PERSON ANNOTATION IN VIDEO SEQUENCES

A Master's Thesis
Submitted to the Faculty of the

Escola Tècnica Superior d'Enginyeria de Telecomunicació de Barcelona

Universitat Politècnica de Catalunya

By

Carolina Fernández-Pedraza Jorde

In partial fulfilment
of the requirements for the degree in

MASTER IN TELECOMMUNICATION'S ENGINEERING

Advisor:

Elisa Sayrol Clols, Josep Ramón Morros Rubió

May, 2019

Abstract

In the recent years, the demand for video tools to automatically annotate and classify large audiovisual datasets has increased considerably. One specific task in this field applies to TV broadcast videos, to determine who and when a person appears in a video sequence.

This work starts from the base of the ALBAYZIN evaluation series presented in the IberSPEECH-RTVE 2018 in Barcelona [1], and the purpose of this thesis is trying to improve the results obtained and compare the different face detection and tracking methods. We will evaluate the performance of classic face detection techniques and other techniques based on machine learning on a closed dataset of 34 known people. The rest of characters on the audiovisual document will be labelled as "unknown".

We will work with small videos and images of each known character to build his/her model and finally, evaluate the performance of the ALBAYZIN algorithm over a 2h video called "La noche en 24H" whose format is like a news program. We will analyze the results and the type of errors and scenarios we encountered as well as the solutions we propose for each of them if there is any. In this work, We will only focus on a monomodal basis of face recognition and tracking.

Resumen

En los últimos años, la demanda de herramientas de anotación automática de vídeo para clasificar bases de datos de gran volumen ha aumentado considerablemente. Una tarea específica en este campo recae sobre los vídeos televisivos, para saber quién y cuándo aparece alguien en una secuencia.

Este trabajo empieza con la base de ALBAYZIN evaluation series que se presentó en el IberSPEECH-RTVE 2018 en Barcelona [1], y el propósito de este TFM es intentar mejorar los resultados que se obtuvieron y comparar los distintos métodos. Se evaluará el rendimiento de métodos clásicos de detección de caras con otros basados en machine learning sobre un dataset cerrado del que conocemos a 34 personas. El resto las clasificamos como "unknown".

Trabajaremos con pequeños videos e imágenes de cada persona que conocemos para construir su modelo y finalmente, evaluar el rendimiento del algoritmo de ALBAYZIN sobre un vídeo de 2h titulado "La noche en 24H" cuyo formato es como un telediario. Analizaremos los resultados y los distintos tipos de errores y escenarios que nos hemos encontrado así como una solución que hemos propuesto para cada uno de ellos si es que existe. En este trabajo nos centraremos solo en la parte monomodal que trabaja con el reconocimiento de caras y el tracking.

ACKNOWLEDGEMENTS

Lo primero, quiero dar las gracias a mis tutores Ramón y Elisa por toda la ayuda prestada y por tanta paciencia. Gracias a vosotros he conseguido terminar este trabajo y este máster. A Albert, por su amabilidad y tener siempre tener la solución que necesitaba.

Gracias a mi familia elegida Claudia, Ana, Fede, Javi, Alberto y Carlos por estar ahí desde que tengo uso de razón.

Gracias a Marcela, mi amiga, mi hermana mayor. Gracias por tu amor y confianza en mí.

Gracias a mis compañeros de este máster, que son muchos, por poner su granito de arena para ayudarme a que todo esto saliese adelante.

Y por último, a mis padres por esforzarse siempre en dejarme la mejor herencia que una hija puede tener: la educación.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

NOMENCLATURE

Adam	Adaptive Moment Estimation
AFLW	Annotated Facial Landmarks in the Wild
CNN	Convolutional Neural Network
DCT	Discrete Cosine Transform
DER	Diarization Error Rate
FDDB	Face Detection Data set and Benchmark
FEREt	Facial Recognition Technology
FOV	Field Of View
FV	Feature Vector
GPU	Graphics Processing Unit
HOG	Histogran of Oriented Gradients
ImageNet	Image Network
LDML	Local Data Markup Language
LFW	Labeled Face in the Wild
MHT	Multiple Hypothesis Tracking
MMOD	Maximum-Margin Object Detector
MSRA	Microsoft Research Open Data
Pascal VOC	Pascal Visual Object Classes
PCA	Principal Component Analysis
ReLU	Rectified Linear Unit
ResNet	Residual Network

RMSProp	Root Mean Squared Propagation
SVM	Support Vector Machine
VGG	Visual Geometry Group
WIDER	Web Image Dataset for Event Recognition
YTF	Youtube Faces

CHAPTER 1

INTRODUCTION

Nowadays, the vast amount of video data stored in web archives makes their retrieval based on manual text annotations impractical. We need well-established methods for searching, navigating, and retrieving broadcast quality video content rely on transcripts obtained from manual annotating, close captioning and/or speech recognition [7]. For those tasks to be done, we need a selection of indexes derived from the content of the video to help organize video data and metadata that represents the original video stream.

Automated indexing and search technology unlocks the visual intelligence contained in large media datasets, providing an effective way to organize and search visual content.

In this thesis we are going to explore several techniques of face recognition and tracking to accomplish the challenge of video indexing working only with the image data (speech information will not be taken into account). Concretely, in this thesis we are going to put into contest this task with the Albayzin evaluation series, that is part of the IberSpeech-RTVE Multimodal Diarization challenge. This is supported by the Spanish Thematic Network on Speech Technology (RTTH) and Cátedra RTVE en la Universidad de Zaragoza and is organized by Vivolab-Universidad de Zaragoza.

The diarization evaluation consists of segmenting broadcast audiovisual documents according to a closed set of different faces and linking those segments which originate from the same face. For this evaluation, a list of characters to recognize are given. The rest of characters on the audiovisual document will be discarded for the evaluation purposes. System outputs must give for each segment who is/are in the image from the list of characters. For each character, a set of face pictures and short audiovisual document are given.

The goal of this Master's Thesis is to continue with the challenge of Albayzin evaluations and improve the results of the monomodal performance.

1.1 Objectives

We first need to define our objectives and what we expect to achieve in this thesis.

1. *Improve the face detection field exploring different options based on HOG+SVM and Deep Learning.*
2. *Improve the tracking method exploring different options based on optical flow and a correlation tracker.*
3. *With those best technique, try to improve the results obtained by the UPC in this edition of the Albayzin evaluation series.*
4. *Analyze the limitations of Albayzin and the challenges we cannot solve with image processing.*

1.2 Planning

The project breakdown structure has 8 main work packages.

WP 1	State of the Art Research
Description	Do research on the current state of art for video annotation.
Tasks	T1. Read articles on suggested alternatives to face recognition and feature extraction.
Dates	Start: 17/09/2018 End: 08/10/2018

Table 1.1: Work package 1.

WP 2	Get base network
Description	Choose a network that will serve as the base of our project.
Tasks	T1. Choose a pretrained network for our data and download it. T2. Train and finetune it if necessary.
Dates	Start: 08/10/2018 End: 22/10/2018

Table 1.2: Work package 2.

WP 3	Implemet my network
Description	Build a siamese and triplet architecture.
Tasks	T1. Implement the code for both architectures with the previous pre-trained network given. T2. Divide the MSRA dataset into train and validation. T3. Make trials to get the best accuracy possible.
Dates	Start: 22/10/2018 End: 26/01/2019

Table 1.3: Work package 3.

WP 4	First approach to Albayzin project
Description	Understand the existing UPC system of Albayzin.
Tasks	T1. Read the papers and understand the system. T2. Try to replicate the results obtained by the authors.
Dates	Start: 26/01/2019 End: 15/02/2019

Table 1.4: Work package 4.

WP 5	First trial to improve the results
Description	Explore a CNN network as a face recognition tool.
Tasks	T1. Download the pretrained CNN network. T2. Adapt the code to the new face recognition method. T3. Analyze the errors and try to combine the parameters in an optimal way. T4. Compare with the classic face recognition method.
Dates	Start: 18/02/2019 End: 25/03/2019

Table 1.5: Work package 5.

WP 6	Second trial to improve the results
Description	Explore a new tracking method.
Tasks	T1. Download the suggested tracking method by DLIB. T2. Understand the method reading the article in pyimagesearch. T3. Explore and understand the different key parameters that could affect the overall error of the system. T4. Compare with the previous methods.
Dates	Start: 25/03/2019 End: 15/04/2019

Table 1.6: Work package 6.

WP 7	Analyze the limitations of Albayzin
Description	Explore the different scenarios in the video where there are errors.
Tasks	T1. Analyze the results that gives us the scoring script of the different types of errors. T2. Look in the video what happens in that moments and if there is any solution. T3. Compare with the previous tracking method and extract conclusions.
Dates	Start: 15/04/2019 End: 15/05/2019

Table 1.7: Work package 7.

WP 8	Write the thesis
Description	Organize the sections and start writing this thesis.
Tasks	T1. Organize the contents. T2. While I continue working, start writing the state of the art and methodology. T3. Analyze and write the results I have obtained during these months and write the conclusions and future work to continue improving.
Dates	Start: 21/02/2019 End: 15/05/2019

Table 1.8: Work package 8.

Given the work package structure explained, the following Gantt Diagram is generated:

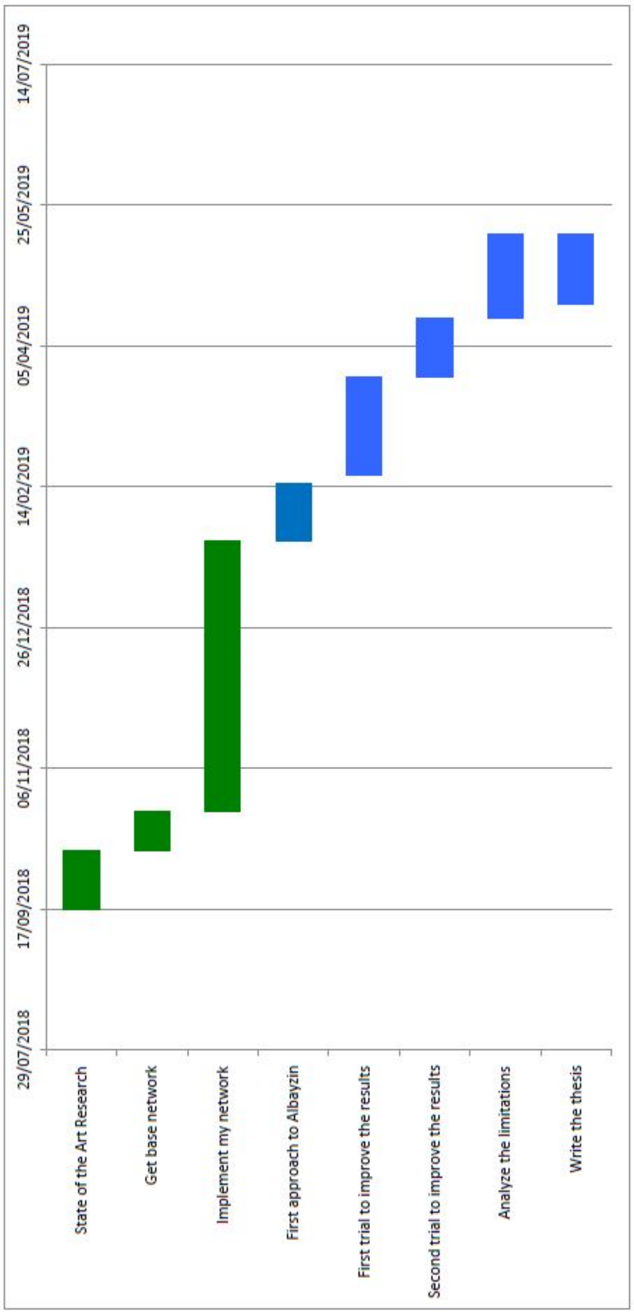


Figure 1.1: Gantt diagram of the project.

