

2021/2022



**esprit** Se former autrement

# Final Report: Data Science Internship Project

# Biometric solution for 2D Faces captured in non-controlled environments

# Realized by:

BELAIDI Nada

# MAGHREBI Racha

# CHARAABI Khaled

**Tutor:  
Nefissa Khiari**

ÉCOLE SUPÉRIEURE PRIVÉE D'INGÉNIERIE ET DE TECHNOLOGIES



Délivrée par la  
Commission  
des Titres  
d'Innovation



## **Summary:**

### **Chapter 1 :**

#### **General introduction**

<u>1- General Introduction</u>	2
<u>2- Project context</u>	
<u>2.1- Methodology</u>	6
<u>2.2- Stack</u>	6
<u>a- JupyterLab</u>	7
<u>b-Google Colab</u>	8
<u>3- Business objectives:</u>	8
<u>4- Data Science objectives:</u>	9
<u>Data Science Project Steps</u>	9
<u>Data Science Objectives</u>	9

### **Chapter 2 : Data Comprehension and preparation**

<u>1-Introduction</u>	
<u>1 -Data Preparation</u>	
<u>1-2-Data understanding</u>	12
<u>2-2- Data preprocessing</u>	13
<u>a-Data visualization</u>	13
<u>b- Preparation of inter and intra class comparison lists</u>	14

### **Chapter 3 : Data Modeling**

<u>1-face detection</u>	
<u>1-1-Cascade classifier</u>	16
<u>1-2-MTCNN</u>	18
<u>2-face alignment</u>	20
<u>3-face recognition</u>	
<u>3-1-Facenet</u>	21
<u>3-2-SVM</u>	25

### **Chapter 4: Comparing and discussing results**

<u>1-face detection</u>	28
<u>2-face recognition</u>	29
<u>3-conclusion</u>	31

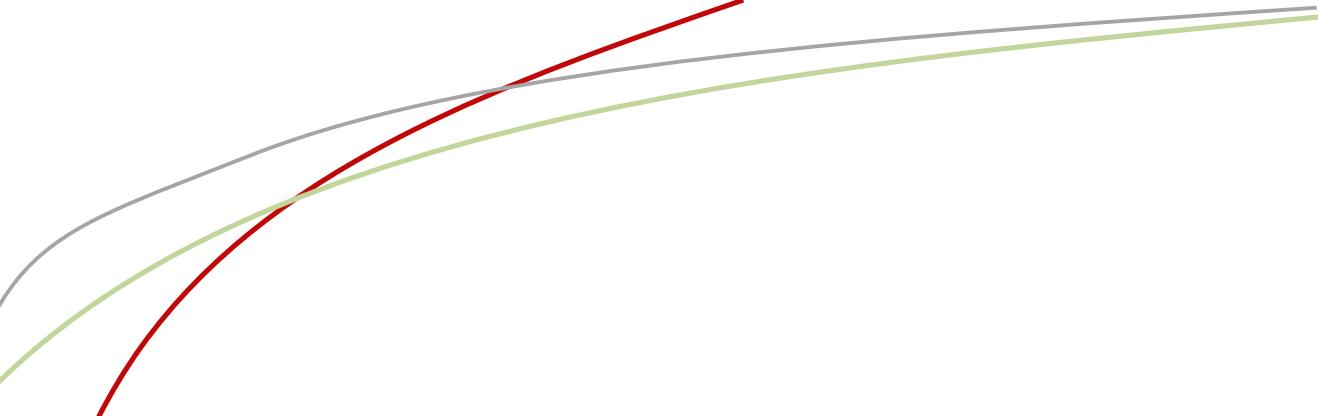
<u>General conclusion</u>	31
---------------------------	----



## General introduction

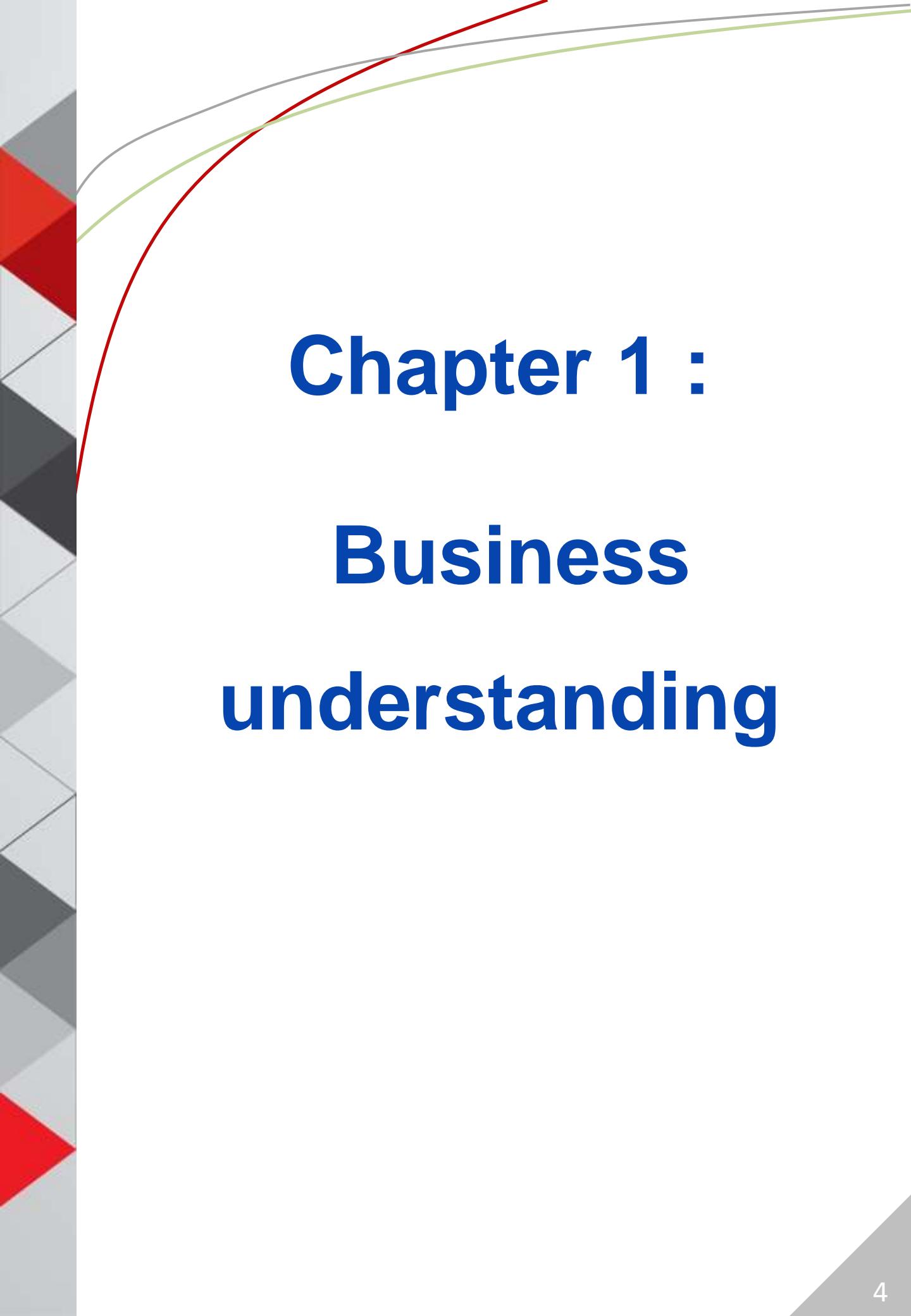
Facial recognition is a biometric technology that uses distinguishable facial features to identify a person. Allied Market Research expects the facial recognition market to grow to \$9.6 billion by 2022. Today, it's used in a variety of ways from allowing you to unlock your phone, go through security at the airport, purchase products at stores and in the case of entertainer and musician Taylor Swift it was used to identify if her known stalkers came through the gate at her Rose Bowl concert in May 2018.

