CSC 511 Advanced Artificial Intelligence: Term Project

Submission To: Dr. Roman Tankelevich
By: Muhammad Zaid Kamil (Student Number: 211890395)

Abstract— **The goal of Deep Learning is to use systems to interpret and understand visual complex problems. By using digital media, we want the machine to accurately identify and classify objects. Some Neural Network problems are trivial for human beings, such as object and face detection. In this Artificial Intelligence Graduate report we will use Neural Network for different techniques involved in Face Detection and Face Recognition. We will first present a general introduction of Neural Networks discussion on the methodology of RCNN algorithm and compare two libraries a and the impact of Face Detection our lives. Then the report will go into the implementation of Face Detection using OpenCV library and Face Recognition using DeepFace library. In the final section, , we will compare the results of the two implementations. Finally, we will conclude the paper.**

***Keywords*: Artificial Intelligence, Face Recognition, Face Detection, RCNN, OpenCV, Computer Vision, Implementation,**

I. Introduction

Face Detection and Face recognition is a technology in the field of Artificial Intelligence and Machine Learning. We want computers to sense media (digital images/videos) similar to human perception. As humans, we are trained a lifetime to tell objects apart, and recognize face from incoming people. We use our retina and optic neurons to discern objects and recognize face. This is where Neural Network comes, where cv trains system to perform this function (face detection and face recognition) with less time and with the limit of cameras, data and algorithms. Although thinking and behaving like human might not be the pinnacle of vision that humans have for a machine, teaching computers how to see is the frontier that could not be ignored. The paper 1 will explore another topic in the field of Computer vision: Face recognition, which we will dive into the methodology of Face recognition algorithm

II. SECTION: Background & Methodology

A. Introduction

The section will evaluate and compare of software for Face recognition. This section will investigate into two main computer vision libraries: OpenCV and dlib. Which will explore the algorithms used for face recognition based on the library. Explore the features, time complexity of the two libraries. The paper also explores two applications for face recognition from the two libraries.

1) Computer Vision applications

One of the popular computer vision problems is phone recognition/biometrics measurement. Initially biometrics was limited to the uniqueness of finger scan to classify individuals. For example, if we talk about smartphone devices, initially users used pin code to unlock their personal devices, then smart phones included an additional unlocking system which uses finger scanner. Now in recent smart phone products, users use face recognition to unlock, and pay from their personal devices. From the figure below we can represent the progress of biometric system in phone.

Fig. 1. Progress of phone biometric system for screen lock

2) Computer Vision Libraries

The result of computer vision applications resulted in formation of different types of computer vision libraries and APIs which developed solution in different computer vision areas in this case for face recognition. The paper will compare the performance using these two libraries (OpenCV and dlib) and build two simple recognition systems. The libraries are mainly written in C++ and python programming language.

The best way to get acquainted with a particular library is to look at the documentation. For OpenCV the documentation contains modules and examples from face analysis to texture mapping. Each module gives a description of the functions used. And the example program presents the C++ code and screenshot of the program. Documentation of other computer vision libraries such as DeepFace (written in python) are in presented in open source libraries like GitHub.

B. Methodology of Algorithm

This section will go in depth of the four steps for the face recognition algorithm: From searching the face to recognizing and identifying the face.

1) Step 1: Search the face in digital images/video.

This step is crucial because if the system skips a face and regard another object as a face, then the recognition will give false results. Using the HOG Algorithm. The system can detect the face.