1.3 Deviations from the original work plan

Initially, when I enrolled the Master's Thesis last summer, we were planning to work on face verification using multimodal learning, but once the course started we decided to change the topic to the ALBAYZIN challenge, specifically face recognition, because it was recently presented from UPC and we considered interesting to explore other face detection and tracking techniques to improve the results presented in this edition of the IberSpeech-RTVE 2018.

Also, at the beginning, I spent several weeks in developing and training my own triplet network as a technique to improve the performance of the face recognition task the Albayzin results presented by the UPC. However, we did not get the results we expected and we decided to start using pre-trained face recognition networks.

CHAPTER 2

STATE OF THE ART

In these last years, the research on video annotation has advanced a lot. To deal with this task, two main question arise: "Who appears in the video?" and "Who is speaking in the video?". In this Master's Thesis we are going to focus on the first question "Who appears in the video?".

This chapter will give an introduction to the techniques of face detection, face recognition and face tracking for automatic video annotation systems without taking into account speaker information (monomodal analysis), that later on we are going to apply to the Albayzin system.

2.1 Technical background about CNN's

First of all, it is needed to give a brief background about CNN's because some methods we are going to use and analyze later on, are based on that techniques.

Convolutional neural networks (CNN's) are a specific type of feed-forward neural networks. They are made up of neurons with learnable biases and weights, followed by a non-linearity. A neuron is the smallest unit of information in a neural network, The task during training is to learn the weights of the connections between neurons. Their inputs are meant to be images, which allows us to make certain assumptions to constrain the architecture, reducing considerably the number of parameters in comparison to a regular neural network.

The simplest type of artificial neural network, called multilayer perceptron, is composed of at least three layers, each layer containing one or more neurons. The first layer serves as input for the neural network; one or more intermediate layers called hidden layers, which have an arbitrary number of neurons, allow the neural network to approximate complex functions; and the output layer, which has the same dimension as the number of classes in the classification problem, carries out the predictions. In a multilayer perceptron, each neuron is fully connected to all neurons in the previous layer, and neurons in a single layer function are completely independently between them and do not share any connections. The last fully-connected layer is called the "output layer" and in classification settings it represents the class scores.

The convolutional layer is the core building block of a Convolutional Network (see Figure 2.1) that does most of the computational heavy lifting.

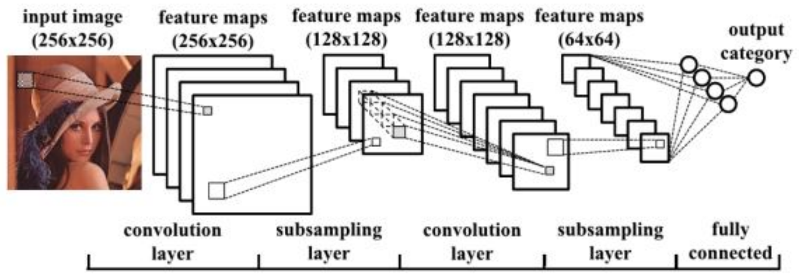


Figure 2.1: Scheme of a basic Convolutional neural network with an input layer, two convolutional layers and a fully connected layer. The relations between the inputs and where each filter is looking at can be seen. Image from [2].

The Convolutional Layer makes use of a set of learnable filters. A filter is used to detect the presence of specific features or patterns present in the original image (input). It is usually expressed as a matrix ($M \times M \times 3$), with a smaller dimension, but the same depth as the input file. This filter is convolved (slided) across the width and height of the input file, and a dot product is computed to give an activation map.

The activation function is the non-linear transformation that we apply after each layer over the input signal. This transformed output is then sent to the next layer of neurons as input, which in latest CNN's tends to be a Rectified Linear Unit or ReLU. The ReLU function basically computes the activation thresholded at zero: *f(x)* = *max(0,x)*. It converts all negative inputs to zero and the neuron does not get activated. It considerably accelerates the convergence of stochastic gradient descent and it does not involve expensive operations like other activation functions such as sigmoid or tanh. One of the main problems about ReLU units is that they can be very fragile and can decay during training, which means that they could end up being in an state in which they are inactive for each input, disabling any backward gradient flow, which is usually due to a learning rate set too high.

Another usual technique to reduce the number of parameters is adding an extra layer after every convolution layer to reduce the spatial size of the output volumes of them and also to capture global and larger features. These layers are called Pooling layers. Nowadays the Max pooling operations are the most widely used. Max-pooling, like the name states; will take out only the maximum from a pool. This is actually done with the use of filters sliding through the input; and at every stride, the maximum parameter is taken out and the rest is dropped (see Figure 2.2). This actually down-samples the network. Unlike the convolution layer, the pooling layer does not alter the depth of the network, the depth dimension remains unchanged.

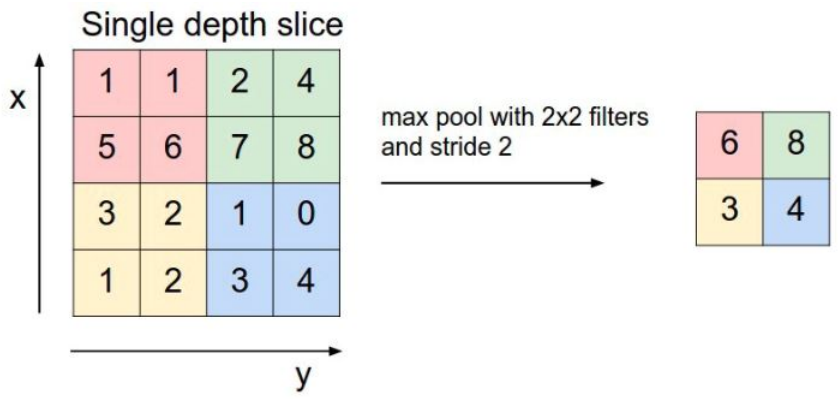


Figure 2.2: Example of how applying max pooling affects to the spatial size reduction. Image from [3].

Then, the resulting feature map is flattened in order to connect it to some FC layers which will take care of the final decision or classification when adding a Softmax, or will give us a feature vector that characterizes the input.

When the CNN architecture is defined, it has to be trained. This is done inserting train images, computing a loss at the output and then updating the weights and biases in order to minimize the chosen loss. Usually this is done through a backward pass called backpropagation. Besides these parameters, which are learnable, every neural network structure has also what is called hyperparameters, non-trainable and which values are up to the network designer.

Training a neural network is the process of finding the set of parameters so that the network's outputs closely match the known outputs found in the training dataset. Once this set of parameters is computed, the model can be used to classify new unseen data. Usually, the parameters of a neural network are optimized iteratively using mini-batch gradient descent, RMSProp (Root Mean Squared Propagation) or Adam (Adaptive Moment Estimation) are the most typical. During an iteration, a subset of the training data is forwarded through the network (the output of the network is computed for this subset of training data). Then, the loss value is obtained, which is a measure of divergence between the output values predicted by the network and the ground truth values provided in the training dataset. Then, in the backward pass, the derivative of the loss value as a function of the parameters of the neural network is computed and the parameters are slightly in the direction which minimizes the loss. This process is carried out until the loss value has converged to a minimum or the neural network achieves a satisfying accuracy in an independent validation dataset. A full pass through the whole training dataset is called an epoch.

2.2 Automatic video annotation

Before neural networks were used, automatic annotation in video sequences was a task that required many complex subsystems working together in a very specific and controlled framework.

Some of the most famous systems were the presented by Everingham, M. et al. [8], where frontal face detection and lip movement detection were used along with the video scripts to annotate the appearing and talking persons.

In [9], by Bredin, H. et al., the structure of the presented system consists of that each monomodal component is processed separately (speaker diarization plus speech-to-text, face detection, tracking and written names on screen) and then a decision-level fusion is performed. The thing is that any of these stages are performed using deep learning (HOG [10]/LDML [11] or DCT/SVM methods are used for face recognition instead).

The focus of this project is on the UPC 2016 MediaEval face verification and recognition part of the whole system, which we aim to improve. Sometimes the terms verification and recognition are used equally, but that is not correct, because these terms solve different problems. A recognition system answers the following question: Who is this person? This is achieved by comparing a query face to all our available models, and deciding to which one it is closer and belongs, which is called 1-to-N matching. Otherwise, a verification system aims to answer "Is this person that specific person?" It gives a boolean answer to the comparison between the query face and one model face at each time (1-to-1 matching).

2.3 Face detection

The existing strategies for face detection can be categorized in several groups, such as knowledge-based methods, feature invariant approaches, face template matching, and appearance based methods [12]. For example, Hsu et al. [13] proposed a color feature based approach that searches face candidates based on the distribution of face

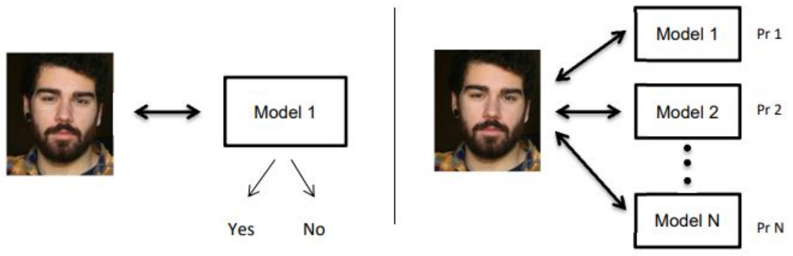


Figure 2.3: Examples of a verification system 1-to-1 (left) and a recognition system 1-to-N (right) procedure [2]

colours in YCbCr space; Lanitis et al. [14] described a face by a set of shape model parameters and matches a query image with flexible appearance models.

In this thesis we explain and analyse the performance two face detection techniques, the first one is made using the now classic Histogram of Oriented Gradients (HOG) feature combined with a linear SVM classifier, an image pyramid, and sliding window detection scheme from Dlib [15]. The second one is based on deep learning with a CNN trained as a triplet network provided by Dlib too [16].

2.3.1 HOG detector + SVM linear classifier

Dlib provides in [15] a pre-trained model that consist of using the now classic Histogram of Oriented Gradients (HOG) feature combined with a SVM (Support Vector Machines) linear classifier, an image pyramid and a sliding window detection scheme. Using both a sliding window and an image pyramid we are able to detect objects in images at various scales and locations.

