Sapienza - University of Rome

CyberSecurity

Biometric Systems

W.O.R.S.

Wide Open-source Revelation System

Alessio Mobilia - 1896932
mobilia.1896932@studenti.uniroma1.it

Andrea Tripoli - 1886723
tripoli.1886723@studenti.uniroma1.it

Friday 8th October, 2021

Contents

1 The project

The main purpose of this project is to identify peoples through surveillance cameras in an automated way in a period where the faces are often obscured by masks.

The idea is to use a behavioural biometric trait, like the gait, that can recognized by video cameras and can easily recorded. Unfortunately the gait recognition cannot be accurate as face recognition, so we chosen to use it as a support to the face recognition that can be used when the face traits aren't easily distinguished.

So we implemented a software that given a database of faces and identities it can extract and record the gait features when a person is recognized in a video through his face, so that when the same person is shown in an other video, in which he cannot be identified by the face recognition, the system will use the gait to identify him

2 Face Recognition

The human face is one of the most used features in the biometric area so that it is possible to verify and recognize the identity of a person. This biometric feature includes the eyes, nose, mouth and the general geometry of the face. Thanks to the peculiarity of simplicity in the verification of these elements and of being "passive" (i.e. no cooperation of the individual is required), face recognition is often used in video surveillance and security environments.

Face recognition involves the computerized recognition of personal identity based on geometric or statistical characteristics derived from face images. It is possible to detect and recognize faces with little and greater processing capacity, even if the current challenge is to create a biometric system that accurately recognizes the face with low light, glasses, cosmetics, etc. . . [16].

An automated face recognition system involves specific tasks to accurately identify a human face: detection of a pattern, face tracking, face verification and face recognition. Face detection learns from statistical models, provided as inputs, of faces and non-faces of persons, and then applies rules (or algorithms) to distinguish a real pattern of a human face from an fictitious pattern. Face tracking predicts the movements of the face and future positions. Finally, face verification provides the identity of the face of the person that the algorithm is trying to identify.

As described in the Figure 1, a face recognition system comprises several steps. An image (or frame of a video sequence) is entered as an input into the face recognition system, the first step is to identify a probable shape and geometry of a human face. Once a face is recognized, the face recognition system performs normalization and cropping operations (for example, align the center of the eyes to detect the angle of the face, rotate or resize the face and finally transform the face region into a vector). This allows us to extract the characteristics of a face and use it for a biometric face recognition system.

The project **W.O.R.S.** is open source and the codes can be found on GitHub [10]

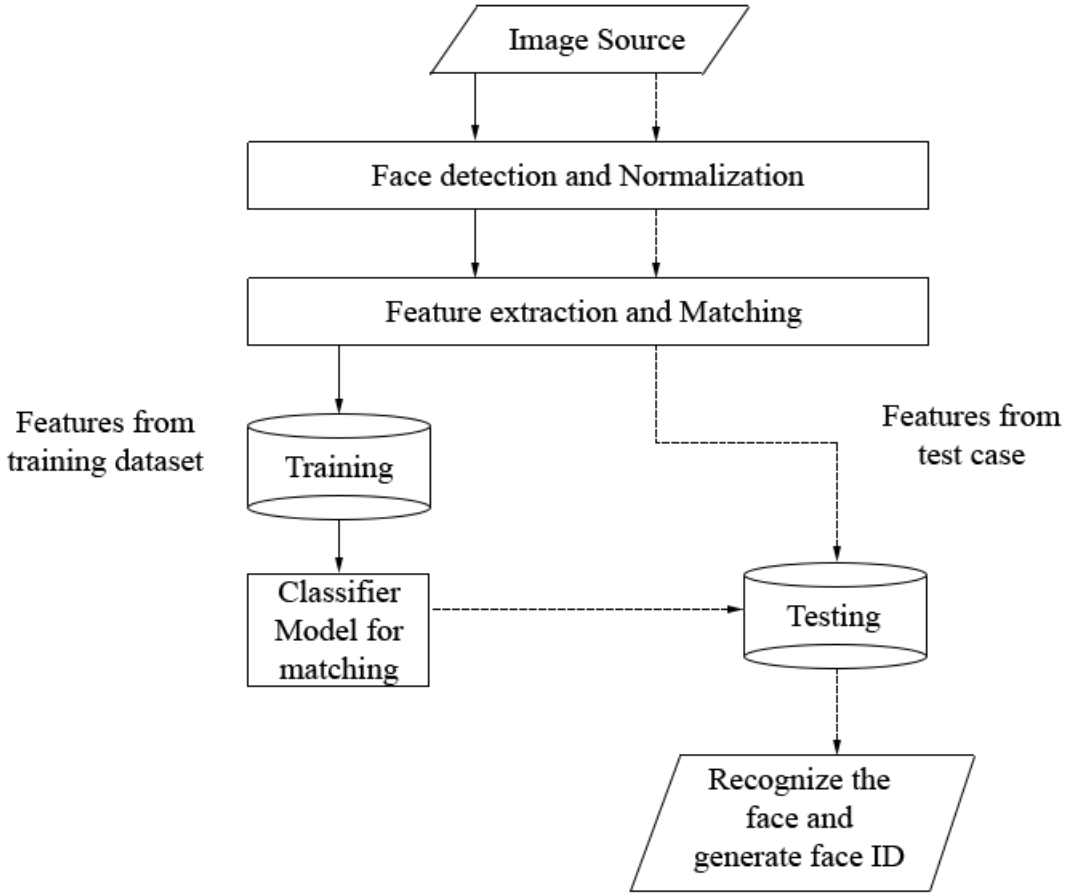


Figure 1: Diagram of a face recognition system

2.1 Face Detection

Face detection is the first step for a biometric face recognition system. Over the years, techniques based on data-based learning have been implemented, such as statistical modeling methods, learning methods based on neural networks, statistical learning theory

and methods based on Support Vector Machine, methods based on random Markov fields and color-based face detection.

Statistical Methods Statistical methods usually start with estimating the distributions of face and non-face patterns, then apply a pattern classifier or face detector to search across a range of possible scales and positions of human faces.

Neural Network-based learning methods Neural network-based methods learn to discriminate implicit distributions of face and non-face models using training samples and network structure, without involving an explicit estimation procedure.An example of a neural network used for face detection is the *convolutional neural network* (CNN).

CNN is a type of feed-forward artificial neural network (i.e. a network in which connections between units do not form loops) in which the connectivity pattern between neurons is inspired by the organization of the animal visual cortex, whose individual neurons are arranged in such a way as to respond to the overlapping regions that tessellate the visual field. CNNs are a fundamental tool in the field of deep learning. In particular they are suitable for image recognition. It is possible to extract features from a previously trained network, and use them to train a linear classifier.

The role of the Convolutional Neural Network (Figure 2) is to reduce the images into a form that is easier to process, without losing the functionalities that are fundamental to obtaining a good prediction through convolution operations. The goal of the convolution operation is to extract high-level features such as edges from the input image. ConvNet (synonym of convolutional neural network) consist of several levels, the so-called ConvLayers, in which each level is responsible for the acquisition of low-level functionalities such as edges, color, gradient orientation, etc.... As the layers increase, the network architecture will have a greater understanding of the images provided as inputs. There are two types of results for the operation: one in which the convoluted feature is reduced in dimensionality relative to the input, and the other in which the dimensionality is increased or remains the same. This is done by applying *Valid Padding* in the first case, or *Same Padding* in the second. With Same Padding we find a convoluted matrix of dimensions equal to the starting image. Instead with the Valid Padding, we find a matrix of dimensions equal to the Kernel itself. Once the convolute matrix is obtained, the Pooling layer will perform a reduction of the spatial dimensions. This will decrease the computing power required to process the data by reducing the size. This will also be useful for extracting dominant

characteristics, keeping the training process effective. The convolutional layer and the pool layer together form the i-th layer of a convolutional neural network. Depending on the complexity of the images, the number of such layers can be increased to further capture low-level details, but at the cost of more computing power. Now that the image has been converted to a multilevel shape, we need to turn it into a column vector. The flattened output is sent to a feed-forward neural network and back-propagation is applied to each iteration of the training [13].