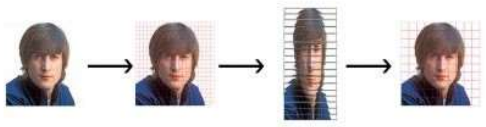


Fig. 2. Visualization of using the HOG algorithm for Face detection

The basic idea of HOG is dividing the image into small connected cells. Then for each cell it computes histogram. Then it combines all histograms to form a final feature vector. Where the vector is unique for each face. The disadvantage of HOG it doesn't work on faces at different angles. Advantage of HOG is to search for faces in official documents: Driving

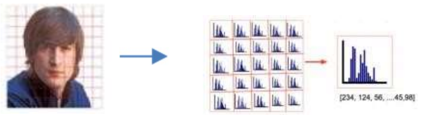


Fig. 3. (a) grid cell for image[1] (b) computes histogram for each cell[2]

The above Figure 3 (b) brings all the histograms together to form feature vector, which is unique for every face, in the above example the vector = [234, 124, 56, …45, 98][2]. This vector is unique to this particular image and face.

2) Step 2: Determine the face position

After finding the face, the task of the system is to position it. Most of the media the face is angular to the lens (side face position), and not center to the lens. Most efficient way to solve this to recognize face is to use the face landmark estimation. algorithm. Where we have to identify key points/landmarks in the face and represent with a numeric number.

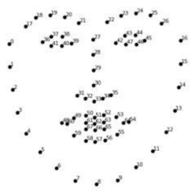


Fig. 4. 68 landmark points in face using Landmark estimation algorithm[1]

From the figure above shows the total 68 landmarks, detecting 68 points in the face which include: mouth, eyes, nose andface structure. Where each landmark is represented by a numeric number from 1-68.

3) Step 3: Determine the unique facial features

3.1) Method 1: Database for face recognition

To distinguish among faces, need to determine unique features. There are two methods to distinguish between faces. The first method to recognize faces is to compare new face with all the other faces stored in the pre-defined database of face images. In each recognition compares to the set of images in database, hoping to find the same face. If there is a large amount of data in the database would be a better option to store the database in the cloud rather than the system. However, it can cause delays between request and replies from the server refer Fig 5(b). This method is beneficial if the task involves identifying a small group of people in the database.

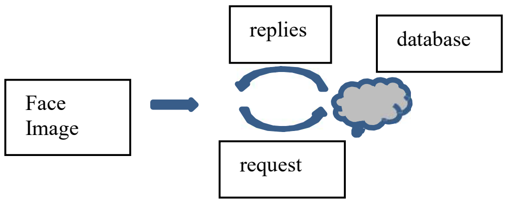


Fig. 5. (a) grid cell for image (b) computes histogram for each cell

3.1) Method 2: Training the D-CNN Neural Network

The second method uses training method such as neural networks DCNN, which will be trained to identify 128 unique facial features.

Step 1: In order to train the DCNN neural network, we first need to upload three images:

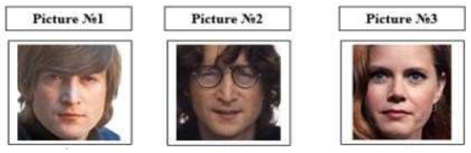


Figure 6: Three representation of Images uploaded to the network[1]

In the above Figure 6, The two images uploaded is of the same person (Picture No 1 and Picture No 2) and the third image (Picture No 3) is another person's face.

Step 2: The neural network will adjust the results get 128 characteristics of Image No1 and No2 are as close as possible and the Image No3 will be different.

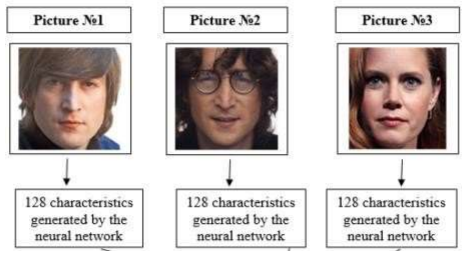

Picture №1
Picture №2
Picture №3
128 characteristics generated by the neural network
128 characteristics generated by the neural network
128 characteristics generated by the neural network

Figure 7: Neural network getting 128 characteristics of Image [1]

Step 3: The process of training neural network to generate numerical features of the face is complex process and requires huge database of faces. We use existing features to train the data set. As soon as neural network is trained, it will be able to generate unique characteristics for any face person. The neural network only needs to be trained once.