In broad terms, the HOG algorithm will check for every pixel all the pixels around it and figures out how dark the current pixel is compared with the directly pixels surrounding it. Then it draws an arrow showing in which direction is getting darker and it breaks down in 16x16 pixels squares, where it counts up how many gradients point in each major direction, then that square is replaced with the arrow direction that was the strongest. The whole process of HOG detection is shown in Figure 2.4.

At each window we extract HOG descriptors and apply the pre-trained SVM classifier to recognise the face. If the classifier detects a face with sufficiently large probability, it will store the bounding box of the window. After we have finished scanning the image with the sliding windows, we apply non-maximum suppression to remove redundant and overlapping bounding boxes.

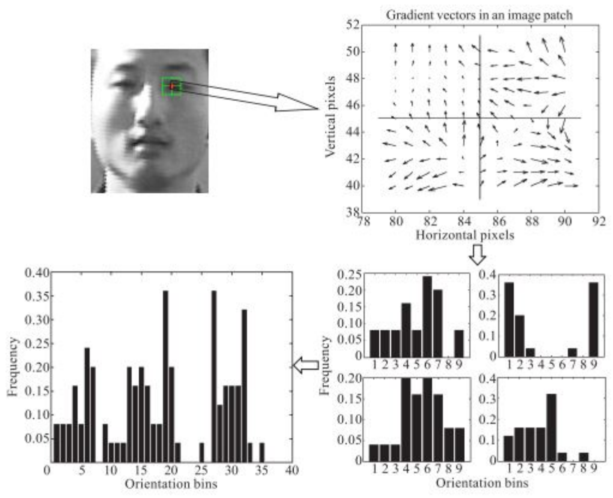


Figure 2.4: Image window divided into small spatial regions (cells). Local 1D histograms of gradient directions or edge orientations are accumulated and concatenated to form the final histogram feature. Image from [4].

2.3.2 Applying Deep Learning (CNN)

There is a CNN face detector available in [16] by Dlib that we want to use in order to detect faces in odd angles as we have so many in our test video. It has been trained as a triplet network on a dataset created by Davis King [17] by finding face images in many publicly available image datasets (excluding the FDDB dataset). In particular, there are images from ImageNet, AFLW, Pascal VOC, the VGG Face dataset, WIDER, and face scrub.

This method uses a Maximum-Margin Object Detector (MMOD) with CNN-based features. This method does not perform any sub-sampling, but instead optimizes over all sub-windows. It improves the normal object detection by replacing the non-maximum suppression with an optimizer that runs over all windows and optimizes the performance of an object detection system in terms of the number of missed detections and false alarms in the final system output. The CNN is going to be evaluated convolutionally over an entire image pyramid like a normal sliding window classifier. This means we need to define a CNN that can look at some part of an image and decide if it is an object of interest. In this case, the CNN architecture we use is defined as a CNN with a receptive field of a little over 50x50 pixels. This is reasonable for face detection since you can clearly tell if a 50x50 image contains a face.

In this example our CNN begins with 3 downsampling layers. These layers will reduce the size of the image by x8 and output a feature map with 32 dimensions, also we use Relu and batch normalization in the standard way. Then, we will pass that through 4 more convolutional layers to get the final output of the network. All of those convolutional layer are 3x3 with batch normalization and Relu. The last layer has only 1 channel and the values in that last channel are large when the network thinks it has found an object at a particular location.

This CNN model is much more accurate than the HOG based model, but takes much more computational power to run, and is meant to be executed on a GPU to attain reasonable speed.

2.4 Face recognition

Face recognition is a K class problem where K is the number of known individuals. This task present several challenges, the most common ones are:

- Sideways faces that can reduce the accuracy of the system over a 10%.
- The effect of the aging process over the time is very challenging due to the ammount of changes that occur.
- Low-Resolution Face Recognition.
- Low-Shot Face Recognition. It happens when we have limited number of shots and with blur, occlusion and different expresions.
- Illumination changes.

SVM's are a binary classification method (one of the classification methods we are going to use later on). In [18] P. J. Phillips developed a SVM-based face recognition algorithm. The face recognition problem is formulated as a problem in difference space, which models dissimilarities between two facial images. They generated a similarity metric between faces that is learned from examples of differences between faces. The SVM-based algorithm is compared with a principal component analysis (PCA) based algorithm on a difficult set of images from the FEREt database. With a 22% of error for SVM and 46% for PCA, [18] indicates that SVM is making more efficient use of the information in face space than the baseline PCA algorithm for face identification.

However, nowadays there are more modern techniques all based on CNN's and Deep Learning that are the most used methods for face recognition. In every type of classification task we need some feature vectors to work with, so we need a feature extraction procedure. CNN's for regular face recognition work as follows: We input a query image to the neural network and as output we obtain the probabilities to pertain to every available class. As a result, different types of loss functions and architectures to train this CNN's are studied in order to obtain the best feature vectors for face recognition and verification. In the next subsection we are going to explain some examples of face recognition based on CNN's that has been used recently and their results.

2.4.1 CNN's for face classification

In 2014, Sun Y.et al also developed a different approach would be the DeepID architecture, presented by Sun Y.et al. [19], where 60 independent CNN's are trained with different patches of the same picture in order to obtain different high-level features, which are extracted from the last FC layers and then linearly combined.

In the same year 2014, Sun Y.et al again developed in [20] a deep learning face representation by joint identification and verification. The Deep IDentification-verification features (DeepID2) are learned with carefully designed deep convolutional networks. The face identification task increases the inter-personal variations by drawing DeepID2 extracted from different identities apart, while the face verification task reduces the intra-personal variations by pulling DeepID2 extracted from the same identity together, both of which are essential to face recognition. They worked with the LFW (Labeled Face in the Wild) dataset [21] where they got a 99.15% of accuracy.

One year later, in 2015, Florian Schroff et al. [22] presented a system called FaceNet (Google), that directly learns a mapping from face images to a compact Euclidean space where distances directly correspond to a measure of face similarity. Once this space has been produced, tasks such as face recognition, verification and clustering can be easily implemented using standard techniques with FaceNet embeddings as feature vectors. Their method uses a deep convolutional network trained to directly optimize the embedding itself, rather than an intermediate bottleneck layer as in previous deep learning approaches. The trained with triplets using only 128 bytes per face and they got an 99.63% of accuracy on the LFW dataset and a 95.12% with Youtube Faces (YTF) dataset.

Also in 2017 Liu W. et al. [23] proposed the angular softmax (A-Softmax) loss that enables CNNs (up to 64 layers with residual units) to learn angularly discriminative features, because the classical Softmax was not discriminative enough. Geometrically, A-Softmax loss can be viewed as imposing discriminative constraints on a hypersphere manifold, which intrinsically matches the prior that faces also lie on a manifold. Moreover, the size of angular margin can be quantitatively adjusted by a parameter m. They further derive specific m to approximate the ideal feature criterion. Extensive analysis and experiments on LFW and YTF show the superiority of A-Softmax loss in FR tasks with a 99.42% and 95.0% of accuracy respectively.

In this Master's Thesis, the task of face recognition is performed by several verifications (one for each model) because of the problem of the "Unknowns" and that we work with a closed dataset. There will be a threshold to decide for each verification if the detected face is similar enough to the corresponding model or, on the contrary, it is "Unknown".

The state of the art verification systems now implement two types of neural network architectures in order to achieve the best distance transformation, which are the following:

2.4.2 Siamese network

Siamese nets were first introduced in the early 1990s by Bromley and LeCun to solve signature verification as an image matching problem [24]. A siamese neural network consists of twin networks which accept distinct inputs but are joined by an energy function at the top, usually the Euclidean distance. The architecture of the twin networks are identical, but the weights have to be shared among them as well for the network to be called *siamese* (see Figure 2.5). The main idea behind siamese networks is that they can learn useful data descriptors that can be further used to compare between the inputs of the respective networks.

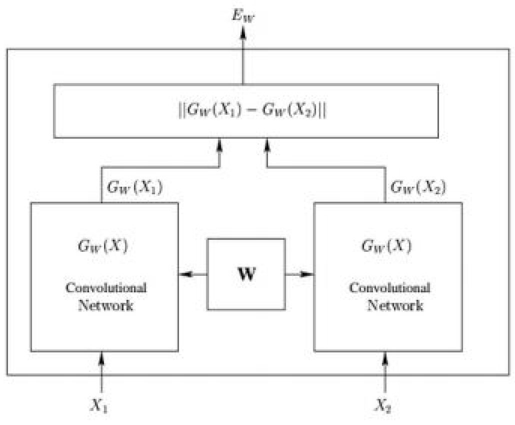


Figure 2.5: Scheme of a Siamese neural network architecture. Image from [2].

Also, the network is symmetric, so that whenever we present two different images to the twin networks, the top conjoining layer will compute the same metric as if we introduce the same images but interchanged to the other twin.

Using this sort of neural network architecture trained with different pairs of images we can obtain feature vectors which, with a proper distance and loss functions, achieve an interesting property: Given two similar input images, its feature vectors extracted at the end of the neural network will be closer in the output Euclidean space than any non-similar sample. A clear example of this is the method submitted by [25], which developed a Siamese neural network that minimizes the intra-class distance relating to a provided minimum threshold and augments the inter-class distance, making the architecture highly suitable for verification tasks.

As previously discussed, one of the main hyperparameters when training the neuronal network is the loss function as long as it is, apart from other factors, directly related to the weights and biases final values. In Siamese architectures the most extended loss function is the submitted by [26], called *contrastive loss function.*

Being D_w the output of the Siamese neural network related to the distance between a pair of labelled samples x_1 and x_2 from some pair corpus P and W the shared weight parameter,

$$D_w = \left\| W\frac{\phi(x_1)}{\|\phi(x_1)\|_2} - W\frac{\phi(x_2)}{\|\phi(x_2)\|_2} \right\|_2 \tag{2.1}$$

and Y=1 if both inputs represent the same identity and Y = 0 in the opposite case (different identities), the *contrastive loss function* is defined as:

$$L = \sum_{x_1, x_2 \epsilon P} (1-Y)\frac{1}{2}(D_w)^2 + Y\frac{1}{2}(max(0, \alpha - D_w)))^2 \tag{2.2}$$

Where α>0 is a margin that defines a radius around the outputs of each single network from the Siamese architecture. The performance that this loss function presents in a facial verification framework is presented in [27] by the same team, restating the utility of this metric learning architecture.

2.4.3 Triplet network

A Triplet network (inspired by "Siamese network") consists on 3 identical feed-forward networks with shared parameters. We feed the triplet network with 3 samples that we denote as x (anchor), x^+ (positive, same class as anchor) and x^- (negative, different class of anchor), and the embedded representation of the network as *Net(x)*, the one before last layer will be the vector:

$$TripletNet(x, x^+, x^-) = \begin{bmatrix} \|Net(x)) - Net(x^-)\|_2 \\ \|Net(x)) - Net(x^+)\|_2 \end{bmatrix} \epsilon\mathbb{R}^2_+ \tag{2.3}$$

The network outputs 2 intermediate values: the L2 distances between anchor-positive and between anchor-negative images.

