POLITECNICO DI TORINO

Master's Degree in Computer Engineering

Master's Degree Thesis

BiometricNet

A deep learning-based approach for biometric authentication

Thesis supervisor

Prof. ENRICO MAGLI

Candidate

STEFANO BRILLI

JULY 2020

Abstract

This thesis presents an analysis of a novel, deep learning-based approach for user verification. Face verification is the task of comparing a candidate face to another and verifying whether it is a match. The traditional approach consists of relying on analytical metrics to shape the classification boundary. Instead of defining a metric, our approach allows the network to inherently learn it by mapping matching and non-matching face pairs onto different statistical distributions. Although any class of target distributions can be applied, using the Gaussians is a logical choice since the natural output of large enough fully-connected layers comes to be Gaussian. Moreover, since their masses tend to stay close to a single value, a threshold-based classification can be employed for the verification.

To Carla, Dario and Riccardo...

Table of Contents

List of Tables

List of Figures

Acronyms

FR

Face Recognition

FV

Face Verification

AI

Artificial Intelligence

ML

Machine Learning

DL

Deep Learning

NN

Neural Network

CNN

Convolutional Neural Network, a family of neural networks.

FC

Fully Connected, referred to fully connected layers employed in neural networks.

TP, FP, TN, FN

Regarding to the confusion matrix, these acronyms mean True Positive, False Positive, True Negative, False Negative.

Chapter 1

Introduction

Authentication systems based on biometric analysis have become very popular in the last years since they both avoid the users to remember passwords and to physically prompt a key for getting access to a system. Moreover, progressions in designing convolutional neural networks (Section 2.2.2) have conducted to achieve very high performances in many computer vision tasks. One of them is the face recognition task, i.e. the ability to analyze and detect useful characteristics from a face. As explained in detail in Section 3, we can be more precise, discriminating between "face identification" which means labeling each face, and "face verification" which means deciding whether two faces represent the same person.

This thesis explores new solutions for implementing face verification systems by using deep learning techniques. In particular, as will be explained in Chapter 4, we employ a novel approach for achieving good results, better than the one performed by state-of-the-art methods.

1.1 Structure of the paper

The thesis is organized as follows:

- **Chapter 2**. An introduction to Machine Learning and Deep Learning
- **Chapter 3**. Introduction to Face Recognition: it presents a comprehensive introduction to the approaches for implementing FR tasks, and some actual applications
- **Chapter 4**. Introduction to BiometricNet: an introduction to the proposed approach for implementing FV, with a complete discussion about it
- **Chapter 5**. Experiments and results: experimental settings and results of our experiments

- **Chapter 6**. Conclusions and future works: an analysis of the possible improvements for BiometricNet and a recap about the work.

Chapter 2

Introduction to Machine Learning and Deep Learning

Machine learning has grown a lot in the last years due to the large set of fields it can be applied: finance, health, academic research, entertainment, and social sciences. Its growth is perhaps ascribed to the availability and reduction in costs of the hardware needed for implementing the algorithms (CPU, GPU, cheap cloud services, etc.).
Another reason is the availability of well-documented frameworks, such as Tensorflow [1], PyTorch [2], Keras [3], etc. Currently there are more than 41k repositories matching *Machine Learning* keyword on GitHub [4] and more than 23k repositories matching *Deep learning* keyword [5].

This chapter provides a comprehensive overview of the principal ML and DL topics and techniques.

2.1 Definition and basic concepts about Machine Learning

Artificial intelligence can be defined as intelligence demonstrated by machines, in contrast with the natural intelligence demonstrated by humans. According to [6], *"Intelligence measures an agent's ability to achieve goals in a wide range of environments."*
Machine learning is *a set of methods that can automatically detect patterns in data, and then use the uncovered patterns to predict future data* [7]. These patterns are learned by computers, that become capable of understanding what data represent and how to manipulate them to get an output. Instead of programming a computer for doing something, it learns how to do that. Instead of writing code for teaching

to a robot how to walk, the robot learns how to walk by trying and trying again.

These examples show a shifting in the programming paradigm: In traditional programming, the programmer writes rules, i.e. the logic (through code) and obtains such an output providing such an input. On the other hand, an ML algorithm can be seen as a black box that gets data and related outputs and, performing some kind of operations, returns the logic of the program, i.e. a way to map inputs onto the outputs (an illustration is reported in Figure 2.1).

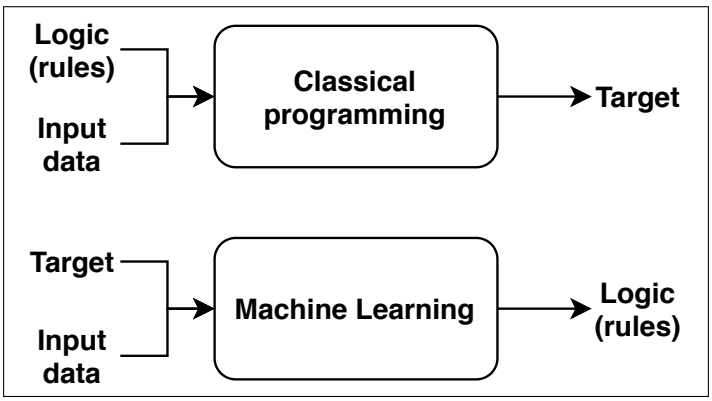


Figure 2.1: Paradigm shifting introduced by ML

Preprocessing of data is a very important step in the model's building pipeline. Since computers can only understand numerical information, it is important to process and convert data in such format. We call *samples* the observations we have in our dataset or database, while we refer to *features* for indicating pieces of information each sample is composed of.
Each feature can assume a value according to the context it is involved in. For instance, if our dataset is composed of samples about people, we may have a feature that refers to the person's age (a numerical value from 0 to 99) and a categorical value that indicates the person's sex (0 for male and 1 for female).

2.1.1 Types of machine learning

Machine learning can be divided into two subcategories:

- **Supervised learning**. In this approach the goal is to learn a mapping between the input data x and the output data y. An example of a task is the spam filter, that can learn how spam and ham (it refers to good emails) emails are mapped onto the output space and take decision when a new email comes. The data already labeled and used by the model to learn the mapping belong to the *training set* since they are used to improve the knowledge of the model. The unlabelled data, i.e. the ones the model looks at in order to take a decision

belong to the *testing set*. During the training phase, just the training set must be used, because the goal is to decrease as much as possible the generalization error, i.e. the model has to learn how to handle data in general, without specializing on training samples.
Supervised tasks can be further divided into two categories, according to the type of output data.

A *classification* task seeks a mapping between input data and categorical labels. For instance, the spam filter reported above belongs to this category.

On the other hand, when the output labels are continuous values the task is called *regression*. Predicting the stock value of a company in the future is a regression task.
Supervised algorithms require a huge human effort because each sample has to be labeled for being used as a training one.

- **Unsupervised learning**. Algorithms belonging to this category do not require labels, because their goal is to split data into categories according to patterns learned by data. An example of an unsupervised algorithm could be the identification of topics in a set of online discussions so that the ones that talk about sports fall into a different cluster with respect to the ones that talk about animals.

2.2 Deep learning

As shown in Figure 2.2, deep learning is part of the ML family, even though there are some important differences.

Conventional machine-learning techniques were limited in their ability to process natural data in their raw form. For decades, constructing a pattern-recognition or machine-learning system required careful engineering and considerable domain expertise to design a feature extractor that transformed the raw data (such as the pixel values of an image) into a suitable internal representation or feature vector from which the learning subsystem, often a classifier, could detect or classify patterns in the input.[8]

Deep learning algorithms work differently and overcome these limitations. They are capable of working with raw data, so no features engineering is required. Moreover, the expert is not required to know the working context and the proper family of function to use to approximate the mapping function.
These methods can approximate complex functions by using multiple levels of non-linear transformations. At each level, data are taken as input, manipulated, and outputted to the next layer, in a very deep fashion.

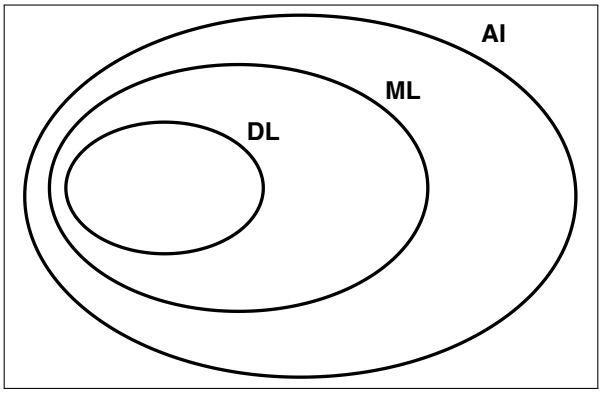


Figure 2.2: Relationship between AI, ML and DL

2.2.1 Introduction to Neural Networks

Neural networks are the most common algorithms for implementing DL. They are computer systems that can learn how to map input data to output labels, without prior knowledge and with no need to specify the rules. The programmer is required to just provide labeled data according to the context, a way to measure the goodness of the predictions, and a way for updating the model itself. The result is achieved by performing a trial-and-error approach until a certain condition is met.

Implementation

Neural networks are an inspiration of the human brain. A NN is composed of a series of nodes called *artificial neurons* (similar to the brain neurons) and edges (like the synapse) that bring signals from a neuron to other ones. Each edge brings a signal that is modulated by its weight, so higher the weight of an edge, stronger the contribution of the neuron that edge starts from. Once all the signals come to a given neuron, they are summed and passed as input to a non-linear function and its output becomes the value to bring to the next neuron through the next edge. Figure 2.3 shows an example of neuron in a NN.

A neuron implements the functions

$$z = \sum_i w_i x_i + b_0 \tag{2.1}$$

$$y = \sigma(z) \tag{2.2}$$

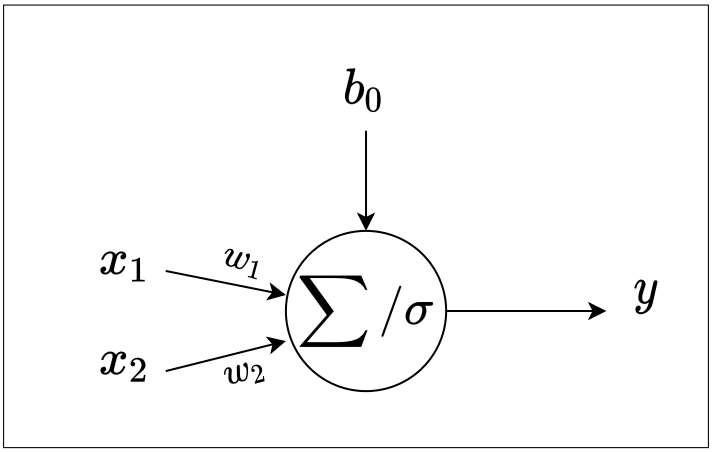


Figure 2.3: The neuron: a scheme

where:

- z is the total signal coming into the neuron
- x_i is the value of the i_{th} feature of the sample
- w_i is the weight of the i_{th} edge
- b_0 is called bias and could be seen an extra input
- σ is the non-linear activation function
- y is the neuron's output

The neurons that perform the same transformation, being at the same depth, are grouped in *layers*.

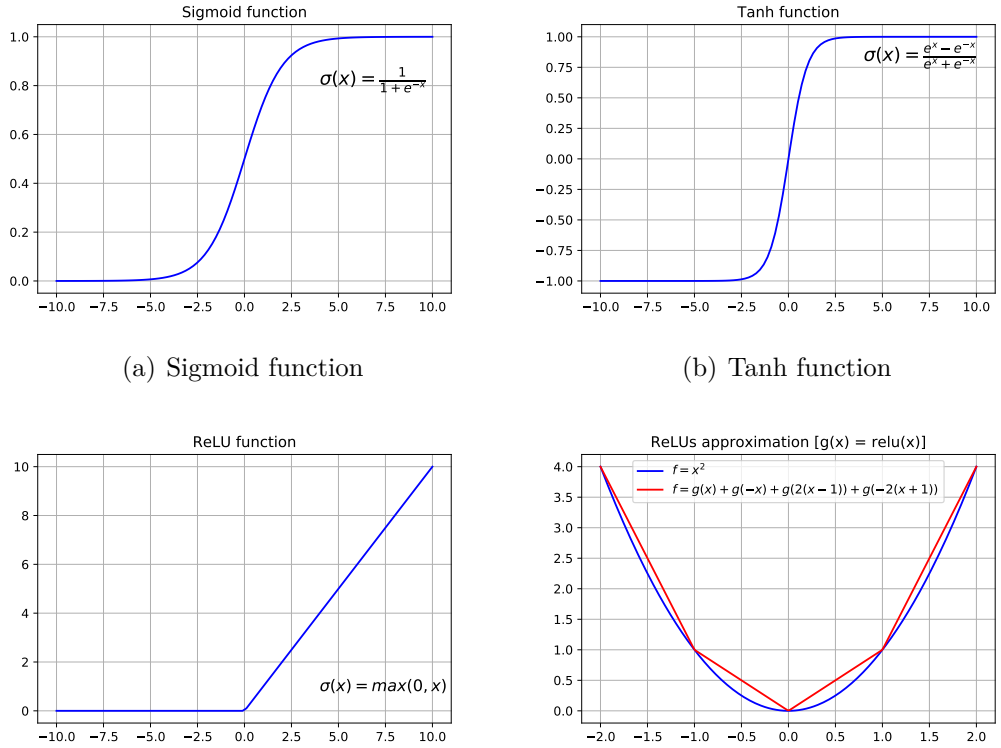
The activation function *sigma* is the most important part of the neuron and, in general, of the model. Utilizing non-linear functions, a DL model can approximate almost any function, even the most complex. The most common activation functions used in DL are the following.