Step 4: Compare the existing data from previous step, 128 features of the uploaded faces with the data about the people. The Support Vector Machine (SVM) is used to check if the data coincide[1]. If the data is homogeneous, the more accurate to determine the face. The linear SVM uses a regression analysis model.

$$(\vec{x}_1, y_1), \ldots, (\vec{x}_n, y_n)$$

Where y is 1 or -1, which indicates a class xi which the the point belongs.
The example given from the paper is as follows:

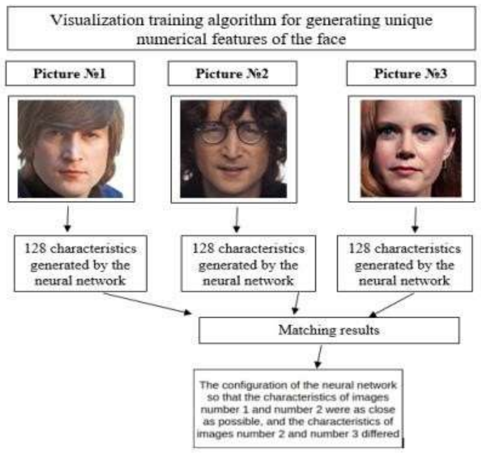
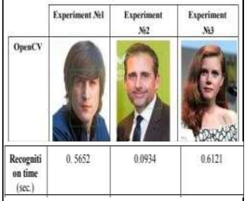
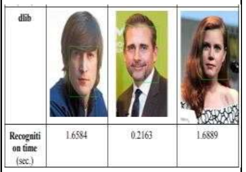

Visualization training algorithm for generating unique numerical features of the face
Picture №1
Picture №2
Picture №3
128 characteristics generated by the neural network
128 characteristics generated by the neural network
128 characteristics generated by the neural network
Matching results
The configuration of the neural network so that the characteristics of images number 1 and number 2 were as close as possible, and the characteristics of images number 2 and number 3 differed

Figure 8: Visualization of the neural learning algorithm to check face match[1]

Table 1. Comparisons of the pseudocode, results and time for detection using OpenCV and dlib libraries [1]

Libraries		
	Pseudocode	**Results**

OpenCV	***Face_detector_opencv.py:***
	# Import the system and OpenCV libraries
	import sys
	import cv2
	from skimage import io
	# Call the basic function library to find faces
	lbp_face_cascade = cv2.CascadeClassifier('data/ lbpcascade_frontalface.xml')
	# Go through all entered images
	for images in sys.argv[1:]:
	  print("Processing image: {}".format(images))
	  image = io.imread(images)
	  faces_detected_img = detect_faces(lbp_face_cascade, image, scaleFactor=1.2)
	  io.imsave('face_recognition.jpg', image)
	  print("Number of faces detected:{}".format (len(faces))
	# Print the result
	plt.imshow(convertToRGB(faces_detected_img))
	#Wait for "Enter" button to continue
	cv2.waitKey(0)
		
		Experiment №1 / Experiment №2 / Experiment №3
		OpenCV
		Recognition time (sec.): 0. 5652 / 0.0934 / 0.6121

Dlib	***Face_detector_dlib.py:***
	# Import the system and dlib libraries
	import sys
	import dlib
	from skimage import io
	# Call the basic function library to find faces
	face_detector = dlib.get_frontal_face_detector()
	# Go through all entered images
	for images in sys.argv[1:]:
	  print("Processing image: {}".format(images))
	  image = io.imread(images)
	  faces_detected_image= face_detector (image, 1)
	  io.imsave('face_recognition.jpg', faces_detected_image)
	  print("Number of faces detected:{}".format (len(faces_detected_image))
	# Print the result
	win = dlib.image_window()
	win.clear_overlay()
	win.set_image(image)
	win.add_overlay(dets)
	#Wait for "Enter" button to continue
	dlib.hit_enter_to_continue()
		