Taking all the previous information into account, a general structure of a triplet would be one as in Figure 2.6. The *Comparator* block in Figure 2.6 allocates the triplet-loss function that will backpropagate through the triplet network and it will carry out the process of learning. The whole process of how a triplet network learns is ilustrated in Figure 2.7.

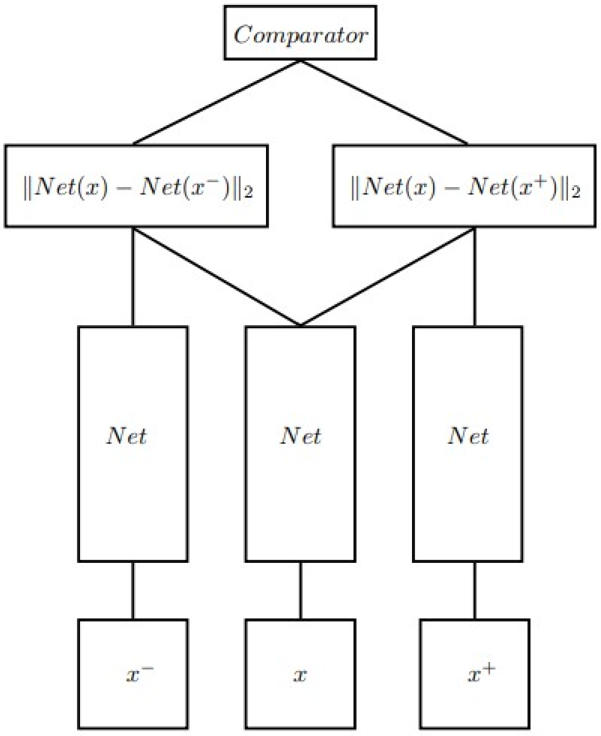


Figure 2.6: Architecture of a generic triplet-loss convolutional neural network, with its convolutional and FC layers sharing its parameters.

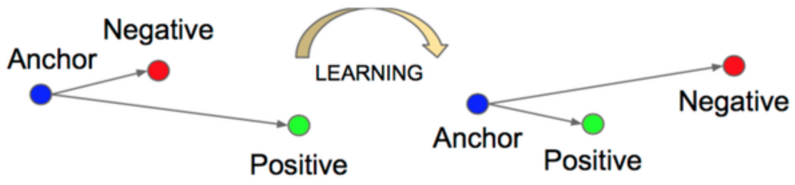


Figure 2.7: Learning process of a triplet network.

It tries to bring close the anchor with the positive as far as possible from the negative. The formula to calculate the triplet-loss is:

$$Loss = \sum_{i=1}^{N} \left[\|f_i^a - f_i^p\|_2^2 - \|f_i^a - f_i^n\|_2^2 + \alpha \right]_+ \tag{2.4}$$

A system that implements this kind of CNN architecture is FaceNet [22] from Google's Schroff, Kalenichenko and Philbin. Its network is based on GoogLeNet Inception models [28] disposed in a triplet-loss way, achieving a 99,63% accuracy in LFW database and a 95,12% with YTF. These results reduced the error rate from the previous state of the art results by 30%.

Another well-known network is the submitted by Parkhi, M et. al [29], from the Visual Geometry Group of the University of Oxford. First its network is trained for face classification and then finetuned for verification using the triplet-loss function, achieving accuracy partially similar to FaceNet and DeepFace in LFW dataset, using less data and simpler network architecture.

2.4.3.1 Hard Negative Mining

There is a very efficient method for selecting the triplets that enter the network. Doing it randomly (respecting that the positive image belongs to the same class as the anchor and the negative image to a different class) is not enough, the problem of this random method is that the number of possible triplets grows quadratically or cubically with the number of examples. It is infeasible to process them all and the training converges slowly and does not learn optimally. There is a solution for this problem called *Hard Negative Mining.*

As long as the negative value is further than the positive value + alpha there will be no gain for the algorithm to condense the positive and the anchor. Let's pretend that:

- $\alpha = 0.2$
- Negative distance = 2.4
- Positive distance = 1.2

The loss function result will be *1.2-2.4+0.2 = -1.* Then when we look at *Max(-1,0)* we end up with 0 as a loss. The Positive Distance could be anywhere above 1 and the loss would be the same. With this reality, it is going to be very hard for the algorithm to reduce the distance between the Anchor and the Positive value. It exists one method to select the triplets that is called *Hard Negative Mining.*

It consists on forcing the selection of the negative image closer to the anchor than the positive. *Hard Negative Mining* solves this problem by performing hard example selection batch-wise. Given a batch sized K, it performs regular forward propagation and computes per instance losses. Then, it finds $M<K$ hard examples in the batch with high loss values and it only back-propagates the loss computed over the selected instances.

2.5 Face tracking

Face tracking is a computer technique whose goal is to keep the trace of similar detected faces within a video sequence. This is not an easy task because the faces may vary its pose and appearance along the time, which means that what is being track will not look the same in every frame. These changes are the reason why face tracking can not be understood as a "continuous face detection".

2.5.1 Challenges in face tracking

Due to face variability, face tracking is a very challenging problem. This section briefly describes the most important challenges that makes tracking a face a hard problem.

- **Illumination changes**: Illumination variations often lead to a lost of track. For example, if the camera is situated inside a car trying to track the driver's head, and suddenly he/she drives into a tunnel, the illumination conditions would change a lot.
- **Pose variations**: In some situations the face that is being tracked is not frontal. These pose changes make the work more challenging; if the face movements are just frontal, applying an affine transformation based tracker can be enough to track the face. However, this technique does not work when pose variation occurs. For example, when the subject turns his face to the right, the whole pattern that is being tracked changes, hence, the tracker has to deal with it and follow a new pattern.
- **Face deformation**: Tracking faces through changes of expressions is another challenging problem.

- **Occlusion and clutter**: As with most tracking problems, occlusion and clutter affect the performance of face trackers. A simple approach to recover from a loss of track is to compute the mean face that has been tracked so far. With this technique, once the occlusion disappears the tracker would be able to find again the face using the mean face.

2.5.2 Some tracking algorithms

In this section we are going to summarise some face tracking methods that exists right now.

In [30] they explain that particle filters are used for tracking algorithms that can resolve non-linear and non-Gaussian problems. The particulate filter algorithm originally designed for problems of signal processing, it was extended in computer vision under the "condensation" algorithm name. The idea is to apply a recursive Bayesian filter to several hypothetical locations of the face, and merge these assumptions based on their likelihood conditional on the predicted state. These are sequential simulation methods Monte Carlo type for estimation of the state vector is not necessarily a Markov linear systems subject to random excitations possibly non-Gaussian. This generic formalism eliminates any restrictive assumption about the probability distributions come into play in the characterization of the problem.

Also in [30] they talk about that better tracking outcomes can be acquired if the motion correspondence choice is overdue until several frames have been observed. The Multiple Hypothesis Tracking (MHT) algorithm upholds several correspondences suggestions for each object at each time frame. The final track of the object is the most likely set of correspondences over the time period of its observation. MHT according to [31] is an iterative algorithm. Iteration begins with a set of existing track hypotheses. Each hypothesis is a crew of disconnect tracks. For each hypothesis, a prediction of object's position in the succeeding frame is made. The predictions are then compared by calculating a distance measure. MHT is capable of dealing tracking multiple object, ability to tracks for objects entering and exit of Field Of View (FOV) and Calculating of Optimal solutions [32].

In 2015 Lijun Wang, Wanli Ouyang, Xiaogang Wang and Huchuan Lu proposed in [33] a new approach for general object tracking with fully convolutional neural network. Instead of treating CNN's as a black-box feature extractor, they conducted in-depth study on the properties of CNN features offline pre-trained on massive image data and classification task on ImageNet. In this work they identified feature maps in conv5-3 as weak object detectors, but they were not discriminative enough between instances of the same class. On the other hand, feature maps from conv4-3 are more sensitive to intra-class appearance variation.

Later, in 2017 Swapnil Vitthal Tathe, Abhilasha Sandipan Narote and Sandipan Pralhad Narote presented in [34] a face tracking framework that is capable of face detection using Haar fea-tures, recognition using Gabor feature extraction, matching using correlation score and tracking using Kalman filter. The method has good recognition rate for real-life videos and robust performance to changes due to illumination, environmental factors, scale, pose and orientations.

In this Master's Thesis we are going to work with two tracking methods, one is an optical flow method based on Lucas-Kanade and the other one is based on the correlation tracker. Both methods will be described more in detail in the Methodology chapter. We use these correlation trackers because in our video, faces do not move much and we do not need very complex tracking algorithms, they are relatively simple and achieve their goals.

CHAPTER 3

METHODOLOGY

In this chapter we are going to explain (1) the database we have used for Albayzin and how it is structured and (2) the detailed process of Albayzin system including both methods of face detection and tracking.

3.1 Albayzin database

The RTVE2018 database donated by RTVE and labeled thanks to the *Spanish Thematic Network on Speech Technology* (RTTH) and *Cátedra RTVE de la Universidad de Zaragoza*, will be used to evaluate multimodal diarization systems. RTVE20184 database has been divided into 4 partitions, a train one, two development partitions *dev1*, *dev2* and finally a test partition.

Detailed information about the RTVE2018 database content can be found in the RTVE2018 database description report [35]. **Only part of the *dev2* and text partitions will be used in this challenge. For this Master's Thesis, speech information will not be taken into account**.

3.1.1 Enrollment

Enrollment files are needed to detect the face of each of the 34 known characters that we take into account for this challenge. Each character has several jpeg pictures of him/herself and a short mp4 video in which only appears the corresponding character. With this information we will build a model for each character that later on it will be used for the verification task against the equivalent results for the test video.

3.1.2 Development

For development, *dev2* partition contains a 2 hours show "La noche en 24H" labeled with speaker and face timestamps. It is a talk show that contains interviews with some of the protagonists of the day.

3.2 Albayzin system

In this Master's Thesis we focus on the face recognition part of the Albayzin challenge. We want to evaluate only the use of visual information for face diarization.

3.2.1 Overall view

The UPC video system consists on several steps that are ilustrated in Figure 3.1.

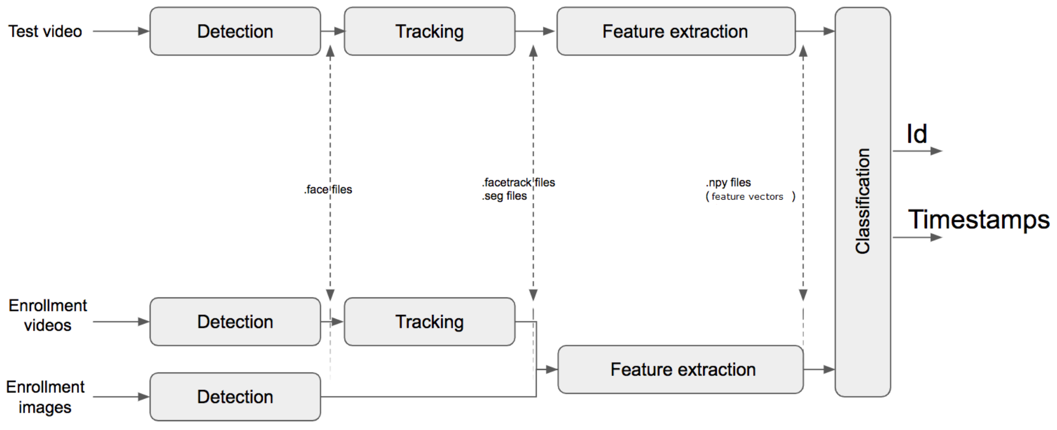


Figure 3.1: General overview of the Albayzin UPC video system.