Today, we are inundated with data of all kinds, but the plethora of photo and video data available provides the dataset required to make facial recognition technology work. Facial recognition systems analyze the visual data and millions of images and videos created by high-quality Closed-Circuit Television (CCTV) cameras installed in our cities for security, smartphones, social media, and other online activity. Machine learning and artificial intelligence capabilities in the software map distinguishable facial features mathematically, look for patterns in the visual data, and compare new images and videos to other data stored in facial recognition databases to determine identity.



One of the major advantages of facial recognition technology is safety and security. Law enforcement agencies use the technology to uncover criminals or to find missing children or seniors. In New York, police were able to apprehend an accused rapist using facial recognition technology within 24 hours of an incident where he threatened a woman with rape at knifepoint. In cities where police don't have time to help fight petty crime, business owners are installing facial-recognition systems to watch people and identify subjects of interest when they come in their stores.

Airports are increasingly adding facial recognition technology to security checkpoints; the U.S. Department of Homeland Security predicts that it will be used on 97 percent of travelers by 2023. When people know they are being watched, they are less likely to commit crimes so the possibility of facial recognition technology being used could deter crime.



# **Chapter 1 :**

# **Business**

# **understanding**

## **1-Introduction:**

During this chapter, we will be going through the business comprehension by detailing the problematic, raising the business questions and ultimately turning them into data science goals, then we are going to present the methodology of the project

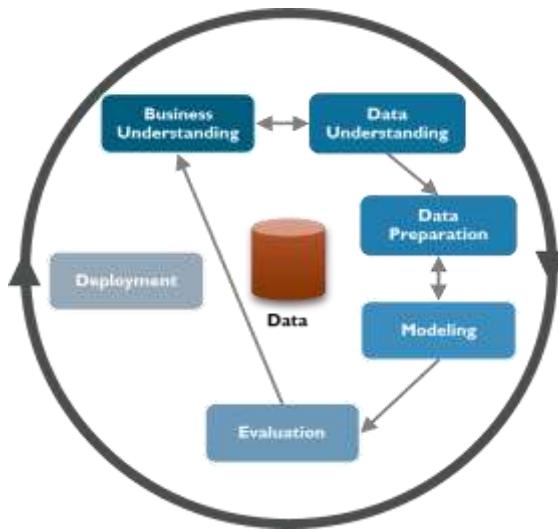
## **2- Project Context:**

As part of our data science internship project, we have to develop a solution for automatic recognition of 2D captured faces in uncontrolled environments.

### **2.1- Methodology:**

To ensure the realization of our project we opted for CRISP methodology

Cross Industry Standard Process is a Data Mining process model developed by IBM in the late 1990s and is considered today as among the best solutions for managing Data Science projects



## 2.2- Stack:

We used several technologies and libraries like Google Colab, JupyterLab, Python,pandas, scikit-learn, TensorFlow, keras, numpy,pandas,xgboost ...

### a- JupyterLab:

**JupyterLab** is a next-generation web-based user interface for Project Jupyter. JupyterLab enables you to work with documents and activities such as Jupyter notebooks, text editors, terminals, and custom components in a flexible, integrated, and extensible manner.



Picture 1: JupyterLab

### c- Google Colab:

**Google Colab or Colaboratory** is a cloud service, offered by Google (free), based on Jupyter Notebook and intended for training and research in machine learning. This platform allows you to train machine learning models directly in the cloud. So without needing to install anything on our computer except a browser. Cool, isn't it? Before presenting this magnificent service, we will recall what a Jupyter Notebook is



Picture 3 : Google-Colab

### 3- Business objectives:

Our application:

- Helps find missing people and identify perpetrators
- Protects businesses against theft
- Strengthens security measures in banks and airports
- Makes shopping more efficient
- Reduces the number of touchpoints
- Improves photo organization
- Improves medical treatment
- Saves time

## **Description of the specific objective selected:**

Develop a solution for automatic recognition of 2D captured faces in uncontrolled environments. The challenge is to maintain a high recognition rate despite the conditions.

## **4- Data Science**

### **Data Science Project Steps:**

Feature engineering	Data understanding.
Data preparation	Preparation of inter and intra class comparison lists.
Data Modeling	Classification
	Scoring.

## **4- Data Science objectives:**

Data Science Project Steps:

Data Science Objectives :

- Developing a solution for automatic recognition of 2D captured faces in uncontrolled environments
  - maintain a high recognition rate despite the conditions
  - Recognizing correctly detected faces
  - Modernize the environment
- Identifying the suitable technologies for our business objectives.
  - Training fast and efficient Deep Learning models.