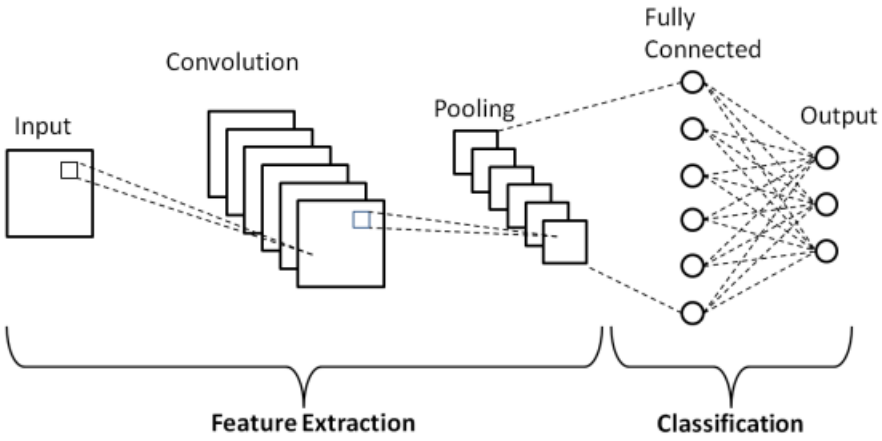


Figure 2: Convolutional Neural Network

Histogram of oriented gradients Histogram of oriented gradients (HOG) is a descriptor of features used in computer vision and image processing for object recognition. The technique counts occurrences of gradient orientation in localized portions of an image (Figure 3).

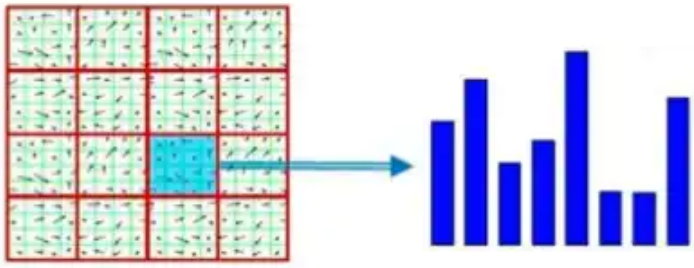


Figure 3: Histogram of oriented gradients

2.2 Face Recognition

Robust face recognition schemes require both low-dimensional feature representation for data compression purposes and advanced discrimination capabilities for subsequent image classification. Representation methods usually begin with a dimensionality reduction procedure, since the high dimensionality of the original space makes statistical estimation very difficult, if not impossible, due to the fact that the high-dimensional space is mostly empty. Some of the most popular representation and classification techniques for face recognition are as follows.

Principal component analysis Principal component analysis (PCA) is a data simplification technique used in multivariate statistics. The aim of the technique is to reduce the more or less high number of variables describing a data set to a smaller number of latent variables, limiting the loss of information as much as possible.

Gabor wavelets Gabor wavelets are commonly used to extract local functionality for various applications such as object detection, recognition and tracking. However, extracting Gabor's features is very computationally intensive, so the features are impractical for real-time applications. The Gabor image representation simulates the function of the human visual system, a design feature that may be important in the field of robotics and machine vision. Gabor's wavelet approach appears to be quite perspective and has several advantages such as invariance to homogeneous changes in lighting, small changes in head balance and robustness against face hair, glasses. .

Fisher Linear Discriminant Fisher Linear Discriminant (FLD) became one of the most widely used techniques in pattern recognition. FLD is an example of a class-specific method, and is a well-known method of classifying the class into arrays of classes of classes and classes that could be taken for classification, as it tries to form interclass and intraclass scattering in order to produce a better ranking [11].

3 Gait Recognition

Gait is defined as the style or manner of walking. Studies in physchophysics suggest that people can identify familiar individuals using just their gait [4].

The Gait is a biometric behavioral trait, that is analysable via non invasive methods. With gait recognition a person can be recognized even with a mask and with sunglasses, very common in this pandemic period.

Unfortunately the gait can be affected and distorted by external factors such walking speed, different kind of shoes, ground sloops, temporary illness and carrying weights and moreover it change by aging.

There are different possible approaches to gait recognition, Machine Vision-based, floor sensor-based and Wearable Sensor-based approaches.

3.1 Gait recognition with machine vision

The process of video of human gait by computer vision method to recognize or to identify a parson based on their body shape and walking styles is a passive mode of acquiring biometric data from distance.

Contrary to other method, the machine vision-based approach, is way more affordable and easier to setup because it can use pre-existing video-surveillance camera, do not force each individual to wear specific devices, and can be used to identify non collaborative peoples.
Furthermore the machine vision-based approach is used to analyze other biometric traits, such face recognition, so the it can be implemented as a support to other type of biometric recognition.

This approach is negatively affected by common machine vision problems, such occlusion, pose and illumination changes, perspective with respect to camera, carried object and voluminous clothes. Some of this problems can be mitigated with use of multiple cameras and infrared cameras.

Furthermore they use algorithms that are quite demanding in terms of computational resources and is really difficult to track multiple users in the same environment.

There are different possible method to recognize a person's gait in a video, the most common relies on a shape based approach. This class of approaches emphasizes the

silhouette shape similarity and underplays the temporal information. So far, they have resulted in the highest demonstrated performance [14] . The direction chosen by us involves the transformation of the silhouette sequence into a single image representation such GEI (Gait energy image, Figure 4) that preserves the dynamic and static information of a gait sequence, however, there is no consideration of the time that normalizes each silhouette within the GEI.

The GEI image can be calculated using this formula:

$$G(x, y) = \frac{1}{N} \sum_{N}^{t=1} B(x, y, t)$$

Where $B(x, y, t)$ is the gait cycle image sequence, N is the number of frames in a gait sequence of a cycle, and t is the number of gait frames [9] .

Figure 4: Example of Gait Energy Image (GEI) extracted by our system for the person 8