- **Sigmoid**. It is defined as $\sigma(x) = \frac{1}{1+e^{-x}}$ and squashes any value between 0 and 1 (Figure 2.4a).
- **Softmax**. It is a function that takes a vector of real numbers and normalizes it into a probability distribution. After having applied the softmax function to these numbers, each entry of the vector contains a probability score and the sum of all entries is one. It is defined as $\sigma(x_i) = \frac{e^{x_i}}{\sum_j e^{x_j}}$.

- **Hyperbolic Tangent (Tanh)** It is defined as $\sigma(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$ and squashes any value between -1 and 1 (Figure 2.4b).

- **Rectified Linear Unit (ReLU)** It is defined as $\sigma(x) = max(0, x)$ (Figure 2.4c).

Figure 2.4d shows that a combination of ReLU functions can approximate pretty well a parable function.



(a) Sigmoid function

(b) Tanh function

(c) ReLU function

(d) Parable approximation

Figure 2.4: Example of activation functions

Loss function and weights update

"Most deep learning algorithms involve optimization of some sort. Optimization refers to the task of either minimizing or maximizing some function $f(x)$ by altering x." [9, p. 82]

The loss function also called cost function, is a function that maps an event onto a real number. In implementing DL algorithms, we use such a loss function for getting the amount of error our model did in mapping inputs during the last epoch. If the loss function returns a large number it means that we have to modify a lot our network since it is performing badly. A small number, conversely, means that our model can approximate well the function that maps input onto the outputs, and the network parameters should not change too.

To decide how much each weight should be updated we use a technique called *backpropagation* [9, pp. 204–210], which consists of computing the partial derivative (using the chain rule) of the loss for each network parameter. The rationale is that computing the derivative of the loss for such parameters, we can understand how much the loss function changes when its inputs change. Updating the weights in the decreasing direction we can minimize the loss function and improve our model performances.

The last actor in this process is the *optimizer*. While the loss function defines how much error we have to backpropagate, the optimizer defines how to technically do that (it defines the updating algorithm). Examples of optimizers are SGD [10] and ADAM [11].

It is worth citing two common problems that can occur during the backpropagation phase. *Exploding gradient* occurs when the gradient becomes too large to be fit in memory, causing overflow and making very unstable the network. Overflow is caused by the chain rule since if such a deep layer has a gradient > 1, it will go through continuous matrix multiplications until it becomes *NaN*. On the other hand, if the gradient of deeper layers is < 1 it will be continuously multiplied for weight matrices until it will become too small. This situation is called *vanishing gradient*, and it causes the model to stop learning.

2.2.2 Convolutional Neural Networks

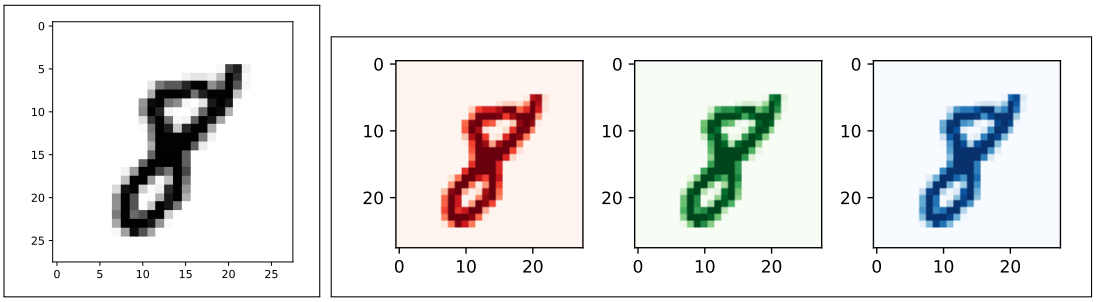
In fully connected neural networks each neuron is linked to all neurons of the next layer. Building a classifier by using such a network is possible and efficient for a large set of tasks, like solving the XOR problem, but it could be infeasible for other tasks, like implementing an images classifier.
Building an image classifier by using an FC network could lead to at least three problems related to the very high number of parameters required by such a network [12, pp. 3–4]:

- **Overfitting**. If the dataset is small and there are a lot of trainable parameters, the network could learn a perfect representation of the training data and the loss could go to zero. Nevertheless, if fresh data are different from the ones used in the training phase, the performances of the model could degrade a lot.

- **Out-of-memory issues**. Having too many parameters could lead to a memory overflow.
- **Structural limitations**. FC networks do have no built-in invariance concerning translations or distortions of the input image.

CNNs are a class of deep neural networks primarily used for analyzing images. They were inspired by the experiments of *Hubel* and *Wiesel* on the mammalian visual cortex in [13].

CNN is similar to a traditional neural network. It has an input layer, some hidden layers, biases, and an output layer. However, such kind of networks differs for the shape of input data and weights. As we have seen in the previous section, traditional neural networks do have inputs and weights as scalar values.
Inputs and weights of a CNN are instead multidimensional, like images. A black and white image is composed of just two dimensions (Figure 2.5a), while an RGB image has a third dimension that indicates the channels, i.e. the amount of red, green and blue each pixel is composed by (Figure 2.5b). In this case, we indicate such input as a *tensor*, a structure of dimensionality higher than two. Differently from traditional network parameters, that are scalars, CNNs have multidimensional parameters too.



(a) A black and white image (b) An RGB image

Figure 2.5: Examples of CNN inputs

They combine two architectural ideas that solve the problems listed above. Indeed, its hidden layers are characterized by a *local connectivity*, ensuring that neurons can extract simple features like edges and corners and high layers will combine them to extract more complex features [12, p. 5].
Moreover, the layers' weights are shared among neurons, and this ensures both a lower number of free parameters, solving the out-of-memory problem and the fact that the same feature can be extracted independently by its position within the image. [12, p. 5]

CNN is commonly composed of three types of layers:

- **Convolutional layer**. It consists of a set of filters that are *convoluted* across the input image. Convolution is the linear operation that gives the name to CNNs [9, p. 330] and is defined as

$$s(t) = (x * w)(t) = \sum_{a=-\infty}^{\infty} x(a)w(t-a) \tag{2.3}$$

  The result of this operation is called *feature map*.

- **Non-linear layer**. The feature map is passed as input to a non-linear activation function to apply the non-linearity to the signal. Some activation functions have been presented in Section 2.2.1

- **Pooling layer**. This layer is used to reduce the dimensionality of the representation and so the number of free parameters the next convolutional layer will handle. This allows controlling overfitting as well.

2.2.3 ResNet: an overview

Residual Networks are a special type of neural networks introduced in [14].

The special feature of ResNet is the use of the so-called *identity shortcut connection*, that allows the signal to flow over the network by skipping some layers. The motivation for skipping layers is to avoid the vanishing gradient problem (Section 2.2.1) by reusing the activations of previous layers until the adjacent layer learns its weights.

As will be explained in the next chapters, we will employ ResNet as part of our models since they exhibit good performances for our purposes.

Chapter 3

Introduction to Face Recognition

A facial recognition system is a technology capable of identifying or verifying a person from a digital image or a video frame from a video source.

FR systems are widely used in the security area. Such systems can be found in any place where facial verification or recognition is required for granting access to users or to validate some kind of badge: airports, universities, banks, stadiums, and so on.

The face recognition field can be subdivided into three more specific fields of application:

- **Face identification**. It answers the question "who is this person?". Identification is the action of guessing the identity of a person, so the set of data must contain at least one picture of all people we want to build a classifier on. Such a classifier can be used, for instance, for automatically labeling a picture according to the subject within it. It computes a one-to-many similarity since the probe face is compared with all the pictures in the dataset to find the match.

- **Face verification**. It answers the question 'are these pictures representing the same person?'. Verification is the act of deciding if two representations (an image, a video, and so on) are referred to the same person. It can be employed in situations in which is more important to detect a match between two people instead of labeling a person. It computes a one-to-one similarity between the probe image and each image of the dataset, to decide whether the two images are of the same person.

- **Face clustering**. It means to find some common features between faces and grouping people according to them. Clustering algorithms can be employed for

grouping faces of the same person in the same group of pictures, and pictures of other people in other groups.

It is worth mentioning that there exist other recognition techniques based on human beings' characteristics: among others, there are fingerprint and iris recognition systems. Both of them can only work in a very constrained situation since the person has to put her finger into a scanner or look at an iris camera by a short distance. So they can be only employed when the person is close to the scanner and nothing can occlude the scanning; these techniques are invasive but they do not suffer any distortion or occlusion situation.

On the other hand, face recognition techniques can work in almost any situation: the person can be pretty far from the camera and she could not even notice the presence of it. However, lightness paucity and an unconstrained position of the person could lead to misclassifications and so to an error. Researchers are working on systems able to overcome these drawbacks.

3.1 Some famous applications of face recognition

Face recognition systems are studied by a lot of researchers around the world and are far from working perfectly. However, such systems are widely used in some contexts. In the following are reported some examples of common daily usage of face recognition, and one example of a situation in which it failed.

The Boston Marathon

During the marathon on April 15, 2013, two bombs were placed, killing three people. FBI tried to use a face recognition system to find and catch the suspects. Their idea was simple: for each face captured by surveillance cameras, the system had to compare each of them with faces contained into the Massachusetts Registry of Motor Vehicles database, where driving license pictures of all citizens were stored. They failed for two reasons [15]:

- The driving license pictures were taken in a very constrained manner: same angle, same position, same quality, and same lightness. Instead, pictures provided by the surveillance cameras were taken from unconstrained angles, with variable lightness, the subject could have never looked toward the camera and the quality was low.

- The driving license's picture of Dzhokhar Tsarnaev, the suspect the FBI was looking for, was taken when he was 16, so the system was not able to detect a match.

This story highlights the challenges researchers are facing nowadays. Having systems that work only in constrained situations is useless since real situations happen in unconstrained manners. They are working for building systems that can recognize people regardless the age, lightness, or position.

Face unlocking system on smartphones

Smartphones have become commonplace in our life; we use them for taking pictures, chatting with friends, and keeping secrets. To keep hidden all these things, the devices have to offer a way to not allowing access to anyone who is not the owner.

First sold smartphones could be unlocked by prompting a security code or a symbol. These approaches were useful, but not safe and pretty uncomfortable. Years later fingerprint scanners were implemented onto the devices, by allowing much more high security but not solving the inconvenience since the user had to physically touch the scanner.

In the last years, more advanced smartphones were sold with an integrated face recognition scanner. Such a system allows very high security and a comfortable action for the user since it has just to look at the frontal camera for unlocking the device. This is an example of a face verification process, that will be explained in the next section.

3.2 Preprocessing of the data

In Section 3 we talked about the differences between face authentication and authentications based on other biometric features, such as iris and fingerprint. We said that building authentication systems based on these two latter biometric features could be easier, at least for the quality of the provided biometry, since the user has to look at, or put his finger on, a scanner.
However, having few sources of error comes at the cost of a more intrusive situation for the user.

On the other hand, an authentication system based on face recognition can be employed with much fewer constraints for the user. Face pictures can be taken at a distance, and such systems can be much less invasive to the ones we talked about above. The cost of this low level of constraints is that there can occur a lot of variations and modifications in the quality of the picture that the recognition could not work properly. For instance, using a face recognition system for surveillance purposes lead to take faces with a lot of variations in pose and illuminations, or detecting a partial or total occlusion of the face. In general, taking pictures in an uncontrolled situation could lead to a degradation of performances because of noise in the picture.

As pictures were taken in uncontrolled situations can contain a lot of extra things, like animals and cars, or the same picture could contain different people, we need a system that can detect people and align their faces. However, detecting and aligning faces are challenging operations due to the already discussed variations in pose, illumination, and occlusion. A group of researchers has recently demonstrated that a poor alignment on low-resolution pictures taken in a controlled situation, is the major problem that causes a poor recognition performance, while poor performance in using pictures taken in uncontrolled situations is caused both by the variations in pose and illumination and a poor alignment [16].

3.2.1 Detection and alignment

For preprocessing data we follow the strategy employed by other works such as [17][18]. MTCNN (Multi-Tasking Convolutional Neural Network)[19] is a complex but powerful system for face detection and alignment. In the following will be reported a description of how it works. MTCNN aims to detect a face (or faces) within a picture and then align it utilizing five facial detected landmarks. Having an input image, different copies of different sizes are created by the system, to form a pyramidal structure.

MTCNN is composed of three convolutional neural networks and each of them is involved in a preprocessing step. Figure 3.1 shows the stages of MTCNN.

- The first network is called *Proposal Network* (P-Net). It takes the test image as input and produces the candidate's facial windows of the image. These windows can be seen as regions of interest of the image. Since there could exist a lot of overlapping windows, a *Non-Maximum Suppression* (NMS) algorithm is used for merging too many similar windows.

- The second network is called *Refine Network* (R-Net). It takes as input all the proposal windows produced by P-Net. It performs calibration and again applies the NMS algorithm for merging too many similar windows. During this phase, a lot of false-positive candidates are rejected.

- The third and last network is called *Output Network* (O-Net). It performs similar to the previous network since it applies again NMS algorithm. This network identifies five regions of interest of the face (left eye, right eye, nose, left corner of the mouth, right corner of the mouth) and returns just one single bounding box, the one containing the aligned face.

For BiometricNet we employ MTCNN to generate normalized facial crops of size 160 x 160 pixels, performing an alignment based on five facial points. In particular, our benchmark pictures are cropped to size 160x160 px, while the training pictures are cropped to size 182x182 px with a margin of 22 pixels on each side to perform

random crops, enhancing data augmentation, and reducing overfitting during the training. Moreover, the images are mean normalized and constrained in the interval [-1,1], as done in [17][18].