Key Results :

- Using MTCNN for face detection.
- Using Cascade classifier for face detection.
- Using SVM for face recognition.
- Using Facenet for face recognition.

## **5. Conclusion:**

During this chapter we realized the first phase of the project which consists of business understanding and data science objectives.

# Chapter 2 :

# Data

# Comprehension

# And preparation

## **1- Introduction:**

During this chapter we are going to understand and prepare data for the modeling

## **2- data preparation:**

### 2-1- data understanding:

The dataset we will be using for this project is the LFW database. It's the most popular uncontrolled face database. It consists of 13233 images from 5749 individuals. All the images are captured from the internet. The number of images per person varies from 1 to 530. The images contain large variations in pose, illumination, expression, gender, age, etc.

Each picture is centered on a single face, and every image is encoded in RGB. The original images are of the size 250 x 250. The dataset contains 1680 directories, each representing a celebrity. Each directory has 2-50 images for the celebrity.

Link to the dataset: <http://vis-www.cs.umass.edu/lfw/>



## 2-2 Data Preprocessing:

### a- Data visualization:

The existence of 143 people who have more than 10 photos going up to 530 photos which will lead to the overfitting of the models from where we decide to eliminate them.

we will stick to:

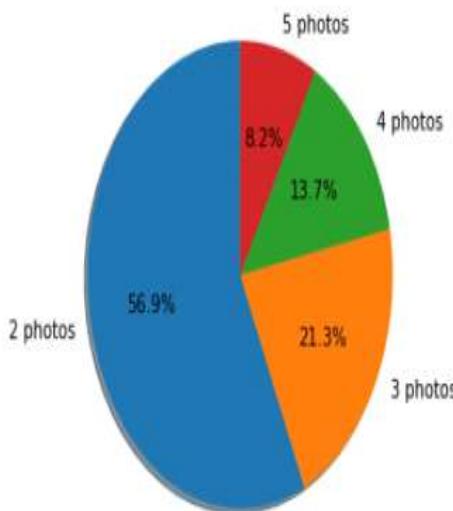
**categorie 2 photos : 779 personnes**

**categorie 3 photos : 291 personnes**

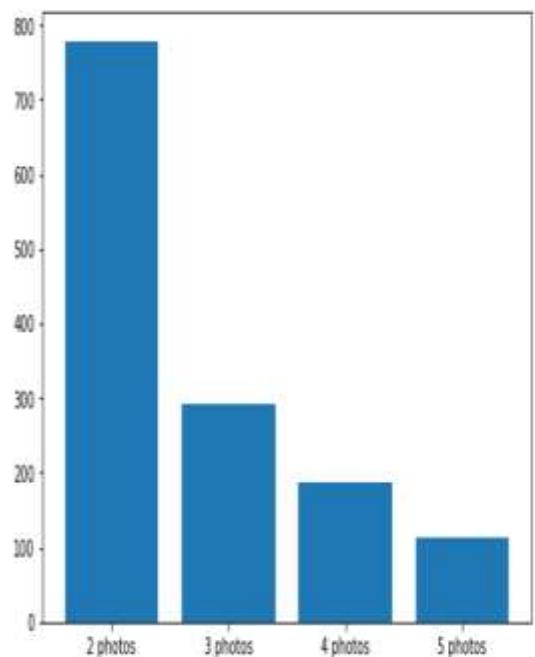
**categorie 4 photos : 187 personnes**

**categorie 5 photos : 112 personnes**

```
ax1.axis('equal') # Equal aspect ratio ensures that p  
plt.show()
```



```
plt.show()
```



## b- Preparation of inter and intra class comparison lists.

We organize the dataset in the form of 70% of data of each category in trainset and 30% of each category in testset.

These results are all the possible combinations for intra-class comparisons, but we must have the same number of intra-class comparisons as between classes. Thus, we extract randomly from intra\_train and intra\_test to have the same number of lines as inter\_train and inter\_test respectively.

```
] inter_train_trim
```

	id1	img1	id2	img2
14041	484	0.jpg	532	0.jpg
58	444	0.jpg	463	0.jpg
5309	50	0.jpg	115	0.jpg
18252	20	0.jpg	30	0.jpg
14564	382	0.jpg	448	0.jpg
...	...	...	...	...
15773	67	0.jpg	76	0.jpg
8954	441	0.jpg	474	0.jpg
15655	389	0.jpg	464	0.jpg
19726	314	0.jpg	341	0.jpg
7961	186	0.jpg	212	0.jpg

2699 rows × 4 columns

```
] inter_test_trim
```

	id1	img1	id2	img2
1331	72	0.jpg	81	0.jpg
211	116	0.jpg	161	0.jpg
1619	255	0.jpg	282	0.jpg
1288	358	0.jpg	374	0.jpg
398	81	0.jpg	109	0.jpg
...	...	...	...	...
1999	154	0.jpg	209	0.jpg
1207	265	0.jpg	340	0.jpg
1311	293	0.jpg	307	0.jpg
1648	245	0.jpg	280	0.jpg
1693	61	0.jpg	133	0.jpg

1195 rows × 4 columns

```
] intra_test
```

	id1	img1	id2	img2
0	676	0.jpg	676	4.jpg
1	676	0.jpg	676	2.jpg
2	676	0.jpg	676	3.jpg
3	676	0.jpg	676	1.jpg
4	676	4.jpg	676	2.jpg
...	...	...	...	...
1190	122	0.jpg	122	3.jpg
1191	122	0.jpg	122	1.jpg
1192	122	2.jpg	122	3.jpg
1193	122	2.jpg	122	1.jpg
1194	122	3.jpg	122	1.jpg

1195 rows × 4 columns

```
] intra_train
```

	id1	img1	id2	img2
0	206	0.jpg	206	1.jpg
1	1409	0.jpg	1409	1.jpg
2	1047	0.jpg	1047	1.jpg
3	1647	0.jpg	1647	1.jpg
4	444	0.jpg	444	4.jpg
...	...	...	...	...
2694	454	3.jpg	454	1.jpg
2695	400	0.jpg	400	2.jpg
2696	400	0.jpg	400	1.jpg
2697	400	2.jpg	400	1.jpg
2698	456	0.jpg	456	1.jpg

2699 rows × 4 columns

# Chapter 3 :

# Data

# Modeling

## 1. Introduction

Modeling is the phase in which our work begins to be more clear. After and building all the necessary data, the next step in this project is to create the most suitable model that will meet our business objectives.