		dlib
		Recognition time (sec.): 1.6584 / 0.2163 / 1.6889

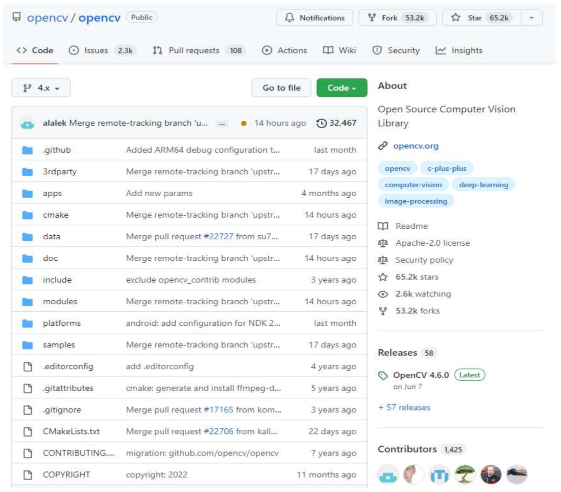
The above table compares the pseudocode, and time spent to search for the face using two face detection systems, one is from the OpenCV library and second from the dlib libraries. The following pseudocode was written in python for both libraries. As from the results it can be seen the OpenCV was the more effective library since the recognition time is almost three times faster compared to the algorithm from dlib library.

III. SECTION: Implementation

1) *Implementation of Face recognition and Face Detection tools*

1.1) Implementation of Face Detection

A. *Step 1:* Installing OpenCV from open source – Github. Version OpenCV



alalek Merge remote-tracking branch 'u... 14 hours ago 32,467

.github	Added ARM64 debug configuration t...	last month
3rdparty	Merge remote-tracking branch 'upstr...	17 days ago
apps	Add new params	4 months ago
cmake	Merge remote-tracking branch 'upstr...	14 hours ago
data	Merge pull request #22727 from su7...	17 days ago
doc	Merge remote-tracking branch 'upstr...	14 hours ago
include	exclude opencv_contrib modules	3 years ago
modules	Merge remote-tracking branch 'upstr...	14 hours ago
platforms	android: add configuration for NDK 2...	last month
samples	Merge remote-tracking branch 'upstr...	17 days ago
.editorconfig	add .editorconfig	4 years ago
.gitattributes	cmake: generate and install ffmpeg-d...	5 years ago
.gitignore	Merge pull request #17165 from kom...	3 years ago
CMakeLists.txt	Merge pull request #22706 from kall...	22 days ago
CONTRIBUTING....	migration: github.com/opencv/opencv	7 years ago
COPYRIGHT	copyright: 2022	11 months ago

About

Open Source Computer Vision Library

opencv.org

opencv c-plus-plus computer-vision deep-learning image-processing

Readme
Apache-2.0 license
Security policy
65.2k stars
2.6k watching
53.2k forks

Releases 58

OpenCV 4.6.0 Latest
on Jun 7

+ 57 releases

Contributors 1,425

Figure 1: Importing the OpenCV library

B. *Step 2:* Add the environment variables and directories

We add the environment variables, build directories and library directories by going to Project-Properties-Configuration Properties. This will allow the vscode to compile during execution.

C. *Step 3:* Add the Images in the File Directory

We need to add the database for the face detection algorithm. Where all the images are stored in the pre-defined database of face images. For this implementation, named a folder called Resources to store the images as shown in the below Figure.