4 Implementation

4.1 Overview

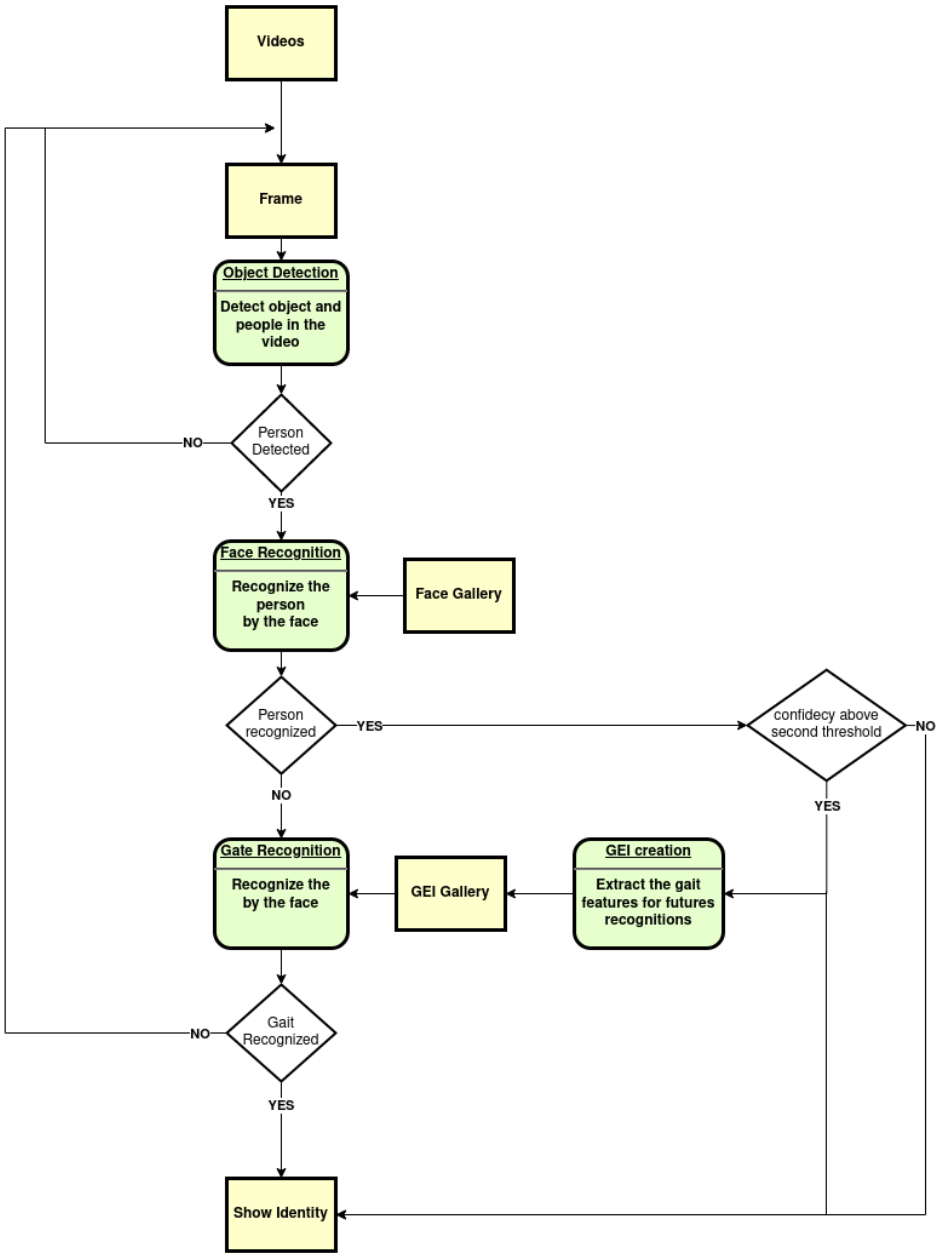


Figure 5: Structure of the software

The goal of the program is to identify the people in an open-set environment through the face traits and the gait.

The program receive as input one or more videos and a gallery of faces with their identities. It will parse the video and analyze each frame.

Each frame is passed to the object detection that will recognize if there are one or more

people. In case of positive result the box containing the person is passed to the face recognition that will extract the face features and than will compare them with the gallery.

The comparison will return a distance measure. If It is below the threshold the person is labeled as recognized. But we considered an other threshold, a little lower. If the distance is below this second threshold too, then the frame is elaborated and will be used to extract the gait features and the resulted GEI will be added to the gallery for the Gait Recognition. (To understand how the two threshold has been chosen read the evaluation section).

In case the face will not recognize the person, the frame is passed to the Gait recognition that will try to recognize it comparing the extracted features with the ones in the gallery. Is important to notice that the gait recognition need multiple frames to recognize the person.

The gait recognition, using a behavioral trait, is more prone to errors respect to face recognition. For this reason we give the priority to the latter. Furthermore, for the face recognition, we considered a lower threshold below which the GEI are saved, so we limit cascading errors.

4.1.1 Directory subdivision

```
project
├── data
│   ├── encodingsFace.pickle
│   ├── data0
│   └── ...
├── dataset
│   ├── person1
│   │   ├── img1.jpg
│   │   ├── gei0_0.jpg
│   │   └── ...
│   └── ...
├── deep_sort
│   ├── detection.py
│   ├── generate_detections.py
│   └── ...
├── detection
│   ├── __init__.py
│   ├── centroidtracker.py
│   ├── encodingFace.py
│   └── utils.py
├── icon
│   └── ...
├── input
│   ├── video1.mp4
│   └── ...
├── main.py
├── model_data
│   ├── coco
│   │   └── ...
│   └── ...
├── output
│   ├── video1.mp4
│   └── ...
├── report
│   ├── img
│   │   ├── img.jpg
│   │   └── ...
│   ├── video1.txt
│   └── ...
├── requirements.txt
├── tools
│   └── ...
├── venv
│   └── ...
└── yolov3
    └── ...
```

In the input folder can be added the video to analyze in mp4 format, after the execution of the program in the output folder you can find a video for each video in the input folder.

In the report folder the program will store a csv file for each video analyzed that contain the information of people reconized. Inside report/img/ there will be the images of recognized people through face recognition.

In dataset/ there are a folder for each person containing the face images that the program will need to recognize the person in the videos and it will store also the pictures of the GEI saved.

In data/ there are saved some binary files readeble only by the program, this files contain data neccessary for the recognition, like the GEI and the features extracted from the faces.

in tools/ and detection/ there are various libraries created and imported needed

The folders venv/, yolov3/ and model_data/ contains models and libraries for the DNN used in the system.

4.2 Object Detection

Before performing face recognition, the biometric system identifies a human figure (of a person) and then passes the box to facial recognition as an argument, this will allow the algorithm to perform facial recognition on a restricted region and not on the whole frame. To do this we will use the *YOLOv3 model* with the *DEEP Sort framework*

Therefore, it is important to perform an object detection (in this case the object is the person). When we go to detect an object, we detect an object in a frame, putting a bounding box or a mask around it, classifying the object. But tracing an object leads to numerous challenges, the most common below.

Occlusion Object occlusion in videos is one of the most common obstacles to seamless object tracking. For example, as shown in Figure 6, the challenge is to identify the same boy once the two individuals cross each other. So we must try to elaborate the character and characteristics of the individual of the next frame on a previous frame.

Figure 6: Source from www.researchgate.net

Camera not fixed When the camera used to track a particular object is also moving to the object, it can often lead to unwanted consequences. For example, many trackers consider the characteristics of an object to follow them. These trackers may not work in scenarios where the object looks different due to camera movement (the object would appear smaller or larger). A robust tracker may be able to track the following objects on its own (as is the case with drones).

4.2.1 Kalman Filter

In most of the problems of engineering, artificial vision, driving, navigation, etc. . . , the concept of "Kalman filter" recurs.

The Kalman filter is based on the idea of using available detections and previous predictions to arrive at the best guess of the current state, while maintaining the possibility of errors in the process.

For example, we can train a good enough YOLOv3 object detector that detects a person. To effectively track and predict a person's next state we assume a "constant velocity model". Now once we have defined the simple model according to the laws of physics, given a current survey, we can make a hypothesis about where the person will be in the next frame. But this idea presents a problem! Because we will have:

- Noise associated with the constant speed pattern that a person would follow. It is not possible that the same constant speed always occurs. This noise is called "process noise".
- The detector output may not be accurate, so you will have "measurement noise"