In this section, the development of different techniques used will be discussed and explained thorough

### **1-Face Detection :**

#### 1-1 Cascade Classifier :

It is a machine learning based approach where a cascade function is trained from a lot of positive and negative images. It is then used to detect objects in other images.

Since we don't have any data to train this model we proceeded to use a pre-trained version of it imported from the OpenCV framework.

```
[1] import matplotlib.pyplot as plt
import cv2
import os
import fnmatch
from PIL import Image
import glob
import numpy as np
from pathlib import Path
```

```
trained_face_data = cv2.CascadeClassifier(cv2.data.haarcascades + 'haarcascade_frontalface_alt.xml')
trained_eye_data=cv2.CascadeClassifier(cv2.data.haarcascades+ 'haarcascade_eye.xml')
def detect_faces_eyes(image):
    height = 224
    width = image.shape[1]*height/image.shape[0]
    image = cv2.resize(image, (int(width), height), None, 0.5, 0.5, interpolation=cv2.INTER_AREA)
    cv2.waitKey(0)
    gray_img = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    face_coordinates = trained_face_data.detectMultiScale(gray_img)

    for (x, y, w, h) in face_coordinates:
        face = gray_img[y:y + h, x:x + w]
        eye_coordinates = trained_eye_data.detectMultiScale(face,scaleFactor=1.04,minNeighbors=3,flags=0, )
        colors = np.random.randint(1, 255, 3)
        cv2.rectangle(image, (x, y), (x + w, y + h), (int(colors[0]), int(colors[1]), int(colors[2])), thickness=2)
        for (x2, y2, w2, h2) in eye_coordinates:
            eye_colors = np.random.randint(1, 255, 3)
            eye_center = (x + x2 + w2 // 2, y + y2 + h2 // 2)
            eye_radius = int(round((w2 + h2) * 0.25))
            cv2.circle(image, center=eye_center, radius=eye_radius,
                       color=(int(eye_colors[0]), int(eye_colors[1]), int(eye_colors[2])))
    cv2.imshow(image)
    cv2.waitKey(0)
```

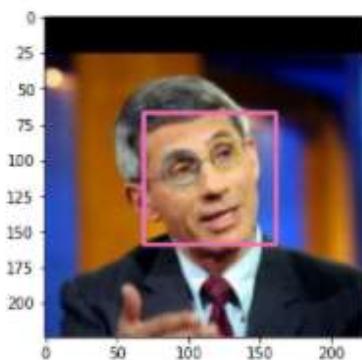
The scaleFactor parameter is used to determine how many different sizes of eyes the function will look for. Usually this value is 1.1 for the best detection. Setting this parameter to 1.2 or 1.3 will detect eyes faster but doesn't find them as often, meaning the accuracy goes down.

minNeighbors is used for telling the detector how sure he should be when detected an eye. Normally this value is set to 3 but if you want more reliability you can set this higher. Higher values means less accuracy but more reliability

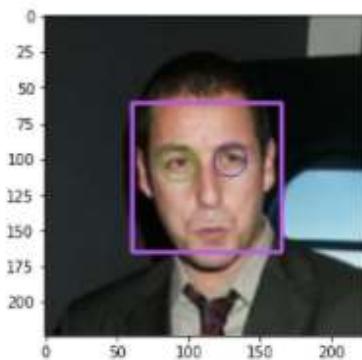
The flags are used for setting specific preferences, like looking for the largest object or skipping regions. Default this value = 0. Setting this value can make the detection go faster.

We set the parameters in a way that helped us get the best results possible, these are some of them:

```
[ ] img=cv2.imread("/content/drive/MyDrive/LFW/Face Dataset Train/100/1.jpg")
detect_faces_eyes(img)
```



```
[ ] img=cv2.imread("/content/drive/MyDrive/LFW/Face Dataset Train/10/0.jpg")
detect_faces_eyes(img)
```



## 1-2-MTCNN:

Multi-Task Cascaded Convolutional Neural Networks is a neural network which detects faces and facial landmarks on images.

The MTCNN algorithm works in three steps and use one neural network for each. The first part is a proposal network. It will predict potential face positions and their bounding boxes like an attention network in Faster R-CNN. The result of this step is a large number of face detections and lots of false detections. The second part uses images and outputs of the first prediction. It makes a refinement of the result to eliminate most of false detections and aggregate bounding boxes. The last part refines even more the predictions and adds facial landmarks predictions (in the original MTCNN implementation).

The first step should be installing MTCNN and TensorFlow:

```
[ ] pip install mtcnn
```

```
Collecting mtcnn
  Downloading mtcnn-0.1.1-py3-none-any.whl (2.3 MB)
    |████████| 2.3 MB 5.4 MB/s
Requirement already satisfied: keras>=2.0.0 in /usr/local/lib/python3
Requirement already satisfied: opencv-python>=4.1.0 in /usr/local/lib
Requirement already satisfied: numpy>=1.14.5 in /usr/local/lib/python
Installing collected packages: mtcnn
Successfully installed mtcnn-0.1.1
```

```
[ ] pip install --upgrade tensorflow
```

```
Requirement already satisfied: tensorflow in /usr/local/lib/python3.7/dist-packages (2.6.0)
Requirement already satisfied: grpcio<2.0,>=1.37.0 in /usr/local/lib/python3.7/dist-packages (from tensorflow)
Requirement already satisfied: tensorflow-extensions<2.7.4 in /usr/local/lib/python3.7/dist-packages
```

Importing the model and performing face detection on a random picture:

```
[ ] # confirm mtcnn was installed correctly
import mtcnn
import matplotlib.pyplot as plt
# print version
print(mtcnn.__version__)

0.1.8

[ ] path = '/content/drive/myDrive/TF/Face_Dataset_Train/100/1.jpg'

[ ] detector = mtcnn.MTCNN()
# detect faces in the image
pixels=plt.imread(path)
faces = detector.detect_faces(pixels)
for face in faces:
    print(face)

WARNING:tensorflow:5 out of the last 11 calls to <function Model.make_predict_function.<locals>.predict_function at 0x7F79e0233710> triggered tf.function retracing. Tracing is expensive and the excess
  'box': [35, 74, 82, 118], 'confidence': 0.9995027700532837, 'keypoints': {'left_eye': (108, 119), 'right_eye': (141, 107), 'nose': (131, 135), 'mouth_left': (122, 158), 'mouth_right': (152, 149)}
```

