DEEP LEARNING FOR FACE RECOGNITION IN SURVEILLANCE VIDEOS

by

Paul-Darius Sarmadi

A dissertation submitted in partial fulfillment of the requirements for the degree of
Master of Science in
Computer Science

Examination Committee:	Dr. Matthew Dailey (Chairperson) Dr. Mongkol Ekpanyapong Prof. Manukid Parnichkun
Nationality:	French
Previous Degree:	Bachelor of Science in Math, Physics TELECOM SudParis, France
Scholarship Donor:	Telecom SudParis, France - AIT

Asian Institute of Technology
School of Engineering and Technology
Thailand
May 2016

Acknowledgments

I want to thank my advisor Dr. Matthew Dailey for his precious help during this research work and all the committee members for their advice and remarks.

I also want to thank my family for all they brought me.

In particular, I want to thank my little brother Maximilien and my little sister Melanie. It is for them that I work and it is thank to them that I smile.

Abstract

The problem of recognizing a previously identified criminal, hoping to follow him or her through video cameras feeds is a key issue. However, police require a great deal of human resources to perform this task. Automating the face verification process in surveillance video seems to be a feasible solution to this problem. Deep learning algorithms have recently reached particularly high levels of accuracy in automated face verification. For example, recent approaches reached over 98% accuracy on the "Labeled Faces in the Wild" (LFW) database. The goal of this research is to explore the possibility of adapting such methods to the particular conditions and constraints of video surveillance. I performed experiments on a database of surveillance videos acquired in a crowded shopping mall. With the neural network architecture described in this article, an accuracy of 88.04% was reached on the database.

Keywords: face verification, surveillance video, deep learning, deep neural network, convolutional neural network.

Table of Contents

List of Figures

Chapter 1

Introduction

1.1 Background

Cameras are everywhere, in front of stores, businesses and on our streets. They are used for several purposes:

- After a crime has been commited, the video can be used as clues or proof. The goal is to get more information on the person or on the event that has occurred.
- In the few minutes after a crime, to intercept the person who committed it. Some police are always scanning surveillance videos, monitoring subways, streets, and so on to be able to catch the perpetrator in the corridor or before he or she could escape.
- Before a crime, typically, to dissuade potential thieves from stealing anything. Sometimes, even cameras that are not working, not recording, or are fakes can be effetive deterrents.

In the context of global terrorism, the problem of recognizing a previously identified terrorist, hoping to follow him or her through video cameras through several feeds sadly appears to be a key issue. More generally, following the path of any criminal using surveillance videos sounds like a main concern.

In this work we can assume that police face several difficulties:

- The number of cameras they may have access to, depending on the locality's policy. This number may be very high, and is growing everywhere very quickly.
- The low quality of the video. It is sometimes very hard to recognize a person in a video with low resolution.
- The crowd. It takes a second to recognize a previously identified thief on a video when he or she is alone. What if there are 30 other people on the video, in a crowded street for example?

Recognizing previously identified criminals in video surveillance feeds adds to the above-mentioned difficulties, requiring a great deal of human resources. These issues lead to a simple conclusion. It is expensive work in terms of time, money, and human resources, but it is nevertheless extremely important to uphold the national security of any country.

Automating the face recognition process in surveillance video seems to be an interesting answer to address some of these problems.

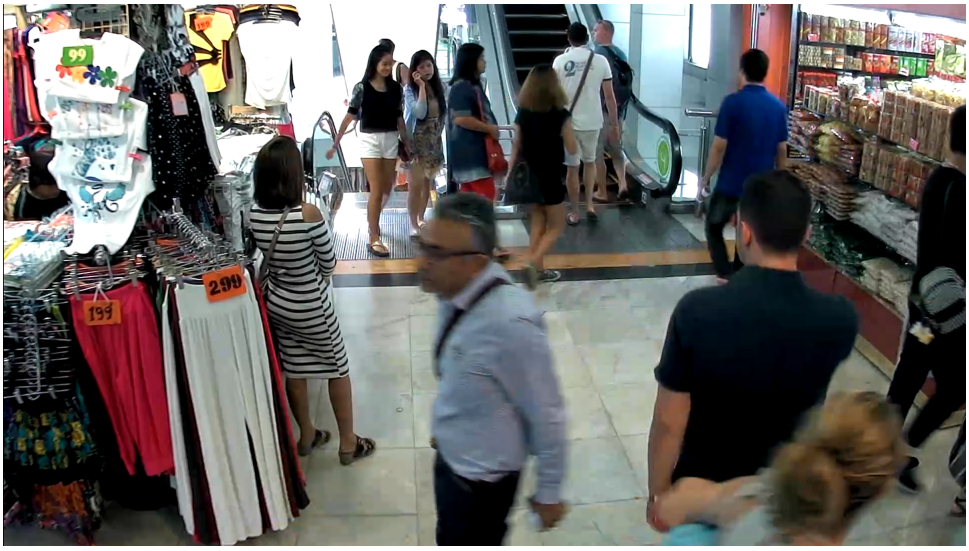


Figure 1.1:: A frame from a video in the MBK dataset.

1.2 Problem Statement

Face identification is a key machine learning issue at present. The best results obtained this far use a "deep" neural network (any artificial neural network with more than one hidden layer). In 2014, DeepFace reached an accuracy of 97.35% on "the Labeled Faces in the Wild (LFW)" dataset for face verification (Taigman, Yang, Ranzato, Wolf, 2014). In 2015, FaceNet reached a 99.63% accuracy on the "LFW" dataset for identification (Schroff, Kalenichenko, Philbin, 2015).

In surveillance video, face recognition faces difficult issues such as blurriness, low resolution, and unexpected poses of faces. Though the problem of face recognition in surveillance video has already been studied, deep learning techniques may improve performance in this scenario.

The goal of this research was to build a deep neural network for face recognition on surveillance videos. This model had to be based on the latest algorithms and best practices deep learning offers.

1.3 Objectives

The objectives of this research were to develop and evaluate methods for a database already provided for the study. This database contained recorded videos from the surveillance camera network at the MBK shopping mall in Bangkok. Figure 1.1 shows a frame of one of these videos. Three of our researchers appeared over several seconds in some of the videos. They were walking like anyone else in the mall.

Choosing this database served one goal: to be placed in the theoretical situation of policemen looking

for one or several criminals in the streets or in the corridors of public places, using a few sample pictures of them to try to track and find out where they have gone.

Our researchers are playing the role of the criminals we are looking for. The main objective was to use deep learning techniques to build an automated solution for the task of finding people we have a few photos of over a larger collection of video streams.

More precisely, considering this database, there were three main objectives in this research study:

- Create a database of images containing faces extracted from the surveillance videos.
- Build a deep neural network for face recognition. The general idea is that the network should be provided with a few photos of our researchers from a training set, and try to find these individuals in a testing set.
- Testing the resulting model. Compare the model's accuracy with other experiments using different techniques.

1.4 Limitations and Scope

There were two main limitations to this research:

- Time. This project was three months long. There are many deep learning techniques existing, but due to the time limit, not all could be explored.
- Material limitations. The laboratory has provided a single NVIDIA GeForce 780 GTX GPU card for the computation. This places some limitations on the mini-batch size for stochastic gradient descent. Preliminary results showed that the size will be limited to 20 samples, while the usual size is 128.

Deep learning gives astonishing results in the task of face recognition. That is why despite those limitations, we were able to get an interesting model for the context of surveillance videos.

1.5 Research Outline

I organize the rest of this dissertation as follows.

In Chapter 2, I provide a review of the relevant literature.

In Chapter 3, I propose my methodology.

In Chapter 4, I present the results of the study.

Finally, in Chapter 5, I conclude my thesis.

Chapter 2

Literature Review

As said in the previous section, the goal of this study was to exploit *deep learning* algorithms for face recognition in surveillance videos.

2.1 About Deep Learning

According to "Deep Learning Methods and Applications" (Deng, Yu, 2014):

> Deep learning is a set of algorithms in machine learning that attempt to learn in multiple levels, corresponding to different levels of abstraction. It typically uses artificial neural networks. The levels in these learned statistical models correspond to distinct levels of concepts, where higher-level concepts are defined from lower-level ones, and the same lowerlevel concepts can help to define many higher-level concepts.

One of the most well-known deep learning algorithms is the *Convolutional Neural Network* (LeCun, Bottou, Bengio and Haffner, 1998), a variant of the multilayer perceptron (Rosenblatt, 1961) which involves the use of convolutional layers (see Figure 2.1). Biologically inspired by mammalian visual mechanisms, convolutional neural networks are historically known for an application called *LeNet*, able to recognize hand-written digits (LeCun et al., 1989). In the last 10 years, the interest in convolutional networks has grown quickly. In fact, with the evolution of technology — basically massive usage of always more powerful GPU cards — it has been possible to use these networks more widely and for various tasks. They have been used widely for human face classification tasks and have given impressive results.

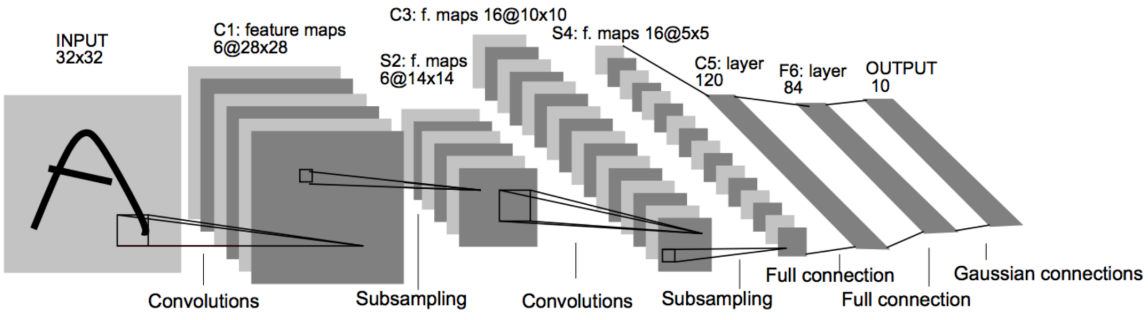


Figure 2.1:: Architecture of LeNet-5 for digit recognition. Extracted from LeCun, Bottou, Bengio and Haffner (1998).

