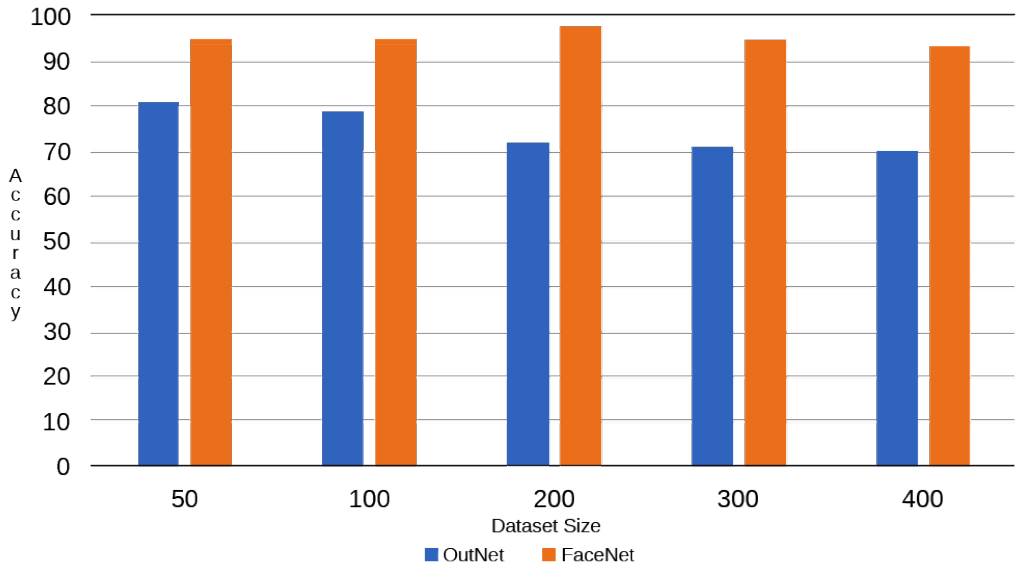


Figure 3: OurNet, on the left, and FaceNet on the right, GARs

For the identification task, after the classification, we measured the performance obtained under different conditions, with different dimension of the dataset. For each of these, we measured the accuracy obtained by the classifier, testing each class with an image never seen before. This test is made taking in account that we are in a close-set, where we know that each class will be in our database and we have to determine from which class the subject belongs.

In the following graph we report the different results obtained for the accuracy for each of the two models changing the number of classes: 50, 100, 200, 300, 400. As we can note our network is more sensitive to variations, going from 70%, with 400 classes, to 81%, with 50; otherwise, FaceNet is more stable remains constant at about 95%. This robustness is probably given by the greater distance between the samples; results obtained thanks

to the use of the mentioned semi-hard triplets, as already described, with respect to the one used by us, random-triplets.



We try also to test the system with the open-set assumption, where we don't have the prior knowledge if the face that we have to classify is in the database or not; we implemented two different approaches: one based on the SVM classifier and the other on a simple Neural Network.

As a first thing, we have to find a threshold to establish if the predicted result can be accepted using a measure of confidence in the prediction made.

Since the SVM is trained using the cross-validation evaluation, its aim is to increase the accuracy, so the prediction probabilities can't be used as a confidence value. For this reason, we implemented a identification-verification system. It works in the following way; given an image to the SVM, it predicts an identity and, then, we use the distance between this new image and one of the predicted identity[11] as a confidence value on which we can apply a threshold.

For the second approach, the neural network, things are easier. The network, is trained aiming at reducing the categorical cross-entropy loss; this means that the output probabilities are more reliable and they can be used as a confidence values for the threshold.

[11]That is stored in the database

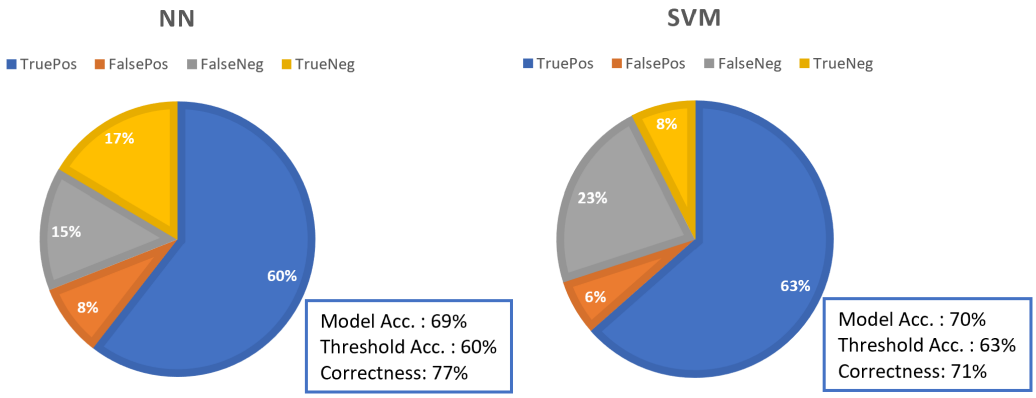


Figure 4: OurNet

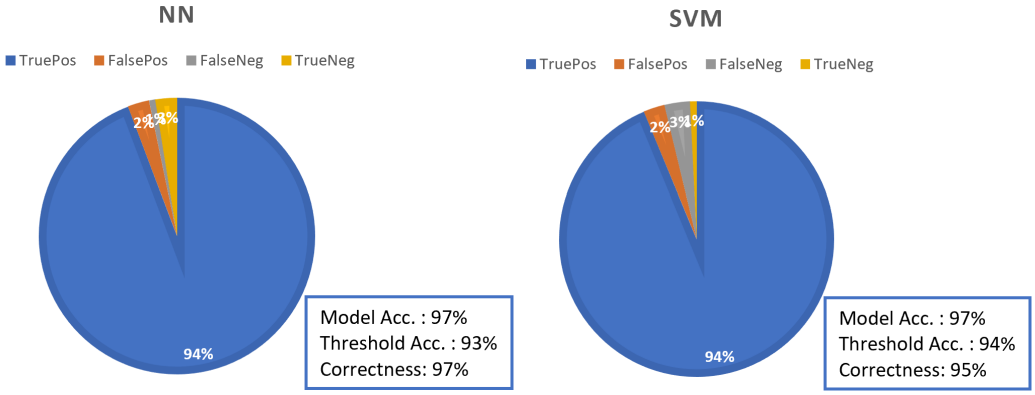


Figure 5: FaceNet

As we can see, in the graphs above, we reported a comparison between the two tested classifier models, SVM and a simple NN, for both the models.These images have been computed by classifying 400 different faces and then applying an optimal threshold to accept or reject the prediction.

In particular they show the True Positive, True Negative, False Positive, and False Negative[12] values. Also, are reported, the model Accuracy, the Threshold Accuracy, and the Correctness[13].

It is possible to see how both the classifiers achieved similar results, as just

[12]Correctly accepted faces, Correctly rejected faces, Incorrectly rejected faces, and Incorrectly accepted faces

[13]Correct predictions without applying the threshold, Accepted correct predictions filtered by the threshold, and Accepted correct prediction plus rejected wrong prediction

introduced in the classifier chapter, while there is quite a difference between OurNet and FaceNet.

We can conclude by saying that, although our network is simpler, it has obtained good results, probably its best application would be for flights on short distances where the number of people, classes, is between 100/150. Obviously, for a larger-scale application, the use of FaceNet is better in every aspect.