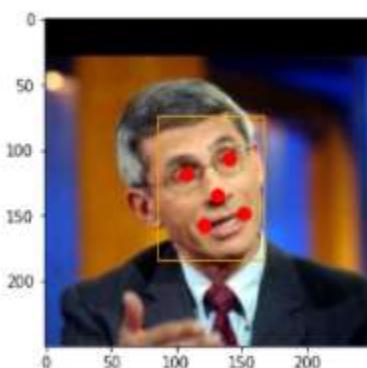
As we can see the result is a dictionary with each feature coordinates along with the face cropping box,

We proceeded then to building a function to draw the results on the image in order to give us a better look at our model performance.

```
[ ] def draw_facebox(path, result_list):
    # load the image
    data = plt.imread(path)
    # plot the image
    plt.imshow(data)
    # get the context for drawing boxes
    ax = plt.gca()
    # plot each box
    for result in result_list:
        # get coordinates
        x, y, width, height = result['box']
        # create the shape
        rect = plt.Rectangle((x, y), width, height, fill=False, color='orange')
        # draw the box
        ax.add_patch(rect)
        # draw the dots
        for key, value in result['keypoints'].items():
            # create and draw dot
            dot = plt.Circle(value, radius=5, color='red')
            ax.add_patch(dot)
        # show the plot
    plt.show()

# filename = 'test1.jpg' # filename is defined above, otherwise uncomment
# load image from file
# pixels = plt.imread(filename) # defined above, otherwise uncomment
# detector is defined above, otherwise uncomment
#detector = mtcnn.MTCNN()
# detect faces in the image
faces = detector.detect_faces(pixels)
# display faces on the original image
draw_facebox(path, faces)
```

# DISPLAY FACES ON THE ORIGINAL IMAGE  
draw\_facebox(path, faces)



## 2-Face alignment :

The results we got from MTCNN were more accurate so we used them for face alignment. We built a function from scratch.

This is a part of our alignment function:

```
[ ] def align_face(path, reference_pts=None,
                  crop_size=(96, 112),
                  align_type='smilarity'):
    src_img = plt.imread(path)
    detector = mtcnn.MTCNN()
    # detect faces in the image
    faces = detector.detect_faces(pixels)
    facial_pts = []
    for key, value in faces[0]['keypoints'].items():
        facial_pts.append(value)
    if reference_pts is None:
        if crop_size[0] == 96 and crop_size[1] == 112:
            reference_pts = REFERENCE_FACIAL_POINTS
        else:
            default_square = False
            inner_padding_factor = 0
            outer_padding = (0, 0)
            output_size = crop_size

            reference_pts = get_reference_facial_points(output_size,
                                                         inner_padding_factor,
                                                         outer_padding,
                                                         default_square)

    ref_pts = np.float32(reference_pts)
    ref_pts_shp = ref_pts.shape
    if max(ref_pts_shp) < 3 or min(ref_pts_shp) != 2:
        raise FaceWarpException(
            'reference_pts.shape must be (K,2) or (2,K) and K>2')
    if ref_pts_shp[0] == 2:
        ref_pts = ref_pts.T

    src_pts = np.float32(facial_pts)
    src_pts_shp = src_pts.shape
    if max(src_pts_shp) < 3 or min(src_pts_shp) != 2:
        raise FaceWarpException(
            'facial_pts.shape must be (K,2) or (2,K) and K>2')

    if src_pts_shp[0] == 2:
        src_pts = src_pts.T
```

And these are our results:

```
[[ 13,  21,  29],
 [ 7,  20,  26],
 [ 9,  19,  26],
 ...,
 [ 46,  94, 237],
 [ 45,  92, 238],
 [ 45,  91, 238]]], dtype=uint8)
```



### 3-Face Recognition :

After finishing the process of face detection , *recognition* is the attempt to confirm the identity of persons which is used for verification . We used two different models Facenet and SVM.

#### **3-1-Facenet:**

A one-shot model that directly learns a mapping from face images to a compact Euclidean space where distances directly correspond to a measure of face similarity.

Facenet uses triplets in ordre to learn face recognition, meaning it needs a similar (same person) and a dissimilar (different person ) for each image of the training set.

In order to make that possible we started with merging two of our pre-prepared lists :

```

✓ [9] df = pd.DataFrame()

✓ [10] #Generating triplets : img2_x is the positive one img2_y is the negative one
0s df = pd.merge(intratrain, intertrain, how='inner', on=['id1','img1'])

✓ [11] df
0s

      id1  img1  id2_x  img2_x  id2_y  img2_y
0       0  0.jpg      0   1.jpg      9  0.jpg
1       0  0.jpg      0   1.jpg     56  0.jpg
2       0  0.jpg      0   1.jpg      1  0.jpg
3       0  0.jpg      0   2.jpg      9  0.jpg
4       0  0.jpg      0   2.jpg     56  0.jpg
...
4777  98  0.jpg    98   2.jpg    174  0.jpg
4778  98  0.jpg    98   2.jpg    146  0.jpg
4779  98  0.jpg    98   2.jpg    145  0.jpg
4780  98  0.jpg    98   2.jpg    116  0.jpg
4781  98  0.jpg    98   2.jpg    135  0.jpg

```

Now that the dataframe is ready, we imported all the necessary libraries:

```
[ ] import torch
import torch.nn as nn
import torch.nn.functional as F
import torchvision
from torchvision import transforms, models
from torch.utils.data import Dataset, DataLoader
import torchvision.datasets as datasets
import random
import PIL
from PIL import Image
from pathlib import Path
import tqdm
from tqdm import tqdm_notebook
import pandas as pd
import glob
%matplotlib inline
```