2.2 About Convolutional Neural Networks

A convolutional neural network is generally made of one or more convolutional layers followed by a standard neural network. Convolutional layers are similar to standard layers. The difference is that in this specific case, we connect only adjacent neurons of a layer to the next one, whereas, standard layers connect all the neurons of a layer to the next one. Then, most frequently a subsampling of the convolutional layer is applied and the output of this step is used as an input to the next layer.

The next figures show a typical convolutional neural network represented in 3 dimensions. Its role is to identify a hand-written digit in an image as an example of a particular written number. This 3D representation has been made by Harley, in 2015.

Figure 2.2 shows the full network. The bottom layer is the input, a hand-written “2” digit.

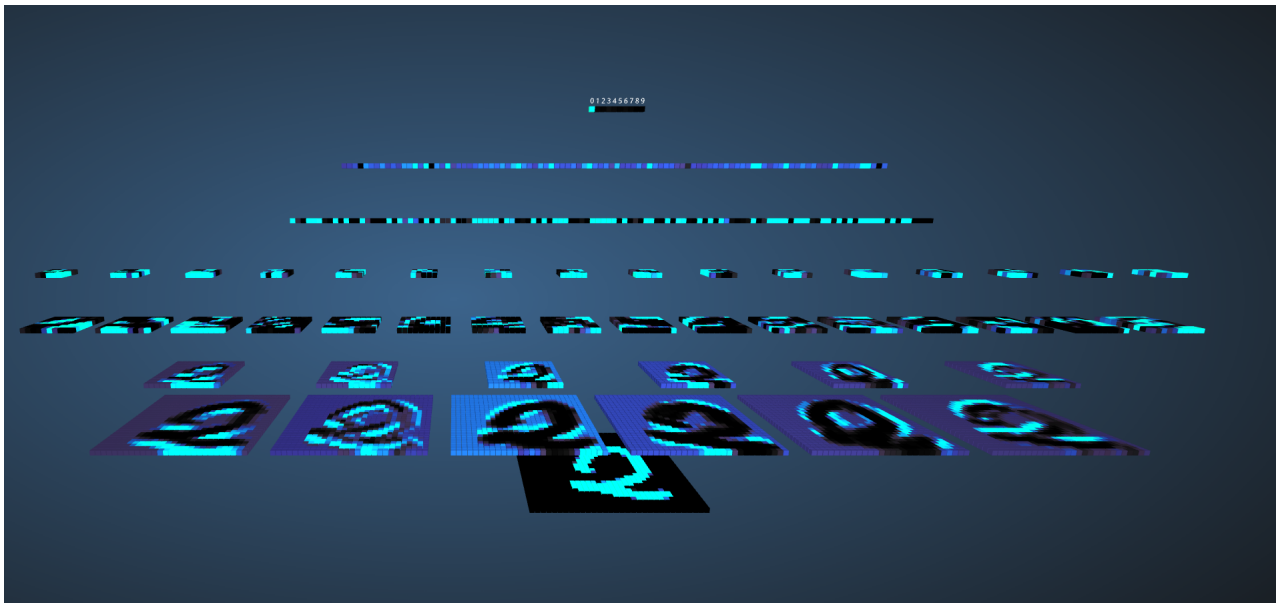


Figure 2.2:: 3-dimensional representation of a typical convolutional neural network. Generated by Harley, 2015.

The next layer is the first convolutional one. See Figure 2.3. Some restricted sub-regions of the input are received by the neurons of the next layer. Convolutional units have tied weights, so that the same processing is applied at every local subregion of the input layer. This idea is biologically inspired by cells from the visual cortex of animals, which are sensitive to fixed small regions of the visual field and perform similar computation over the entire visual field.

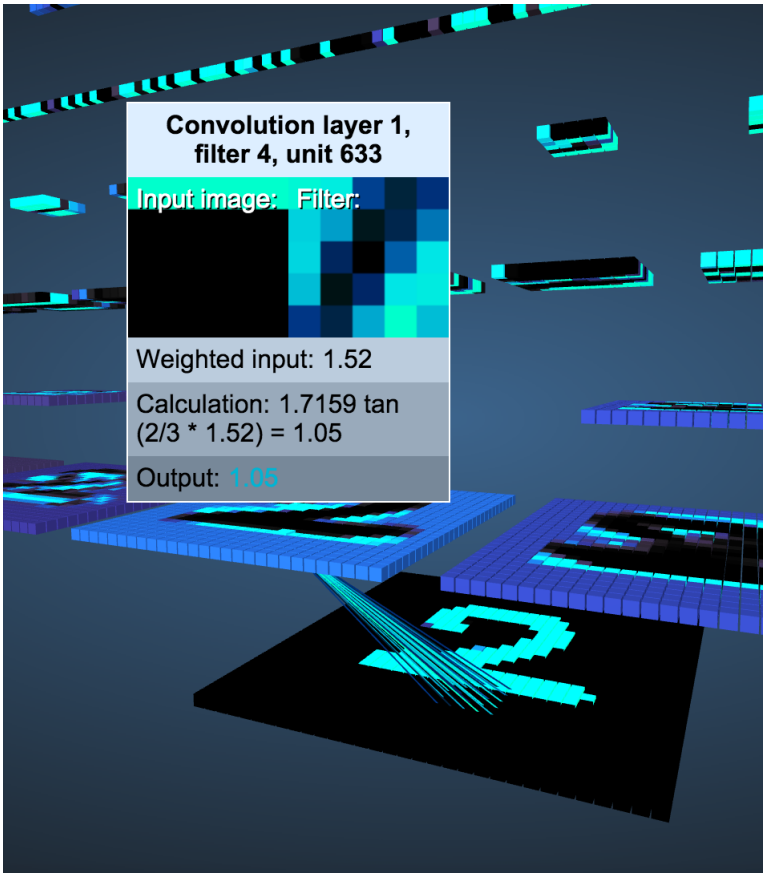


Figure 2.3:: The first convolutional layer. Generated by Harley, 2015.

In the next layer of the convolutional network, subsampling occurs. Subsampling reduces sensitivity to small translations of similar features. Different types of subsampling exist. The most used is "max-sempling" which extracts the maximum value among the pixels of a region. See Figure 2.4.

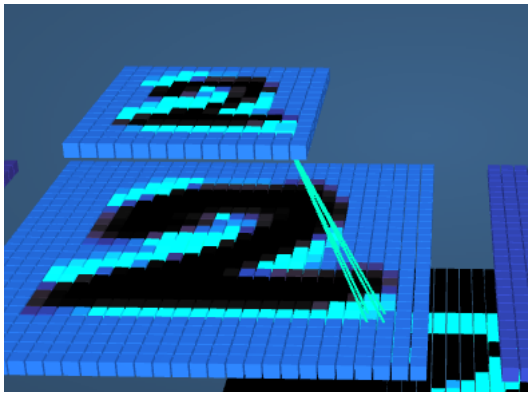


Figure 2.4:: The first subsampling layer. Generated by Harley, 2015.

Figure 2.5 represents the second convolutional layer. Its input is the output of the first subsampling layer which represents the result of the computations of the first convolutional layer.

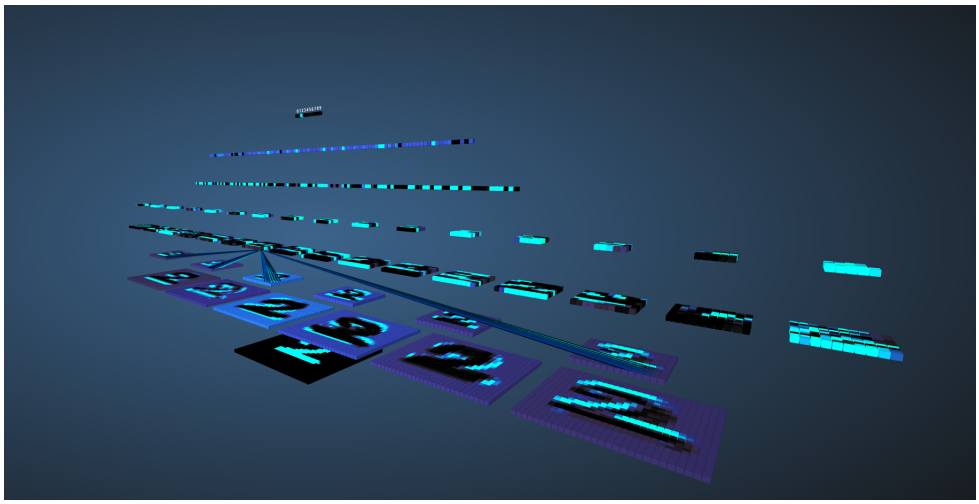


Figure 2.5:: The second convolutional layer. Generated by Harley, 2015.

The two last layers before the final output represent standard fully-connected layers. See Figure 2.6.