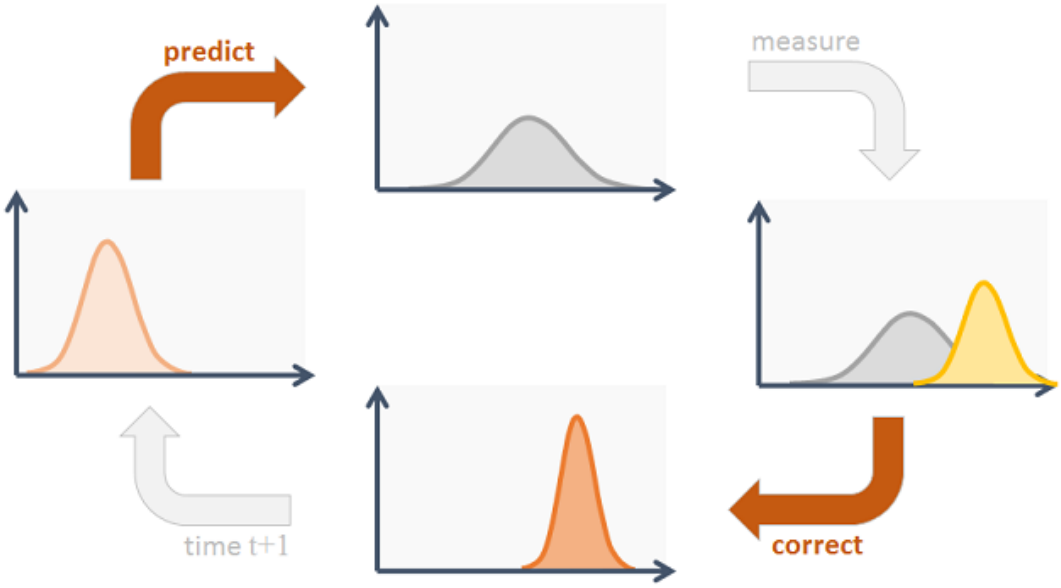


Figure 7: Source from www.researchgate.net

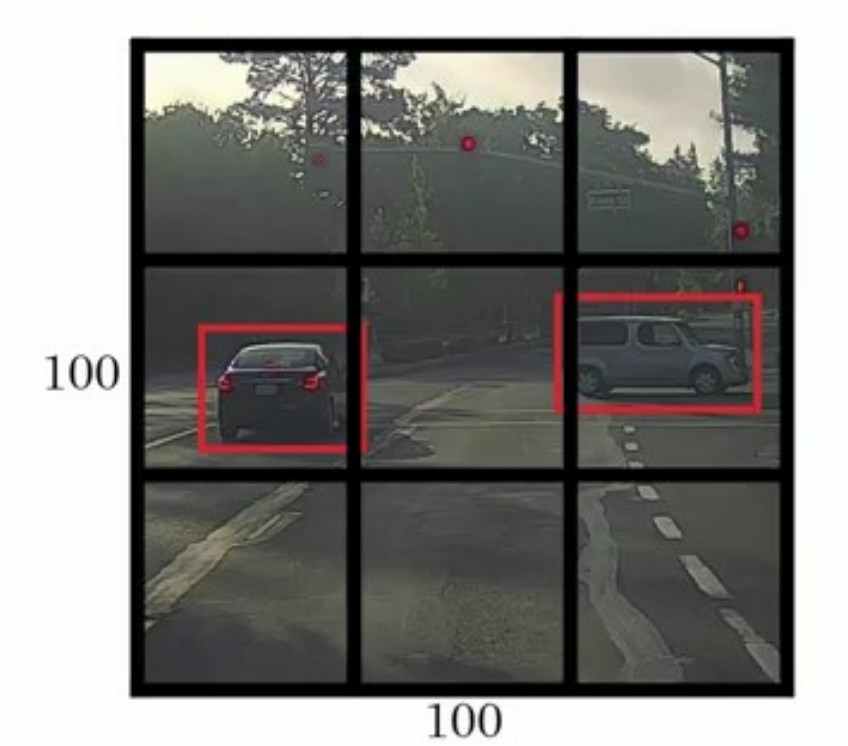
As shown in Figure 7, the Kalman filter has a recursive function. This function takes current readings (to predict current status), takes measurements, and updates forecasts. Hence, it essentially boils down to inferring a new distribution from the previous one.

4.2.2 Yolo

The YOLO (You Only Look Once) framework is one of the most used frameworks for object detection. It takes the entire image in a single instance and predicts the bounding

box coordinates and class probabilities for these boxes. The biggest advantage of using YOLO is its superb speed - it's blazingly fast and can process 45 frames per second. The function is quite simple [15]:

1. YOLO first takes an input image
2. The framework then divides the input image into grids (let's say a 3 X 3 grid)
3. Image classification and location are applied on each grid. YOLO then predicts bounding boxes and corresponding class probabilities for object



4.2.3 Deep Sort

The most widely used object tracking framework is Deep Sort. In the Deep Sort tracker, the Kalman filter is a crucial component.

The Kalman plot scenario is defined on the eight-dimensional state space (u, v, a, h, u ', v', a ', h') which contains the center position of the bounding box (u, v) , are the centers of the bounding boxes, a is the aspect ratio and h, the height of the image. The other variables are the respective speeds of the variables.

The Kalman filter helps us to account for noise in the detection and uses the previous state to predict a good fit for the bounding boxes.

For each survey, we create a "Track", which contains all the necessary status information. It also has a parameter for tracking and deleting tracks that had their last successful detection a long time ago, as those objects would have left the scene. To eliminate duplicate tracks, there is a minimum detection threshold for the first few frames.

Now, when we have drawn the new bounding boxes from the Kalman filter, the next problem is to associate the new detections with the new predictions. Since they are processed independently, here's the *assignment problem* because we have no idea how to associate **track_i** with **detection_k** detection.

To solve this problem, we need 2 things: a distance metric to quantify the association and an efficient algorithm to associate the data.

The Deep Sort authors decided to use the *Mahalanobis distance* squared (effective metric when it comes to distributions) to incorporate the uncertainties of the Kalman filter. Thresholding this distance can give us a very good idea of current associations. This metric is more accurate than Euclidean distance since we are actually measuring the distance between 2 distributions (all is a distribution under the Kalman filter). In this case, the authors offered to use the standard Hungarian algorithm, which is a very effective and simple combinatorial optimization algorithm that solves the assignment problem.

4.2.4 Implementation

In the following code [3], the YOLOv3 + Deep Sort model will be used for the tracker implementation.

Listing 1: Yolo + Deep Sort (for Tracker)

```
import os
import cv2
import numpy as np
import tensorflow as tf
from yolov3.yolov3 import Create_Yolov3
from yolov3.utils import load_yolo_weights, image_preprocess,
    postprocess_boxes, nms, draw_bbox, read_class_names
import time
from yolov3.configs import *

from deep_sort import nn_matching
from deep_sort.detection import Detection
```

```
12 from deep_sort.tracker import Tracker
13 from deep_sort import generate_detections as gdet
14
15 input_size = YOLO_INPUT_SIZE
16 Darknet_weights = YOLO_DARKNET_WEIGHTS
17 if TRAIN_YOLO_TINY:
18     Darknet_weights = YOLO_DARKNET_TINY_WEIGHTS
19
20 video_path   = "./IMAGES/test.mp4"
21
22 yolo = Create_Yolov3(input_size=input_size)
23 load_yolo_weights(yolo, Darknet_weights) # use Darknet weights
24
25 def Object_tracking(YoloV3, video_path, output_path, input_size=416,
       show=False, CLASSES=YOLO_COCO_CLASSES, score_threshold=0.3,
       iou_threshold=0.45, rectangle_colors='', Track_only = []):
26     # Definition of the parameters
27     max_cosine_distance = 0.5
28     nn_budget = None
29
30     #initialize deep sort object
31     model_filename = 'model_data/mars-small128.pb'
32     encoder = gdet.create_box_encoder(model_filename, batch_size=1)
33     metric = nn_matching.NearestNeighborDistanceMetric("cosine",
       max_cosine_distance, nn_budget)
34     tracker = Tracker(metric)
35
36     times = []
37
38     if video_path:
39         vid = cv2.VideoCapture(video_path) # detect on video
40     else:
41         vid = cv2.VideoCapture(0) # detect from webcam
42
43     # by default VideoCapture returns float instead of int
44     width = int(vid.get(cv2.CAP_PROP_FRAME_WIDTH))
45     height = int(vid.get(cv2.CAP_PROP_FRAME_HEIGHT))
46     fps = int(vid.get(cv2.CAP_PROP_FPS))
47     codec = cv2.VideoWriter_fourcc(*'XVID')
48     out = cv2.VideoWriter(output_path, codec, fps, (width, height)) #
       output_path must be .mp4
```

```
49
50      NUM_CLASS = read_class_names(CLASSES)
51      key_list = list(NUM_CLASS.keys())
52      val_list = list(NUM_CLASS.values())
53      while True:
54          _, img = vid.read()
55
56          try:
57              original_image = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
58              original_image = cv2.cvtColor(original_image, cv2.
    COLOR_BGR2RGB)
59          except:
60              break
61          image_data = image_preprocess(np.copy(original_image), [
    input_size, input_size])
62          image_data = tf.expand_dims(image_data, 0)
63
64          t1 = time.time()
65          pred_bbox = YoloV3.predict(image_data)
66          t2 = time.time()
67
68          times.append(t2-t1)
69          times = times[-20:]
70
71          pred_bbox = [tf.reshape(x, (-1, tf.shape(x)[-1])) for x in
    pred_bbox]
72          pred_bbox = tf.concat(pred_bbox, axis=0)
73
74          bboxes = postprocess_boxes(pred_bbox, original_image,
    input_size, score_threshold)
75          bboxes = nms(bboxes, iou_threshold, method='nms')
76
77          # extract bboxes to boxes (x, y, width, height), scores and
    names
78          boxes, scores, names = [], [], []
79          for bbox in bboxes:
80              if len(Track_only) !=0 and NUM_CLASS[int(bbox[5])] in
    Track_only or len(Track_only) == 0:
81                  boxes.append([bbox[0].astype(int), bbox[1].astype(int),
     bbox[2].astype(int)-bbox[0].astype(int), bbox[3].astype(int)-bbox
    [1].astype(int)])
```

```
                scores.append(bbox[4])
                names.append(NUM_CLASS[int(bbox[5])])

        # Obtain all the detections for the given frame.
        boxes = np.array(boxes)
        names = np.array(names)
        scores = np.array(scores)
        features = np.array(encoder(original_image, boxes))
        detections = [Detection(bbox, score, class_name, feature) for
    bbox, score, class_name, feature in zip(boxes, scores, names,
    features)]

        # Pass detections to the deepsort object and obtain the track
    information.
        tracker.predict()
        tracker.update(detections)

        # Obtain info from the tracks
        tracked_bboxes = []
        for track in tracker.tracks:
            if not track.is_confirmed() or track.time_since_update > 1:
                continue
            bbox = track.to_tlbr() # Get the corrected/predicted
    bounding box
            class_name = track.get_class() #Get the class name of
    particular object
            tracking_id = track.track_id # Get the ID for the
    particular track
            index = key_list[val_list.index(class_name)] # Get
    predicted object index by object name
            tracked_bboxes.append(bbox.tolist() + [tracking_id, index])
     # Structure data, that we could use it with our draw_bbox function

        ms = sum(times)/len(times)*1000
        fps = 1000 / ms

        # draw detection on frame
        image = draw_bbox(original_image, tracked_bboxes, CLASSES=
    CLASSES, tracking=True)
        image = cv2.putText(image, "Time: {:.1f} FPS".format(fps), (0,
    30), cv2.FONT_HERSHEY_COMPLEX_SMALL, 1, (0, 0, 255), 2)
```

```
113
114         # draw original yolo detection
115         #image = draw_bbox(image, bboxes, CLASSES=CLASSES, show_label=
    False, rectangle_colors=rectangle_colors, tracking=True)
116
117         #print("Time: {:.2f}ms, {:.1f} FPS".format(ms, fps))
118
119         if output_path != '': out.write(image)
120         if show:
121             cv2.imshow('output', image)
122             if cv2.waitKey(25) & 0xFF == ord("q"):
123                 cv2.destroyAllWindows()
124                 break
125
126     cv2.destroyAllWindows()
127
128 Object_tracking(yolo, video_path, '', input_size=input_size, show=True,
    iou_threshold=0.1, rectangle_colors=(255,0,0), Track_only = [])
```