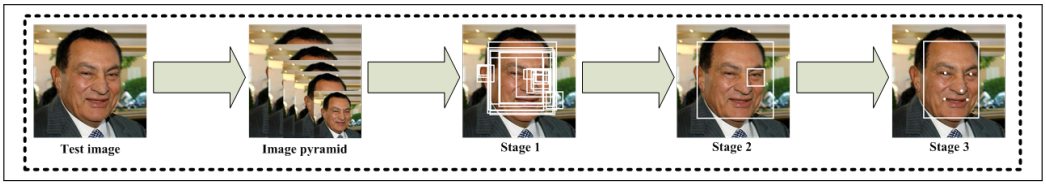


Figure 3.1: MTCNN pipeline

3.2.2 Datasets

This section provides a brief description of the datasets we used for conducting experiments. Choosing good training sets is crucial for achieving satisfying performances and, in the FR field, that means choosing a pretty large set that contains as much generalization between faces as possible.

For each dataset, some statistics (for instance number of pictures and number of different people) and some example pictures are reported.

Training sets

The following datasets were used for training our models. It is worth saying that we have used them by following no specific rule, in the sense that we could have used one of them for the first two phases of training and another one for the final finetuning. However, a detailed description of the datasets used for each phase will be reported in the experimental section.

Dataset	#photos	#subjects
CASIA Webface[20]	0.49M	10K
MS1M-DeepGlint[21]	3.9M	87K
Asian-DeepGlint[21]	2.83M	94K

Table 3.1: Training sets

Benchmark sets

In order to test the goodness of our models, we employ the following datasets. As said earlier, the work behind this thesis is focused on face verification so, for all

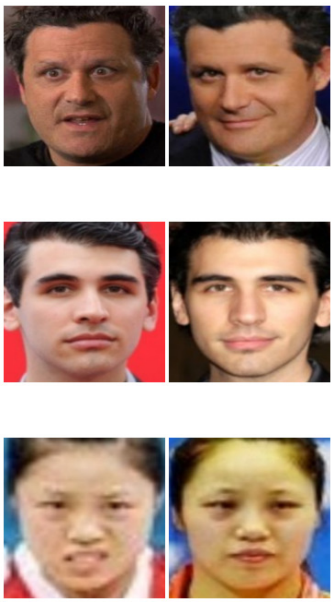


Figure 3.2: Pictures taken from CASIA, MS1M-DeepGlint and Asian-DeepGlint respectively.

the datasets, we report the results achieved by testing 6000 pairs of face images, 3000 matching pairs, and 3000 non-matching pairs.

Dataset	**#photos**	**#subjects**
Labelled Faces in the Wild (LFW)[22]	13.2K	5.7K
YouTube Faces (YTF)[23]	3.4K	1.6K
Celebrities in Frontal-Profile (CFP-FP)[24]	7K	500
Cross-Age LFW (CALFW)[25]	12.1K	5.7K
Cross-Pose LFW (CPLFW)[26]	11.6K	5.7K

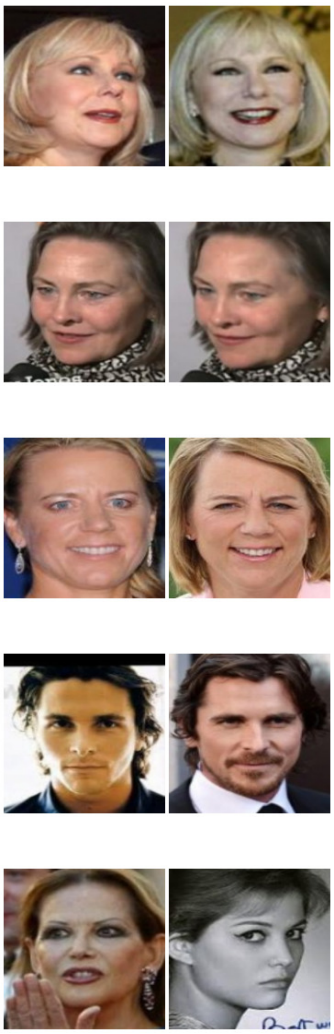


Figure 3.3: Pictures taken from LFW, YTF, CFP-FP, CALFW and CPLFW respectively.

3.3 Face verification steps

Since the topic of this thesis is to build a face verification system (technical details later) it is worth listing the steps involved in this kind of approach.

Enrollment

The user is required to provide a picture of himself in a constrained situation in terms of position, lightness, and without any accessory (sunglasses, hats, etc.). The

system extracts useful features from the picture and saves them to a local database. These features will be compared with the fresh image the user will provide for getting access to the system. An illustration is reported in Figure 3.4.

Verification

The user provides a fresh image of his face to the system to get access. The system extracts the features from the image as done in the previous phase, loads the stored one from the local database, and performs a comparison. The decision is taken by employing such a metric so, if the result is above the threshold the user can get access. Otherwise, the system rejects the request and asks a new fresh image to the user. An illustration is reported in Figure 3.5

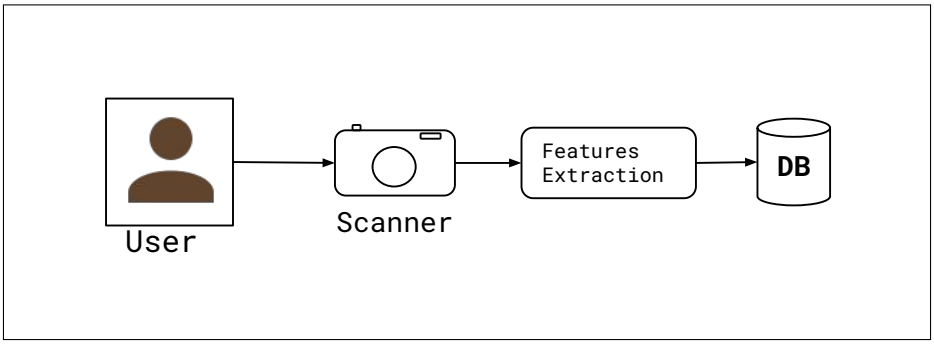


Figure 3.4: Enrollment phase

3.4 Literature review

The study of FR became popular in the early 1990s when an approach based on computing the eigenvectors of the covariance matrix of the set of face images was introduced: Eigenface [27].
The idea was to represent every image in the dataset as a linear combination of the eigenfaces. For performing the recognition, the weights of the test image were computed and compared with the weights of all the images into the dataset. The classification was performed by taking the label of the training image with the least error.

This and other methods developed in those years worked by making assumptions about the distribution of the faces. But these kinds of methods fail to address the uncontrolled facial changes, that cause a deviation from their prior assumptions [28, p. 1].

In the early 2010s methods based on learning local descriptors were introduced. In [29] an unsupervised method is used to learn an encoder, then the PCA (Principal

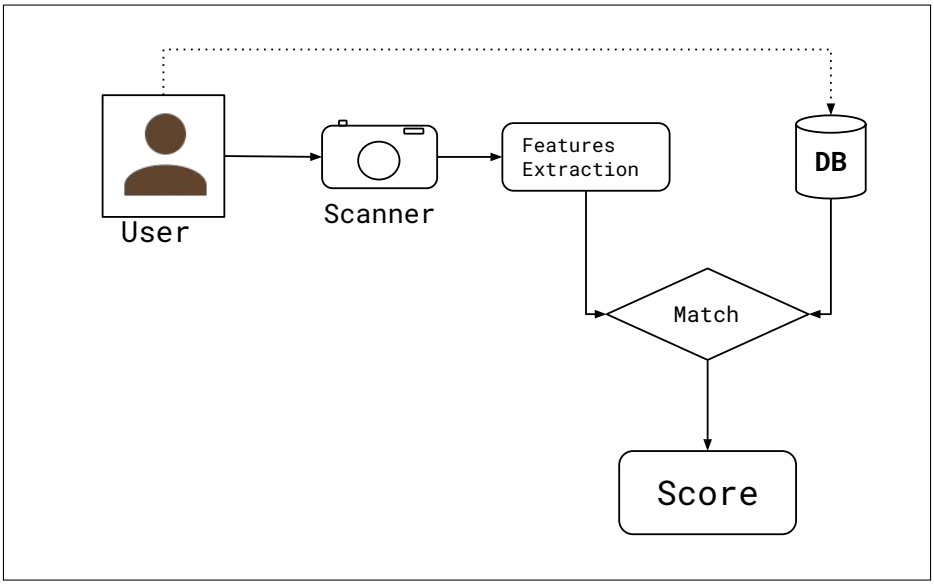


Figure 3.5: Verification phase

Component Analysis, a technique used for reducing the features space and keeping as most information as possible) is applied to reduce the dimensionality of the features and get better compactness.

The performances of such FR systems were unstable because of the difficulties encountered in extracting good features in the pose, light, and age variations [28, p. 1].

In 2012, when AlexNet won the ImageNet competition (a challenge that evaluates algorithms for object detection and image classification at large scale), the situation changed. AlexNet was a CNN designed by Alex Krizhevsky and described in [30]. It achieved a top-5 test error rate of 15.3% in that competition, more than 10% lower than the second-place network (26.2%). From that event, deep learning was employed in almost all researches about FR around the world. It reshaped all the aspects involved in such research areas, such as algorithms, training/test sets, applications, and evaluation protocols [28, p. 2].

As explained in Chapter 2, building an efficient ML model requires large datasets, since the model has to learn as many patterns as possible avoiding to overfit data and decreasing the generalization error. Since there are billions of people on the Earth, and so billions of different faces, FR tasks can be seen as classification problems with billions of classes. However, academic research can not get access to the same, very large, datasets employed by internet companies in building their FR models. For this reason, academic researchers are making an effort in designing loss functions that can overcome the situation of using pretty small datasets [28, p. 5].

In the following, it is described how a classification task is implemented by using a neural network. A traditional NN employed for solving a classification task is composed of the following parts:

- **Input layer**. The ones that take raw data;
- **Hidden layers**. They are involved in extracting features and transforming them;
- **Bottleneck layer**. This layer is the one that outputs the representations of data before feeding them into the classification layer. Such representation is called *pre-logits*;
- **Classification layer**. This layer gets the pre-logits representation and output a probability vector. Each entry of it represents the probability that the sample belongs to the class i_{th} class. Usually, this layer employs a sigmoid or a softmax activation function.

Once the data has been obtained the goodness of the model is measured by means of a loss function. The most used loss function employed in conjunction with softmax is the *cross-entropy loss*. It is defined as

$$loss = -\sum_{c=1}^{M} y_{o,c} log(p_{o,c}) \tag{3.1}$$

where

- M is the number of classes;
- log is the natural logarithm;
- o is an observation, a sample;
- y is a binary indicator. It assumes value 0 or 1 if label c is the correct classification for observation o;
- p is the predicted probability for o belonging to class c.

Cross-entropy loss takes as input a probability value and outputs an error measure. It increases as the predicted probability diverges from the actual label, and it decreases as the predicted probability converges to the actual label (Figure 3.6)

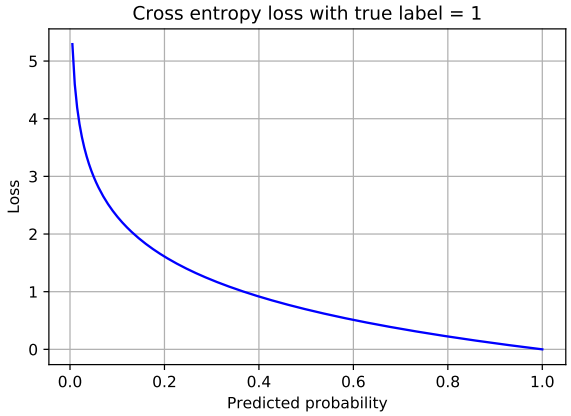


Figure 3.6: Cross entropy function when true label = 1

3.4.1 Related works

This section provides an overview of the most recent and famous works about FR. For each of them, the idea and a brief analysis are comprehended. As will be clear later, such models did not invent new approaches for implementing face recognition tasks. They defined and studied innovative metrics for discriminating pairs of pictures. For generating embedding representations of images, such models use *siamese networks*, i.e. networks that take two samples at the time as input and provide comparable output vectors. Since the network must be regularized for handling two different samples at the time, its weights are shared. Regularizing such networks means to make them able to generate close representations in space when images represent the same person and far representations when images represent different people. The concept of "close" and "far" depends on the distance metric employed, described by the loss function.

DeepFace

DeepFace [31] was one of the first works to use DL for implementing face recognition tasks and one of the first to approach with the results close to the human performance on the LFW dataset [22] (97.35% of DeepFace vs 97.53% of human) [17, p. 1].

Their network is composed of two convolutional layers and a max-pooling layer within them, to extract low-level features. Such representations are supplied to three locally connected layers. Such layers work similarly to convolutional layers, but the weights are unshared, that is, different filters are used for each section of the image. Finally, there are two fully connected layers: the output of the first one is used as the face representation feature vector, and the output of the second one

is provided to a k-class softmax, that performs the classification [31, pp. 3–4].

For implementing a face verification task, the researchers experimented two different approaches.

- Taking the feature vector representation of two faces (provided by the first fully connected layer), they used the weighted-χ^2 similarity [31, p. 5]:

$$\chi^2(f1, f2) = \sum_i \omega_i (f_1[i] - f_2[i])^2 / (f_1[i] + f_2[i]) \tag{3.2}$$

  where f_1 and f_2 are the representations of DeepFace. The weight parameters ω_i are learned using a linear SVM onto $(f_1[i] - f_2[i])^2/(f_1[i] + f_2[i])$.