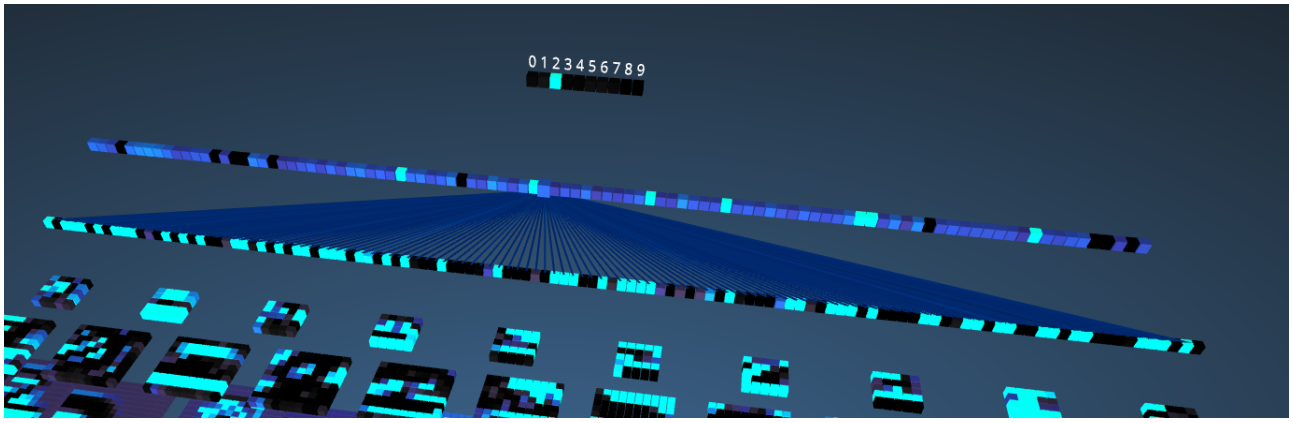


Figure 2.6:: The fully-connected layers. Generated by Harley, 2015.

2.3 Previous Work

2.3.1 Deep learning for face recognition

There are two main schemes for face recognition:

- Recognition by person identification. In this case, a network takes an image as an input and returns a label that identifies one and only one person as an output. The literature on this type of architecture is extremely abundant. For face verification, DeepFace (Taigman, Yang, Ranzato, Wolf, 2014) reached 97.35% accuracy on the Labeled Faces in the Wild (LFW) dataset. The state of the art for face identification is FaceNet (Schroff, Kalenichenko, Philbin, 2015).

- Recognition by comparison. The "Same/Not Same" algorithms. The basic idea is that the training dataset is made of pairs of images linked to the label "1" if they represent the same object — in our case, the face of a person — and "0" otherwise. In deep learning, the standard same/not same network is called a "Siamese Network" (Chopra, Hadsell, LeCun, 2005). A Siamese network is built out of two convolutional networks. The input to the first is the first image of the pair and the input to the second is the second image. The two networks share the same parameters and return two values each. Those values are used to compute an energy. If the energy is high, the two images are considered to be "very different". Otherwise they are considered to be "similar." The energy is a function of a loss function used to update the parameters of the two convolutional networks by contrastive gradient descent (Figure 4.9). Intuitively, the computed energy is similar to gravitational potential energy. If a mass is far from Earth, its GPE is high, and the mass will be considered as not belonging to the planet. If the mass is stuck to the Earth, its GPE will be low, and the mass is considered part of the planet itself.

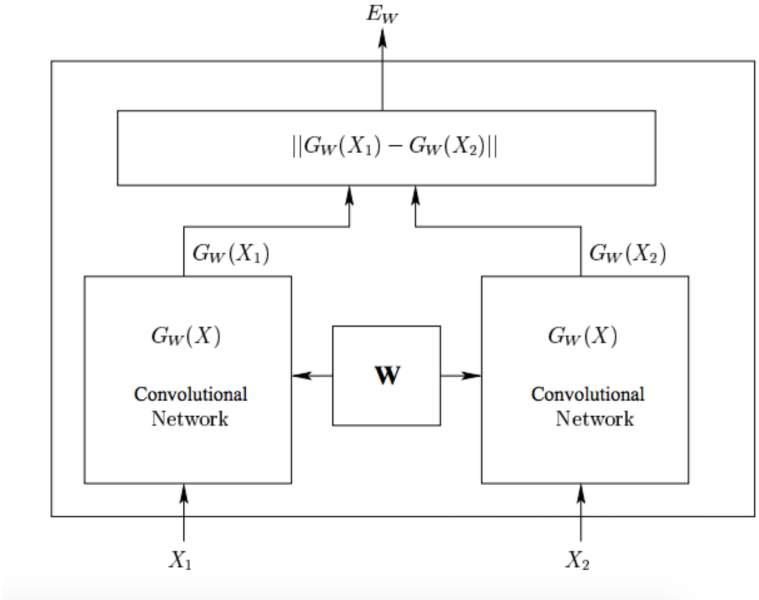


Figure 2.7:: Architecture of a Siamese network. Extracted from Chopra, Hadsell, LeCun, 2005.

2.3.2 Video-Based Face Recognition

On automated face recognition for surveillance video, a number of articles have been published. They are using a wide range of methods. Liu and Chen (2003) used a Hidden Markov Model for video-based face recognition. Le An, Kafai and Bhanu (2012) used a Dynamic Bayesian Network. Goswami, Bhardwaj, Singh and Vatsa (2014) proposed an interesting methodology. First, an algorithm extracts the most "memorable" frames in the video. Then, the chosen frame is applied to a deep learning network performing face recognition. This idea provides the state-of-the-art in low false acceptance rates.

2.4 Conclusion

Video-based face recognition is a very important area of research. Though many articles are published on this subject, some of the latest deep neural network techniques have yet to be applied to the context of images extracted from surveillance videos.

Chapter 3

Methodology

3.1 System Design

Figure 3.1 shows the design of the final product.

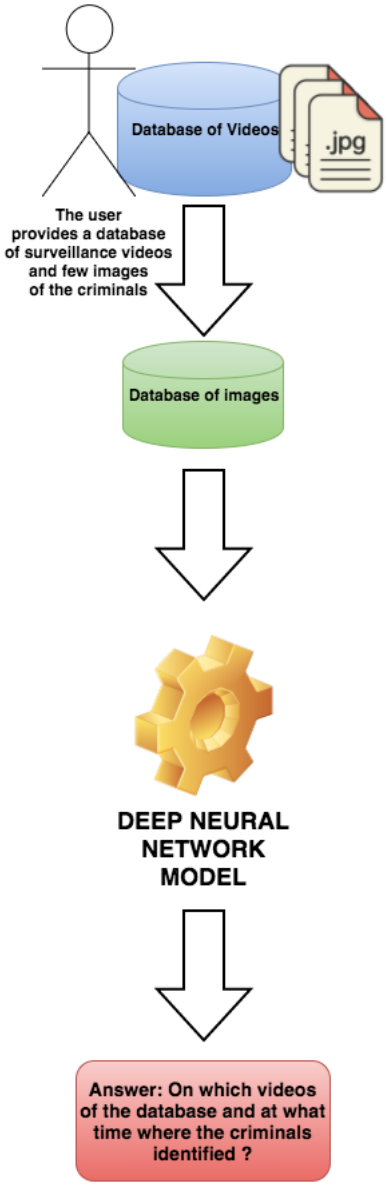


Figure 3.1:: Design of the final product.

3.2 Solution overview

The goal of this research is to build and evaluate a system able to detect our three researchers in the surveillance videos of a mall, as shown in the previous section. My solution is composed of five steps.

- Initially, we have a database of videos.
- Then, these videos are processed to extract the faces of every person appearing in them. We have now a database of faces.
- From this database, I extract training patterns for the learning and test processes.
- A model is trained to recognize our researchers or an existing model is acquired from available models.
- The model is tested on our database of videos.

3.3 Solution Design

Figure 3.2 presents two main ideas. In blue, the steps described in the above section are shown. In green, the solutions used to go from one step to the next are described.

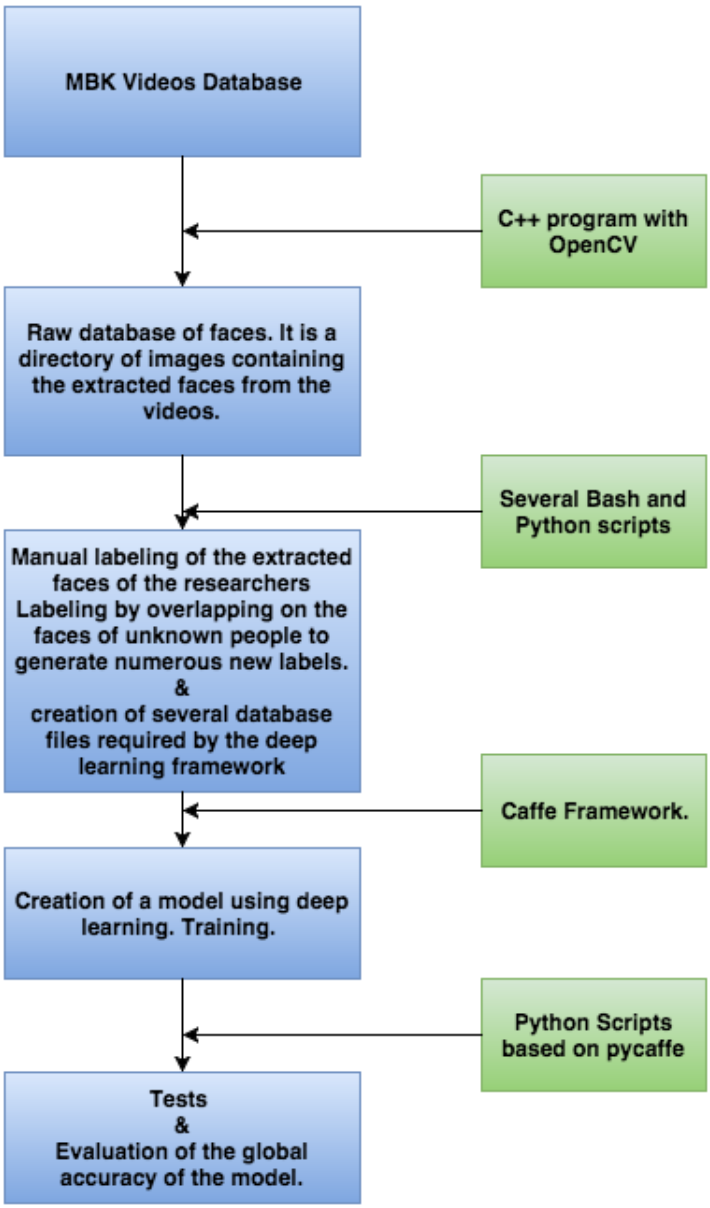


Figure 3.2:: An overview of the global design of the study.