Where is it:

- *Yolo* is the YOLOv3 model
- *Video_path* is the video path
- *Track_only* is used to define the object to track (For example ["person"] to track only people)
- *Model_filename* is a CNN pre-trained *deep_SORT* monitoring model
- *Tracker(metric)* is an object of the *deep_sort* library, which deals with the creation, maintenance of the track and the eventual cancellation of all the tracks

Finally, the detections are transmitted to the *deep_sort* to extrapolate the trace, with 2 lines:

Listing 2: Yolo + Deep Sort (for Tracker)

```
1 tracker.predict ()
2 tracker.update (detections)
```

4.3 Face Recognition

The implementation of the biometric system for face recognition is based on the technique of deep metric learning (DML). This method aims to train a CNN-based non-linear feature extraction module (or an encoder such as hog), which embeds the extracted image features that are semantically similar, in close positions while pushing away dissimilar image features using an appropriate distance metric (e.g. Euclidean) [12].

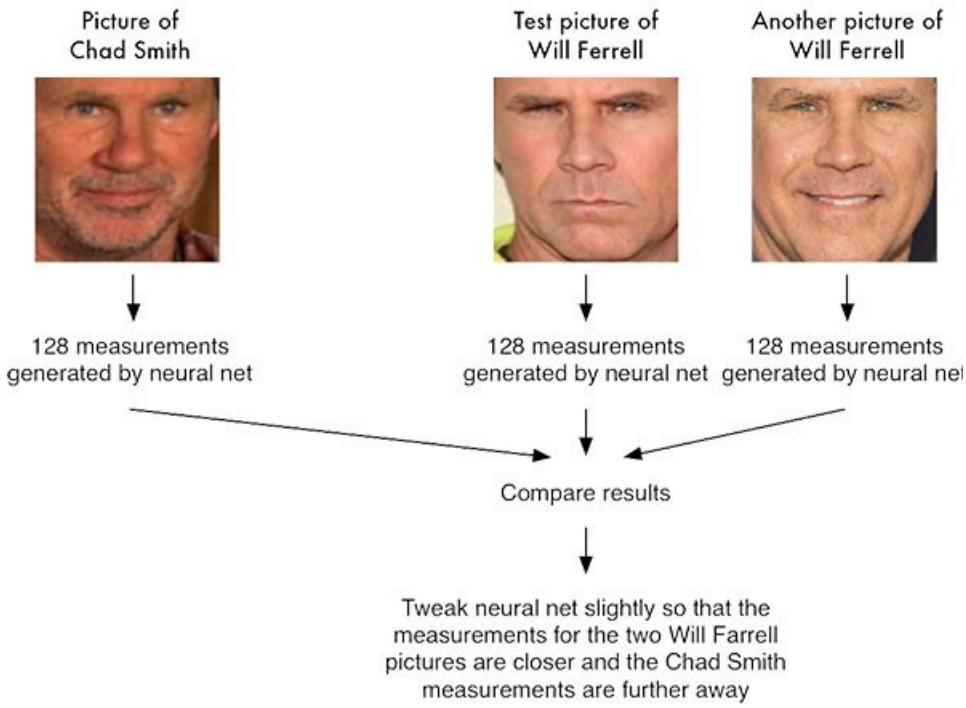


Figure 8: Comparison of incorporations of 128-d

In the implementation of face recognition, a network architecture for face recognition based on ResNet-34 will be used, with fewer layers and number of filters than the traditional cnn method. The neural network will quantify the faces, constructing the embedding (i.e. quantification) of 128-d each. In this way, the measurements of two images of the same person will have an almost similar weight, but further away from the measurement of a third person (Figure 14).

To implement deep metric learning, the dlib library, managed by David King, will be used, which will allow us to build visa embeds for the face recognition process with an identification probability of 99.38% [7]. Finally, the face_recognition library, created by Adam Geitgey, built around the dlib library to carry out and simplify the identification process [6].

Below is an outline of the processes that will be dealt with later

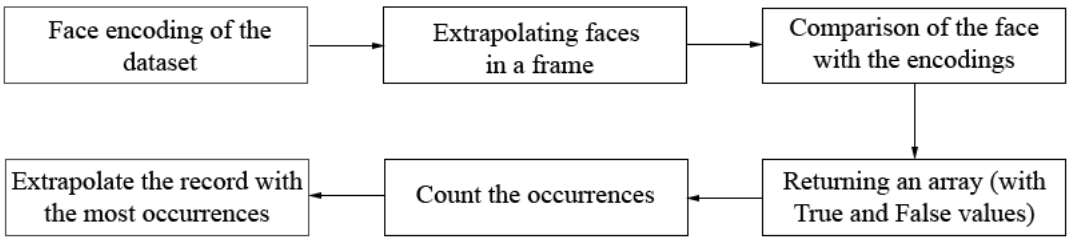


Figure 9: From face coding to face recognition

4.3.1 Face coding

The first step before performing face recognition is to build 128-d embeds for each of the faces in our dataset. In this case, we will use a trained black network to create 128-d embeddings (Figure 10).

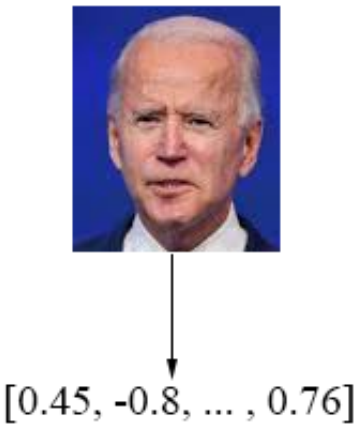


Figure 10: 128-d incorporation construction

In the following code, the algorithm will construct a 128-d vector for each face (image) present in the dataset. Then it will do the following:

- Read an image
- Convert the color space to RGB (given the numerous formats RGB, BGR, HSV, ...)
- Use the face_locations function to extrapolate the x and y coordinates of faces with the cnn method (this method is advisable to use it if you have a powerful computer and do not consider using it in real time, otherwise it is advisable to use the "hog" method)
- Compute faces by building 128-d vectors

- Insert the vectors in the knownEncodings array (and then stores them in a pickle file) and the names in the knownNames array (extrapolated from the names of the folders that contain the photos of each person)

Listing 3: Face coding

```
# initialize the list of known encodings and known names
knownEncodings = []
knownNames = []

# loop over the image paths
for (i, imagePath) in enumerate(imagePaths):
    # extract the person name from the image path
    print("[INFO] processing image {}/{}".format(i + 1,
                                                 len(imagePaths)))
    name = imagePath.split(os.path.sep)[-2]
    # load the input image and convert it from BGR (OpenCV ordering
    # )  to dlib ordering (RGB)
    image = cv2.imread(imagePath)
    rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

    # detect the (x, y)-coordinates of the bounding boxes
    # corresponding to each face in the input image
    boxes = face_recognition.face_locations(rgb,
                                            model="cnn")
    # compute the face embedding for the face
    encodings = face_recognition.face_encodings(rgb, boxes)

    # loop over the encodings
    for encoding in encodings:
        # add each encoding + name to our set of known names and
        # encodings
        knownEncodings.append(encoding)
        knownNames.append(name)
```

4.3.2 Face recognition

Face recognition, using the face_recognition library, will allow us to simplify the process of identifying and using a more efficient K-nearest neighbors. This algorithm allows to compare the face just identified with those present in the dataset.