- After having regularized the face identification network they removed the top layer and doubled the architecture, obtaining a siamese network. For each pair of images, they computed the absolute difference between the feature vectors and regularized a new top fully connected layer with a sigmoid activation function [31, p. 5].

  The distance metric they employed for regularizing the classification space is defined as $d(f_1, f_2) = \sum_i \alpha_i |f_1[i] - f_2[i]|$, where α_i are trainable parameters.

FaceNet

According to Facenet[32] researchers, training a verification model following the traditional approach is inefficient since one has to hope that the bottleneck layer outputs a representation that can well generalize all new faces [32, p. 2].

Their approach consists of directly training and optimizing a network that produces an embedded representation for each face (the representation provided by the bottleneck layer), minimizing the distance between faces of the same person, and maximizing the distance between faces of different people. In order to implement such an approach, they introduced the *triplet-loss* [32, p. 3], defined as

$$\sum_i^N [||f(x_i^a) - f(x_i^p)||_2^2 - ||f(x_i^a) - f(x_i^n)||_2^2 + \alpha]_+ \tag{3.3}$$

where

- x_i^a is the *anchor* image, i.e. a picture of a specific person;
- x_i^p is the *positive* image, another picture of the same person;
- x_i^n is the *negative* image, a picture of another person;
- α is the forced margin between positive and negative pairs.

In order to reach a fast convergence it is crucial to choose the most difficult triplets, i.e. given an anchor x_i^a, they want to select:

- an hard positive sample x_i^p defined as $argmax_{x_i^p}||f(x_i^a) - f(x_i^p)||_2^2$
- an hard negative sample x_i^n defined as $argmin_{x_i^n}||f(x_i^a) - f(x_i^n)||_2^2$

However they argue that selecting hardest negatives could lead a local minima pretty early in the training phase [32, p. 4], so they select such negatives by imposing the condition $||f(x_i^a) - f(x_i^p)||_2^2 < ||f(x_i^a) - f(x_i^n)||_2^2$, calling these samples *semi-hard*. For their experiments, they employed convulutional neural networks using Stochastic Gradient Descent (SGD) [10] with standard backpropagation, and Adagrad [33] as the optimizer.

Once the training is over, the network can be employed for classifying pairs of face pictures. The network gets the pictures as input and, for each of them, returns a numerical representation in such a dimension. Then it is computed the l_2 distance between those representations and set a threshold. If the distance is above the threshold, the system classifies pictures as belonging to the same person, otherwise not.

This approach has improved the embedding representation provided by a deep neural network. Nevertheless, some researches have highlighted that it is pretty difficult to directly optimize a network like this, and some tricks are needed [34].

SphereFace

In [35], the researchers argued that some methods combine softmax loss with other losses like contrastive loss [36], and use triplet-loss [32] for regulating the embedding learning. According to them, such combinations require to carefully design a pair/triplet selection and it is time-consuming and sensible to the performances.

They reasoned about the possibility that the Euclidean space could not always be suitable for learning discriminative facial features. They highlighted that the features learned by softmax loss have inherent angular distribution, concluding that the Euclidean margin-based losses are incompatible with softmax loss.

For binary-class cases the decision boundary of softmax loss is defined as $(\boldsymbol{W}_1 - \boldsymbol{W}_2)x + b_1 - b_2 = 0$, where $\boldsymbol{W}_i$ are weights and b_i is bias in softmax loss. Defining x as a feature vector and constraining $||\boldsymbol{W}_1|| = ||\boldsymbol{W}_2|| = 1$ and $b_1 = b_2 = 0$, the decision boundary becomes $||x||(\cos\theta_1 - \cos\theta_2) = 0$, where θ_i is the angle between $\boldsymbol{W}_i$ and x. This new decision boundary only depends on θ_1 and θ_2, and the modified softmax loss can directly optimize angles.

To control the angular decision boundary, researchers introduced an integer $m(m \geq 1)$, defining a modified version of softmax loss called *A-Softmax*. For a binary-class case the decision boundaries for class 1 and class 2 become

$||x||(\cos m\theta_1 - \cos\theta_2) = 0$ and $||x||(\cos\theta_1 - \cos m\theta_2) = 0$ respectively, while m controls the size of the angular margin. *SphereFace* was the first work to show the effectivness of angular margin in face recognition [35, p. 2].

A-Softmax loss is formulated as

$$L_{A-softmax} = \frac{1}{N}\sum_i -\log(\frac{e^{||\boldsymbol{x}_i||\psi(\theta_{y_i,i})}}{e^{||\boldsymbol{x}_i||\psi(\theta_{y_i,i})} + \sum_{j\neq y_i} e^{||\boldsymbol{x}_i||\cos(\theta_{y_i,i})}}) \tag{3.4}$$

subject to $\psi\theta_{y_i,i} = (-1)^k\cos(m\theta_{y_i,i}) - 2k$, $\theta_{y_i,i} \in [\frac{k\pi}{m}, \frac{(k+1)\pi}{m}]$ and $k \in [0, m-1]$.

SphereFace reached very competitive results, close to the ones achieved by models trained on higher-quality datasets. Moreover, it started the study of the angular losses for face recognition tasks, preparing the way for coming state-of-the-art models.

CosFace

Authors of CosFace [17] pointed out that the standard cross-entropy loss can not be employed to learn separable features if they are not discriminative enough. They also stated that an angular margin would be preferable since there is an intrinsic consistency between softmax and the cosine of the angle.

Following the same idea of SphereFace [35], they proposed a modified version of softmax loss, normalizing both features and weight vectors to remove radial variations [17, p. 2]. Like the previous architecture, they introduced a cosine margin term m for maximizing the decision margin in the angular space. However, they used this term differently and the decision boundary is given by $\cos(\theta_1) - m = \cos(\theta_2)$, where θ_i is the angle between the feature and the weight of the class i.

They explained this reformulation by saying that the decision boundary defined by the A-Softmax has difficulty in optimization because of the non-monotonicity of the cosine function. Moreover, they highlighted that the decision boundary of SphereFace depends on θ, leading to different margins for different classes [17]. Their loss overcomes such drawbacks.

The final formulation of the loss they called *Large Margin Cosine Loss* [17, p. 3] is then

$$L_{lmc} = \frac{1}{N}\sum_i -\log\frac{e^{s(\cos(\theta_{y_i,i})-m)}}{e^{s(\cos(\theta_{y_i,i})-m)} + \sum_{j\neq y_i} e^{s\cos(\theta_{j,i})}} \tag{3.5}$$

subject to

$$W = \frac{W^*}{||W^*||}, \tag{3.6}$$

$$x = \frac{x^*}{||x^*||}, \tag{3.7}$$

$$\cos(\theta_j, i) = W_j^T x_i, \tag{3.8}$$

where:

- N is the number of training samples,
- x_i is the i-th feature vector corresponding to the ground-truth class of y_i,
- Σ_j is the weight vector of the j-th class,
- θ_j is the angle between Σ_j and x_i.

ArcFace

According to ArcFace's [18] researchers, both softmax-loss-based and triplet-loss-based can achieve excellent performances on face recognition, but both of them have some drawbacks:

- Softmax loss:
  - the linear transformation matrix $W \in \mathbb{R}^{d \times n}$ increases in size with the number of identities n;
  - the learned features can well separate identities in a closed-set classification problem, but they are not discriminative enough for the open-set face recognition problem.
- Triplet loss:
  - the number of face triplets increases a lot for large datasets, leading to an increase in the number of iteration steps;
  - training the models with semi-hard samples is a difficult task.

In their work, they analyze the improvements reached by SphereFace in introducing the idea of an angular loss. However, they stated that the A-Softmax loss requires some approximations to be computed, and then the training is unstable. [37, p. 2].

While CosFace directly adds the cosine margin penalty to the target logit, ArcFace performs differently. By using the arc-cosine function, it computes the angle between the feature and the target weight. Then, it adds the angular margin to the target angle and re-compute the cosine of this new representation. Finally, after having re-scaled all logits by a feature norm, this loss behaves as the softmax loss.

To define their loss function, they imposed $b_j = 0$ and transformed the logit as $W_j^T x_i = ||W_j||\ ||x_i||\ cos\theta_j$, where θ_j is the angle between the weight W_j and the feature x_i. Then, they fixed the individual weight $||W_j|| = 1$ by $L2$ normalization and fixed the embedding feature $||x_i||$ by $L2$ normalization, re-scaling it by s [37, p. 2]. Finally, they added the margin term m between x_i and W_{y_i} to shrink the

margin between intra-class samples and enlarge the margin between inter-class samples. The ArcFace loss is defined as

$$L_{arcface} = -\frac{1}{N}\sum_{i}\log\frac{e^{s(\cos(\theta_{y_{i,i}}+m))}}{e^{s(\cos(\theta_{y_{i,i}}+m))}+\sum_{j\neq y_i}e^{s\cos(\theta_{j,i})}} \tag{3.9}$$

Chapter 4

Introduction to BiometricNet

In the previous chapter, we have seen that CNNs have improved the performances of FR tasks since they can learn complex non-linear approximations and extract features without handcrafting them. We have also discussed the choice of the loss function, talking about the various losses introduced during the last years.

All methods discussed above employ a specific and predefined metric for computing the distance between the embedded representations of images. All of them constraint the model to enlarge the distance between pictures of different people and compress the distance between pictures of the same person. This demonstrates that choosing a good metric is crucial for achieving good results and, conversely, a bad metric could lead to bad accuracies and not so relevant performances.

We propose a different approach for training face verification models. Instead of defining a specific metric for evaluating the distances between the input pair, we regularize the network in such a way that it learns the best metric for comparing images. This is achieved by imposing the output of the network to follow a statistical distribution when input images belong to the same person and another statistical distribution when input images belong to different people.

The idea of imposing the outputs to follow the determined distribution was first introduced and experimented by AuthNet [38] and RegNet [39], even though with a different approach: [38] and [39] aim to solve a one-vs-all task, by building networks capable of mapping the biometric features of authorized user onto a distribution far away from the unauthorized users distribution. This means that it is necessary to re-train the network if the authorized user changes.

BiometricNet works differently since it takes as input facial pairs and maps them on the criteria whether they represent the same person or not. Once a model based on BiometricNet has been trained it can work for any person, with no need

to re-train the network.

BiometricNet works by performing the following steps:

- A pair of pictures is provided to BiometricNet: one of them is the *anchor*, the other one could be a positive or a negative sample (positive if it represents the same person of anchor, negative otherwise). As will be discussed in the next chapter, such images get aligned before being fed to the network;
- Such images are processed by BiometricNet, that extracts useful features layer by layer;
- The images are mapped onto a statistical distribution. In an ideal situation, and looking at Figure 4.1, if the pair of images represents the same person, output should be mapped to the blue distribution, while if the images represent different people the output should be mapped to the red distribution;
- Since BiometricNet regularizes the output to follow a well-defined distribution, we can employ a simple linear boundary for classifying the images;
- The loss is computed by comparing predicted and actual labels and the network parameters are updated.

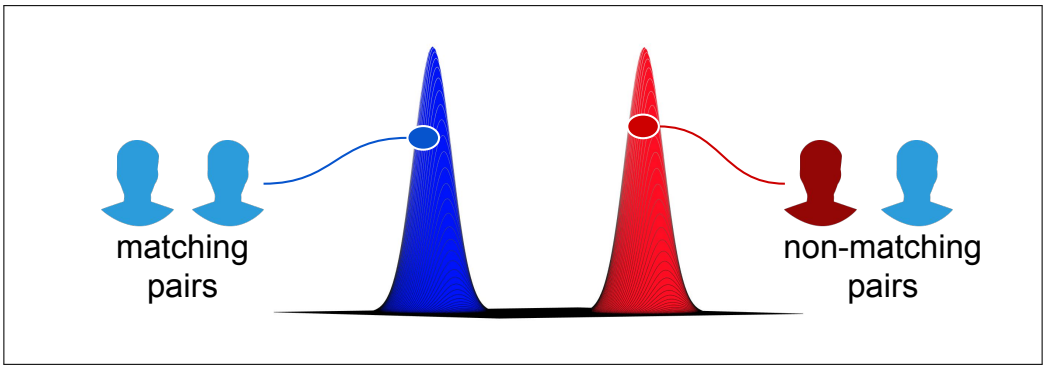


Figure 4.1: Input pair mapping according to label

4.1 Proposed method

The goal of BiometricNet is to learn meaningful features of the faces and, at the same time, a metric for classifying the two faces. In this section, the general architecture of BiometricNet and the loss functions employed in our experiments will be explained.

BiometricNet is constituted of two sub-networks, as depicted in Figure 4.2:

- **FeatureNet**. It is a *siamese* CNN network that takes as input images of facial pairs $x = [x_1, x_2]$ and after having extracted features across the layers, returns an embedding representation of both the images $f = [f_1, f_2]$.
  The size of these representations is set by the user and it is a tuning parameter since increasing the size of this representation means getting more information for each picture, but also much more resources are required for the task. Moreover, using a too large embedding representation could lead the network to overfit the data, causing a drop in performance.

- **MetricNet**. This network maps the embeddings computed by FeatureNet onto a p-dimensional latent space $\mathbf{z}$. The embeddings must be combined before being provided to MetricNet. For instance, they can be concatenated or the difference between them can be computed. The output of this network is used for the final classification.