3.4 Database

As said in the introduction, deep learning algorithms were applied to perform face recognition in a video surveillance system. A database of surveillance videos was required to generate a training set and a test set for our model. The database that was used for the learning process is a set of 14 videos recorded in the MBK Shopping Center in Bangkok. The duration of the videos is variable, from a minute to around 3 minutes and 30 seconds. Three of the researchers of our laboratory appear in the videos, walking in the mall like any other person.

3.5 Raw Database of Faces

A C++ algorithm using OpenCV (Bradski, 2000) provided in Algorithm 1 was used for face detection.

```
for each video in the database do
    for each frame N of the current video do
        Detect all the faces of the frame N;
        Save the P-th detection in "Database/video/FrameNFaceP.jpg";
    end
end
```

Algorithm 1: Face detection algorithm

The algorithm used for face detection uses a machine learning process called Haar feature-based cascade classifiers, described by Viola and Jones (2001).

Once the process is over, a "Database" directory is created with one directory for each video. In each directory, all of the faces are saved as JPEG files.

Figure 3.3:: An example of two faces extracted from the surveillance system.

3.6 Creation of database files

The framework that will be presented in the next section requires two files to work: a `train.txt` file and a `test.txt` file. Their role is trivially linked to their name in a supervised learning process.

3.6.1 Case of direct face identification

In the case of direct face identification, each of the `train.txt` and `test.txt` files share the same structure:

```
/adress/of/the/training/image1.jpg label1\newline
/adress/of/the/training/image2.jpg label2\newline
...\newline
/adress/of/the/training/imageN.jpg labelN
```

The labels are assigned automatically using the simple tracking method described in the next subsection.

The labels of the researchers were obtained by automatically tracking then manually merging tracks of the same researcher in different video segments.

3.6.2 Face tracking

for *each video in the database* **do**
  for *each frame of the current video* **do**
    Detect all the faces of the frame N;
    for *each face of the current frame* **do**
      if *there is a face detected in the three previous frames whose distance with the current face is inferior to the size of the face* **then**
        The label L of the current face := The label of the nearest of the faces of the three previous frames;
        Save the detected face in “Database/video/LabelLFrameNFaceP.jpg”;
      end
      else
      end
      The label L of the current face := Biggest label given until now + 1 Save the detected face in “Database/video/LabelLFrameNFaceP.jpg”;
    end
  end
end

Algorithm 2: Automatic Labeling Algorithm

A representation of the final database is given below:

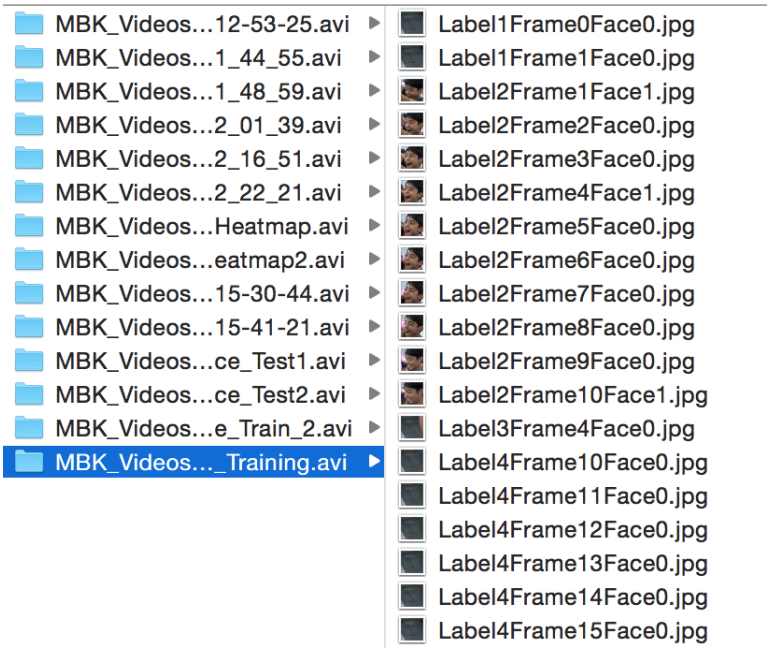


Figure 3.4:: The database after automatic labeling.

3.6.3 Case of a "Same/Not same" model

In the case of a "Same/Not same" model, the problem was a bit different. A slightly unusual structure is best for this model. The solution is to generate two files for training and two files for testing. Let's name them `train1.txt`, `train2.txt`, and `test1.txt` and `test2.txt`. Their individual structure are as explained before.

/adress/of/the/training/image1.jpg label1
/adress/of/the/training/image2.jpg label2
...
/adress/of/the/training/imageN.jpg labelN

In line K, train1.txt and train2.txt contain respectively "address/to/file1K.jpg labelK" and "address/to/file2K.jpg labelK". The labels are identical. However, the images are different. If both the images represent the same person, the label will be 1, and 0 otherwise. `test1.txt` and `test2.txt` will work the same way. Only one of the two labels are used for the supervised learning, the other one, being identical, is not used. The syntax of the training and testing files being fixed with Caffe, this solution seems to be the most adapted to the problem encountered with the "Same/Not Same" models. The fact that two face images are considered as different or identical is determined by the face tracking algorithm described above.

3.6.4 Conclusion

It was a feasible task to create a series of Python scripts that run one after the other to create the required train.txt and `test.txt` files. These python scripts are indirectly launched through bash scripts. A README file will be provided to explain which commands to type and which options to select to generate the required files from the database.

The purpose of the designed architecture is to make those scripts reusable for any new database and generalizable for an arbitrary number of labels. If a user provides another database of videos, and follows the process described in the `README` file, the required `train.txt` and `test.txt` files will be generated, and the models presented in the next section can be used for learning or testing directly on these data.

3.7 Model

3.7.1 Framework

The chosen deep learning framework for this study is Caffe (Jia et al., 2014).

3.7.2 Strategy

Different strategies have been considered for this modelisation. As explained in the previous chapter, there are two usual schemes for face recognition.

- The first one is the "same/not same" or "face verification" scheme. The idea is to readjust the Siamese network described by Lecun et al. (2005) to our particular dataset.

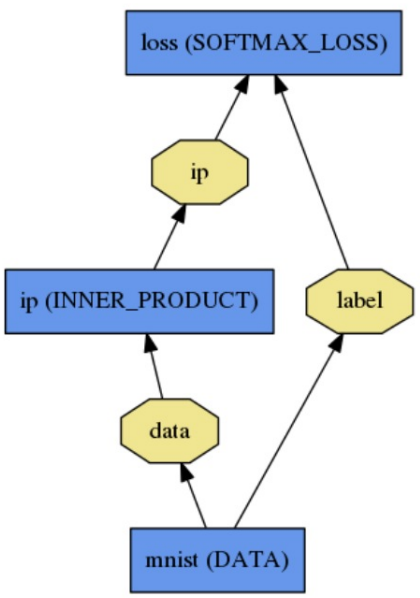
- The second scheme is direct face identification. For this purpose, the idea is to modify the last layer of a state-of-the-art deep neural network architecture for face identification. The output of this last modified layer should be binary. Either the input image represents one of our researchers — in practice, a criminal —-, or it does not. The previous layers should already be trained, and the training should be done in the last layer only.

3.7.3 The learning process with Caffe

The Caffe framework requires several files to train a model.

- First, `train.txt` and `test.txt` files, which give paths to all images with corresponding labels.

- Second, a `train_test.prototxt` file that describes the architecture of the network. This file is written with protobuf. According to the `README` file available on the project's GitHub site, "Protocol Buffers (a.k.a., protobuf) are Google's language-neutral, platform-neutral, extensible mechanism for serializing structured data.". Figure 3.5 shows how a logistic regression classifier is easily defined in a train_test.prototxt file with Caffe.




```
name: "LogReg"
layer {
  name: "mnist"
  type: "Data"
  top: "data"
  top: "label"
  data_param {
    source: "input_leveldb"
    batch_size: 64
  }
}
layer {
  name: "ip"
  type: "InnerProduct"
  bottom: "data"
  top: "ip"
  inner_product_param {
    num_output: 2
  }
}
layer {
  name: "loss"
  type: "SoftmaxWithLoss"
  bottom: "ip"
  bottom: "label"
  top: "loss"
}
```

Figure 3.5:: Logistic regression classifier definition with Caffe. Extracted from the official website of the framework.

- Third, a `solver.prototxt` file, which contains information on the batch size or on the variables related to the used loss function.

The learning process produces two files: A `.caffemodel` and a `.solverstate`. These files are used to store the value of the parameters of the designed model after a number of steps of learning chosen in the `.solverstate` file.