For facial recognition, two algorithms were used that will be compared in terms of performance:

1. Based on *occurrences*: that is, the individual with more "images" present in the database that exceed the threshold is the identified individual.

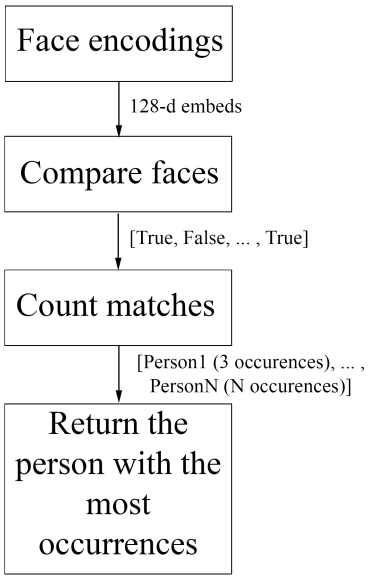
2. Based on the *highest score*: The individual who has the highest similarity score is the person identified.

The algorithms will both be evaluated because they have a noticeable difference. The occurrence-based algorithm has been implemented to avoid that a single similarity can "win" against dozens of "similarities" of an individual. For example, the person in a moving video may happen to have a high resemblance (for a photo) to an erroneous individual. Unlike the second algorithm which is more stable in the case of high quality frames and frontal photos.

Figure 11: Algorithm 1

Figure 12: Algorithm 2

By comparing both algorithms, you can see that algorithm 1 is more accurate (a video was used as input) since Zac Efron has more images with Anne Hathaway similarities, but not the highest score. Unlike algorithm 2 which has the similarity to the highest confidence rate, but this induces an error in this case.
The implementations of the following algorithms will be shown below. The *occurrence-based* algorithm does the following:



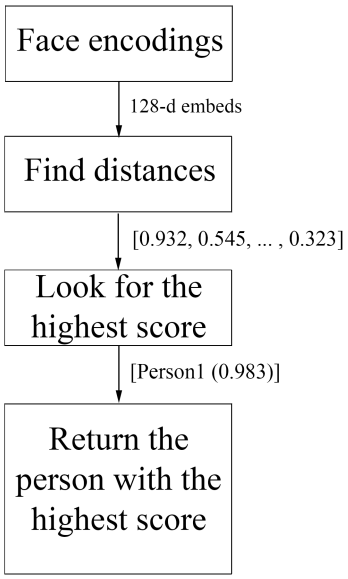
- Extract the x and y coordinates from the RGB image provided as an argument
- Encodes and builds 128-d embeds of newly recognized faces
- The compare_faces function will compare the detected faces with the faces present in our dataset. This function will return Boolean values (True and False) based on the Euclidian distance between the candidate and the saved encodings. If the Euclidian distance is greater than the predetermined threshold, the candidate has a possible similarity to the encoding saved in the dataset (and therefore will be assigned the value True). If the Euclidian distance is less than the threshold, the candidate has a low probability of being the person in the dataset (and therefore will be assigned the value False). If the person does not exist in the dataset, the candidate will be given the name unknown
- For each True value returned, it counts the occurrences of that person and saves them in a dictionary
- Extrapolates the result with the greatest number of occurrences. That record will be the biggest candidate

Listing 4: Face recognition (Algorithm 1)

```
boxes = face_recognition.face_locations(rgb, model="cnn")

encodings = face_recognition.face_encodings(rgb, boxes)
names = []

# loop over the face embeddings
for encoding in encodings:

    # attempt to match each face in the input image to our known
    # encodings
    matches = face_recognition.compare_faces(data["encodings"],
                                             encoding)
    name = "Unknown"
    # check to see if we have found a match
    if True in matches:
        # find the indexes of all matched faces then initialize a
        # dictionary to count the total number of times each face
        # was matched
```

```
19          matchedIdxs = [i for (i, b) in enumerate(matches) if b]
20          counts = {}
21          # loop over the matched indexes and maintain a count for
22          # each recognized face face
23          for i in matchedIdxs:
24              name = data["names"][i]
25              counts[name] = counts.get(name, 0) + 1
26          # determine the recognized face with the largest number
27          # of votes (note: in the event of an unlikely tie Python
28          # will select first entry in the dictionary)
29          name = max(counts, key=counts.get)
30
31      # update the list of names
32      names.append(name)
```

Unlike the occurrence-based algorithm, the second algorithm is based on the *highest confidence score* found:



- Extract the x and y coordinates from the RGB image provided as an argument
- Encodes and builds 128-d embeds of newly recognized faces
- The face_distance function will extract Euclidian distances greater than the threshold established between the probe and the database

- For each True value returned, it counts the occurrences of that person and saves them in a dictionary
- Extrapolates the result with the highest confidence score

Listing 5: Face recognition (Algorithm 2)

```
boxes = face_recognition.face_locations(rgb, model="cnn")

encodings = face_recognition.face_encodings(rgb, boxes)
names = []

# loop over the facial embeddings
for encoding in encodings:
    name = "Unknown"

    # Calculate the distance between the database and the probe
    face_distances = face_recognition.face_distance(data["encodings"],
                                                    encoding)
    score = 1

    # Find indexes of results
    matchedIdxs = [i for (i, b) in enumerate(face_distances) if b]

    # Loop to find the highest score match
    for i in matchedIdxs:
        if(face_distances[i] < x):
            if(score > face_distances[i]):
                score = face_distances[i]
                name = data["names"][i]

    # Insert into list the name of the individual found
    names.append(name)
```

4.4 Gait Recognition

The gait recognition use as input the box and the frame in which as been recognized a person by the object detection.
It need at least 30 frames to extract the features of the gait.
The implementation of the gait recognition follow this main steps for each frame:

- **Background removal**

  The first frame of the video is used as background, likely in the first frame should not appear any person.

  This image is removed from other frames using the OpenCV library, and the whole frame is converted in a black and white image.

- **Extraction of the gait features of the recognized person**

  the image of the person that as already been recognized by the neural net is used to generate the GEI as explained previously.

- **Save the GEI in the database or try to identify the person**

  If the person as already been recognized by the face recognition with an high confidence the GEI extracted are saved in the database.

  Otherwise, the GEI is compared with the ones already stored in the database to recognize the person. If the person is not recognized the GEI is temporally stored, in case the face recognition recognize the person's identity at a later time.

  The information of the GEI will be deleted if the person exit from the scene without been recognized by the face recognition.

Listing 6: Compare GEI with gait database (return a score between 0 and 1)

```
1 for q in range(num):  # loop all the gei
2         gei_to_com = gei[q, :, :]
3         score = np.exp(-(((gei_query[:] - gei_to_com[:]) / (128 * 88))
              ** 2).sum())
```

5 Evaluation

In order to evaluate the system in a better way and to decided the appropriate threshold for the final system we decided to evaluate the two main recognition system separately. Furthermore do a complete evaluation on the final system could be more difficult, because is quite impossible to find a good database with a lot of images of good quality that permit to analyze both face and gait of the same person.

5.1 Face Recognition

The face recognition as been evaluated using CelebA dataset [8] and youtube videos containing celebrities.
We have inserted in the gallery 104 people with 44 images for each and we have tested the face recognition over 25 youtube videos, with 13 people who were in the gallery and 25 impostors.
The face was evaluated with both the algorithm.