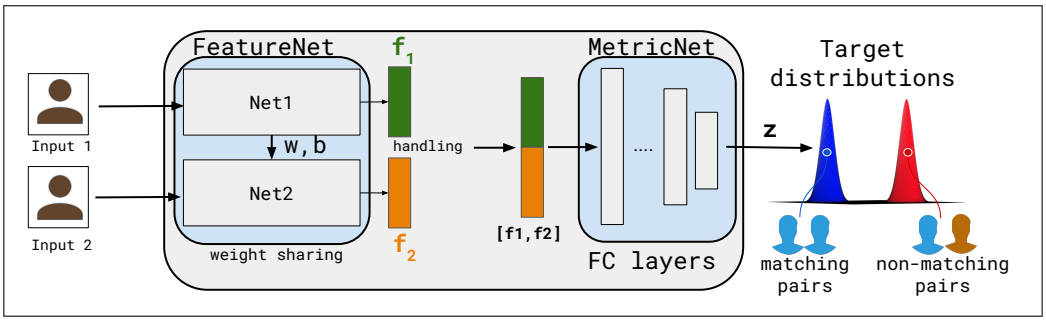


Figure 4.2: BiometricNet architecture. *FeatureNet* takes images of facial pairs as input and provides two embeddings. *MetricNet* takes such embeddings as input and maps them onto the latent space $\mathbf{z}$.

The novelty of this approach is that no predetermined metric is imposed. The metric is directly learned by the network since the decision space is shaped to follow a target distribution through the loss function.
It is worth mentioning that an arbitrary distribution can be employed as a target distribution. However, employing distributions whose masses tend to stay close to a single value might be the best choice, since a linear boundary can be used for classification. From the *central limit theorem*, it is known that the output of fully connected layers tend to follow a Gaussian distribution [40]. Since the second part of BiometricNet is composed of fully connected layers, as will be described later, we chose to employ multivariate Gaussian distributions as target ones.

Let $\mathbb{P}_m$ and $\mathbb{P}_n$ be the desired target multivariate Gaussian distributions.

$$\mathbb{P}_m = \mathcal{N}(\boldsymbol{\mu}_m, \boldsymbol{\Sigma}_m), \mathbb{P}_n = \mathcal{N}(\boldsymbol{\mu}_n, \boldsymbol{\Sigma}_n) \tag{4.1}$$

where

- $\boldsymbol{\Sigma_m} = \sigma_m^2 \mathbb{I}_p$ and $\boldsymbol{\Sigma_n} = \sigma_n^2 \mathbb{I}_p$ are diagonal covariance matrices
- $\boldsymbol{\mu_m} = \mu_m \mathbf{1}_p^T$ and $\boldsymbol{\mu_n} = \mu_n \mathbf{1}_p^T$ are the expected values

It can be noted that using different values for the variance of the two distributions would complicate the choice of the parameters since the optimal variance would be specific to the considered dataset, in order to match its intra-class and inter-class variances. Moreover, using the same variance for both the distributions allows us to use a hyperplane for taking decisions and discriminating the faces. For those reasons, we set $\boldsymbol{\Sigma_m} = \boldsymbol{\Sigma_n}$ for BiometricNet.

We define $\boldsymbol{x}_m$ as a pair of matching faces and $\boldsymbol{x}_n$ as a pair of non-matching faces. Then we define $\boldsymbol{f}_m$ and $\boldsymbol{f}_n$ as the corresponding feature representations outputted by FeatureNet.
MetricNet can be seen as a generic encoding function $\mathcal{H}(\cdot)$ of the input feature pairs $\boldsymbol{z} = \mathcal{H}(\boldsymbol{f})$, where $\boldsymbol{z} \in \mathbb{R}^p$, such that $\boldsymbol{z}_m \sim \mathbb{P}_m$ if $\boldsymbol{f} = \boldsymbol{f}_m$ and $\boldsymbol{z}_n \sim \mathbb{P}_n$ if $\boldsymbol{f} = \boldsymbol{f}_n$.

As explained before we want to regularize the network to constrain the metric to follow the defined target distributions. Since we have chosen the Gaussian as target distribution, we can write the Kullback-Leibler (KL) divergence between samples and targets in a closed-form, as a function of first and second-order statistics.

The Kullback-Leibler divergence was introduced by Kullback and Leibler in [41], and is a measure of how a probability distribution is different from a second distribution.
Starting from the standard mathematical definition of KL divergence, it can be computed its formulation for multivariate Gaussian distributions [42]:

$$\mathcal{L}_m = \frac{1}{2}[\log \frac{\det(\boldsymbol{\Sigma}_m)}{\det(\boldsymbol{\Sigma}_{\mathcal{S}m})} - p + tr(\boldsymbol{\Sigma}_m^{-1}\boldsymbol{\Sigma}_{\mathcal{S}m}) + (\boldsymbol{\mu}_m - \boldsymbol{\mu}_{\mathcal{S}m})^T \boldsymbol{\Sigma}_m^{-1}(\boldsymbol{\mu}_m - \boldsymbol{\mu}_{\mathcal{S}m})] \quad (4.2)$$

where $\mathcal{S}$ refers to the sample statistics. $\mathcal{L}_n$ can be obtained similarly and it is not indicated for brevity.

It is worth noting that, since we only need the first and second-order statistics of $\boldsymbol{z}$, we can capture this information batch-wise.
As will be explained in detail in Section 4.3, during the training the network gets $b/2$ difficult matching face pairs and $b/2$ difficult non-matching face pairs extracted from the training dataset, being b the batch size.

Letting $\mathbf{X} \in \mathbb{R}^{b \times r}$ with r the size of a face pair, the network outputs a collection of latent space points $\mathbf{Z} \in \mathbb{R}^{b \times p}$. We then have to compute the first and second order statistics of the encoded representations $\mathbf{Z}_m$, related to the matching input faces $(\boldsymbol{\mu}_{S_m}, \boldsymbol{\Sigma}_{S_m})$, and $\mathbf{Z}_n$, related to the non-matching input faces $(\boldsymbol{\mu}_{S_n}, \boldsymbol{\Sigma}_{S_n})$.

Let us denote as $\mathbf{\Sigma}_{S_m}^{(ii)}$ the i-th diagonal entry of the sample covariance matrix $\mathbf{Z}_m$. The assumption made about the diagonal covariance allows us to simplify 4.2 as

$$\mathcal{L}_m = \frac{1}{2}[\log \frac{\sigma_m^{2p}}{\prod_i \mathbf{\Sigma}_{S_m}^{(ii)}} - p + \frac{\sum_i \mathbf{\Sigma}_{S_m}^{(ii)}}{\sigma_m^2} + \frac{\|\boldsymbol{\mu}_m - \boldsymbol{\mu}_{S_m}\|_2}{\sigma_m^2}] \tag{4.3}$$

$\mathcal{L}_n$ can be obtained similarly and it is not indicated for brevity. This loss captures the statistics of the matching and non-matching pairs, enforcing the distribution $\mathbb{P}_m$ and $\mathbb{P}_n$ respectively.

The overall loss function we have to minimize, and that covers the entire network (FeatureNet and MetricNet), is then given by $\mathcal{L} = \mathcal{L}_m + \mathcal{L}_n$

4.2 Architectural details

- **FeatureNet**. The goal of FeatureNet is to provide a good representation for each pair of images. This is the most important part of BiometricNet since having bad representations could lead to poor performances at the end of the training process. We employ convolutional neural networks for this task because of their ability to extract features from images. In particular, we employ residual networks because of their fast convergence.

- **MetricNet**. The goal of MetricNet is to map representations provided by FeatureNet onto the latent space, before the classification step. For this task, we employ six fully connected layers.

  We set the output size of the first FC layer equals to the size of the combination of the representations provided by FeatureNet. For instance, if FeatureNet maps images to $p = 512$ and we concatenate those representations, the first fully connected layer of MetricNet will have 1024 neurons. At the output of each layer, the *leaky ReLU* activation function is employed, while the last layer linearly maps the latent space onto target distributions. In some experiments, we regularize these layers using weight decay and dropout regularizations.

4.3 Pairs selection

Choosing meaningful face pairs is crucial for achieving the goal. For the training phase, BiometricNet selects the most difficult matching and non-matching face pairs, i.e. the ones that are far from their target distributions and very close to the decision threshold. The distance between a face pair and its target distribution

is computed during the selection phase pass. Each pair is given as input to BiometricNet and it outputs the corresponding value of $\boldsymbol{z}$.

At the end of the selection phase, for each mini-batch, if the subset of matching face pairs whose output $\boldsymbol{z}_m$ is enough distant from the mass center of $\mathbb{P}_m$, i.e. $\|\boldsymbol{z}_m - \boldsymbol{\mu}_m\|_\infty \geq 2\sigma_m$, is selected. Similarly, the selected non-matching face pairs are the ones whose output $\boldsymbol{z}_n$ is enough distant from the mass center of $\mathbb{P}_n$, i.e. $\|\boldsymbol{z}_n - \boldsymbol{\mu}_n\|_\infty \leq 2\sigma_n$. Being b the batch size, we want to select $b/2$ difficult matching face pairs and $b/2$ difficult non-matching face pairs. If during the selection pass this condition is not met, the batch is discarded and the procedure repeated.
Once the condition is met the forward pass starts and the loss is computed. Then, the backward pass is performed to reduce the loss.

The rationale of this choice is that face pairs falling close to the threshold are the ones with high uncertainly, and so the ones that can improve the regularization of the network. It is worth noting that, as the training proceeds and the loss decreases, difficult face pairs of such an epoch get correctly classified and they will not be selected anymore during the next epoch.

4.4 Verification

Once the network has been trained, we can provide it pairs of face pictures for verifying and authenticating users. This phase does not update layers weights anymore and the input pictures are processed by the entire network. As explained in the previous chapter, BiometricNet computes and outputs the metric value $\boldsymbol{z}$ related to inputs. The classification is then performed by employing a hyperplane that separates target distributions, by means of the test:

$$(\mu_m - \mu_n)^T \boldsymbol{z} \lessgtr (\mu_m - \mu_n)^T (\mu_m + \mu_n)/2 \tag{4.4}$$

If we use a latent space of size $p = 1$ then the above test changes, and we have to compare the scalar $\mathbf{z}$ with a threshold

$$\tau = (\mu_m + \mu_n)/2 \tag{4.5}$$

We take into consideration a different approach for looking at better performances. Following other works, such as [17], we compute metric value $\mathbf{z}$ for all combinations of original and horizontally flipped images, obtaining four different metrics. Then we compute the final value $\overline{\mathbf{z}}$ as the mean value of those metrics (Table 4.1).

Image 1	**Image 2**	**Metric value**	**Final metric value**
Original	Original	$\mathbf{z}_1$	$\overline{\mathbf{z}} = \frac{1}{4}\sum_{i=1}^{4}\mathbf{z}_i$
Original	Flipped	$\mathbf{z}_2$	
Flipped	Original	$\mathbf{z}_3$	
Flipped	Flipped	$\mathbf{z}_4$	

Table 4.1: Summary of all possible images combinations with related metric value $\boldsymbol{z}$

Chapter 5

Experiments

This chapter reports the experiments conducted. Since directly employing the KL loss leads to a slow convergence, we decided to split the regularization into the following steps:

- The first step consists of regularizing FeatureNet, the part of BiometricNet that has to output an embedding representation for each image provided as input. Even though we argued that using a softmax classifier could not guarantee high performances, we implemented a softmax classifier based on cross-entropy loss for regularizing FeatureNet because of good generalization and fast convergence properties.

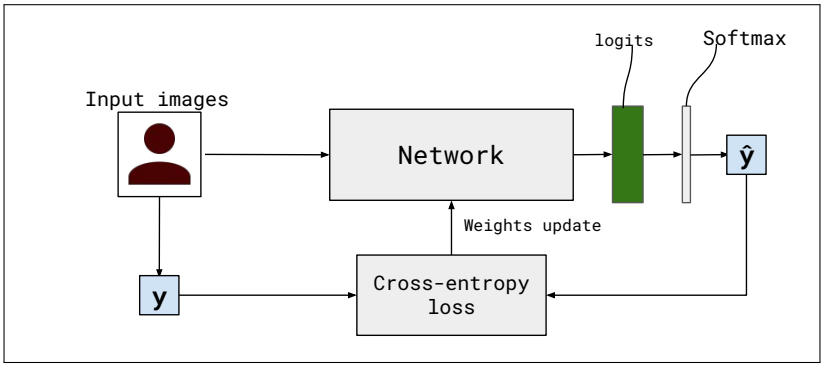


Figure 5.1: FeatureNet regularization

- The second step consists of regularizing MetricNet layers to obtain good representations of images in the latent space. In this phase we only updated weights of MetricNet, not updating the FeatureNet and, of course, loading the weights obtained in the first step. This part ends when the loss on the benchmark datasets stops decreasing. In this step we use the loss described in Section 4.1.

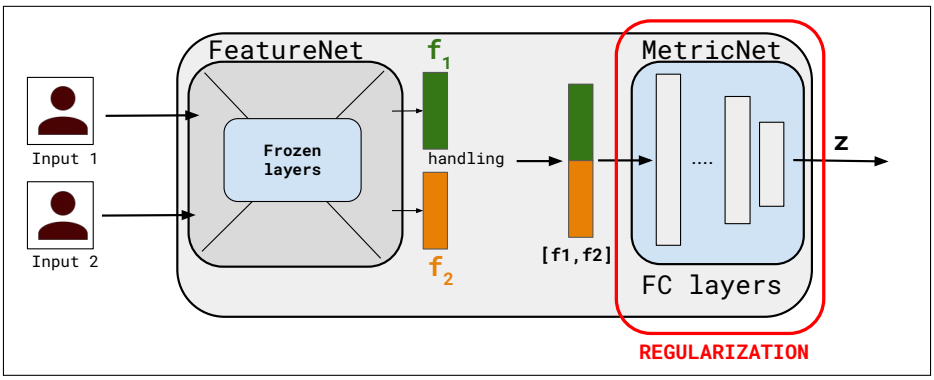


Figure 5.2: MetricNet regularization

- In the third step the entire network is fine-tuned to achieve results comparable to state-of-the-art. It is characterized by a very small learning rate since we do not want to change too the network's weights. This step ends when the measured loss stops to decrease and the network becomes stable.