3.8 Testing

As we just said, the output of a face identification model should be binary. A Python script to test the accuracy of the model can be written with no difficulty. On the contrary, the input of a Siamese network is two images and the output can be interpreted as an energy. This energy is high for two images representing two different people and low otherwise (in some models, it is the opposite). A threshold on this energy has to be determined to make classification possible with the network. Thus,

before any test, a script had to be written to determine a good threshold on the training set. This script was written using pycaffe, the caffe model for python. Then, and only then could the tests be done.

Chapter 4

Results

This chapter describes the results I obtained during this research. First, the deep neural network model used in the study is described. Then, the results obtained with this network in the context of face verification are given. After that, I describe the final experimental software, which is available online and the results one can expect from it. Finally I describe the results of the program on a particular video of the database.

4.1 Preliminary results

Before the proposal, some parts of the solution had been built.

- Scripts to generate the database of face images from a video surveillance sequence had been written. The database had been generated and was usable for a direct face identification model. However, the overlap calculation for face detection was not written yet. Hence, the number of available classes was limited, decreasing the performances of a "Same/Not Same" network. A direct face identification model would not affected by this issue, and it could have been built with this version of the database. The database contained 55,203 images. After manual labeling of the faces in the database, 183 images were labeled 1 for the first researcher, 325 were labeled 2 and 15 were labeled 3.
- The scripts to generate the database files mentioned in Figure 3.2 were fully written, and worked both for a direct face identification model and for a Siamese network. I executed them for this second scenario. Four files train1.txt, train2.txt, test1.txt, and test2.txt, as described in section 3.6.2, had already been built.
- I designed a Siamese model with Caffe and trained it on the generated database for 50,000 epochs. This training required one night, with batches of size 20, on the 780 GTX GPU card given by the laboratory. The accuracy of the resulting network could not be tested at that moment because of the singular output generated by such a network, as mentioned in section 2.3.1. A script was being written to face this issue.

4.2 Evolution after the proposal

After the proposal, the direction taken by the study slightly evolved.

- A Python script to interpret the results of the Siamese training was written. Its role was to determine a threshold for the distance between two images as described in the previous chapter. The results were not satisfying as the output vector for each image through the Siamese

network was always a 0 vector. I hypothesized that the problem was that only the images representing the researchers were labeled.

- Consequently, I adapted the code for face detection to make it a face tracker. The scripts for the database generation were modified accordingly. Then the Siamese network was trained again considering the fact that there were a large range of classes with few images in each. However, again, the results were not satisfying, and the exact cause of this remains uncertain. An error in the code is improbable, as very similar code was used for digit recognition and worked fine. The most probable cause is the non-convergence of the network.
- Then, I studied an other network. The idea was to test a pre-trained network on which the MBK database could be used with only minor modifications. A particularly light and efficient face verification network which reached state-of-the-art was selected. The details concerning this network's architecture and the obtained results were particularly interesting and are explained in the next sections.

4.3 Network used in this research

Wu, He and Sun (2015) published a light convolutional neural network for face verification. Its particularity is that it is extremely light but still reaches state-of-the-art results.

4.3.1 Architecture

Figure 4.1 describes the architecture of the Wu, He and Sun model.

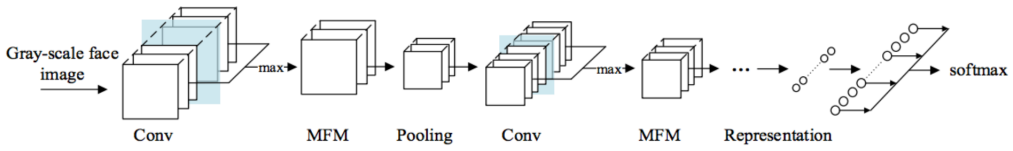


Figure 4.1:: Architecture of the lightened convolution neural network. Extracted from Wu, He, Sun, 2015.

First, as an input, a gray-scale face image is provided. The article suggests the use of two possible models. The one used in this research is built with 4 convolution layer with Max-Feature-Map activation functions, 4 max-pooling layers and 2 fully connected layers. The details are given in Figure 4.2.

A				B			
Name	Filter Size /Stride	Output Size	#param	Name	Filter Size /Stride, Pad	Output Size	#param
input	-	$144 \times 144 \times 1$	-	input	-	$144 \times 144 \times 1$	-
crop	-	$128 \times 128 \times 1$	-	crop	-	$128 \times 128 \times 1$	-
conv1_1	$9 \times 9/1$	$120 \times 120 \times 48$	3.8K	conv1_1	$5 \times 5/1, 2$	$128 \times 128 \times 48$	1.2K
conv1_2	$9 \times 9/1$	$120 \times 120 \times 48$	3.8K	conv1_2	$5 \times 5/1, 2$	$128 \times 128 \times 48$	1.2K
mfm1	-	$120 \times 120 \times 48$	-	mfm1	-	$128 \times 128 \times 48$	-
pool1	$2 \times 2/2$	$60 \times 60 \times 48$	-	pool1	$2 \times 2/2$	$64 \times 64 \times 48$	-
				conv2_a	$1 \times 1/1$	$64 \times 64 \times 48$	0.04K
conv2_1	$5 \times 5/1$	$56 \times 56 \times 96$	2.4K	conv2_1	$3 \times 3/1, 1$	$64 \times 64 \times 96$	0.8K
conv2_2	$5 \times 5/1$	$56 \times 56 \times 96$	2.4K	conv2_2	$3 \times 3/1, 1$	$64 \times 64 \times 96$	0.8K
mfm2	-	$56 \times 56 \times 96$	-	mfm2	-	$64 \times 64 \times 96$	-
pool2	$2 \times 2/2$	$28 \times 28 \times 96$	-	pool2	$2 \times 2/2$	$32 \times 32 \times 96$	-
				conv3_a	$1 \times 1/1$	$32 \times 32 \times 96$	0.09K
conv3_1	$5 \times 5/1$	$24 \times 24 \times 128$	3.2K	conv3_1	$3 \times 3/1, 1$	$32 \times 32 \times 192$	1.7K
conv3_2	$5 \times 5/1$	$24 \times 24 \times 128$	3.2K	conv3_2	$3 \times 3/1, 1$	$32 \times 32 \times 192$	1.7K
mfm3	-	$24 \times 24 \times 128$	-	mfm3	-	$32 \times 32 \times 192$	-
pool3	$2 \times 2/2$	$12 \times 12 \times 128$	-	pool3	$2 \times 2/2$	$16 \times 16 \times 192$	-
				conv4_a	$1 \times 1/1$	$16 \times 16 \times 192$	0.19K
conv4_1	$4 \times 4/1$	$9 \times 9 \times 192$	3K	conv4_1	$3 \times 3/1, 1$	$16 \times 16 \times 128$	1.1K
conv4_2	$4 \times 4/1$	$9 \times 9 \times 192$	3K	conv4_2	$3 \times 3/1, 1$	$16 \times 16 \times 128$	1.1K
mfm4	-	$9 \times 9 \times 192$	-	mfm4	-	$16 \times 16 \times 128$	-
pool4	$2 \times 2/2$	$5 \times 5 \times 192$	-	pool4	$2 \times 2/2$	$8 \times 8 \times 128$	-
				conv5_a	$1 \times 1/1$	$8 \times 8 \times 128$	0.12K
				conv5_1	$3 \times 3/1, 1$	$8 \times 8 \times 128$	1.1K
				conv5_2	$3 \times 3/1, 1$	$8 \times 8 \times 128$	1.1K
				mfm5	-	$8 \times 8 \times 128$	-
				pool5	$2 \times 2/2$	$4 \times 4 \times 128$	-
fc1	-	256	1,228K	fc1	-	256	524K
fc2	-	10,575	2,707K	fc2	-	10,575	2,707K
loss	-	10,575	-	loss	-	10,575	-
total			3,961K				3,244K

Figure 4.2:: Details of the architecture of the lightened convolutional neural networks. Extracted from Wu, He, Sun (2015).

4.3.2 Training

The network was trained on a GTX980 during two weeks on the CASIA-WebFace dataset.

The Dropout algorithm is used for the fully connected layers with a ratio of 0.7. The momentum was set to 0.9, and the weight decay to 5e-4 except for fc2 layer for which the rate is set to 5e-3. The learning rate was set to 1e-3, and its value was reduced step by step to finally reach 5e-5. The parameter initialization for the convolution layers was Xavier and it was Gaussian for the fully connected layers.

4.3.3 Runtime performance

To extract one face image representation on a single core i7-4790, 71ms on the first model and 67ms on the second one are needed. This implies the model can be used in real-time applications.

4.3.4 Results

As said previously, these networks reach state-of-the-art results. They were trained on the CASIA-WebFace database and tested both on LFW and YTF (YouTube Faces Database). The results are given in the next two tables, compared with other state-of-the-art methods.

Method	#Net	Accuracy	TPR@FAR=0.1%	Protocol	Rank-1	DIR@FAR=1%
DeepFace [24]	1	95.92%	-	unsupervised	-	-
DeepFace [24]	7	97.35%	-	unrestricted	-	-
Web-Scale [25]	1	98.00%	-	unrestricted	82.1%	59.2%
Web-Scale [25]	4	98.37%	-	unrestricted	82.5%	61.9%
DeepID2 [19]	1	95.43%	-	unsupervised	-	-
DeepID2 [19]	4	97.75%	-	unsupervised	-	-
DeepID2 [19]	25	98.97%	-	unsupervised	-	-
WebFace [27]	1	96.13%	-	unsupervised	-	-
WebFace+PCA [27]	1	96.30%	-	unsupervised	-	-
WebFace+Joint Bayes [27]	1	97.30%	-	unsupervised	-	-
WebFace+Joint Bayes [27]	1	97.73%	80.26%	unrestricted	-	-
FaceNet [17]	-	99.63%	-	unrestricted	-	-
VGG [16]	1	97.27%	81.90%	unsupervised	74.10%	52.01%
Our model A	1	97.77%	84.37%	unsupervised	84.79%	63.09%
Our model B	1	**98.13%**	**87.13%**	unsupervised	**89.21%**	**69.46%**

Figure 4.3:: Comparison with other state-of-the-art methods on LFW. Extracted from Wu, He, Sun (2015).