The algorithm based on the occurrences showed this results

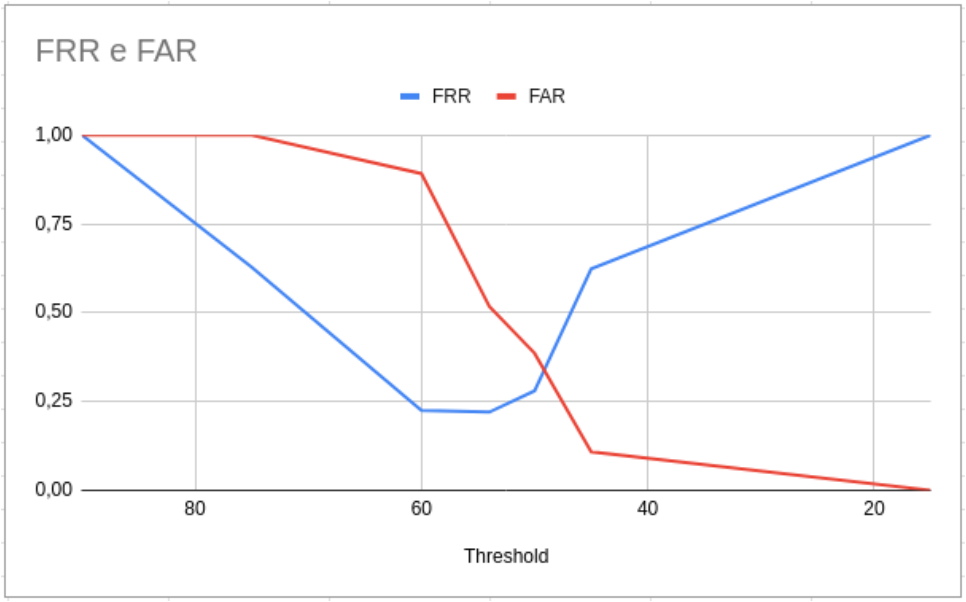


Figure 13: test for the face algorithm based on the occurrences

In this case the FRR is grow both when the distance is low or high and is lower only near the ERR. This is because the algorithm tend to recognize wrongly nearly all the people, because is possible that other people have more images in the gallery that are higher then a low threshold.

Instead the algorithm based on the best scores showed this results:

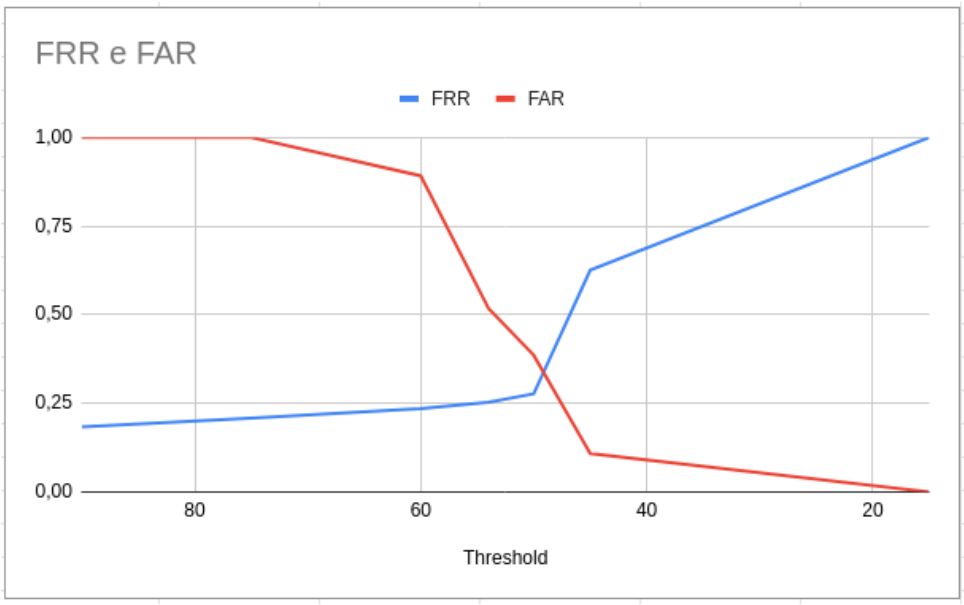


Figure 14: test for the face algorithm based on best scores

This results are more like a classic algorithm, the result seems quite good in with both algorithms.

5.2 Gait Recognition

The gait recognition has been evaluated using the CASIA Gait Database A [1].
The database contains 12 videos for 20 people that walk. We inserted the background given in the dataset as first frame for each video.
We extracted and inserted in the gallery the GEI taken from 5 videos (chosen randomly) for the first 10 people and from all 12 videos for the people 18, 19 and 20.
The remaining videos have been used as probes.
So the system should recognize 10 people over 17 without identify people as person 18, 19 or 20 despite their feature are saved in the gallery.
The test as been done first using the algorithm for the detection of the people without the deep learning.
The result was totally unusable.

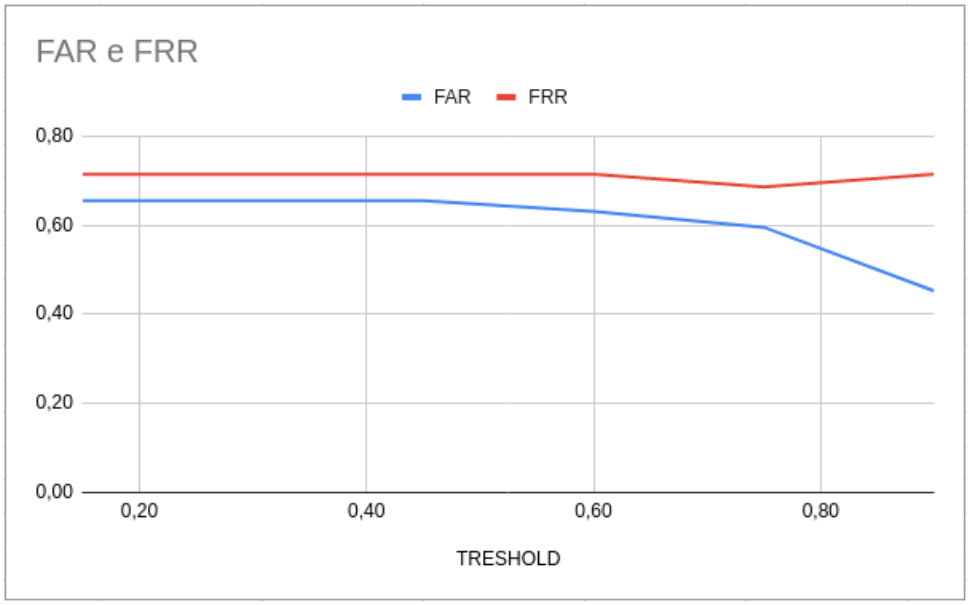


Figure 15: first test for gait recognition

After the test we decided to use an algorithm based on a deep neural network. This algorithm seems to recognize better people and help the gait algorithm to extract the GEI with a better quality.

As the figure suggest the system has very low performance, but can be used as a support to the face recognition. But we believe that the low quality of the images affected negatively the evaluation.

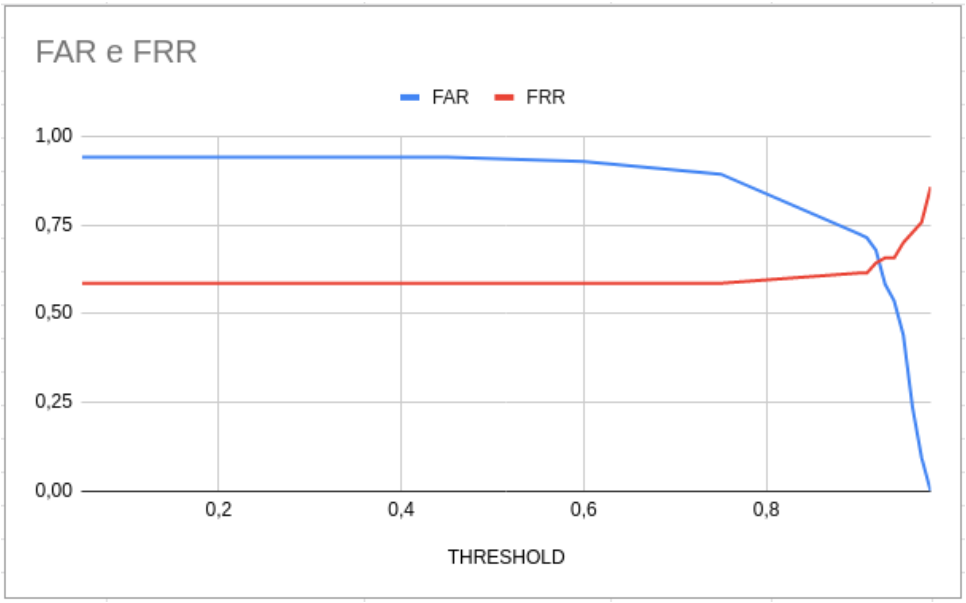


Figure 16: second test for gait recognition

Accordingly to the graph we chosen to use the EER as threshold: 0.92

Furthermore we have just done some tests to select the best number of frames on which calculate the GEI. This seems to be 30.