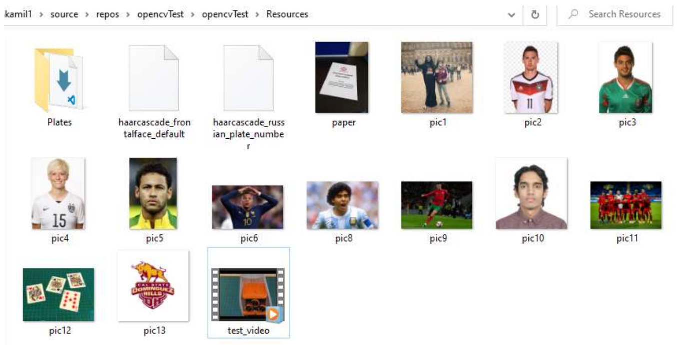
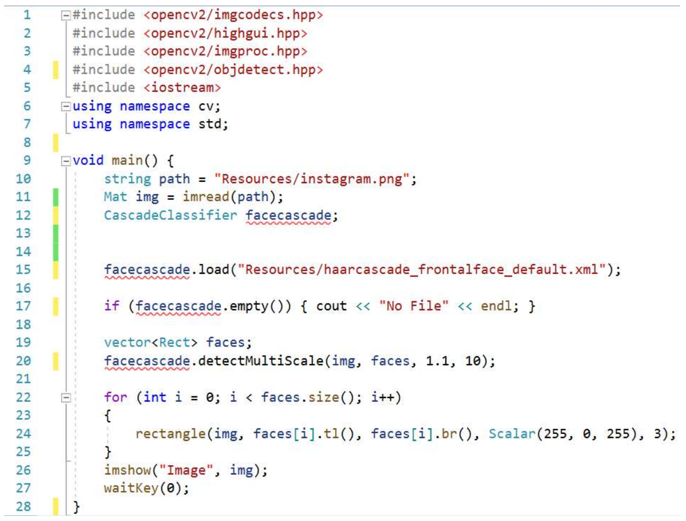

kamil1 › source › repos › opencvTest › opencvTest › Resources
Search Resources
Plates
haarcascade_frontalface_default
haarcascade_russian_plate_number
paper
pic1
pic2
pic3
pic4
pic5
pic6
pic8
pic9
pic10
pic11
pic12
pic13
test_video

Figure 2: Folder containing the images used for Face Detection

D. *Step 4:* Implementation using VisualStudio

To implement face detection, we used the face cascade from Haar Cascade algorithm from OpenCV library code. The code was written in C++ using Visual Studio IDE. The source for the projects is mentioned in the references below [7].



```
1   #include <opencv2/imgcodecs.hpp>
2   #include <opencv2/highgui.hpp>
3   #include <opencv2/imgproc.hpp>
4   #include <opencv2/objdetect.hpp>
5   #include <iostream>
6   using namespace cv;
7   using namespace std;
8
9   void main() {
10      string path = "Resources/instagram.png";
11      Mat img = imread(path);
12      CascadeClassifier facecascade;
13
14
15      facecascade.load("Resources/haarcascade_frontalface_default.xml");
16
17      if (facecascade.empty()) { cout << "No File" << endl; }
18
19      vector<Rect> faces;
20      facecascade.detectMultiScale(img, faces, 1.1, 10);
21
22      for (int i = 0; i < faces.size(); i++)
23      {
24          rectangle(img, faces[i].tl(), faces[i].br(), Scalar(255, 0, 255), 3);
25      }
26      imshow("Image", img);
27      waitKey(0);
28  }
```

Figure 3: C++ Implementation using Visual Studio

<<comments>>

Line 4: Header file to work with Haar Cascade
Line 10: Adding Image file path in directory
Line12: Loading the cascade model
Line 15: XML file to load Haar Cascade model to detect Face in Image
Line24: Forming of the boundary on detected faces.

We first imported OpenCV library. Under the main function. We have to input the image path from the directory folder. We use the facsCascade algorithm
We first imported OpenCV library. Under the main function. We have to input the image path from the directory folder. We use the faceCascade algorithm from the CascadeClassifier. To form the bounding boxes, we use a for loop. If Face is detected, a rectangular bounding box is formed. Then the image is showed using imshow with a wait time of infinity since waitKey(0).