Method	#Net	Accuracy	Protocol
DeepFace [24]	1	91.40%	supervised
WebFace [27]	1	88.00%	unsupervised
WebFace+PCA [27]	1	90.60%	unsupervised
VGG [16]	1	92.80%	unsupervised
Our model A	1	90.72%	unsupervised
Our model B	1	91.60%	unsupervised

Figure 4.4:: Comparison with other state-of-the-art methods on YTF. Extracted from Wu, He, Sun (2015).

4.4 Results on the MBK database

4.4.1 The MBK database

The MBK database is an experimental database of faces extracted from surveillance videos in the MBK mall in Bangkok. The labeling of the faces is done both through automatic overlapping and manual labeling for the researchers. This leads to around 85,000 images of faces (a first experiment described in the "Preliminary Results" section lead to around 50,000 images but a new experiment with different parameters lead to the detection of more faces). Few mistakes were made by the automatic process and it was not feasible to correct them. Furthermore, the faces are directly extracted from frames and their resolution is often very poor while their blurriness is important.

4.4.2 Test

The network A presented in the previous section and pre-trained for the LFW database was used on this database of face images from surveillance video. It was tested on 13881 negative pairs and 13881 positive pairs. The first element of each pair was taken in the direct order of the database. The second element was taken such that each first element was the origin of as many positive and negative pairs. For each label, each positive pair which could have been created were actually created. Concerning the negative pairs, they were chosen randomly.

4.4.3 Result

The calculation of the following curves was made under Matlab using the VLFeat library.

The ROC curve obtained is shown in the following figure. The accuracy of the model reaches 88.04%.

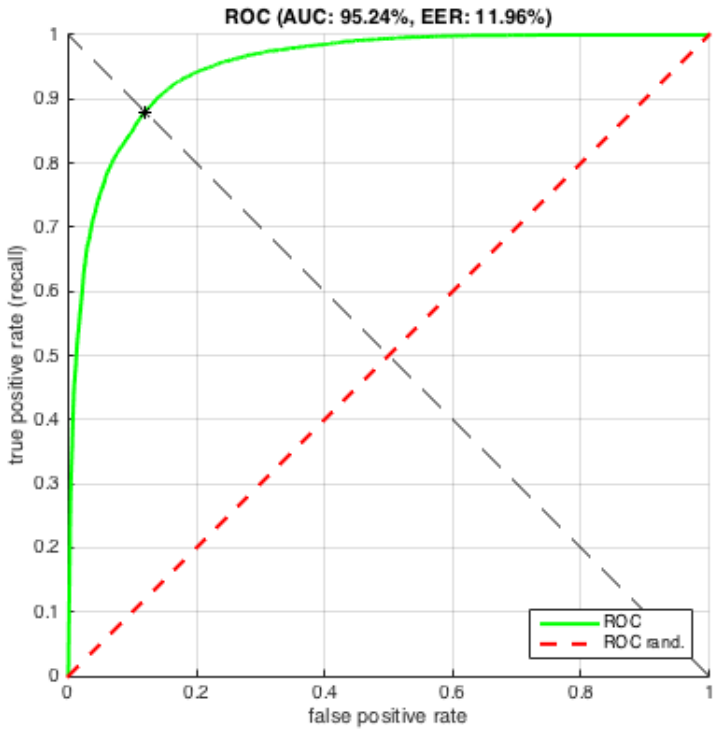


Figure 4.5:: ROC curve of the network on the MBK database.

The ROC curve obtained on the MBK database is compared with other newtorks such as DeepFace on YTF database in the following figure. This comparison is a simple indication as the curves do not concern the same databases, though they are both made of face images extracted from videos.

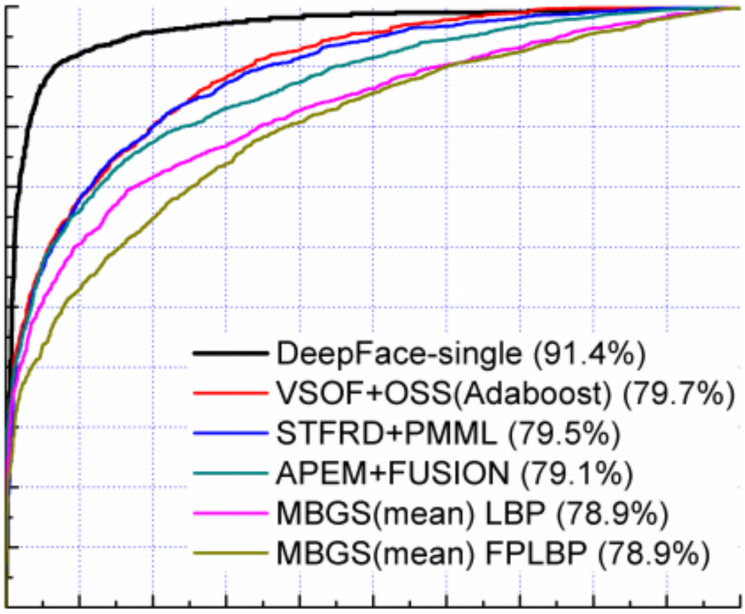


Figure 4.6:: ROC curve of other network on the YTF database. Extracted from DeepFace (Taigman, Yang, Ranzato, Wolf, 2014).

A histogram describing the repartition of the positive and negative pairs relatively to their score is also provided. The negative pairs are represented in blue and the positive ones are represented in red. Few errors on the automatic process of overlapping were made. The details and the consequences of these errors are described in the next section.

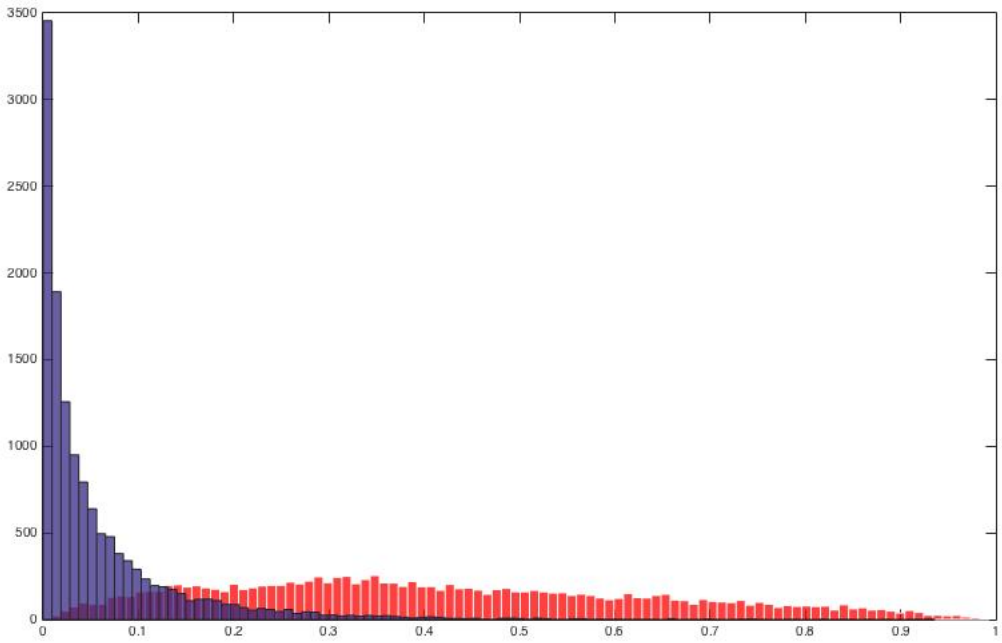


Figure 4.7:: Repartition of the positive and negative pairs according to their score.

The ROC curve is built using this histogram. The idea is that a threshold between 0 and 1 separates the histogram according to the following illustration.

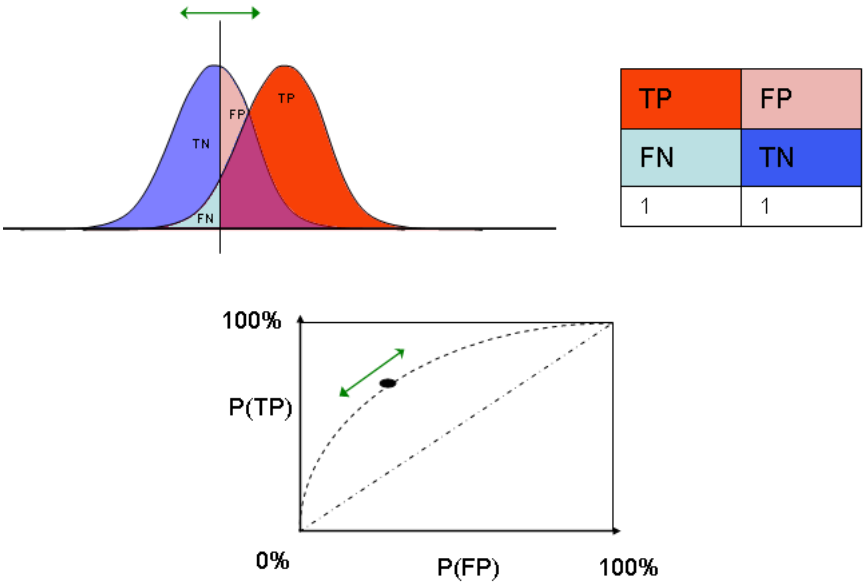


Figure 4.8:: Calculation of the ROC curve. Extracted from Wikipedia.