And then we proceeded to build our data loader from scratch:

```
[ ] class MyDataset(Dataset):
    def __init__(self, path, df, num_triplets, tfms=None):
        self.path = path
        self.df = df
        self.num_triplets = num_triplets
        self.tfms = tfms
        self.training_triplets = self.generate_triplets(self.df, self.num_triplets)
    @staticmethod
    def generate_triplets(df, num_triplets):
        triplets = []
        for i in range(num_triplets):
            neg_id = df["img2_y"][i]
            pos_id = df["img2_x"][i]
            anc_id = df["img1"][i]
            pos_name = df["id1"][i]
            neg_name = df["id2_y"][i]
            triplets.append([anc_id, pos_id, neg_id, pos_name, neg_name])
        return triplets
    def __getitem__(self, i):
        anc_id, pos_id, neg_id, pos_name, neg_name = self.training_triplets[i]
        anc_img = (self.path / str(pos_name) / str(anc_id))
        pos_img = (self.path / str(pos_name) / str(pos_id))
        neg_img = (self.path / str(neg_name) / str(neg_id))
        anc_img = PIL.Image.open(anc_img)
        pos_img = PIL.Image.open(pos_img)
        neg_img = PIL.Image.open(neg_img)
        pos_name = torch.from_numpy(np.array([pos_name]).astype('long'))
        neg_name = torch.from_numpy(np.array([neg_name]).astype('long'))
        sample = [anc_img, pos_img, neg_img, pos_name, neg_name]

        if self.tfms:
            sample[0] = self.tfms(sample[0])
            sample[1] = self.tfms(sample[1])
            sample[2] = self.tfms(sample[2])
        return sample
```

Building our model and our loss function:

```
[ ] class Net(nn.Module):
    def __init__(self, emb_size, num_classes, pretrained=False):
        super(Net, self).__init__()
        self.model = models.resnet34(pretrained)
        self.emb_size = emb_size
        self.lin1 = nn.Linear(1000, emb_size)
        self.lin2 = nn.Linear(emb_size, num_classes)
    def l2_norm(self, input):
        input_size = input.size()
        buffer = torch.pow(input, 2)
        normp = torch.sum(buffer, 1).add_(1e-10)
        norm = torch.sqrt(normp)
        _output = torch.div(input, norm.view(-1, 1).expand_as(input))
        output = _output.view(input_size)
        return output
    def forward(self, x):
        x = self.model(x)
        x = self.lin1(x)
        self.features = self.l2_norm(x)
        alpha = 10
        self.features*=10
        return self.features
    def forward_classifier(self, x):
        features = self.forward(x)
        res = self.lin2(features)
        return res

class TripletLoss(nn.Module):
    def __init__(self, margin):
        super(TripletLoss, self).__init__()
        self.margin = margin
        self.pdist = PairwiseDistance(2)
    def forward(self, anc, pos, neg):
        pos_dist = self.pdist(anc, pos)
        neg_dist = self.pdist(anc, neg)

        hinge_dist = torch.clamp(self.margin+pos_dist-neg_dist, min=0.0)
        loss = torch.mean(hinge_dist)
        return loss
```

Building a transformer and the training function:

```
[ ] batch_size = 32
triplets=4782
num_workers = 0
train_tfms = transforms.Compose([
    transforms.Resize(256),
    transforms.RandomHorizontalFlip(),
    transforms.RandomCrop(224),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
])
```

```

for epoch in range(5):
    triplet_loss_sum = 0.0
    scheduler.step()
    model.train()
    torch.cuda.synchronize()
    for batch_idx, batch_sample in tqdm_notebook(enumerate(train_dl)):
        anc_img = batch_sample[0].to(device)
        pos_img = batch_sample[1].to(device)
        neg_img = batch_sample[2].to(device) ***

```

And then we proceeded to training:

```

[ ] 150/? [09:04<00:00, 2.84s/it]
/usr/local/lib/python3.7/dist-packages/torch/nn/functions.py:100: UserWarning: max_pool2d_implementation is deprecated, please use max_pool2d instead.
  return torch.max_pool2d(input, kernel_size, stride, padding,
Epoch 0
Loss: 0.01819251548765295
[ ] 150/? [04:10<00:00, 1.42s/it]
Epoch 1
Loss: 0.01646104187633342
[ ] 150/? [04:09<00:00, 1.42s/it]
Epoch 2
Loss: 0.015947358817522718
[ ] 150/? [04:09<00:00, 1.43s/it]
Epoch 3
Loss: 0.015764608630222637
[ ] 150/? [04:14<00:00, 1.45s/it]
Epoch 4
Loss: 0.015605942181392177

```

In order to test our model accuracy, we had to build another data loader and transformer with the same steps as the ones used with the training set and then feeded them to our model .

Our predictions first looked like this:

```

[ ] np.linalg.norm(out1.cpu().detach().numpy()-out2.cpu().detach().numpy(), axis=1)

array([0.23789228, 0.33970106, 0.37581587, 0.8266433 , 0.         ,
       0.         , 0.6529719 , 0.         , 0.         , 0.         ,
       0.0189697 , 0.38108963, 0.7768045 , 0.         , 0.         ,
       0.06506015, 0.         , 0.         , 0.7282962 , 0.6931193 ,
       0.8522287 , 0.         , 0.5727874 , 0.         , 0.73321635,
       0.5830866 , 0.30988678, 0.02236481, 0.         , 0.81673306,
       0.14445217, 0.         ], dtype=float32)

```

The zeros meant that the images weren't for the same person,  
We to make our prediction result look more understandable:

	<b>id1</b>	<b>img1</b>	<b>id2</b>	<b>img2</b>	<b>target</b>
2077	733	0.jpg	733	2.jpg	1
1362	1312	0.jpg	1312	2.jpg	1
1667	254	0.jpg	254	1.jpg	1
2038	71	0.jpg	71	2.jpg	1
718	175	0.jpg	199	0.jpg	0
...	...	...	...	...	...
1229	1183	0.jpg	1183	2.jpg	1
960	73	0.jpg	138	0.jpg	0
360	320	0.jpg	380	0.jpg	0
912	382	0.jpg	403	0.jpg	0
1601	1677	0.jpg	1677	1.jpg	1

2232 rows × 5 columns

The final step was calculating the accuracy:

```
[ ] def accuracy(df):
    x=0
    for i in range(2232):
        if (df["id1"][i]==df["id2"][i]) and (df["target"][i]==1):
            x=x+1
        if (df["id1"][i]!=df["id2"][i]) and (df["target"][i]==0):
            x=x+1
    return x/(2232)

[ ] accuracy(df1)
0.9762544802867383
```

## 2-SVM :

Algorithm generates a decision surface separating the two classes. For face recognition, we re-interpret the decision surface to produce a similarity metric between two facial images .