There exist a pre-procesing that will be explained in the next section where we perform an indexing step for each video and we also define the shot limits, but now we will give an overall view of the most important steps.

The first step is face detection. This task is performed for enrollment data as well as for the test video. The result file contains the coordinates of all the bounding boxes that allocate a face. The output is a text file where each line corresponds to a detected face. Each line has the following format: *frame_number tlx tly width height confidence pose*, where *frame_number* is the frame where a face has been detected, *t1x* and *t1y* are the coordinates of the corner of the bounding box (corner up and left) and *width* and *height* give us the size of the bounding box. An example would be *273 345 139 186 186 4 1* that means that in the frame 273 we have detected a face marked with a bounding box (rectangle) that starts in the coordinates (345,139), it has a width of 186 pixels and a height of 186 pixels.

The second step is face tracking, but it will be only performed for enrollment videos and the test video because with the enrollment images it has no sense as there is no motion to track. The output consists of two text files: .facetrack and .seg. The facetrack file associates each face detection with a track ID. Each line corresponds to a detection and contains the following information: *frame_ number trackID tlx tly width height* (example: *282 1 345 139 186 186*). The seg file contains one line per track. Each line informs about the track start and stop, both in terms of frame number and timestamp. The info in each line is: *videoID timestamp_ ini timestamp_ end frame_ ini frame_ end track_ count trackID num_frames* (example: *LN24H-20151125 000015.716 000022.116 000392 000552 000003 000121 161*).

The third step is feature extraction where with all the information extracted before, we will build the model for each character with the all enrollment data (one FV for each image and one FV for the enrollment video averaging all FV's extracted). Furthermore, with the test video we will just store all the FV's that we found. For each track, we extract all the faces and we store just one FV that is the average of all the FV's extracted in each track and will characterize the person that appears in that moment.

In the fourth step, we perform the recognition step with several verifications comparing each FV of the test video with each model of our known characters. The identity of the person detected in the test video will belong to the identity which the enrollment FV is the most similar. The output consists of a text file with the following information: *video_ ID time_ ini duration frame_ ini num_frames id track_ ID*, but only the *id*, *time_ ini* and *duration* will be the most relevant.

Finally, we execute the existing scoring script that will give as the overall error and all the details of what has happened with the automatic annotation process.

In the following sections we will explain more in detail each of the steps.

3.2.2 Pre-processing

There are two important steps to perform before starting to detect any face:

1. **Indexing**: We need to create an index file (.idx) for each video, containing the relationship between timestamps and frame numbers. In this case we do this with FFMPEG [36].

2. **Define shot limits**: We also need to create shot limits files for each video (enrollment and development). A shot is the continuous footage or sequence between two edits or cuts. Detecting the shot limits is useful and necessary to help the tracking step. It is important to say that all tracks terminate when the shot terminates. This is done with the *scenedetect* tool in [37].

3.2.3 Detection

In this Master's Thesis we use two different face detection techniques, one is based on Histogram of Oriented Gradients combined with a linear SVM classifier [15] and the other one is based on a pre-trained CNN trained as a triplet network [16], because the idea was to try to learn better not frontal faces. Both techniques has been described in the chapter of *State of the art* in the chapter of *Results* they will be compared.

Now that we have the data of all the faces that appears in the video, that is, all dimensions of the bounding boxes that contains a face in each frame, it is time to introduce the tracking step. This step will provide continuity to the faces that belongs to the a single track, that means that all the faces of the same person in a single shot will be associated to the same track.

3.2.4 Tracking

As in the previous section, we used two different tracking methods:

- **By detection approach**: Using the output of the face detection in the previous step, face tracking relates detections of the (potentially) same person in successive frames using optical flow vectors.

  The method we perform to relate detections of the same person in successive frames, is an optical flow method based on Lucas-Kanade as explained in [5]. Optical flow is defined as spatio-temporal image brightness variations. To track a face across a sequence of frames, we use Lucas-Kanade optical flow algorithm [38] to estimate the motion of extracted facial features between adjacent frames in the sequence.

  The movement of a point can be described as a 2D vector $[u,v]^T$, as shown in Figure 3.2. Describe the brightness of point *(x, y)* at time t as *I(x, y, t)*, then, according to brightness constancy assumption in equation 3.1,

$$I(x, y, t-1) = I(x + u(x, y), y + v(x, y), t) \tag{3.1}$$

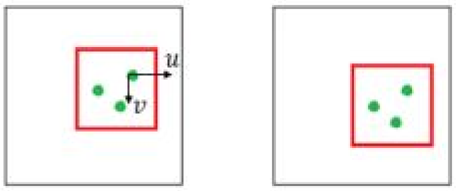


Figure 3.2: Points movement between two subsequent frames. Image from [5].

Linearize it by Taylor expansion, we can get

$$\bigtriangledown I \cdot [u,v]^T + I_t = 0 \tag{3.2}$$

Since the movements of neighbour points are almost the same, we can approximate the motion within a window of n points as

$$\begin{bmatrix} I_x(p_1) & I_y(p_1) \\ . & . \\ . & . \\ . & . \\ I_x(p_n) & I_y(p_n) \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = - \begin{bmatrix} I_t(p_1) \\ . \\ . \\ . \\ I_x(p_n) \end{bmatrix} \tag{3.3}$$

Then, we can estimate the positions of these points on the next frame as $p+[u,v]^T$. For future face replacement, we adjust it to an upright rectangle by computing its minimum bounding rectangle, and set it as the new bounding box.

- **Correlation tracker+Centroid association**: This tracking method combines the centroid association [6] and a correlation tracker [39]. As this method was designed for not having to detect in every frame, we apply the centroid association in the frames where it is time to detect to associate the current detections to the previous tracks in order to check if the current detections belongs to a person who has been already tracked (and we should continue that track, not creating a new one) or if we should start a new track because a new person has appeared.

  Then, between detection frames, it is when we perform the correlation tracking itself developed by Dlib. We will explain now in both complementary methods for our tracking step.

  - <u>The Correlation tracker</u>: As explained in [39], it is based on Danelljan et al. [40].

  Their work, in turn, builds on the popular MOSSE tracker from Bolme et al.'s 2010 work [41] However, their MOSSE tracker works well for objects that are translated, it often fails for objects that change in scale.

The work of Danelljan et al. proposed utilizing a scale pyramid to accurately estimate the scale of an object after the optimal translation was found. This breakthrough allows us to track objects that change in both (1) translation and (2) scaling throughout a video stream and furthermore, we can perform this tracking in real-time.

This object lets you track the position of an object as it moves from frame to frame in a video sequence. To use it, you give the correlation tracker the bounding box of the object you want to track in the current video frame. Then it will identify the location of the object in subsequent frames.

- Centroid association: As explained in [6], this object tracking algorithm is also called association tracking as it relies on the Euclidean distance between (1) existing object centroids (i.e., objects the centroid tracker has already seen before) and (2) new object centroids between subsequent frames in a video. The centroid association algorithm is a multi-step process.

1. Accept bounding box coordinates and compute centroids.

The centroid association algorithm assumes that we are passing in a set of bounding box (x, y) coordinates for each detected object in every single frame.

These bounding boxes can be produced by any type of object detector we want (CNN, color thresholding + contour extraction, Haar cascades, HOG + Linear SVM, SSDs, Faster R-CNNs, etc.), provided that they are computed for every frame in the video.

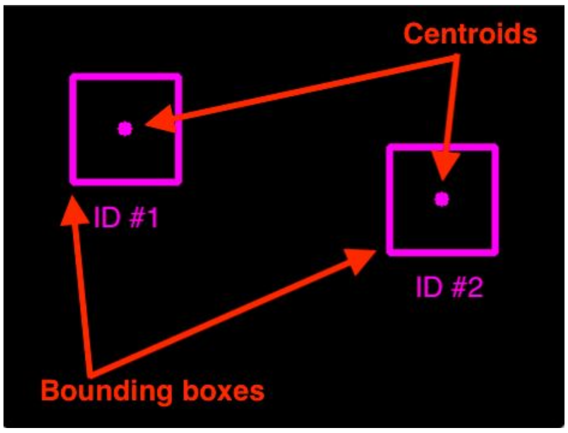


Figure 3.3: To build a simple object tracking algorithm using centroid tracking, the first step is to accept bounding box coordinates from an object detector and use them to compute centroids. Image from [6].

Once we have the bounding box coordinates we must compute the centroid, that is, the center (x, y) coordinates of the bounding box. Figure 3.3 demonstrates accepting a set of bounding box coordinates and computing the centroid.

Since these are the first initial set of bounding boxes presented to our algorithm we will assign them unique IDs.

2. Compute Euclidean distance between new bounding boxes and existing objects.

For every subsequent frame in our video, we apply Step 1 of computing object centroids; however, instead of assigning a new unique ID to each detected object (which would defeat the purpose of object tracking), we first need to determine if we can associate the new object centroids (yellow) with the old object centroids (purple) and check if it could be the same object. To accomplish this process, we compute the Euclidean distance (highlighted with green arrows) between each pair of existing object centroids and input object centroids.

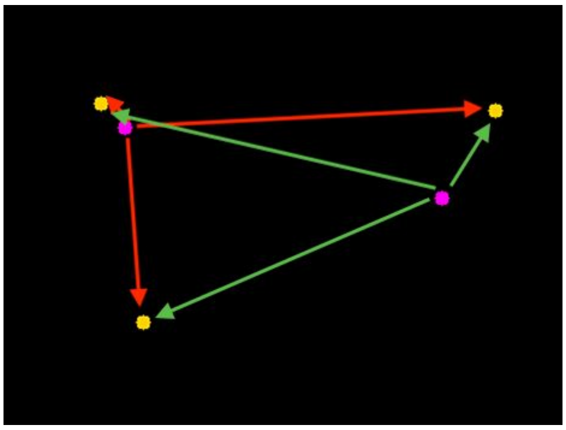


Figure 3.4: Three objects are present in this image for simple object tracking with Python and OpenCV. We need to compute the Euclidean distances between each pair of original centroids (red) and new centroids (green). Image from [6].

From Figure 3.4 we can see that this time we have detected three objects in our image. The two pairs that are close together are two existing objects.

We then compute the Euclidean distances between each pair of original centroids (yellow) and new centroids (purple). In order to use the Euclidean distances between these points to match them and associate them, we go through Step 3.