For a given threshold, some of the pairs will be considered as positive being actually positive (true positive or TP). Some will be considered as positive while they are actually negative (false positive or FP). Some will be considered as negative being actually negative (true negative or TN). Finally, some will be considered as negative while they are actually positive (false negative or FN).
The ROC curve is the curve made of all the points determined by a threshold between 0 and 1, for which the abscissa is the false positive rate and for which the ordinate is the true positive rate.

The threshold which is generally kept is the one for which we have an equal probability of miss-classifying a positive or negative sample. This point is obtained by intersecting the ROC curve with a diagonal of the unit square. Its abscissa is given by the EER. In our case, the EER = 11.96%. The corresponding threshold is 0.14.

4.5 Description of the final product

The final experimental product proposed at the end of this study is made of several parts.

4.5.1 Face detection overlapping

As said in the third chapter, the face detection algorithm was written with C++ and OpenCV. The algorithm uses Haar feature-based cascade classifiers. A simple face detector was not enough and a face tracking algorithm providing automatic labeling of the faces had to be written for two reasons.

- First, to test the network we needed positive and negative pairs of images. The problem being that there are almost 90,000 images of faces in the database after the face detection was done and it was not possible to create enough pairs manually.
- The second point is described in the last section of this chapter.

Consequently, a face tracking algorithm was written. Its behavior is described in the chapter Methodology, section "Creation of database files".

The face tracker is efficient enough in our case but no study has been made on its real accuracy. Practically, two errors can be made.

- A single face appearing on the video is considered with two different labels.
- Several faces appearing on the video are considered with the same label.

The first error does not change much about the results of the study. The negative pairs are chosen randomly in the list of all the faces. The possibility that two images from the same person are considered as negative pairs is very low. However the second scenario, more rare but still existing, is more dangerous for the precision of the analysis of the results as the positive pairs are created with all the available images of the same label. Consequently, the number of pairs considered as positive with a relatively low score is slightly overestimated while the number of pairs with a relatively low score is slightly underestimated. Hence, the global accuracy of the product is slightly underestimated.

4.5.2 Face verification

After the face detection process is done, the second part of the product can start. It performs several successive actions.

- Asks the user to enter the address of the images of the person he is looking for in the database.
- Process those images in the neural network which outputs an array of features (numbers) from each image. The above-mentioned distance between two images is directly computed as a cosine-score of their two corresponding arrays of features.
- Asks for the address of the database containing the images of face.
- The array of features of each image is computed by feed-forwarding the image in the neural network. The cosine-score of each image of a detected face with each image of the person the user is looking for is computed. The algorithm decides whether it represents the same person or not.

4.5.3 Selection of the best test

For the rest of this chapter, let's call the detected images on the videos the "suspect images" and the images provided by the user the "criminal images".
The initial naive idea was to consider that if the score of a suspect image with each criminal image is above the threshold then the suspect is the criminal, and else he or she is not. The problem is that this idea surely increases the true negative rate of the test, but also decreases the true positive rate, which is practically not acceptable.
This is where automatic labeling becomes particularly useful. Let's consider a number N of suspect images that we know are from the same person, thank to the automatic labeling. Let's also consider a number n of criminal images. The problem underlined is "How many pictures among the N images of suspects have to be identified by at least how many n images of criminals to maximize the accuracy of the model?"

The answer is expressed in terms of probability. For a given suspect image and a given criminal image, let's call D the event "a suspect is detected as a criminal on one picture" and P the event "the suspect is actually the criminal on the picture". Then:

$$P(P|D) = \frac{TPR}{TPR + FPR} = p$$

where TPR = True Positive Rate and FPR = False Positive Rate.

Let's assume that the fact that a suspect image is detected as a criminal image on one picture is independant of the fact that the same suspect image is detected as a criminal image on an other picture. Then the probability that the suspect image is detected k times among the n images of the criminal follows a binomial distribution written as follow:

$$P_{exactlyk,n}(P|D) = C_n^k p^k (1-p)^{n-k}$$

And the probability that the suspect is detected at least k times among the n images is:

$$P_{k,n}(P|D) = \sum_{k'=k}^{k'=n} C_n^{k'} p^{k'} (1-p)^{n-k'} = A_{k,n}$$

Let's assume now that the fact that a suspect image is detected at least k times among n as criminal image is independant of the fact that an other suspect image with the same label is also detected as a criminal at least k times among n. Then it also follows a binomial law written:

$$P_{l,N,k,n}(P|D) = \sum_{l'=l}^{l'=N} C_N^{l'} A_{k,n}^{l'} (1 - A_{l',n})^{N-l'}$$

In conclusion, we have formulated the probability that for a given set of N images of a suspect, the probability that he or she is actually a criminal knowing that at least l images among the set are de-

tected as criminals at least k times among the total number of criminal images.

This defines a matrix of probabilities M such that:

$$M_{i,j} = P_{i,N,j,n}$$

In a scenario where we have five suspect images and three criminal images, the resulting matrix has the aspect presented in the following picture:

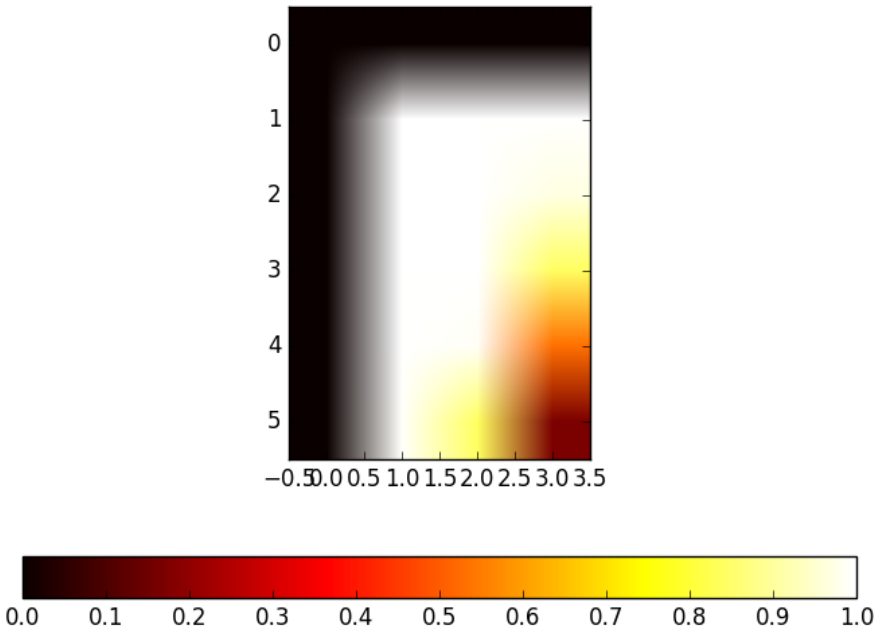


Figure 4.9:: The matrix of probabilities.

Here are the values of the matrix:

[0, 0, 0, 0]
[0, 0.9999999999999951, 0.9999999525689869, 0.9976629875412226]
[0, 0.9999999999823158, 0.9999932742823311, 0.9700927023132484]
[0, 0.9999999742413335, 0.9996171531778806, 0.839991448518568]
[0, 0.9999812348061731, 0.9890255828765551, 0.533024434956926]
[0, 0.9931600804077683, 0.8398962730338903, 0.1708882346261404]

Obviously, this matrix just gives an indication and can not be trusted. The main reason is that the binomial law is not really applicable because the experiences are not independant. In fact, the suspect

images are not so different from a frame to the other. Still, the matrix can give an interesting indication on which test is the best one to work with on the current label.

Theoretically, other cases exist. For example with two suspect images and three criminal images, how to decide if the suspect is the criminal and with which accuracy when the first suspect image is never detected as a criminal and the second one is detected three times as a criminal? This case has not been studied.

The final point is that there are others complex and interesting ways to decide whether a suspect is a criminal after the tests. The law of large numbers can be used. Basically if

$$Number\_of\_positive\_tests/N$$

has a distance to accuracy which is smaller than the distance to

$$1 - accuracy$$

, one can conclude that the test is positive.

4.5.4 Processing of group of images by label

Given a label, meaning given a group of suspect images representing one person, the test to decide whether or not the person appearing on the images is the criminal works like this:

- Let N be the number of suspect images and n the number of criminal images.
- The matrix mentioned in the previous subsection is computed and the line L and column C corresponding to the value are extracted. The suspect images will have to be detected as criminals at least C times for at least L of them.
- The tests are made by feed-forwarding the images in the neural network. If the minimum conditions given by the previous matrix are filled, the algorithm stops and the images are saved in a directory at an address similar to "Captured_Faces/VideoX/LabelYFrameZFaceF.jpg".

When the algorithm is finished, the user has access to all faces detected as criminals and can see, according to the filename the frame number and the name of the video in which he or she is detected. This is precisely filling the goal of the study described in the third section of the third chapter.

4.6 A test on a video of the database

A test of the software has been done on one video of the database where the three researchers appear. 9848 faces were detected on the video. 47 of them contained one of our researcher. A picture of this researcher was provided and the tests were made for each image and no decision taking algorithm as explained in the section "Selection of the best test" was used. The threshold determined earlier was

a bit smaller than the actual one because of the mistakes contained in the face tracker and thus in the test. For this experience, we selected a threshold of 0.2.
46 out of the 47 pictures of the researchers were detected. The true positive rate is 97.87%. This score may be quite overestimated by the fact that the given picture of the researcher was very similar to the ones detected in the video. Among the 9801 face images that did not represent our researcher, 762 were detected as positive. Consequently, the false positive rate is 7.77%. The accuracy of the system on this video in particular is 92.25%. This result underlines the fact that the global accuracy of the system was underestimated by the errors in the face tracker.
The 9848 detected faces of the video which was 1 minutes and 14 seconds long with 30 frames per second were processed by the neural network and compared with the picture of the researcher in three hours and twenty minutes on a MacBook Pro with a 2.4GHz Intel Core i5 CPU.

