FIND THE MEMBER

Pablo Garcia Arribas (714120290033)
Soohyun Lee (714120290013)
Begaiym Omurkanova (516120990004)
Hannah Rowe (516120990001)
Dastan Yusupov (516120990003)

DESIGNED

FIND THE MEMBER

TABLE OF CONTENTS

PROBLEM WE AIM TO EXPLORE

After studying photo and video simulation and manipulation during our last few classes, we aimed to explore python language and create a program that we would **both enjoy coding and using**!

Our team members were especially interested in what we learnt in lectures on **python libraries such as cv3 and machine learning with scikit-learn.**

FIND THE MEMBER

PROBLEM WE AIM TO EXPLORE

FACE DETECTION + MACHINE LEARNING

A PROGRAM THAT IDENTIFIES A PERSON
AFTER TRAINING THE COMPUTER WITH PHOTOS

DESIGNED BY L@RGO

FIND THE MEMBER

FUNCTIONS TO BE INCLUDED

- ✓ generate 100 pictures of each of us from a video made with laptop's webcam
- ✓ train the data by using a supervised classifier method called **K-Nearest Neighbors (KNN)**
- ✓ resize an image to size 150x150 and **flatten it into a line of pixels** that will then be compared with our data sample
- ✓ capture the faces in the video and compare the face to the 'K' nearest faces of the data we previously gathered
- ✓ **print the registered person's name** it believes to be in the video

- ✓ **avoiding the mirror effect** that happens when you take "selfies".
- ✓ change the images into **black and white** to make them more comparable.
- ✓ a "keyboard command" such as the letter "Q" to stop the video whenever the user wants.

DESIGNED BY L@RGO

FIND THE MEMBER

OUR CODES

1. CREATE DATASETS
2. WEBCAM

DESIGNED BY L@RGO

OUR CODES
1. CREATE DATASETS

- This is the first program that runs
- This script takes 100 photos of the face of the front camera of the computer
- We import 2 libraries: **numpy and cv2**.
    - Numpy is to place the data as an array.
    - Cv2 is a library that help us to work with videos and photos

FIND THE MEMBER

OUR CODES

1. CREATE DATASETS

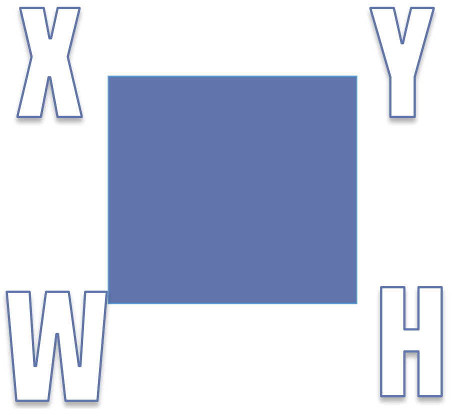
- A face detection algorithm has to be imported from opencv. In our case we found that frontal face default is good for this kind of project
- We use the function **VideoCapture** inside the library cv2 to open the webcam of your computer.
- Then we create a while that will do the following 100 times:
  - If the user presses the letter "q" it will stop running
  - One frame of the video will be taken with with the function cap.read
  - We will flip the picture horizontally (right to left) to avoid the "mirror effect"

DESIGNED BY L@RGO

FIND THE MEMBER

OUR CODES : 1. CREATE DATASETS

```
# Put the frame in gray
        gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY
# Detect faces comes next
        faces = face_cascade.detectMultiScale(gray, 1.3, 5)
        if faces != ():
# Get the rectangle representing the first face
        (x,y,w,h) = faces[0]
#Put the face in color into 'roi_color'
        roi_color = frame[y:y+h, x:x+w]
```



DESIGNED BY L@RGO

FIND THE MEMBER

OUR CODES : 1. CREATE DATASETS

```
# Save the face into a file
        cv2.imwrite('dataset/'+str(i+(n*100))+'.png', roi_color)
        i = i + 1
        print(i)
# Show the webcam output
        cv2.imshow('frame', frame)
```