3. Update (x, y) coordinates of existing objects.

The primary assumption of the centroid tracking algorithm is that a given object will potentially move in between subsequent frames, but the distance between the centroids for frames F_t and F_{t+1} will be smaller than all other distances between objects.

Therefore, if we choose to associate centroids with minimum distances between subsequent frames we can build our object tracker.

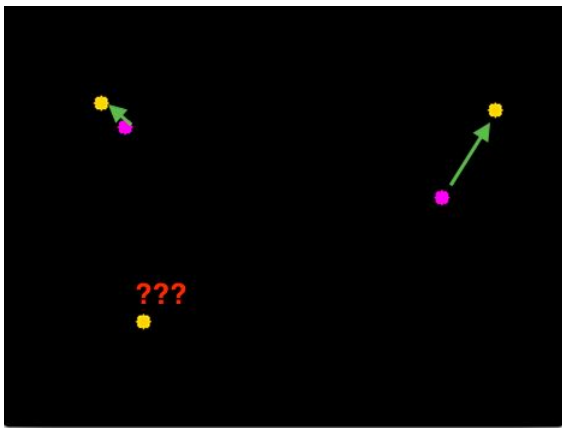


Figure 3.5: Our simple centroid object tracking method has associated objects with minimized object distances. Image from [6].

In Figure 3.5 you can see how our centroid tracker algorithm chooses to associate centroids that minimize their respective Euclidean distances. In Step 4 we will see what to do with the object in the bottom left.

4. Register new objects.

In the event that there are more input detections than existing objects being tracked, we need to register the new object. "Registering" simply means that we are adding the new object to our list of tracked objects by:

Assigning it a new object ID Storing the centroid of the bounding box coordinates for that object We can then go back to Step 2 and repeat the pipeline of steps for every frame in our video stream.

Figure 3.6 demonstrates the process of using the minimum Euclidean distances to associate existing object IDs and then registering a new object.

5. Deregister old objects.

Any reasonable object tracking algorithm needs to be able to handle the situation when an object has been lost, disappeared, or left the field of view.

Exactly how we handle these situations is really dependent on where your object tracker is meant to be deployed, but for this challenge, we will deregister old objects when they cannot be matched to any existing objects for a total of N subsequent frames or we are in a initial shot frame.

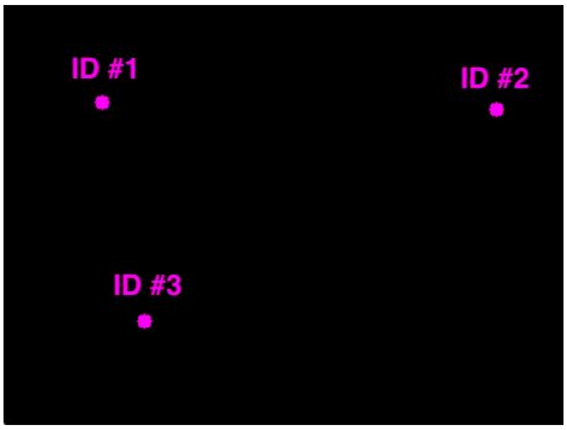


Figure 3.6: In this object tracking example, we have a new object that was not matched with an existing object, so it is registered as object ID #3. Image from [6].

3.2.5 Feature extraction

For the test video, one feature vector (128 D) per face track is extracted. The process is first to extract a FV for each detected face in the track. Later, all FV of a same track are averaged to get just one FV that characterizes the person that has been tracked. The technique used is the Dlib CNN face feature vector extractor in [42] based on face landmarks like eyes, eyebrows, nose, mouth and jawline.

For the enrollment videos, the same process is used. Also one FV is generated per each still image. All these FV are stored with the FV generated from the enrollment videos to build a model for each known character, and finally compare with each FV extracted from the test video. This is explained in the next step.

3.2.6 Classification

Each test FV extracted from the test video is compared against the enrollment FV's. As a classification method, we can use any indeed, but the one we are actually using consists on verify against all the identities using a threshold on the euclidean distance between FV's that indicates how similiar should be two FV's to consider or not a person as "unknown".

In case of multiple verifications for a given track, the identity with the smallest distance is selected. We perform several verifications because of the problem of "Unknown" people and because we have a closed dataset.

3.2.7 Scoring

The diarization performance scoring will evaluate the accuracy of indexing a TV show in terms of the people present in the image. This process is explained in [43] and there exists a script that do all this scoring task.

To measure the performance of the proposed systems, the Diarization Error Rate (DER) will be computed as the fraction of face time that is not correctly attributed to that specific character. This score will be computed over the entire file to be processed; including regions where more than one character is present (overlap regions).

This score will be defined as the ratio of the overall diarization error time to the sum of the durations of the segments that are assigned to each class in the file.

Given the dataset to evaluate Ω, each document is divided into contiguous segments at all faces change points found in both the reference and the hypothesis, and the diarization error time for each segment n is defined as

$$DER = \frac{\sum_{n\epsilon\Omega} E(n)}{\sum_{n\epsilon\Omega}(T(n)N_{ref}(n))}, \tag{3.4}$$

where *T(n)* is the duration of segment n, $N_{ref}(n)$ is the number of faces that are present in segment n, $N_{sys}(n)$ is the number of system faces that are present in segment n and $N_{correct}(n)$ is the number of reference faces in segment n correctly assigned by the diarization system.

The diarization error time includes the time that is assigned to the wrong face, missed face time and false alarm face time:

- **Face Error Time**: The Face Error Time is the amount of time that has been assigned to an incorrect face. This error can occur in segments where the number of system faces is greater than the number of reference faces, but also in segments where the number of system faces is lower than the number of reference faces whenever the number of system faces and the number of reference faces are greater than zero.
- **Missed Face Time**: The Missed face Time refers to the amount of time that a face should be detected, but we are not detecting anything.
- **False Alarm Time**: The False Alarm Time is the amount of time that a face has been labeled by the diarization system but is not present in segments where the number of system faces is greater than the number of reference faces.

We will also consider a forgiveness collar of 0.25s. This is a time we add before and after each reference boundary in order to take into account both inconsistent human annotations and the uncertainty about when a face begins or ends.

3.2.8 Critical Parameters

Other thing that can affect the overall error of the system is the configuration of the parameters of the correlation tracker and the classification script. The most critical for this challenge are:

From the correlation tracker:

- *Detector Model*: Model for the dlib's DNN face detector. If not given, defaults to the HOG+SVM detector. In the script that performs the correlation tracker, we can also perform the face detection step and with this parameter we can choose which model we want to apply.
- *Detection Parameter*: dlib detection parameter. This is the upsampling or downsampling factor we apply to the image before detecting faces. An advantage would be that if the factor is bigger than 1 (upsampling), we can detect smaller faces if there is somebody far away for example. On the other hand, it implies more computational power.
- *Skip Frames*: Number of skip frames between detections. That means that it is not necessary to perform detection on every frame. This has a clear advantage of saving computation resources. However, it could happen that if a face appears in the frames between detections, we are not going to detect it until the next detection frame and there will be several frames where we are not detecting a face when we should be doing so (missed face time).
- *Max dissapeared*: Number of maximum consecutive frames a given object is allowed to be marked as "disappeared" until it is deregistered from tracking. This is useful in case that an object or another person occludes somebody's face so we can keep a continuous track (with its corresponding and unique track ID) instead of fragmenting it, because the face we are trying to detect, appears and disappears.

From the classification script:

- *'Unknown' threshold*: When all the FV's are computed, there is a threshold based on the Euclidean distance between FV's to determine how far or how near is a

FV to decide if it belongs to a known character or if it is unknown. To fix this threshold, we compute a scanning into a range of thresholds (between 0 and 1) and we finally choose the one that generates the minimum overall error.

CHAPTER 4

RESULTS

In this chapter we are going to compare the different face detections and tracking methods and analyze the results and limitations of the system. We are going to follow the evolution and improvement of the overall error through the different techniques employed to improve the initial results.

4.1 Starting point

When the UPC presented their implementation of the Albayzin challenge, the best overall error they could get was 40.1%. Specifically, the proportion of each type if error was:

Type of error	% of time
Overall	**40.1**
Missed Face	37.2
False Alarm	2.2
Face Error	2.6

Table 4.1: UPC initial results.

These results were obtained with the face detection method based on HOG+SVM and, as a tracking method, the optical flow based on Lucas-Kanade.

We can see that the *Missed Face Error* is the biggest one with a 37.2% of the time of the test video that we are not detecting any person (we are missing detections). The optimal "unknown" threshold in this case is 0.47.

From now, this error will be the main challenge we will try to reduce with the different techniques explained in the State of the art and Methodology chapters.

4.2 CNN for face detection

Taking into account the previous result, we thought that there would be a good option to apply the other face detector based on CNN. The idea was that applying machine learning, the detector could learn better from the enrollment data and when it comes with the test video, it will not miss so many faces as before. As a tracking method we maintain the same as before (optical flow based on Lucas-Kanade) and for the moment, we keep the *Detection parameter* = 1. The results are shown in Table 4.2.

Type of error	% of time
Overall	**37.69**
Missed Face	31.7
False Alarm	1.6
Face Error	4.4

Table 4.2: Results with CNN face detector + optical flow tracking.

Now we have managed to reduce the overall error 2.41%. At the same time, we have reduced the Missed Face Time in a 5.5% and the False Alarm (we detect somebody when there is nobody) in a 0.6%, but in return, the Face Error Time has increased a 1.8% (we detect a face but we identify him/her wrong). The optimal "unknown" threshold in this case is 0.45.

We have managed to reduce the overall error as well as the Missed Face Time, but we will try now to reduce it even more with a the correlation tracker.

4.3 Correlation tracker

Now that we have seen the results of both face detection methods with the optical flow tracker, we are going to take the best face detector (CNN) and change to the correlation tracker.

Here, as we had several parameters to adjust, we tried first with the default ones to understand how this correlation tracker works. This default configuration was:

- Detector Model: CNN.
- Skip frames: 25.

- Detection Parameter: 1.
- Max dissapeared: 40.
- 'Unknown' threshold: 0.45.

And the results we obtained are shown in Table 4.3.

Type of error	% of time
Overall	**48.62**
Missed Face	29.6
False Alarm	5.3
Face Error	13.8

Table 4.3: Results with CNN face detector + correlation tracking.

We can see that the Missed Face Time has been reduced a 2.1% as we wanted at the start point. However, all the other types of errors have increased (including the overall error), so, although we have managed to reduce the Missed Time, this is not the solution we wanted, because the Overall Error is still more important.

At this point we started to imagine that if we decreased the missed time, the overall and the other errors would increase and the solution will not be as simple as we would like.

4.4 Configurations of the correlation tracker

There is still one possible option to optimize this tracker, that is, try to find the optimal configuration of its parameters for this type of videos that we are working with.