SVM is a classification algorithm that wouldn't work out very well unless we have big number of images per person, in this case we won't be using the similar/dissimilar pairs and the data we prepared previously since the number of images per person is between 2 and 5. We will be importing the LFW data set from sklearn.

```
▶ from sklearn.datasets import fetch_lfw_people

faces = fetch_lfw_people(min_faces_per_person=60)
print(faces.target_names)
print(faces.images.shape)

↳ Downloading LFW metadata: https://ndownloader.figshare.com/files/5976012
Download LFW metadata: https://ndownloader.figshare.com/files/5976009
Download LFW metadata: https://ndownloader.figshare.com/files/5976006
Download LFW data (~200MB): https://ndownloader.figshare.com/files/5976015
['Ariel Sharon' 'Colin Powell' 'Donald Rumsfeld' 'George W Bush'
 'Gerhard Schroeder' 'Hugo Chavez' 'Junichiro Koizumi' 'Tony Blair']
(1348, 62, 47)
```

We will be plotting a few of these faces to see what we're working with: These faces are already aligned and cropped.

```
▶ import matplotlib.pyplot as plt
fig, ax = plt.subplots(3, 5)
for i, axi in enumerate(ax.flat):
    axi.imshow(faces.images[i], cmap='bone')
    axi.set(xticks=[], yticks=[], xlabel=faces.target_names[faces.target[i]])
```

Colin Powell George W. Bush Gerhard Schroeder Hugo Chavez  
George W. Bush Junichiro Koizumi George W. Bush Tony Blair Ariel Sharon  
Donald Rumsfeld George W. Bush George W. Bush George W. Bush George W. Bush

Each image contains [62x47] or nearly 3,000 pixels. We could proceed by simply using each pixel value as a feature, but often it is more effective to use some sort of preprocessor to extract more meaningful features; here we will use a principal component analysis to extract 150 fundamental components to feed into our support vector machine classifier. We can do this most straightforwardly by packaging the preprocessor and the classifier into a single pipeline:

```
[ ] from sklearn.svm import SVC
from sklearn.decomposition import PCA
pca = PCA(n_components=150, whiten=True, random_state=42)
from sklearn.pipeline import make_pipeline

svc = SVC(kernel='rbf', class_weight='balanced')
model = make_pipeline(pca, svc)
```

Splitting data into testing and training sets:

```
[ ] from sklearn.model_selection import train_test_split
Xtrain, Xtest, ytrain, ytest = train_test_split(faces.data, faces.target,
                                              random_state=42)
```

Finally, we can use a grid search cross-validation to explore combinations of parameters. Here we will adjust C (which controls the margin hardness) and gamma (which controls the size of the radial basis function kernel), and determine the best model:

```
[ ] from sklearn.model_selection import GridSearchCV
param_grid = {'svc_C': [1, 5, 10, 50],
              'svc_gamma': [0.0001, 0.0005, 0.001, 0.005]}
grid = GridSearchCV(model, param_grid)

%time grid.fit(Xtrain, ytrain)
print(grid.best_params_)

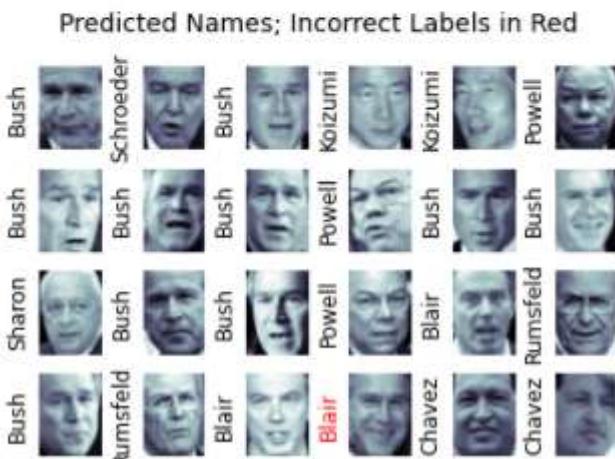
CPU times: user 1min 5s, sys: 38.7 s, total: 1min 43s
Wall time: 1min 1s
{'svc_C': 10, 'svc_gamma': 0.001}
```

The optimal values fall toward the middle of our grid; if they fell at the edges, we would want to expand the grid to make sure we have found the true optimum.

Now with this cross-validated model, we can predict the labels for the test data, which the model has not yet seen:

```
[ ] model = grid.best_estimator_
yfit = model.predict(Xtest)
```

```
[ ] fig, ax = plt.subplots(4, 6)
for i, axi in enumerate(ax.flat):
    axi.imshow(Xtest[i].reshape(62, 47), cmap='bone')
    axi.set(xticks=[], yticks[])
    axi.set_ylabel(faces.target_names[yfit[i]].split()[-1],
                  color='black' if yfit[i] == ytest[i] else 'red')
fig.suptitle('Predicted Names; Incorrect Labels in Red', size=14);
```



This was the accuracy we got using SVM, it is below facenet's accuracy.

```
from sklearn.metrics import classification_report
print(classification_report(ytest, yfit,
                            target_names=faces.target_names))
```

	precision	recall	f1-score	support
Ariel Sharon	0.65	0.73	0.69	15
Colin Powell	0.80	0.87	0.83	68
Donald Rumsfeld	0.74	0.84	0.79	31
George W Bush	0.92	0.83	0.88	126
Gerhard Schroeder	0.86	0.83	0.84	23
Hugo Chavez	0.93	0.70	0.80	20
Junichiro Koizumi	0.92	1.00	0.96	12
Tony Blair	0.85	0.95	0.90	42
accuracy			0.85	337
macro avg	0.83	0.84	0.84	337
weighted avg	0.86	0.85	0.85	337