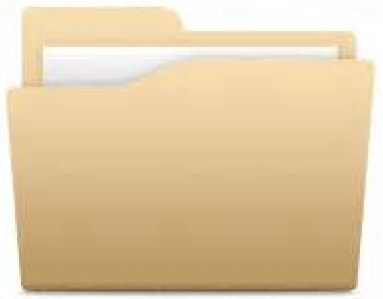
DESIGNED BY L@RGO

FIND THE MEMBER

OUR CODES : 1. CREATE DATASETS

FINALLY:

```
# Release the webcam
    cap.release()
# Close all the windows
    cv2.destroyAllWindows()
```



100 PICTURES OF THE SAME PERSON IN OUR FOLDER

YOU HAVE TO REPEAT THE PROCESS AS MANY TIMES PEOPLE YOU WANT TO IDENTIFY

DESIGNED BY L@RGO

FIND THE MEMBER

OUR CODES

2. WEBCAM

- This code detects faces in front of the camera and tries to match each face based on the faces registered previously
- Define function "**img_to_vector**"
- Import the face detection algorithm of **OpenCV**
- **Arrays, NumPy** used to store the registered images
- Right to left flip to avoid 'mirror effect'
- **KNN nearest neighbors** machine learning

DESIGNED BY L@RGO

FIND THE MEMBER

OUR CODES : 2. WEBCAM

Function to resize an image in 150x150 and flatten this image into a line of pixels.
*Here we have created a function/definition in python's cv2 to resize the image to size 150*150. We prefer this size because it is not too large or too small. This function also flattens the image (which makes it easier to be manipulated)*

```
def img_to_vector(img):
    val = cv2.resize(img, (150,150)).flatten()
    return val.astype(np.float32)
```

Import the face detection algorithm of OpenCV

```
face_cascade =cv2.CascadeClassifier('C:₩Users₩Administrator₩Downloads₩TP_Video₩TP_Video₩haarcascades₩haarcascade_frontalface_default.xml')
```

Arrays to store the registered images and their labels.. *Here we make a place to store the images of the registered persons for the program and their names.*

```
train_data = np.empty((900,150*150), dtype=np.float32)
labels_data = np.empty(900, dtype=np.float32)
```

DESIGNED BY L@RGO

FIND THE MEMBER

OUR CODES : 2. WEBCAM

Read all the registered photos and store them into 'train_data'.

Then, here we actually put the data we have saved (the images and their respective names) into the arrays we made.

The maximum (here 500) needs to be adapted to the number of persons in photos

For example, here the maximum is 500 because we have 5 persons with each 100 photos

```
for i in range(0, 500):
    img = cv2.imread('dataset/'+str(i)+'.png', 0)
    train_data[i] = img_to_vector(img)
```

#Define the names of the registered persons in the order they were registered.

Order here is very important because if the data is not replicated just like how it was saved th ere will be errors when the program runs as it will not be able to accurately say who is who. T he gathered data must correspond with one another. The registered images of the registered persons must correspond with their names.

```
res_labels = ['Pablo', 'Aimee', Hanna', 'Sue', 'Dastan']
```

DESIGNED BY L@RGO

FIND THE MEMBER

OUR CODES : 2. WEBCAM

Define for each photo the index corresponding to the name in 'res_labels'
For example, from 0 to 100 are the photos of Pablo, which is at the qqindex 0 in 'res_labels' and etc.

```
for i in range(0, 100):
labels_data[i] = 0
for i in range(100, 200):
labels_data[i] = 1
for i in range(200, 300):
labels_data[i] = 2
for i in range(300, 400):
labels_data[i] = 3
for i in range(400, 500):
labels_data[i] = 4
```

Define the font to print the names

```
font = cv2.FONT_HERSHEY_SIMPLEX
```

DESIGNED BY L@RGO

FIND THE MEMBER

OUR CODES : 2. WEBCAM

```
# Open the webcam in 'cap‘
        cap = cv2.VideoCapture(0)
# Right to left flip to avoid 'mirror effect'
        frame = np.fliplr(frame).copy()
# Put the frame in gray
        gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
# Detect faces
        faces = face_cascade.detectMultiScale(gray, 1.3, 5)
```

FIND THE MEMBER

OUR CODES : 2. WEBCAM

#Create empty NumPy arrays to store detected faces and their positions
In our case, number of faces times number of pixels and datatype float32

```
np.empty((len(faces), 150*150), dtype=np.float32)
```

#Detect the face and for each face we store its flattened version (img_to_vector) into 'test_data'

```
test_data[idx] = img_to_vector(roi_gray)
```