First of all, we wanted to find the configuration of parameters to replicate the result obtained with the CNN face detector and the optical flow tracking (37.69%) that is the best one we have so far. Then, from that result, tune the parameters with criteria after seeing where and when the actual method fails for our specific video, and optimize the configuration. The most important parameter here is *Skip Frames* because the previous tracker detected in every frame, so we tuned that parameter to 1 and the rest

Type of error	% of time
Overall	**39.26**
Missed Face	32.0
False Alarm	1.7
Face Error	5.5

Table 4.4: Results with CNN face detector + correlation tracking with *Skip Frames*=1.

were set to default with a *Unknown' threshold = 0.45*. The results are shown in Table 4.4.

These results are still worse than the better we have so far. We know that the FV of each track ID is the average of all the FV's within that track, and observing the video, we see that there are some frames where an object (person's hair, other person, a mug...) is occluding the face we are trying to detect. In Figure 4.1 there are some examples of scenarios when that happens.

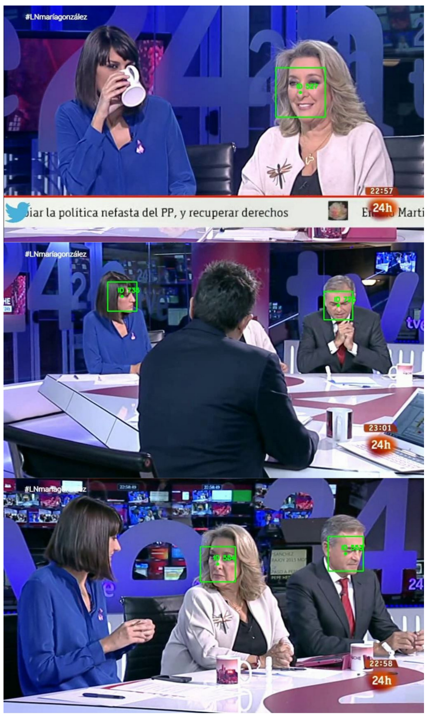


Figure 4.1: Scenarios where someone or something is occluding someone's face.

Now, we want to get rid of those frames because they can be disturbing the mean

FV that will represent the person who appears in that track ID. The solution is setting the parameter *Max disappeared* to 0 in order to not taking into account for the track, the frames where the face are not clearly visible and if one of those frames appears, deregister the track until the face is visible again. With this reasoning, our new configuration of parameters are:

- Detector Model: CNN.
- Skip frames: 1.
- Detection Parameter: 1.
- Max disappeared: 0.
- 'Unknown' threshold: 0.45.

And the results are shown in Table 4.5.

Type of error	% of time
Overall	**38.95**
Missed Face	31.1
False Alarm	2.0
Face Error	5.9

Table 4.5: Results with CNN face detector + correlation tracking with *Skip Frames*=1 and *Max disappeared*=1.

Now, the results are more approximated to 37.69%, but the great advantage of this tracker is not having to detect in every frame, so we will try now to increase the *Skip Frames* parameter to improve the result or at least replicate it with less computational power requirements.

We will try with this configuration:

- Detector Model: CNN.
- Skip frames: 5.
- Detection Parameter: 1.
- Max disappeared: 0.
- 'Unknown' threshold: 0.45.

The results obtained are shown in Table 4.6.

Type of error	% of time
Overall	**37.91**
Missed Face	33.0
False Alarm	1.0
Face Error	3.9

Table 4.6: Results with CNN face detector + correlation tracking with *Skip Frames*=5, *Max disappeared*=1.

Now we have almost replicated our best result and with a decrease of the computational power, detecting 1 out of 5 frames.

Also, we realised seeing the video that there are still many faces that are not detected, because the people are far away, their faces are small or in an odd position and the resolution of the video is not so high as we would like. Some examples of these scenarios are in Figure 4.2. So apart from increasing the *Skip Frames* parameter, we will also increase the detection parameter in order to upsample the frames $\times 2$ and detect smaller faces.

Let's try with this configuration:

- Detector Model: CNN.
- Skip frames: 5.
- Detection Parameter: 2.

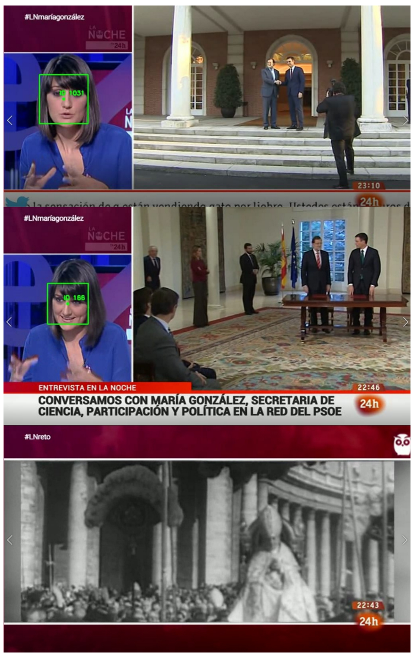

#LNmariagonzález
23:10
24h
#LNmariagonzález
ENTREVISTA EN LA NOCHE
22:46
CONVERSAMOS CON MARÍA GONZÁLEZ, SECRETARIA DE CIENCIA, PARTICIPACIÓN Y POLÍTICA EN LA RED DEL PSOE
24h
#LNreto
22:43
24h

Figure 4.2: Pope John XXIII, Mariano Rajoy and Pedro Sánchez are known characters for our system, but we are not even detecting them.

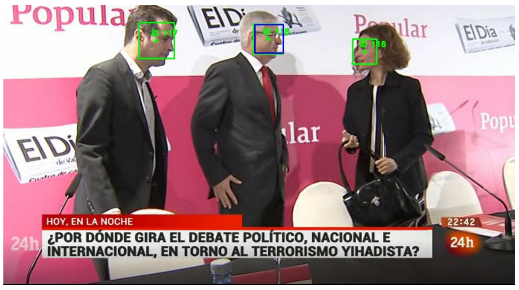
Type of error	% of time
Overall	**37.65**
Missed Face	30.6
False Alarm	1.6
Face Error	5.4

Table 4.7: Results with CNN face detector + correlation tracking with *Skip Frames*=5, *Max disappeared*=1 and *Detection parameter*=2.

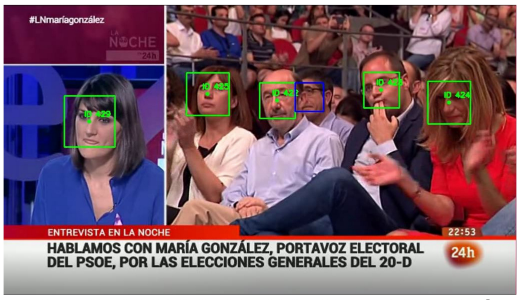
- Max disappeared: 0.
- 'Unknown' threshold: 0.45.

The results obtained are shown in Table 4.7.

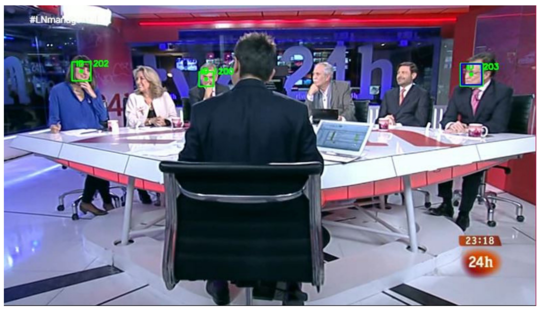
Now, we have managed to reduce the overall error 0.04% of what we have got previously. Also with this last change we have reduce 1.4% the Missed Time Error, but the False Alarm time has increased a 0.6%. It is not much but it is logic because


HOY, EN LA NOCHE
¿POR DÓNDE GIRA EL DEBATE POLÍTICO, NACIONAL E INTERNACIONAL, EN TORNO AL TERRORISMO YIHADISTA?
22:42
24h

FACE LN24H-20151125 1 752.036 1.680 <NA> <NA> Julio_Somoano 116


#LNmariagonzález
ENTREVISTA EN LA NOCHE
HABLAMOS CON MARÍA GONZÁLEZ, PORTAVOZ ELECTORAL DEL PSOE, POR LAS ELECCIONES GENERALES DEL 20-D
22:53
24h

FACE LN24H-20151125 1 1398.640 7.120 <NA> <NA> Maria_Gonzalez 425



FACE LN24H-20151125 1 2893.040 3.800 <NA> <NA> Jose_Maria_Villegas 1200

Figure 4.3: The blue boxes represent the detections without upsampling and the green ones with upsampling. The line below each frame represent the final results obtained by the system. The last two fields are the name of the person and the track ID that we can also see it in the image. In the first frame, the detector with upsampling detects two more faces than without upsamplig, but for example the track 116 is wrongly identified because that woman is not Julio Somoano. Similar things happens with the track 425 in the second image and the track 1200 in the third one. All of these three cases have faces in odd positions that we detect only with the upsampling step, but are wrongly identified.

if we upsample the frames, there will be some probability to detect a face where there are nothing (False Positive or False Alarm.).

The most interesting thing we found is the increasing of the Face Error Time in a 1.5%, that means that we detect a person but we are not identifying him/her well. There are many moments when we detect more faces than without the upsampling step. That should be good, but in the case of our detector and our feature extractor does not work well. This happens because the feature extraction technique (triplet network) has not been trained with faces in extreme face rotations and the feature extractor is not very precise when it finds faces like that. In Figure 4.3 we can see some examples of scenarios when we detect more faces, but wrongly.

We expected better results with the upsampling solution, but we concluded that is the feature extractor which does not work well, because the faces are detected indeed. We tried with other configuration of parameters like increasing the *Skip Frames* parameter or the *Max disappeared* parameter but the results are not going better.

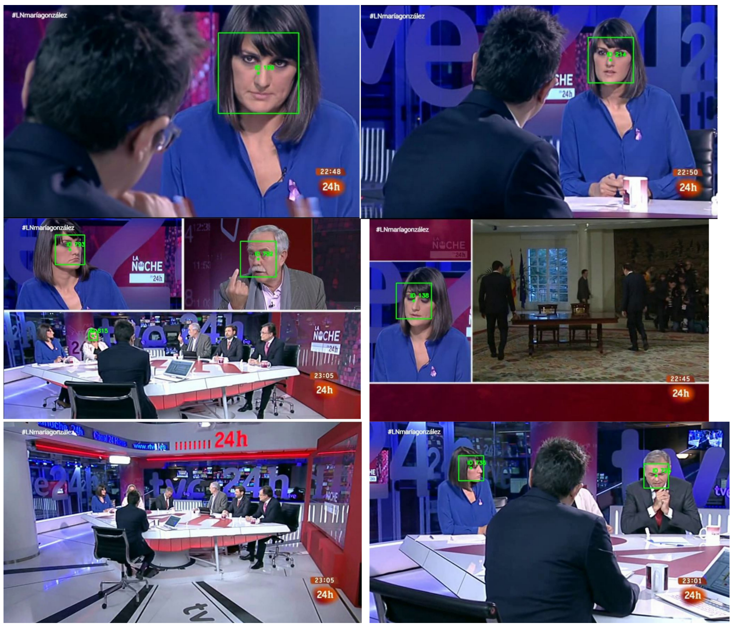


Figure 4.4: Six examples of moments when there is somebody backwards.