# Chapter 4:

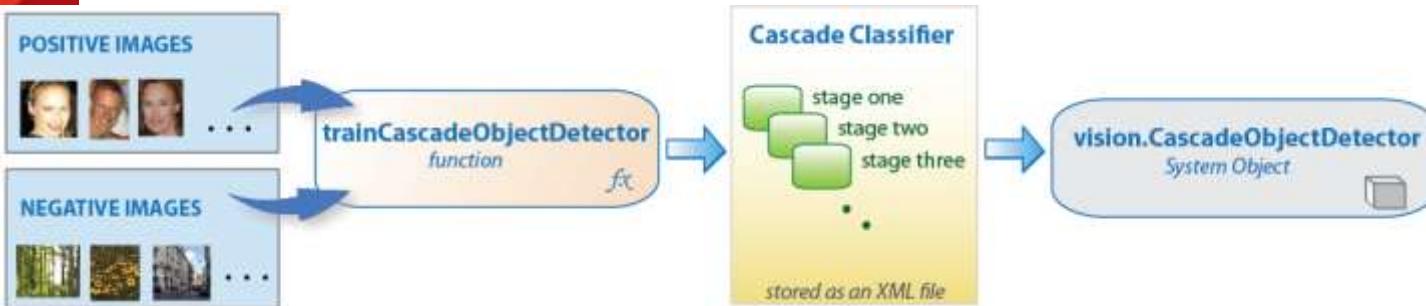
# Comparing and

# discussing

# results

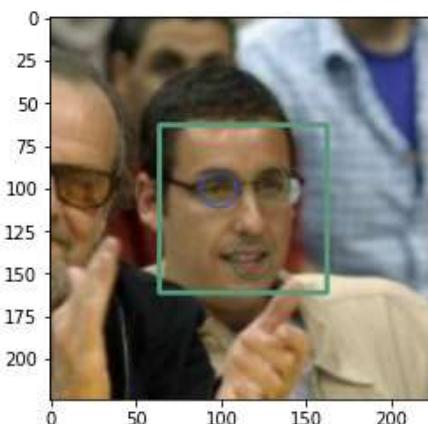
## 1-Face detection :

Cascade classifier relies on big sized datasets containing negative and positive images in order to be trained ( in our case positive : has a face in it, negative : doesn't contain any faces).



In our case it was able to detect faces perfectly, but not eyes. Especially when the face is tilted or when the person in the picture is wearing glasses.  
Here are some of the cases when Cascade classifier failed to predict the eyes position correctly.

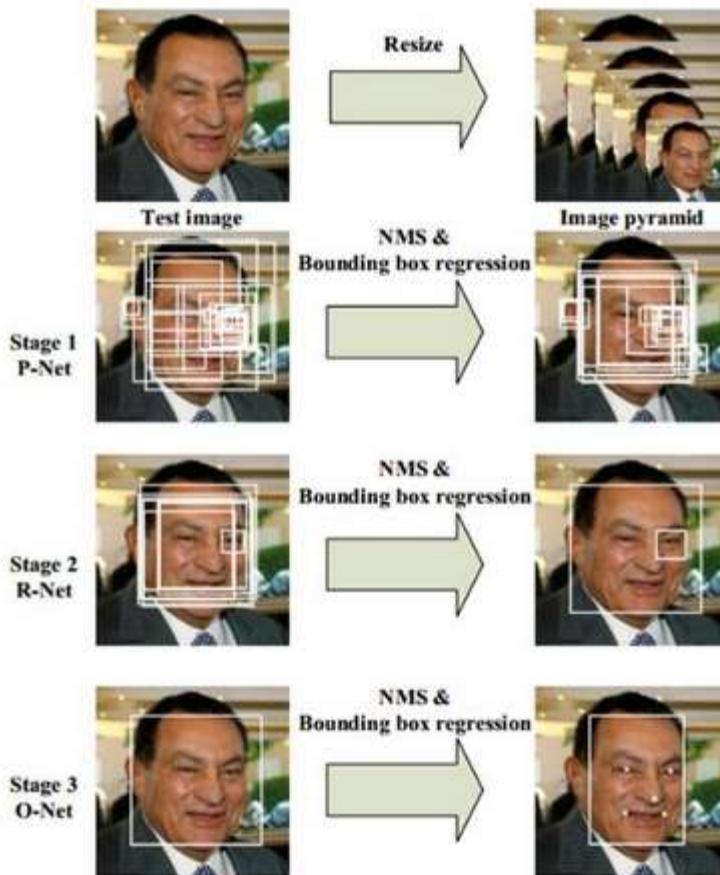
```
▶ img=cv2.imread("/content/drive/MyDrive/LFW/Face Dataset Train/10/3.jpg")
detect_faces_eyes(img)
```



In this case some of the images won't be correctly aligned nor recognized.  
Also, this classifier needs a lot of time and a lot of computational resources in order to be trained since it initially needs a lot of positive images (images of faces) and negative images (images without faces).

**The cascade classifier is 89% accurate.**

MTCNN is a 3 stages face detector:



First, the image is resized multiple times to detect faces of different sizes. Then the P-network (Proposal) scans images, performing first detection. It has a low threshold for detection and therefore detects many false positives, even after NMS (Non-Maximum Suppression), but works like this on purpose.

The proposed regions (containing many false positives) are input for the second network, the R-network (Refine), which, as the name suggests, filters detections (also with NMS) to obtain quite precise bounding boxes.

The final stage, the O-network (Output) performs the final refinement of the bounding boxes. This way not only faces are detected, but bounding boxes are very right and precise.

We almost could not find any wrong results when working with MTCNN.

**MTCNN is 0.95% accurate.**

## 2-Face recognition :

Facenet uses triplets in order to learn face recognition, meaning it needs a similar (same person) and a dissimilar (different person ) for each image of the training set. Facenet allowed us to use the comapairing lists .

**Facenet was 0.976% accurate.**

While SVM, a classification algorithms needing bigger numbers of images per each person performed poorly with maximum one 5 pictures per person.

We resorted then to working with the unfiltered dataset (350 images per person maximum) in order to get better result.

**SVM in this case was 0.85% accurate.**

## 3-Conclusion:

We recommend using MTCNN and Facenet together for optimal results.

## **General conclusion :**

In this project, we used our deep learning prerequisites as well as a lot of research, and we tried several models to get the best out of it, to Develop a solution for automatic recognition of 2D captured faces in uncontrolled environments maintaining a high recognition rate despite all conditions.