5.3 Final System

The complete system is hard to evaluate, we need surveillance videos with a good resolution, that permit to analyze the faces and that capture the whole body of the peoples to extract useful GEI, furthermore the cameras should be motionless, to allow the software to remove the background and extract the GEI.

We found no dataset that provided video of acceptable quality with the identities and the faces of the people recorded. So we will do more test in future.

At the moment we used some videos of the database *Piropo* [2], where seems that some people appearing in the videos are the author itself, so we extracted some publicly available picture of them.

In the final system, we opted for a strategy to decrease the error threshold for saving an individual's walk, this will allow not to associate the walk with the wrong individual (but the walk will be saved if the face recognition threshold is high). The implementation of this strategy was carried out using two thresholds for facial recognition: the *minimum*

threshold and the *optimal threshold*. The two thresholds will be used for:

- **Minimum threshold**: It is the threshold with the lowest value so that it is possible to carry out a face recognition. If the probe has a greater confidence than the lower threshold, the face recognition will be saved (in the csv file) and analyzed with respect to the second threshold.
- **Optimal threshold**: It is the threshold with the greatest value so that it is possible to ascertain the identity of the individual with greater accuracy. If the probe has a confidence value greater than the optimal threshold, the individual's walk will be associated with the identified person.

For a choice dependent on the low quality resolution of the videos, we opted to choose these thresholds:

1. *Minimum threshold*: **0.60**
2. *Optimal threshold*: **0.52**
3. *GEI threshold*: **0.92**

Below, the result of the *csv* file showing each recognition (face or gait) in the input videos:

Class	Id	Name	Face Image	Gait Image	Video Time
Person	1	Carlos	report/person1(432493236).jpg		0:00:04
Person	2	John		dataset/John/gei0_49	0:01:05
Person	2	John	report/person2(323232243)		0:01:15

The above attributes indicate:

- *Class*: The class of the recognized object
- *Identifier*: The identification number
- *Name*: Name of the person
- *Face Image*: Image of the person (when the face is recognized)
- *Gait Image*: Image of the person (when the gait is recognized)
- *Video Time*: Milliseconds in the video

From the tests carried out on the database, the face recognition do a good job, and identify them despite the video quality.

The gait recognition recognized the person when they was too far from the camera and when the face was covered, but as we have seen the FAR is very high. But when the person is easily recognizable by his face the recognition is more accurate and the final result is improved.

We believe that with many videos the results can improve adding informations for the gait recognition.

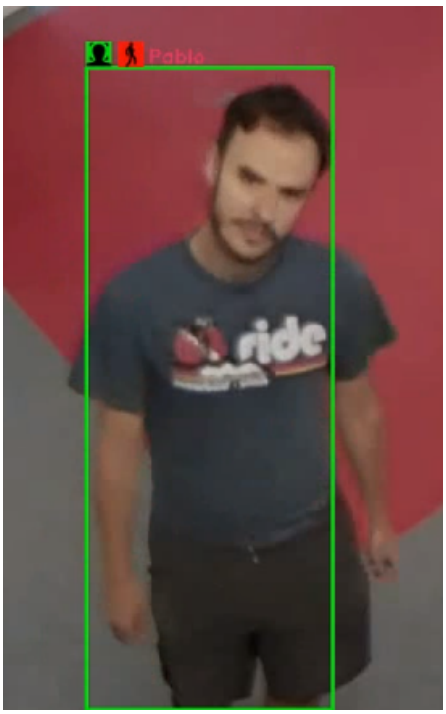
(a) Carlos correctly recognized by the gait recognition

(b) Carlos identified by the face recognition too

Figure 17: Carlos was identified correctly by both recognition systems and a new GEI will be created

Using a double threshold reduces an incorrect association of gait to the wrong individual, improving performance and reliability!

(a) Pablo identified as Carlos by the gait recognition



(b) Pablo identified by the face recognition

Figure 18: The face recognition corrected the wrong result of the gait, so the correct GEI will be added

6 Conclusion

The project developed have a good potential usage, but the gait could be improved using some other technique based on temporal traits of the gait and the whole system should be tested in a real environment.

There could be some good application inside offices and in general inside buildings, where the employers can be identified and localized and the strangers could be noticed if detected in area where they are not allowed to enter.

The system could be used to find a person that wont to be recognized trying to cover his face, like criminals. For example could be very useful in banks or shops where, usually, thieves inspect the place acting like a normal customer, so the system could analyze the face and record his gait, and identify them at the time of the robbery.

The system could be adapted and improved according to the use case.

References

[1] URL: http://www.cbsr.ia.ac.cn/english/Gait%20Databases.asp.

[2] URL: https://sites.google.com/site/piropodatabase/.

[3] Rokas Balsys. *YOLO v3 Real-Time Object tracking with Deep SORT*. 2020. URL: https://pylessons.com/YOLOv3-TF2-DeepSort/.

[4] Rama Chellappa, Ashok Veeraraghavan, and Narayanan Ramanathan. "Gait Biometrics, Overview". In: *Encyclopedia of Biometrics*. Ed. by Stan Z. Li and Anil Jain. Boston, MA: Springer US, 2009, pp. 628–633. ISBN: 978-0-387-73003-5. DOI: 10.1007/978-0-387-73003-5_33. URL: https://doi.org/10.1007/978-0-387-73003-5_33.

[5] Lukas Erhard. *Hausarbeit, Universität Bremen, Fachbereich 08, Soziologe*. URL: https://www.overleaf.com/latex/templates/hausarbeit-universitat-bremen-fachbereich-08-soziologe/kbndqymfsjtb.

[6] Adam Geitgey. *Face Recognition*. URL: https://github.com/ageitgey/face_recognition.

[7] Davis King. *High Quality Face Recognition with Deep Metric Learning*. 2017. URL: http://blog.dlib.net/2017/02/high-quality-face-recognition-with-deep.html.

[8] *Large-scale CelebFaces Attributes (CelebA) Dataset*. URL: https://mmlab.ie.cuhk.edu.hk/projects/CelebA.html.

[9] Jing Luo et al. "Gait Recognition Using GEI and AFDEI". In: *International Journal of Optics* 2015 (2015), pp. 1–5. DOI: 10.1155/2015/763908.

[10] Alessio Mobilia and Andrea Tripoli. *XyperByte/WORS*. URL: https://github.com/XyperByte/WORS.

[11] *Research of Face Recognition with Fisher Linear Discriminant*. 2018.

[12] Adrian Rosebrock. *Face recognition with OpenCV, Python, and deep learning*. 2018. URL: https://www.pyimagesearch.com/2018/06/18/face-recognition-with-opencv-python-and-deep-learning/.

[13] Sumit Saha. *A Comprehensive Guide to Convolutional Neural Networks - the ELI5 way*. 2018. URL: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53.

[14] SudeepSarkar sarkar and Zongyi Liu. “Gait Recognition”. In: *Encyclopedia of Cryptography and Security*. Ed. by Henk C. A. van Tilborg and Sushil Jajodia. Boston, MA: Springer US, 2011, pp. 503–506. ISBN: 978-1-4419-5906-5. DOI: 10.1007/978-1-4419-5906-5_741. URL: https://doi.org/10.1007/978-1-4419-5906-5_741.

[15] Pulkit Sharma. *A Practical Guide to Object Detection using the Popular YOLO Framework – Part III (with Python codes)*. 2018. URL: https://www.analyticsvidhya.com/blog/2018/12/practical-guide-object-detection-yolo-framewor-python/.

[16] James Wayman et al. “Biometric Systems”. In: *Technology, Design and Performance Evaluation*. Springer US, 2005. ISBN: 1-85233-596-3.

It was used the LaTeXtemplate of *Lukas Erhard* [5] and it has been modified to meet the needs.