The only solution we could think regarding image processing is training other feature extractor more sophisticated and precise when detecting faces and extracting features. In fact, I tried to implement my own triplet network to extract better features, but the results were bad because the accuracy was always around 50% and that means that the network was not learning. We believe that this happens because of the selection of the triplet that was random, but there is a method called *Hard Negative Mining* that has been explained in the State of the Art chapter.

Also we have assumed that there is a minimum error we cannot decrease. There are lots of moments when there are people with their back towards the camera (see Figure

4.4) and there is nothing we can do with it, because their faces are not visible at all. But this happens in our specific video because for example the conductor of the show/debate is most of the time backwards asking the collaborators or occluding someone that is exactly behind him. That affect mainly the Missed Face Time, that is why it is always the highest error. However, these parts were annotated because the Albayzin challenge was intended to deal with multimodal diarization. With a multimodal analysis, the speech information would solve the recognition problem when someone's face is not visible.

CHAPTER 5

CONCLUSIONS

In this Master Thesis we have developed an automatic video indexing system exploring several face detection and tracking methods, ones based on machine learning and other ones more classical. With the last experiments, we realized that the face detector based on a CNN worked quite well (better than without machine learning) because, ultimately, its goal is to detect faces and give the bounding box where there is a face. That goal is more or less accomplished even with sideway faces and this face detector detected more faces than the classical one based on HOG+SVM. The main problem came with small faces because most of them are missed even with an upsampling of the frames, maybe because of the resolution of the video and the quality of the image. And obviously, when the person is backwards, it is impossible to detect his/her face.

Also, once a face is detected, after seeing the resulting video, we saw that the tracking task worked well (the tracks start and terminate when they should do it and it follows the correct face track during all the shot the majority of times). With that improvements, we managed to decrease a 2.45% the overall error and reduce a 6.6% the Missed Time Face that has been the highest all the time (and still is). Despite of that, there were an increment of a 1.5% in face recognition errors and it must be because of the feature extractor that does not work well in our video circumstances. Although a face is well detected and tracked even in sideway positions, the feature extractor does not identify the person correctly because it has not been trained with faces in the positions where the feature extractor fails.

We said in the chapter of Results that the solution using image processing is applying a more sophisticated and precise feature extractor. In this Master's Thesis I tried to implement my own triplet network based on VGG faces [29] and train it and test it with the MSRA dataset, but the results were quite bad, because the accuracy was always around 50%, that means that the network was not learning. The conclusion we could get was that the triplets were not optimally selected as it has been explained the subsection *Hard Negative Mining* from the State of the Art chapter.

There is also another powerful solution to improve mainly the Missed Face Time. It is out of the scope of this Thesis, but it could be a very interesting future project that consists on introduce the multimodal analysis and using the speech information to recognise a person even if his/her face is not visible. In fact, the Albayzin system was designed to deal with speech information too, although in this Thesis we only worked with image processing.

We expected better results with the deep learning face detection method and the correlation tracker, but as we said in the chapter of Results and taking into account the limitations we have just described, there will be always a minimum error we will not be able to reduce until we introduce speech information or other face detection and feature extraction techniques that can deal mainly with small faces and low resolution videos which are also important problems that we have had. Probably, with other database and the same system, we would get better results.

BIBLIOGRAPHY

[1] http://iberspeech2018.talp.cat/, “IberSPEECH 2018 | IberSpeech Conference, Barcelona 2018.”

[2] D. R. Navarro, “A DEGREE THESIS Multimodal Deep Learning methods for person annotation in video sequences,” tech. rep., 2017.

[3] http://cs231n.github.io/convolutional-networks/, “CS231n Convolutional Neural Networks for Visual Recognition.”

[4] C. Shu, X. Ding, and C. Fang, “Histogram of the Oriented Gradient for Face Recognition *,” *Tsinghua Science & Technology*, vol. 16, p. 216â224, 2011.

[5] Q. Cao and R. Liu, “Real-Time Face Tracking and Replacement,” tech. rep.

[6] A. Rosebrock, “Simple object tracking with OpenCV - PyImageSearch,” 2018.

[7] Y. Taigman, M. Yang, M. Ranzato, and L. Wolf, “DeepFace: Closing the Gap to Human-Level Performance in Face Verification,” in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2014.

[8] M. R. Everingham, J. Sivic, and A. Zisserman, “Hello! My name is... Buffy” – Automatic Naming of Characters in TV Video,” pp. 1–92, 2012.

[9] H. Bredin, J. Poignant, M. Tapaswi, G. Fortier, V. B. Le, T. Napoleon, H. Gao, C. Barras, S. Rosset, L. Besacier, J. Verbeek, G. Quénot, F. Jurie, and H. K. Ekenel, “Fusion of Speech, Faces and Text for Person Identification in TV Broadcast,” in *Proceedings of the 12th International Conference on Computer Vision - Volume Part III*, ECCV’12, (Berlin, Heidelberg), pp. 385–394, Springer-Verlag, 2012.

[10] N. Dalal and W. Triggs, “Histograms of Oriented Gradients for Human Detection,” *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition CVPR05*, vol. 1, no. 3, pp. 886–893, 2004.

[11] M. Guillaumin, T. Mensink, J. Verbeek, and C. Schmid, “Face Recognition from Caption-Based Supervision,” *International Journal of Computer Vision*, vol. 96, no. 1, pp. 64–82, 2012.

[12] W.-Y. Zhao, R. Chellappa, P. J. Phillips, and A. Rosenfeld, “Face Recognition: A Literature Survey,” *ACM Comput. Surv.*, vol. 35, pp. 399–458, 2003.

[13] R.-L. Hsu, M. Abdel-Mottaleb, and A. K. Jain, “FACE DETECTION IN COLOR IMAGES,” tech. rep.

[14] A. Lanitis, C. Taylor, and T. F. Cootes, “An Automatic Face Identification System Using Flexible Appearance Models,” *Image and Vision Computing*, vol. 13, pp. 393–401, 1995.

[15] D. King, “Dlib HOG+SVM Face detector - http://dlib.net/face_detector.py.html.”

[16] D. King, “Dlib Cnn Face Detector - http://dlib.net/cnn_face_detector.py.html.”

[17] D. King, “Dataset - http://dlib.net/files/data/dlib_face_detection_dataset-2016-09-30.tar.gz.”

[18] P. J. Phillips, “Support Vector Machines Applied to Face Recognition,” *Advances in Neural Information Processing Systems*, vol. 11, 2001.

[19] Y. Sun, X. Wang, and X. Tang, “Deep Learning Face Representation from Predicting 10,000 Classes,” in *Proceedings of the 2014 IEEE Conference on Computer Vision and Pattern Recognition*, CVPR ’14, (Washington, DC, USA), pp. 1891–1898, IEEE Computer Society, 2014.

[20] Y. Sun, X. Wang, and X. Tang, “Deep Learning Face Representation by Joint Identification-Verification,” tech. rep.

[21] G. B. Huang, M. Ramesh, T. Berg, and E. Learned-Miller, “Labeled Faces in the Wild: A Database for Studying Face Recognition in Unconstrained Environments,” tech. rep.

[22] F. Schroff, D. Kalenichenko, and J. Philbin, “FaceNet: A unified embedding for face recognition and clustering,” pp. 815–823, 2015.

[23] W. Liu, Y. Wen, Z. Yu, M. Li, B. Raj, and L. Song, “SphereFace: Deep Hypersphere Embedding for Face Recognition,” tech. rep.

[24] J. Bromley, I. Guyon, Y. LeCun, E. Säckinger, and R. Shah, “Signature Verification Using a "Siamese" Time Delay Neural Network,” in *Proceedings of the 6th International Conference on Neural Information Processing Systems*, NIPS’93, (San Francisco, CA, USA), pp. 737–744, Morgan Kaufmann Publishers Inc., 1993.

[25] J. Hu, J. Lu, and Y.-P. Tan, “Discriminative Deep Metric Learning for Face Verification in the Wild,” in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2014.

[26] R. Hadsell, S. Chopra, and Y. LeCun, “Dimensionality Reduction by Learning an Invariant Mapping,” in *Proceedings of the 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition - Volume 2*, CVPR ’06, (Washington, DC, USA), pp. 1735–1742, IEEE Computer Society, 2006.

[27] S. Chopra, R. Hadsell, and Y. LeCun, "Learning a Similarity Metric Discriminatively, with Application to Face Verification," in *Proceedings of the 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05) - Volume 1 - Volume 01*, CVPR '05, (Washington, DC, USA), pp. 539–546, IEEE Computer Society, 2005.

[28] C. Szegedy, W. Liu, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," tech. rep.

[29] O. M. Parkhi, A. Vedaldi, and A. Zisserman, "Deep Face Recognition," in *Procedings of the British Machine Vision Conference 2015*, pp. 1–41, 2015.

[30] A. Sadiq, A. Rabhi, and A. Mouloudi, "Face Tracking: state of the art," 2015.

[31] J. Joshan and A. P. Suresh, "Systematic Survey on Object Tracking Methods in Video," Tech. Rep. 8, 2012.

[32] A. Acm Reference Format: Yilmaz, O. Javed, and M. Shah, "Object tracking: A survey," *ACM Comput. Surv*, vol. 38, p. 45, 2006.

[33] L. Wang, W. Ouyang, X. Wang, and H. Lu, "Visual Tracking with Fully Convolutional Networks," tech. rep.

[34] S. V. Tathe, A. Sandipan Narote, and S. P. Narote, "Face Recognition and Tracking in Videos," *Technology and Engineering Systems Journal*, vol. 2, no. 3, pp. 1238–1244, 2017.

[35] E. Lleida, A. Ortega, A. Miguel, V. Bazán, C. Pérez, M. Gómez, and A. De Prada, "RTVE2018 Database Description," tech. rep., 2018.

[36] "FFmpeg - https://ffmpeg.org/."

[37] "PySceneDetect - https://pyscenedetect.readthedocs.io/en/latest/."

[38] B. D. Lucas and T. Kanade, "An Iterative Image Registration Technique with an Application to Stereo Vision," in *Proceedings of the 7th International Joint Conference on Artificial Intelligence - Volume 2*, IJCAI'81, (San Francisco, CA, USA), pp. 674–679, Morgan Kaufmann Publishers Inc., 1981.

[39] A. Rosebrock, "Correlation tracker with dlib - PyImageSearch."

[40] M. Danelljan, G. Häger, F. Shahbaz Khan, and M. Felsberg, "Accurate Scale Estimation for Robust Visual Tracking," tech. rep.

[41] D. S. Bolme, J. R. Beveridge, B. A. Draper, and Y. M. Lui, "Visual Object Tracking using Adaptive Correlation Filters," tech. rep.

[42] D. King, "Dlib Feature Extractor - https://face-recognition.readthedocs.io/en/latest/_modules/face_recognition/api.html#face_encodings."

[43] E. Lleida, A. Ortega, A. Miguel, V. Bazán, C. Pérez, M. Zotano, and A. De Prada, “Albayzin Evaluation: IberSPEECH-RTVE 2018 Multimodal Diarization Challenge,” tech. rep.