Chapter 5

Conclusion

5.1 Limitations. Possible improvements.

Several improvements can be made.

- First, concerning the face detection. The face detection used in this experience has a high rate of false positive. Despite the fact that the neural network we used is particularly efficient, its behavior is a bit unexpectable when the image of a face is compared to something which is not one. Using an other library like dlib with a lower false positive rate would be more efficient. Such a script was written in python, but tests have still to be done.
- The second point concerns the face tracking. Though the written algorithm has an acceptable experimental accuracy, other techniques like Mean Shift are known as way more efficient for face tracking.
- The last point is that, as mentioned in the previous chapter, the technique used to test whether or not a group of labeled face images should be considered as a criminal or not can be improved. The solution used in this software is not the best solution. The law of large numbers is probably a better solution. It was not implemented here.

5.2 How to use this software

This section is used as a `README` text for the project, available on GitHub. The address is:
`http://github.com/Paul-Darius/ait-research-study-finished`

5.2.1 Face detection

Installation :

```
chmod+x install.sh ; ./install.sh
```

Generate Database :

```
chmod +x generate\_database.sh;
./generate\_database.sh /path/to/database/ [demo\_mode]
```

- `/path/to/database` is either an absolute or relative (from the current directory) path to the directory containing all the videos.

- `demo_mode` is an option to decide whether you want to see in live and save in the Database directory a video showing you the database generation or not. If the demo mode is set, the generation will be slower but you will see in a window what the algorithm is doing. If the value is not set or set to 0, the database will be generated without video. If the value is set to any other number, the demo mode is activated.

The `.jpg` files are now saved with this structure:

```
Main\_Directory/Database/video\_name/LabelXFrameYFaceZ[Profile].jpg
```

- `Database`: main directory of the files.
- `video_name`: the name of the video where the current face was seen.
- `X`: The label of the face. The face detection algorithm follows the faces. If a person appears walking on several successive frames, the algorithm will know that each picture of the persons face corresponds to the same person and will save the images representing the same face on different frames with the same label.
- `Y`: the frame number
- `Z`: The current picture is the Z-th detected face of the Y-th frame.
- `Profile`: appears if and only if the detected face is in profile.

Note that the automatic labeling procedure:

- is not 100% accurate
- will not remember a person. If a person disappears from the screen and appears again later on the same video or on an other one, the algorithm will consider the face with a new label.

Consequently, the labels may have to be reviewed.

A `.txt` file is also provided. It is saved in:

```
Main\_Directory/Caffe\_Files/full\_database.txt
```

Each line represent a specific .jpg image of of this file has the following structure:

```
Main\_Directory/Database/videoM/LabelX[Profile]FrameYFaceZ.jpg B
```

`B` is automatically computed from `M` and `X` so that the label appearing in the file can be used directly by caffe.

5.2.2 How to manually improve the face detection?

Supposing that the automatic labeling was not efficient enough for your particular case -typically if some main faces appeared on several videos-, you may have to manually change the labels of the concerned images.

Lets assume you have to modify few labels to face this kind of issues. If you manually changed the name of some images so that the label of each file is the first character of the file name (ie for example `1Frame2.jpg` has label `1` and `4Frame23.jpg` has label `4`), then, the next commands will generate a new database file called `labelised_full_database.txt` in the directory `Caffe_File` with the required labels. Note that it works only with labels between 1 to 9. With a minor modification, more labels can be modified. Other few pictures with labels 1 to 9 can be manually modified directly in the file if needed.

First, lets regenerate `full_database.py` with the new names:

```
python src/python/fulldbfile.py
```

Then the script:

```
python src/python/labelisation.py
```

You simply have to copy paste that to continue the work:

```
mv Caffe\_Files/full\_database.txt Caffe\_Files/full\_database\_old.txt
mv Caffe\_Files/labelised\_full\_database.txt Caffe\_Files/full\_database.txt
```

5.2.3 To use the face verification network

Modify the database with this script for preprocessing:

```
python src/python/preprocessing\_face\_id.py
```

5.2.4 To test the newtork

We only use the `full_db.txt` file.
First we generate the pairs readable by matlab with:

```
python src/python/pairs\_generator.py
```

To generate the file `src/face_id/face_verification_experiment-master/code/mbk_pairs.mat`

Then:

```
python

src/face\_id/caffe\_ftr.py

src/face\_id/face\_verification\_experiment-
master/proto/LightenedCNN\_A\_deploy.prototxt

src/face\_id/face\_verification\_experiment-
master/model/LightenedCNN\_A.caffemodel

.

Caffe\_Files/full\_database.txt

prob

src/face\_id/face\_verification\_experiment-
master/results/mbk\_result\_CNN\_AA.mat
```

To generate the file
`src/face_id/face_verification_experiment-master/results/mbk_result_CNN_A.mat`

- In matlab, launch:

`face\_verification\_experiment-master/code/evaluation.m`

to evaluate the work.

5.2.5 To use the face verification script

```
python src/software/soft.py
```

You will be asked the address of the pictures of the face of the person you want to recognise. The result is the faces which have been recognised in the Database directory. They are saved in a directory called `Captured_Faces`.

5.2.6 To use the experimental script for face tracking with DLIB and then face verification

In `src/software`:
- Copy the images of the criminal you want to find in the video in `criminal_pictures/`
- Copy the video you want to find the criminal in in `video/`
- python script.py

The detected faces will be saved in `detected/`

5.2.7 Informations about the Siamese Network

The Siamese Network does not work properly. However, you can try to recreate the experience I made by following these intructions:

Generate Train and Validate caffe files :

```
chmod +x trainandval.sh;  ./trainandval.sh [number\_of\_files]
```

- number_of_files is an option. If it is set to 1, then only a `train.txt` file will be generated as a copy of `full_database.txt`. If it is set to 2, then `train.txt` and `test.txt` will be generated. These two files will contain the same number of lines, i.e. the same number of files (plus or minus one). Finally, if it is set to 3, it will contain `train.txt`, `test.txt`, and `valid.txt`. The three

files will have the same size.

Files generation from `train.txt` and `test.txt`:
For this usage, you should have done `./trainandval.sh 2` previously.

```
python src/python/create\_siamese\_db.py
```

will create the `train1.txt`, `train2.txt`, `test1.txt` and `test2.txt` files required by the siamese network.

Then :

```
for name in Database/*/*.jpg; do
convert -resize 256x256\! \$name \$name
done
```

The goal is to resize the images of the database. Modify the path `Database/*/*.jpg` if necessary.

```
./src/siamese/train\_siamese\_mbk.sh
```

for training and:

```
python src/siamese/analyse.py
```

for testing.

5.2.8 Other Scripts

`src/python/5pt_preprocess.py` does not work properly. Initially it was meant to generate the preprocessing files required for face_id to be used. Unhappily, the github page was in my opinion not documented enough and I did not have enough time to understand it. So I used an other more simple preprocessing technique, which seems to be working fine.

Bibliography

[1] Le An, Mehran Kafai, and Bir Bhanu. Face recognition in multi-camera surveillance videos using dynamic bayesian network. In *Sixth International Conference on Distributed Smart Cameras (ICDSC)*, pages 1–6. IEEE, 2012.

[2] G. Bradski. *Dr. Dobb's Journal of Software Tools*, 2000.

[3] Sumit Chopra, Raia Hadsell, and Yann LeCun. Learning a similarity metric discriminatively, with application to face verification. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR.*, volume 1, pages 539–546. IEEE, 2005.

[4] Ronan Collobert, Koray Kavukcuoglu, and Clément Farabet. Torch7: A matlab-like environment for machine learning. In *BigLearn, NIPS Workshop*, number EPFL-CONF-192376, 2011.

[5] Li Deng and Dong Yu. Deep learning: Methods and applications. *Foundations and Trends in Signal Processing*, 7(3–4):197–387, 2014.

[6] Gaurav Goswami, Romil Bhardwaj, Rajdeep Singh, and Mayank Vatsa. Mdlface: Memorability augmented deep learning for video face recognition. In *IEEE International Joint Conference on Biometrics (IJCB).*, pages 1–7. IEEE, 2014.

[7] Adam W Harley. An interactive node-link visualization of convolutional neural networks. In *ISVC*, pages 867–877, 2015.

[8] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. In *Proceedings of the ACM International Conference on Multimedia*, pages 675–678. ACM, 2014.

[9] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[10] Xiaoming Liu and Tsuhan Chen. Video-based face recognition using adaptive hidden markov models. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition. Proceedings.*, volume 1, pages I–340. IEEE, 2003.

[11] Frank Rosenblatt. Principles of neurodynamics. perceptrons and the theory of brain mechanisms. Technical report, DTIC Document, 1961.

[12] Florian Schroff, Dmitry Kalenichenko, and James Philbin. Facenet: A unified embedding for face recognition and clustering. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 815–823, 2015.

[13] Yaniv Taigman, Ming Yang, Marc'Aurelio Ranzato, and Lior Wolf. Deepface: Closing the gap to human-level performance in face verification. pages 1701–1708, 2014.

[14] Xiang Wu, Ran He, and Zhenan Sun. A lightened CNN for deep face representation. *CoRR*, abs/1511.02683, 2015.