It is worth saying that the steps described above represent a generic pipeline. More regularization steps can be required for each phase to reach an acceptable condition of convergence and good performances.

5.1 Preliminary results

Before choosing the best model we conducted a lot of experiments to find the configuration that guarantees best performances. For all the experiments we set the following parameters for the target distributions:

- Mean of matching pairs target distribution $\mu_m = 0$
- Mean of non-matching pairs target distribution $\mu_n = 40$
- Same variance for both the distributions $\sigma_m = \sigma_n = 1$

We used *Python 3.6.7* for writing the code, *Tensorflow* [1] as DL framework and a single *NVIDIA GeForce Titan X GPU (12GB)* as hardware support, to improve the performances and reducing the training time.
We have tracked the loss and accuracy on the LFW dataset every 500 or 1000 steps through *Tensorboard*. Then we performed the testing phase on all the benchmark datasets employing the trained network.

Table 5.1 reports the results achieved by the trained models when $\boldsymbol{z}_1$ (original images only) is employed as metric value. Table 5.2 reports the results achieved when $\overline{\boldsymbol{z}}$ (all possible combinations of original and horizontally flipped images) is employed as metric value.

Architecture	A	B	C	D	E	F
Dataset	**p = 1**	**p = 1**	**p = 1**	**p = 3**	**p = 16**	**p = 64**
LFW	**99.42**	99.15	98.53	98.78	98.80	98.87
YTF	**97.55**	97.00	96.90	97.07	97.25	97.32
CFP-FP	**98.43**	97.98	96.58	96.45	97.28	96.78
CALFW	**96.25**	95.30	95.55	95.47	95.62	95.73
CPLFW	**92.47**	91.67	88.35	88.60	89.13	89.00

Table 5.1: Results achieved by using $\boldsymbol{z}_1$ (original images) as metric value

Architecture	A	B	C	D	E	F
Dataset	**p = 1**	**p = 1**	**p = 1**	**p = 3**	**p = 16**	**p = 64**
LFW	**99.47**	99.23	98.88	98.97	99.03	99.17
YTF	97.48	96.92	97.32	97.45	97.60	**97.67**
CFP-FP	**98.47**	98.17	96.92	96.67	97.40	97.02
CALFW	96.15	95.47	95.98	95.98	96.00	**96.23**
CPLFW	**92.77**	92.30	89.28	89.20	89.90	89.83

Table 5.2: Results achieved by using $\overline{\boldsymbol{z}}$ as metric value

5.2 Final model and comparisons

This section analyzes and discusses the model that achieves the best results in general.

According to Table 5.1 and Table 5.2, Model A is the one that achieved better results. It got higher results on all the benchmark dataset when z_1 is used as a metric, and higher results on three datasets out of five when $\overline{z}$ is employed as a metric. For this reason, we report here a comparison between Model A and state-of-the-art results. Nevertheless, you can look at the appendix A for a more detailed description of other experiments, to reproduce and, maybe, outperform our results.

A description of the architecture of Model A is reported below.

- **FeatureNet**
  - Inception ResNet V1 [43]
  - Embedding size: 512
- **MetricNet**.
  - 6 fully connected layers of sizes {1024, 512, 256, 128, 64, 1}
  - Feature vector: concatenation
  - Latent space dimensionality(p) = 1
  - Target distributions:
    - Same user: mean 0, variance 1
    - Different users: mean 40, variance 1
- Batch size (for each phase): 90/180/120
- This architecture consists of **25,244,369** training parameters.

In the first part of this experiment, we regularized FeatureNet to obtain a good network for extracting features. We employed CASIA WebFace [20] as the training set and ADAM as the optimizer. The starting learning rate was set to 0.05, decreased to 0.005 at step 30000, to 0.001 at step 50000 and 0.0005 at step 80000 until step 100000, when the run was stopped. Moreover, we set the batch size to 90, a dropout rate of 0.2, and a weight decay of $2e-4$. The training lasted approximately 20 hours.

The second part consisted of regularizing MetricNet by keeping fixed the weights of FeatureNet. We employed again CASIA WebFace [20] as the training set and ADAM as the optimizer. This time we increased the batch size to 180. We set the

starting learning rate to $5e-6$, decreasing it by a factor 0.98 every 3000 steps. The training was stopped at step 301,040, with a training time of approximately 1.5 days.

Finally, we load all weights got by the first two phases to regularize the entire network by employing the covariance loss described in Section 4.1. Here we employed MS1M-DeepGlint [21] as the training set and Momentum as the optimizer. We set the batch size to 120 and a starting learning rate of $5e-6$, decreased by a factor 0.98 every 3000 steps. The training was stopped after 646,799 steps, with a training time of approximately 5 days.

5.2.1 Performances comparison

Method	LFW	YTF	CFP-FP	CALFW	CPLFW
SphereFace [35]	99.42	95.00	94.38	90.30	81.40
SphereFace+ [44]	99.47	-	-	-	-
FaceNet [32]	99.63	95.10	-	-	-
VGGFace [45]	98.95	97.30	-	90.57	84.00
DeepID [46]	99.47	93.20	-	-	-
ArcFace [18]	**99.82**	**98.02**	98.37	95.45	92.08
CenterLoss [47]	99.28	94.90	-	85.48	77.48
DeepFace [31]	97.35	91.40	-	-	-
Baidu [48]	99.13	-	-	-	-
RangeLoss [49]	99.52	93.70	-	-	-
MarginalLoss [37]	99.48	95.98	-	-	
CosFace [17]	99.73	97.60	95.44	-	-
BiometricNet	**99.47**	**97.48**	**98.47**	**96.15**	**92.77**

Table 5.3: Verification accuracy of different methods on benchmark datasets LFW, YTF, CFP-FP, CALFW, CPLFW. These results refer to $\overline{z}$ metric.

Dataset	TP	FP	TN	FN	Total
LFW	2984	16	2984	16	
YTF	2923	74	2926	77	
CFP-FP	2947	39	2961	53	6000
CALFW	2873	104	2896	127	
CPLFW	2722	156	2844	278	

Table 5.4: Confusion Matrix of BiometricNet on LFW, YTF, CFP-FP, CALFW and CPLFW. These results refer to $\overline{z}$ metric.

In Table 5.3 are reported the verification accuracies obtained by different methods tested on benchmark datasets we described in Section 3.2.2. Table 5.4 shows the confusion matrix of BiometricNet.

The results on LFW and YTF are comparable to state-of-the-art results. On the other hand, we outperformed state-of-the-art results on more challenging datasets like CFP-FP, CALFW, CPLFW. We decreased the error rate by 0.1%, 0.7%, and 0.69% on CFP-FP, CALFW, and CPLFW datasets respectively.

5.2.2 Analysis of BiometricNet: ROC curves

ROC curves allow us to visualize the Genuine Acceptance Rate (GAR), i.e. the relative number of correctly accepted matching pairs as a function of the False Acceptance Rate (FAR), i.e. the relative number of incorrectly accepted non-matching pairs.

Employing the ROC curves we can understand how the model is capable of generalizing across different datasets. While the results showed in Table 5.3 refers to a binary classification using a specified threshold, as described in Section 4.4, here we can see the behavior of our models if we impose a value for FAR.

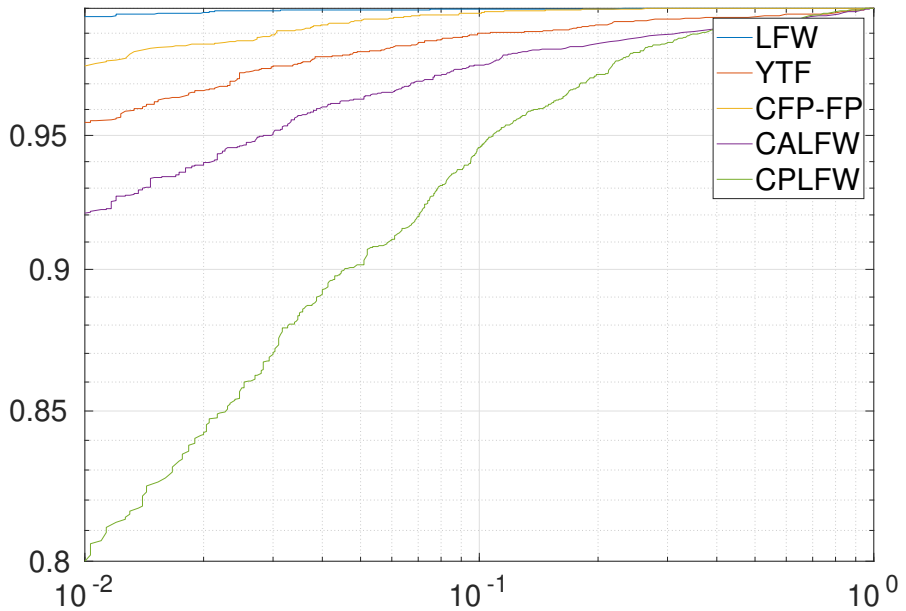


Figure 5.3: ROC curves of BiometricNet on LFW, YTF, CFP-FP, CALFW, CPLFW

Dataset	GAR@$\mathbf{10^{-1}}$FAR%	GAR@$\mathbf{10^{-2}}$FAR%
LFW	99.97	99.67
YTF	99.00	95.50
CFP-FP	99.80	97.70
CALFW	97.73	92.07
CPLFW	94.53	79.40

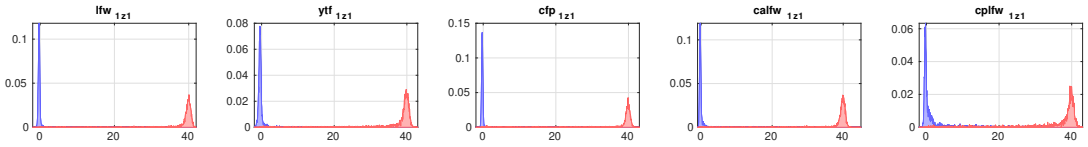
Table 5.5: Genuine Acceptance Rate (GAR) obtained by BiometricNet for LWF, YTF, CFP-FP, CALFW and CPLFW if False Acceptance Rate (FAR) is set to 10^{-2} and 10^{-1}

5.2.3 Discussion about metrics distribution

BiometricNet maps matching and non-matching pairs onto well behaved Gaussian distributions. Figure 5.4 and Figure 5.5 show how BiometricNet maps the benchmark datasets onto those distributions, for z_1 and $\overline{z}$ metrics respectively.

At first sight, it is noticeable that both blue distributions and red distributions have a Gaussian shape, meaning that the approach of BiometricNet works and it can efficiently separate pairs according to their category. More precisely, if we look at non-matching distributions we can observe that they have both mean and variance that follow the ones we imposed. On the other hand, matching distributions do have the right mean we impose, but with a slightly lower variance. This could be possible since the number of non-matching pairs is higher than the number of matching pairs in such a dataset, and so matching and non-matching pairs exhibit different variability, leading to a difficulty in imposing the same variance in the latent space.

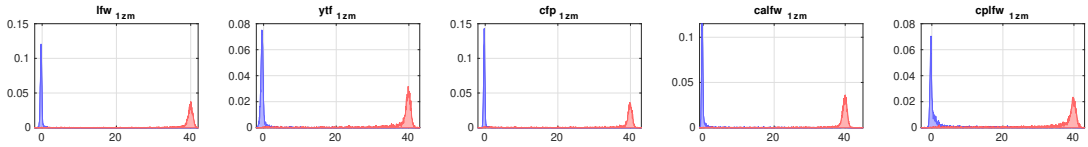
Another thing we can notice is that, for more challenging datasets like CALFW or CPLFW, the matching distributions have heavier tails.



(a) LFW (b) YTF (c) CFP-FP (d) CALFW (e) CPLFW

Figure 5.4: Distribution of z_1 metric for each dataset. The blue area refers to matching pairs, the red area refers to non-matching pairs.

Figure 5.6 and Figure 5.7 show the pairs of LFW misclassified by BiometricNet. It is worth noting that some of the false accepted pictures could be misclassified by



(a) LFW (b) YTF (c) CFP-FP (d) CALFW (e) CPLFW

Figure 5.5: Distribution of $\overline{z}$ metric for each dataset. The blue area refers to matching pairs, the red area refers to non-matching pairs.

Figure 5.6: LFW: False Accept

a human since the people involved look alike. Looking at the false rejected images, we can notice that most of them contain such occlusion or distortion elements, such as sunglasses, hats, profile pictures, and people with closed eyes.