#Execute KNN: Find for each face the 15 nearest registered faces and deduce the matching person (We tried changing the number to other odd numbers check the precision)

```
ret, results, neighbours ,dist = knn.findNearest(test_data, 15)
```

#Write text on the webcam: cv2.putText

```
cv2.putText(frame, str, coordinates of the text, font style, font size, font color)
```

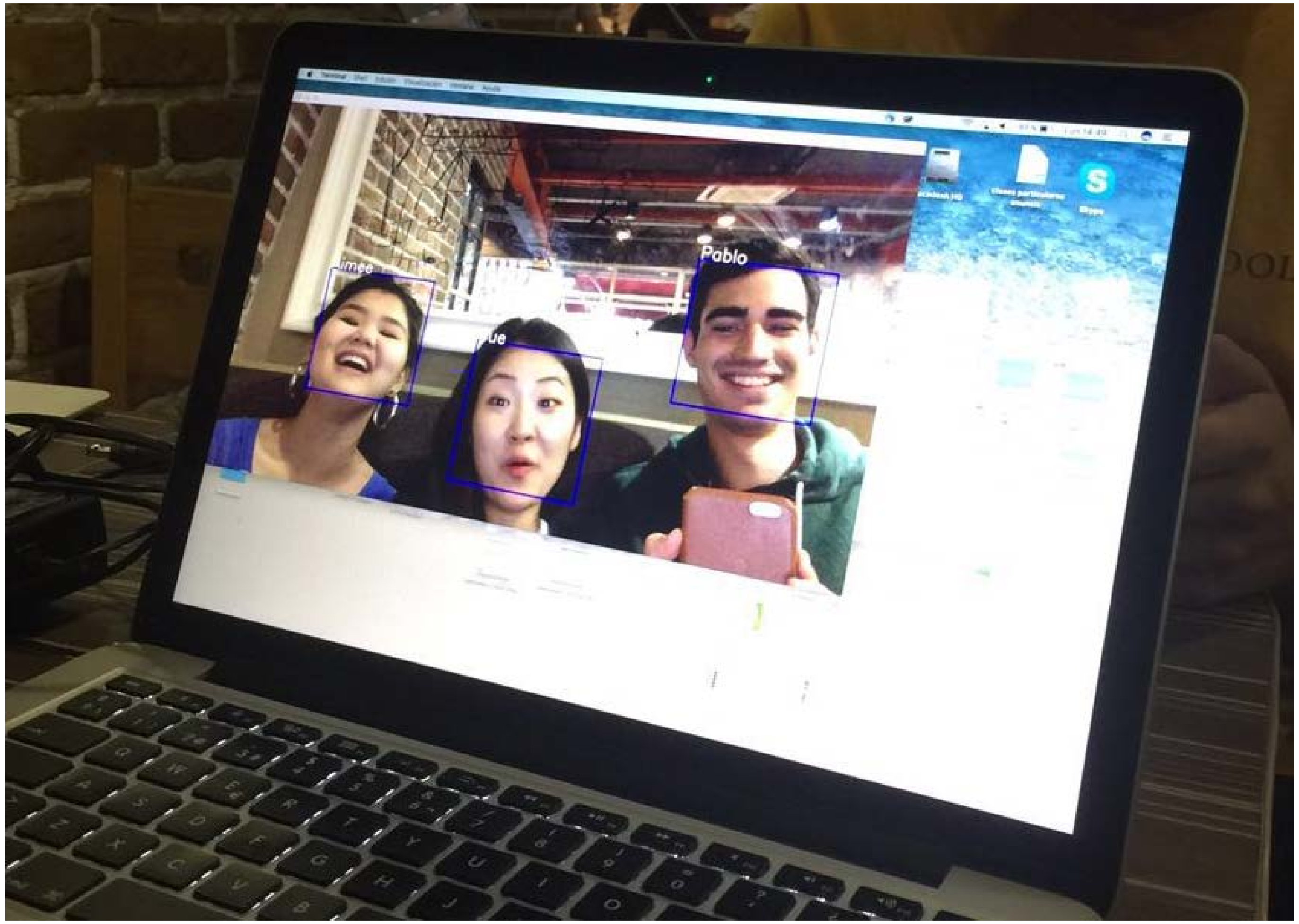
#Show the resulting image

```
cv2.imshow('frame', frame)
```