E. *Results*

The results of the code when run in Visual studio is as follow:

Portrait Face Images:

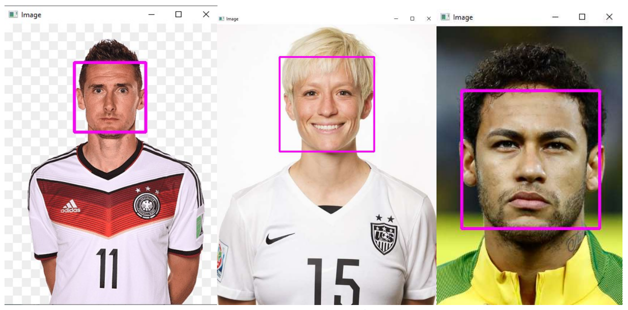


Fig. 4. Portrait Images (a) Pic1 (b) Pic 2 (c) Pic 3

Images with Background/Noise

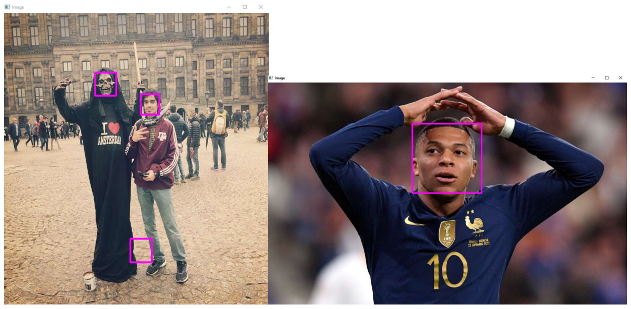


Fig. 5. Face Detection with Images of background noise

The Images with noise represents background such as buildings, people, emotions. The Haar cascade model was able to detect faces with background. As you can see in Figure 5(a) one of the limitations of Haar Cascade is false positive detections which is seen toward the bottom of the image. We can resolve this by manually tuning the parameters in the source code by trial and error.