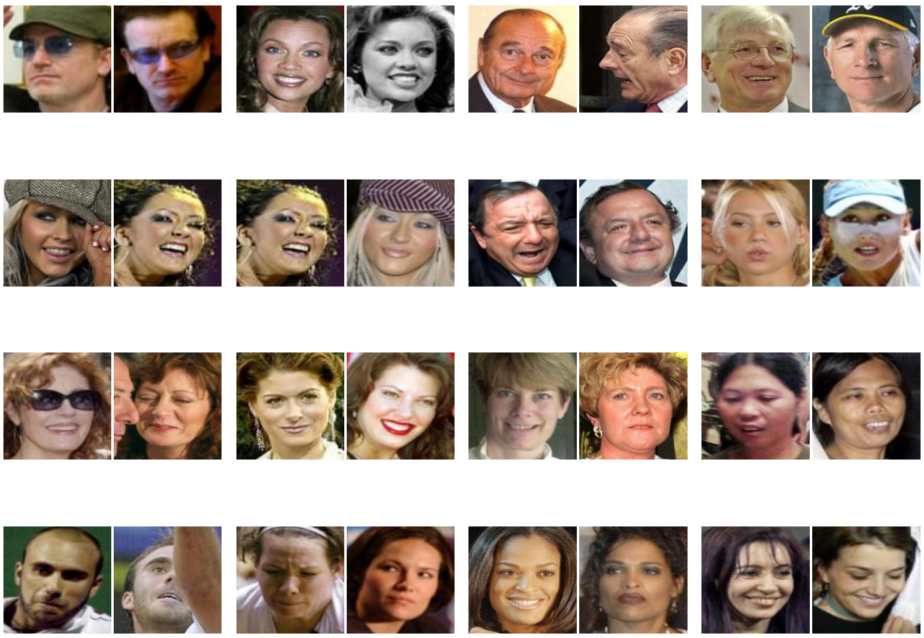


Figure 5.7: LFW: False Reject

Chapter 6

Conclusions

In this thesis, we proposed a novel approach to address the face verification problem with neural networks. Instead of learning a complex classification boundary, our method aims at mapping input images onto well-behaved statistical distributions, allowing us to perform a threshold-based classification. We demonstrated that our approach works by partially outperforming the state-of-the-art results on face benchmarks and achieving high GAR values.

Future work will focus on improving the performances of the model, both by trying different settings for the architecture and different distributions for the latent space. A further improvement for BiometricNet will be its usage in 3D face verification and authentication based on different biological features, such as iris, retina, and fingerprint.

Appendix A

Settings and results of other models

In this section we report settings and results of models we did not take into account in the previous chapter. The reason is that other researchers could be interested in repeating experiments and looking for other configurations that can improve performances. For each model the following information is reported:

- Architecture of FeatureNet;
- Architecture of MetricNet;
- Experimental setting for each training phase.

A.1 Model B

- **FeatureNet**.
  - Inception ResNet V2 [43]
  - Embedding size: 512
- **MetricNet**.
  - 6 fully connected layers of sizes {1024, 512, 256, 128, 64, 1}
  - Feature vector: concatenation
  - Latent space dimensionality(p) = 1
  - Target distributions:
    - Same user: mean 0, variance 1

    - ∗ Different users: mean 40, variance 1

- Batch size (for each phase): 90/180/120
- This architecture consists of **56,871,649** training parameters.

We first regularized FeatureNet for 90,000 steps (approximately 4 days) on CASIA WebFace [20]. We employed an ADAM optimizer and a starting learning rate of 0.05, decreased to 0.005 at step 60,000, to 0.0005 at step 80,000. The batch size was set to 90. We set a weight decay of $5e-4$ and a dropout rate of 0.2.

For regularizing MetricNet we used again CASIA WebFace as the training set and ADAM as the optimizer. We increased the batch size to 180. The training lasted 130,903 steps (approximately 23 hours). We set the initial learning rate to 0.001, decreasing it by a factor 0.98 every 2000 steps.

Finally, we performed a regularization by updating the entire network. We employed the dataset obtained by mixing MS1M-DeepGlint [21] and Asian-DeepGlint [21] sets and ADAGRAD optimizer. We set the batch size to 120 and a dropout rate onto FeatureNet of 0.2. This phase lasted 90,291 steps (approximately 1 day). The starting learning rate was set to $5e-5$ and decreased by a factor of 0.98 every 2000 steps.

Below are reported the results achieved by employing this architecture.

Dataset	**State-of-the-art**	**BiometricNet (Model B)**	**Difference**
LFW	99.83	99.15	-0.68
YTF	98.02	97.00	-1.02
CFP-FP	98.37	97.98	-0.39
CALFW	95.45	95.30	-0.15
CPLFW	92.08	91.67	-0.41

Table A.1: Comparison between state-of-the-art and BiometricNet (Model B) performances when z_1 metric value is used

Dataset	State-of-the-art	BiometricNet (Model B)	Difference
LFW	99.83	99.23	-0.60
YTF	98.02	96.92	-1.10
CFP-FP	98.37	98.17	-0.20
CALFW	95.45	**95.47**	**+0.02**
CPLFW	92.08	**92.30**	**+0.22**

Table A.2: Comparison between state-of-the-art and BiometricNet (Model B) performances when $\overline{z}$ metric value is used

Dataset	TP	FP	TN	FN	Total
LFW	2970	16	2984	30	
YTF	2875	60	2940	125	
CFP-FP	2950	60	2940	50	6000
CALFW	2832	104	2896	168	
CPLFW	2764	226	2774	236	

Table A.3: Confusion Matrix of BiometricNet (Model B) on LFW, YTF, CFP-FP, CALFW and CPLFW. These results refer to $\overline{z}$ metric.

A.2 Model C

- **FeatureNet**.
  - ResNet-101 [14]
  - Embedding size: 512
- **MetricNet**.
  - 6 fully connected layers of sizes {1024, 512, 256, 128, 64, 1}
  - Feature vector: concatenation
  - Latent space dimensionality(p) = 1
  - Target distributions:
    - Same user: mean 0, variance 1
    - Different users: mean 40, variance 1
- Batch size (for each phase): 90/90/180
- This architecture consists of **45,243,521** training parameters.

In the first part of the training we only regularized FeatureNet for 210,000 steps (approximately 4 days) on MS1M-DeepGlint [21]. We employed ADAM optimizer and a starting learning rate of 0.05, decreased to 0.01 at step 50,000, to 0.005 at step 100,000, 0.001 at step 150,000, 0.0005 at step 198,000. The batch size was set to 90.

To improve the ability to generate features, we run a finetuning phase on FeatureNet, by using triplet loss on a dataset obtained by mixing MS1M-DeepGlint [21] and Asian-DeepGlint [21] for 146,212 steps (approximately 1 day) and using ADAM optimizer. The starting learning rate was set to $2e-4$, decreased by 98% every 4000 steps. The batch size was set to 90.

At this point we run three finetuning stages onto the entire network (FeatureNet + MetricNet) by employing loss 4.3, ADAM optimizer, batch size 180, randomly cropped and randomly flipped input images.
For the first stage, we set the learning rate to 0.001, decreased by 98% every 4000 steps. The training took 61,127 steps (approximately 12 hours).
For the second stage, we set the learning rate to 0.00005, decreased by 98% every 4000 steps. The training took 112,545 steps (approximately 2 days).
For the third stage, we set the learning rate to $2.84e-5$, decreased by 98% every 4000 steps. We set a dropout rate of 0.2 between each fully connected layer of MetricNet. The training took 292,417 steps (approximately 5 days).

A comparison between BiometricNet (Model C) and state-of-the-art is reported below.

Dataset	**State-of-the-art**	**BiometricNet (Model C)**	**Difference**
LFW	99.83	98.53	-1.30
YTF	98.02	96.90	-1.12
CFP-FP	98.37	96.58	-1.79
CALFW	95.45	**95.55**	**+0.10**
CPLFW	92.08	88.35	-3.73

Table A.4: Comparison between state-of-the-art and BiometricNet (Model C) performances when z_1 metric value is used

Dataset	State-of-the-art	BiometricNet (Model C)	Difference
LFW	99.83	98.88	-0.95
YTF	98.02	97.32	-0.70
CFP-FP	98.37	96.92	-1.45
CALFW	95.45	**95.98**	**+0.53**
CPLFW	92.08	89.28	-2.80

Table A.5: Comparison between state-of-the-art and BiometricNet (Model C) performances when $\overline{z}$ metric value is used

Dataset	TP	FP	TN	FN	Total
LFW	2972	39	2961	28	
YTF	2911	72	2928	89	
CFP-FP	2936	121	2879	64	6000
CALFW	2883	124	2876	117	
CPLFW	2614	257	2743	386	

Table A.6: Confusion Matrix of BiometricNet (Model C) on LFW, YTF, CFP-FP, CALFW and CPLFW. These results refer to $\overline{z}$ metric.

A.3 Model D

- **FeatureNet**
  - ResNet-152 [14]
  - Embedding size: 512
- **MetricNet**.
  - 6 fully connected layers of sizes {512, 256, 128, 64, 32, 3}
  - Feature vector: difference
  - Latent space dimensionality(p) = 3
  - Target distributions:
    - Same user: mean 0, variance 1
    - Different users: mean 40, variance 1
- Batch size (for each phase): 90/90/90
- This architecture consists of **59,554,499** training parameters.

In the first phase, only FeatureNet is regularized by building a softmax classifier on MS1M-DeepGlint dataset [21] and by using cross-entropy loss. We set ADAM optimizer, weight decay to $5e-4$, and run for 173,000 steps (approximately 4 days). Starting learning rate was set to 0.05, decreased to 0.01 at step 50,000, to 0.005 at step 150,000. The batch size was set to 90 and input images were provided randomly flipped.

In the second phase, we only regularized MetricNet on the MS1M-DeepGlint dataset [21], employing ADAM optimizer, batch size 90 and loss 4.2. Starting learning was set to $5e-3$, decreased by 90% every 4000 steps. This phase took 158,935 steps (approximately 20 hours).

As the last finetuning phase, we employed MOMENTUM (SGD + momentum regularization) optimizer, updating the entire network. The starting learning rate was set to $5e-6$ and decreased every 2000 steps by 98%. The batch size was set to 90.

The results are reported below.

Dataset	**State-of-the-art**	**BiometricNet (Model D)**	**Difference**
LFW	99.83	98.78	-1.05
YTF	98.02	97.07	-0.95
CFP-FP	98.37	96.45	-1.92
CALFW	95.45	**95.47**	**+0.02**
CPLFW	92.08	88.6	-3.48

Table A.7: Comparison between state-of-the-art and BiometricNet (Model D) performances when z_1 metric value is used

Dataset	**State-of-the-art**	**BiometricNet (Model D)**	**Difference**
LFW	99.83	98.97	-0.86
YTF	98.02	97.45	-0.57
CFP-FP	98.37	96.67	-1.70
CALFW	95.45	**95.98**	**+0.53**
CPLFW	92.08	89.20	-2.88

Table A.8: Comparison between state-of-the-art and BiometricNet (Model D) performances when $\overline{z}$ metric value is used

Dataset	TP	FP	TN	FN	Total
LFW	2978	40	2960	22	
YTF	2894	47	2953	106	
CFP-FP	2908	108	2892	92	6000
CALFW	2853	94	2906	147	
CPLFW	2655	303	2697	345	

Table A.9: Confusion Matrix of BiometricNet (Model D) on LFW, YTF, CFP-FP, CALFW and CPLFW. These results refer to $\overline{z}$ metric.

A.4 Model E

- **FeatureNet**
  - ResNet-152 [14]
  - Embedding size: 512
- **MetricNet**.
  - 6 fully connected layers of sizes {512, 256, 128, 64, 32, 16}
  - Feature vector: difference
  - Latent space dimensionality(p) = 16
  - Target distributions:
    - Same user: mean 0, variance 1
    - Different users: mean 40, variance 1
- Batch size (for each phase): 90/90/90
- This architecture consists of **59,554,928** training parameters.

FeatureNet was regularized as done for Model D, so we do not report here the settings. The regularization of MetricNet is pretty equal to the second phase of Model D. For the third phase (regularization of the entire network by employing loss 4.2) we used the same settings of Model D, but with a different learning rate schedule. In fact, the starting learning rate was set to $1e-6$, decreased by 98% every 5000 steps. This phase took 881,266 steps (approximately 11 days).

The results are reported below.

Dataset	State-of-the-art	BiometricNet (Model E)	Difference
LFW	99.83	98.80	-1.03
YTF	98.02	97.25	-0.77
CFP-FP	98.37	97.28	-1.09
CALFW	95.45	**95.62**	**+0.17**
CPLFW	92.08	89.13	-2.95

Table A.10: Comparison between state-of-the-art and BiometricNet (Model E) performances when z_1 metric value is used

Dataset	State-of-the-art	BiometricNet (Model E)	Difference
LFW	99.83	99.03	-0.80
YTF	98.02	97.60	-0.42
CFP-FP	98.37	97.40	-0.97
CALFW	95.45	**96.00**	**+0.55**
CPLFW	92.08	89.90	-2.18

Table A.11: Comparison between state-of-the-art and BiometricNet (Model E) performances when $\overline{z}$ metric value is used

A.5 Model F

- **FeatureNet**
  - ResNet-152 [14]
  - Embedding size: 512
- **MetricNet**.
  - 6 fully connected layers of sizes {512, 384, 256, 192, 128, 64}
  - Feature vector: difference
  - Latent space dimensionality(p) = 64
  - Target distributions:
    - Same user: mean 0, variance 1
    - Different users: mean 40, variance 1
- Batch size (for each phase): 90/90/90
- This architecture consists of **59,757,696** training parameters.

Dataset	TP	FP	TN	FN	Total
LFW	2967	25	2975	33	
YTF	2927	71	2929	73	
CFP-FP	2908	64	2936	92	6000
CALFW	2815	55	2945	185	
CPLFW	2783	389	2611	217	

Table A.12: Confusion Matrix of BiometricNet (Model E) on LFW, YTF, CFP-FP, CALFW and CPLFW. These results refer to $\overline{z}$ metric.

FeatureNet was regularized following the very same steps performed for other models, so we skip to talk about it. For MetricNet, we used six fully connected layers with different sizes for previous models, increasing the number of parameters. The starting learning rate was set at $1e-7$, decreased by 98% every 7500 steps. This model was regularized for 800,719 steps (approximately 11 days).