DESIGNED BY L@RGO

FIND THE MEMBER

FINAL OUTPUT

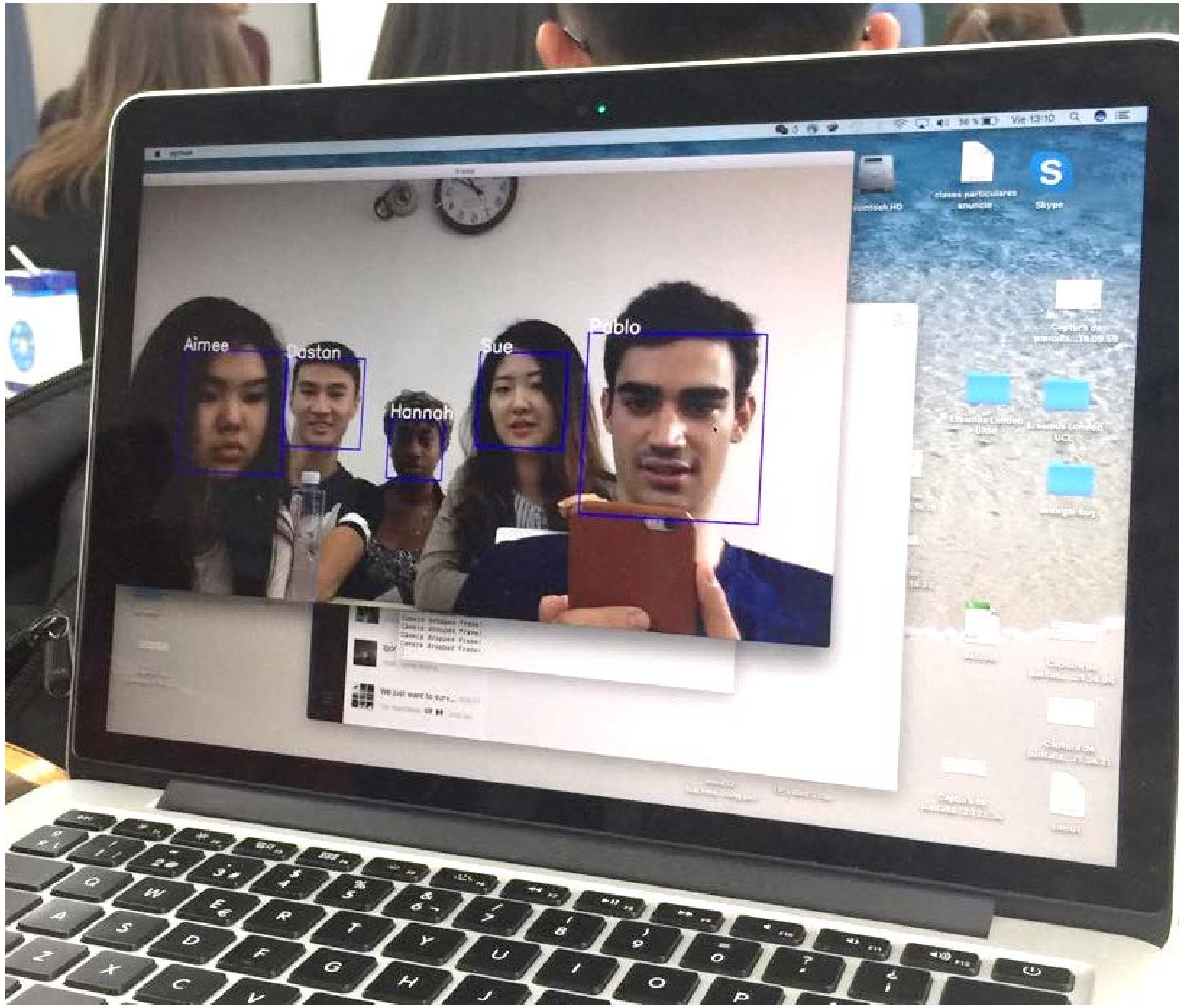


DESIGNED BY L@RGO

FIND THE MEMBER

DESIGNED BY L@RGO

FINAL OUTPUT



LET'S TRY IT OUT!