The results are reported below.

Dataset	State-of-the-art	BiometricNet (Model F)	Difference
LFW	99.83	98.87	-0.96
YTF	98.02	97.32	-0.70
CFP-FP	98.37	96.78	-1.59
CALFW	95.45	**95.73**	**+0.28**
CPLFW	92.08	89.00	-3.08

Table A.13: Comparison between state-of-the-art and BiometricNet (Model F) performances when z_1 metric value is used

Dataset	State-of-the-art	BiometricNet (Model F)	Difference
LFW	99.83	99.17	-0.66
YTF	98.02	97.67	-0.35
CFP-FP	98.37	97.02	-1.35
CALFW	95.45	**96.23**	**+0.78**
CPLFW	92.08	89.83	-2.25

Table A.14: Comparison between state-of-the-art and BiometricNet (Model F) performances when $\overline{z}$ metric value is used

Dataset	TP	FP	TN	FN	Total
LFW	2977	27	2973	23	
YTF	2902	42	2958	98	
CFP-FP	2894	73	2927	106	6000
CALFW	2840	66	2934	160	
CPLFW	2732	342	2658	268	

Table A.15: Confusion Matrix of BiometricNet (Model F) on LFW, YTF, CFP-FP, CALFW and CPLFW. These results refer to $\overline{z}$ metric.

Bibliography

[1] Martın Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: https://www.tensorflow.org/ (cit. on pp. 3, 37).

[2] Adam Paszke et al. «PyTorch: An Imperative Style, High-Performance Deep Learning Library». In: *Advances in Neural Information Processing Systems 32*. Ed. by H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett. Curran Associates, Inc., 2019, pp. 8024–8035. URL: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf (cit. on p. 3).

[3] François Chollet et al. *Keras*. https://keras.io. 2015 (cit. on p. 3).

[4] https://github.com/topics/machine-learning, updated at 04/21/20 (cit. on p. 3).

[5] https://github.com/topics/deep-learning, updated at 04/21/20 (cit. on p. 3).

[6] Shane Legg and Marcus Hutter. *A Formal Measure of Machine Intelligence*. 2006. arXiv: cs/0605024 [cs.AI] (cit. on p. 3).

[7] Kevin P. Murphy. *Machine Learning: A Probabilistic Perspective*. The MIT Press, 2012, p. 1 (cit. on p. 3).

[8] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. «Deep Learning». In: *Nature* 521.7553 (2015), pp. 436–444. DOI: 10.1038/nature14539. URL: https://doi.org/10.1038/nature14539 (cit. on p. 5).

[9] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. The MIT Press, 2016 (cit. on pp. 8, 9, 11).

[10] Sebastian Ruder. «An overview of gradient descent optimization algorithms». In: *arXiv preprint arXiv:1609.04747* (2016) (cit. on pp. 9, 24).

[11] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization*. 2014. arXiv: 1412.6980 [cs.LG] (cit. on p. 9).

[12] Yann LeCun, Yoshua Bengio, et al. «Convolutional networks for images, speech, and time series». In: *The handbook of brain theory and neural networks* 3361.10 (1995), p. 1995 (cit. on pp. 9, 10).

[13] David H. Hubel and Torsten N. Wiesel. «Receptive Fields and Functional Architecture of Monkey Striate Cortex». In: *Journal of Physiology (London)* 195 (1968), pp. 215–243 (cit. on p. 10).

[14] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. *Deep Residual Learning for Image Recognition*. cite arxiv:1512.03385Comment: Tech report. 2015. URL: http://arxiv.org/abs/1512.03385 (cit. on pp. 11, 47, 49, 51, 52).

[15] Sean Gallagher. *Why facial recognition tech failed in the Boston bombing manhunt*. URL: https://arstechnica.com/information-technology/2013/05/why-facial-recognition-tech-failed-in-the-boston-bombing-manhunt (cit. on p. 13).

[16] YUXI PENG, Luuk Spreeuwers, and Raymond Veldhuis. «Low-resolution face recognition and the importance of proper alignment». In: *IET Biometrics* 8 (Jan. 2019). DOI: 10.1049/iet-bmt.2018.5008 (cit. on p. 15).

[17] Hao Wang, Yitong Wang, Zheng Zhou, Xing Ji, Dihong Gong, Jingchao Zhou, Zhifeng Li, and Wei Liu. «CosFace: Large Margin Cosine Loss for Deep Face Recognition.» In: *CVPR*. IEEE Computer Society, 2018, pp. 5265–5274. URL: http://dblp.uni-trier.de/db/conf/cvpr/cvpr2018.html#WangWZJGZL018 (cit. on pp. 15, 16, 22, 25, 33, 39).

[18] Jiankang Deng, Jia Guo, Niannan Xue, and Stefanos Zafeiriou. «ArcFace: Additive Angular Margin Loss for Deep Face Recognition.» In: *CVPR*. Computer Vision Foundation / IEEE, 2019, pp. 4690–4699. URL: http://dblp.uni-trier.de/db/conf/cvpr/cvpr2019.html#DengGXZ19 (cit. on pp. 15, 16, 26, 39).

[19] K. Zhang, Z. Zhang, Z. Li, and Y. Qiao. «Joint Face Detection and Alignment Using Multitask Cascaded Convolutional Networks». In: *IEEE Signal Processing Letters* 23.10 (Oct. 2016), pp. 1499–1503. ISSN: 1070-9908. DOI: 10.1109/LSP.2016.2603342 (cit. on p. 15).

[20] Dong Yi, Zhen Lei, Shengcai Liao, and Stan Z. Li. «Learning Face Representation from Scratch.» In: *CoRR* abs/1411.7923 (2014). URL: http://dblp.uni-trier.de/db/journals/corr/corr1411.html#YiLLL14a (cit. on pp. 16, 38, 46).

[21] http://trillionpairs.deepglint.com/overview (cit. on pp. 16, 39, 46, 48, 50).

[22] 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: 2008 (cit. on pp. 17, 22).

[23] Lior Wolf, Tal Hassner, and Itay Maoz. «Face recognition in unconstrained videos with matched background similarity». In: *CVPR 2011*. IEEE. 2011, pp. 529–534 (cit. on p. 17).

[24] Soumyadip Sengupta, Jun-Cheng Chen, Carlos Castillo, Vishal M Patel, Rama Chellappa, and David W Jacobs. «Frontal to profile face verification in the wild». In: *2016 IEEE Winter Conference on Applications of Computer Vision (WACV)*. IEEE. 2016, pp. 1–9 (cit. on p. 17).

[25] Tianyue Zheng, Weihong Deng, and Jiani Hu. «Cross-age lfw: A database for studying cross-age face recognition in unconstrained environments». In: *arXiv preprint arXiv:1708.08197* (2017) (cit. on p. 17).

[26] Tianyue Zheng and Weihong Deng. «Cross-pose lfw: A database for studying cross-pose face recognition in unconstrained environments». In: *Beijing University of Posts and Telecommunications, Tech. Rep* 5 (2018) (cit. on p. 17).

[27] Matthew Turk and Alex Pentland. «Eigenfaces for recognition». In: *Journal of cognitive neuroscience* 3.1 (1991), pp. 71–86 (cit. on p. 19).

[28] Mei Wang and Weihong Deng. «Deep Face Recognition: A Survey.» In: *CoRR* abs/1804.06655 (2018). URL: http://dblp.uni-trier.de/db/journals/corr/corr1804.html#abs-1804-06655 (cit. on pp. 19, 20).

[29] Zhimin Cao, Qi Yin, Xiaoou Tang, and Jian Sun. «Face recognition with learning-based descriptor.» In: *CVPR*. IEEE Computer Society, 2010, pp. 2707–2714. ISBN: 978-1-4244-6984-0. URL: http://dblp.uni-trier.de/db/conf/cvpr/cvpr2010.html#CaoYTS10 (cit. on p. 19).

[30] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. «Imagenet classification with deep convolutional neural networks». In: *Advances in neural information processing systems*. 2012, pp. 1097–1105. URL: http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks (cit. on p. 20).

[31] Yaniv Taigman, Ming Yang, Marc'Aurelio Ranzato, and Lior Wolf. «Deepface: Closing the gap to human-level performance in face verification». In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2014, pp. 1701–1708 (cit. on pp. 22, 23, 39).

[32] Florian Schroff, Dmitry Kalenichenko, and James Philbin. *FaceNet: A Unified Embedding for Face Recognition and Clustering*. cite arxiv:1503.03832Comment: Also published, in Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition 2015. 2015. DOI: 10.1109/CVPR.2015.7298682. URL: http://arxiv.org/abs/1503.03832 (cit. on pp. 23, 24, 39).

[33] John Duchi, Elad Hazan, and Yoram Singer. «Adaptive subgradient methods for online learning and stochastic optimization». In: *Journal of machine learning research* 12.Jul (2011), pp. 2121–2159 (cit. on p. 24).

[34] Jian Wang, Feng Zhou, Shilei Wen, Xiao Liu, and Yuanqing Lin. «Deep Metric Learning with Angular Loss.» In: *CoRR* abs/1708.01682 (2017). URL: http://dblp.uni-trier.de/db/journals/corr/corr1708.html#abs-1708-01682 (cit. on p. 24).

[35] Weiyang Liu, Yandong Wen, Zhiding Yu, Ming Li, Bhiksha Raj, and Le Song. «SphereFace: Deep Hypersphere Embedding for Face Recognition.» In: *CoRR* abs/1704.08063 (2017). URL: http://dblp.uni-trier.de/db/journals/corr/corr1704.html#LiuWYLRS17 (cit. on pp. 24, 25, 39).

[36] 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)*. Vol. 2. IEEE. 2006, pp. 1735–1742 (cit. on p. 24).

[37] Jiankang Deng, Yuxiang Zhou, and Stefanos Zafeiriou. «Marginal loss for deep face recognition». In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*. 2017, pp. 60–68 (cit. on pp. 26, 39).

[38] Matteo Testa, Arslan Ali, Tiziano Bianchi, and Enrico Magli. «Learning mappings onto regularized latent spaces for biometric authentication.» In: *MMSP*. IEEE, 2019, pp. 1–6. ISBN: 978-1-7281-1817-8. URL: http://dblp.uni-trier.de/db/conf/mmsp/mmsp2019.html#TestaABM19 (cit. on p. 28).

[39] Matteo Testa, Arslan Ali, Tiziano Bianchi, and Enrico Magli. «Learning mappings onto regularized latent spaces for biometric authentication.» In: *CoRR* abs/1911.08764 (2019). URL: http://dblp.uni-trier.de/db/journals/corr/corr1911.html#abs-1911-08764 (cit. on p. 28).

[40] Radford M Neal. *Bayesian learning for neural networks*. Vol. 118. Springer Science & Business Media, 2012 (cit. on p. 30).

[41] S. Kullback and R. A. Leibler. «On Information and Sufficiency». In: *Ann. Math. Statist.* 22.1 (Mar. 1951), pp. 79–86. DOI: 10.1214/aoms/1177729694. URL: https://doi.org/10.1214/aoms/1177729694 (cit. on p. 31).

[42] John Duchi. «Derivations for linear algebra and optimization». In: () (cit. on p. 31).

[43] Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, and Alex Alemi. *Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning*. 2016. arXiv: 1602.07261 [cs.CV] (cit. on pp. 38, 45).

[44] Weiyang Liu, Rongmei Lin, Zhen Liu, Lixin Liu, Zhiding Yu, Bo Dai, and Le Song. «Learning towards Minimum Hyperspherical Energy.» In: *NeurIPS*. Ed. by Samy Bengio, Hanna M. Wallach, Hugo Larochelle, Kristen Grauman, Nicolò Cesa-Bianchi, and Roman Garnett. 2018, pp. 6225–6236. URL: http://dblp.uni-trier.de/db/conf/nips/nips2018.html#LiuLLLYDS18 (cit. on p. 39).

[45] Omkar M Parkhi, Andrea Vedaldi, and Andrew Zisserman. «Deep face recognition». In: (2015) (cit. on p. 39).

[46] Yi Sun, Yuheng Chen, Xiaogang Wang, and Xiaoou Tang. «Deep Learning Face Representation by Joint Identification-Verification.» In: *NIPS*. Ed. by Zoubin Ghahramani, Max Welling, Corinna Cortes, Neil D. Lawrence, and Kilian Q. Weinberger. 2014, pp. 1988–1996. URL: http://dblp.uni-trier.de/db/conf/nips/nips2014.html#SunCWT14 (cit. on p. 39).

[47] Yandong Wen, Kaipeng Zhang, Zhifeng Li, and Yu Qiao. «A Discriminative Feature Learning Approach for Deep Face Recognition.» In: *ECCV (7)*. Ed. by Bastian Leibe, Jiri Matas, Nicu Sebe, and Max Welling. Vol. 9911. Lecture Notes in Computer Science. Springer, 2016, pp. 499–515. ISBN: 978-3-319-46477-0. URL: http://dblp.uni-trier.de/db/conf/eccv/eccv2016-7.html#WenZL016 (cit. on p. 39).

[48] Jingtuo Liu, Yafeng Deng, Tao Bai, Zhengping Wei, and Chang Huang. «Targeting ultimate accuracy: Face recognition via deep embedding». In: *arXiv preprint arXiv:1506.07310* (2015) (cit. on p. 39).

[49] Xiao Zhang, Zhiyuan Fang, Yandong Wen, Zhifeng Li, and Yu Qiao. «Range loss for deep face recognition with long-tailed training data». In: *Proceedings of the IEEE International Conference on Computer Vision*. 2017, pp. 5409–5418 (cit. on p. 39).