Images with multiple faces

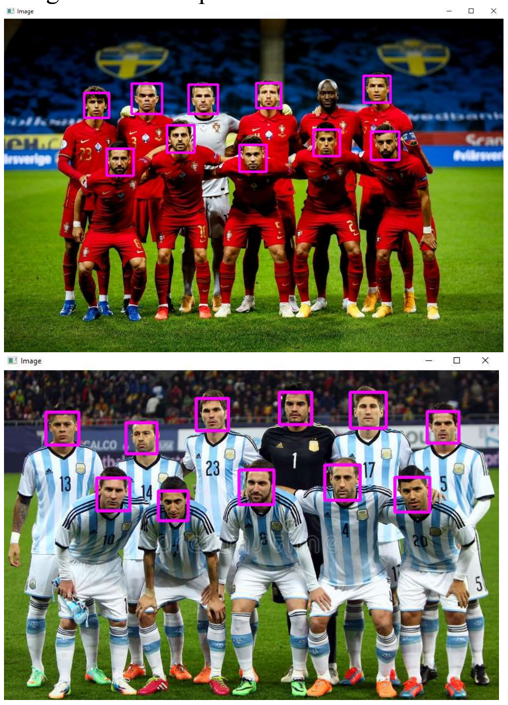


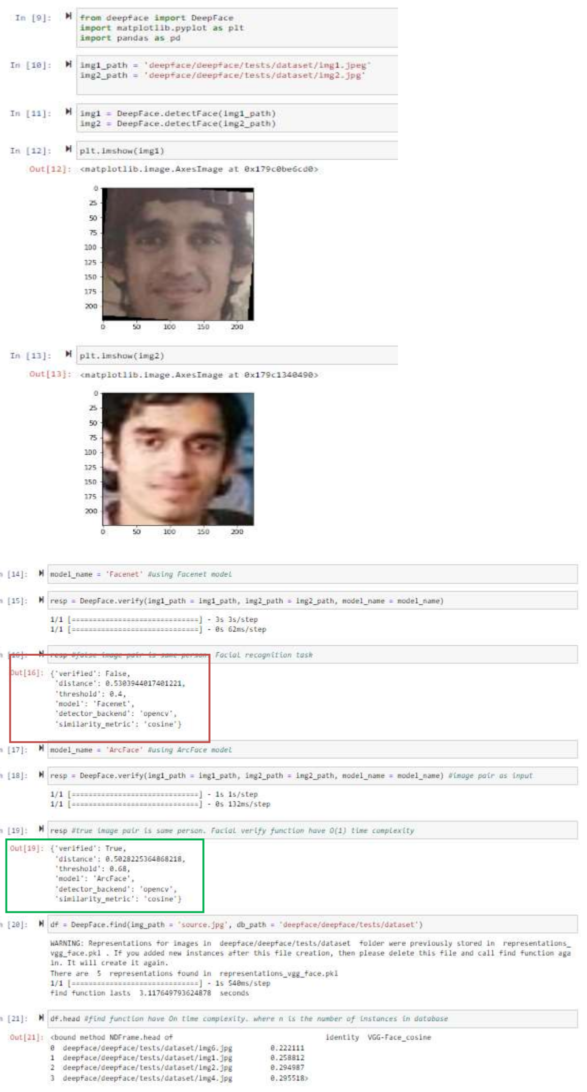
Fig. 5. Face Detection with Images of background noise

The above images represent a Football team where the Total face detected are 11. In Figure 5 a, the number of face detected are 10 out of 11 which gives an accuracy of .90% detection. However, the second image (Figure 5(b)) all the 11 faces are detected.

From the above images, the cascade algorithm is efficient in detecting most of the images. Even with background noise, there were instances of false positives, and 95% accuracy in detecting multiple faces in image.

Implementation of Face Recognition

We were able to implement the code to test our own images for face recognition. For the Face recognition we used DeepFace library written on python on jupyter notebook and for Face Detection we used OpenCV written in C++. The source for the projects is mentioned in the references below.



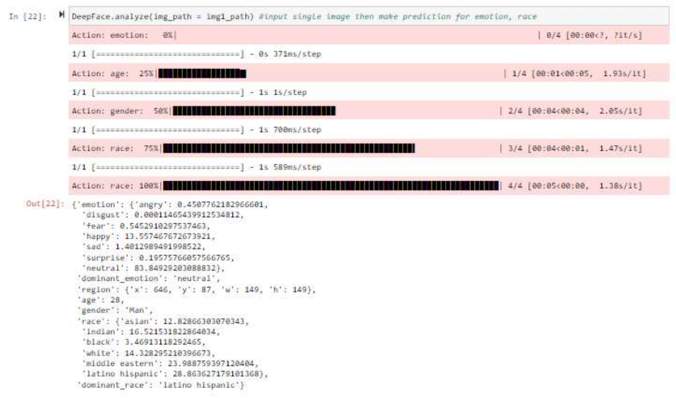
```
In [9]:  from deepface import DeepFace
         import matplotlib.pyplot as plt
         import pandas as pd

In [10]: img1_path = 'deepface/deepface/tests/dataset/img1.jpeg'
         img2_path = 'deepface/deepface/tests/dataset/img2.jpg'

In [11]: img1 = DeepFace.detectFace(img1_path)
         img2 = DeepFace.detectFace(img2_path)

In [12]: plt.imshow(img1)
Out[12]: <matplotlib.image.AxesImage at 0x179c0be6cd0>

In [13]: plt.imshow(img2)
Out[13]: <matplotlib.image.AxesImage at 0x179c1340490>

In [14]: model_name = 'Facenet' #using Facenet model

In [15]: resp = DeepFace.verify(img1_path = img1_path, img2_path = img2_path, model_name = model_name)

         1/1 [==============================] - 3s 3s/step
         1/1 [==============================] - 0s 62ms/step

In [16]: resp #false image pair is same person. Facial recognition task
Out[16]: {'verified': False,
          'distance': 0.5303944017401221,
          'threshold': 0.4,
          'model': 'Facenet',
          'detector_backend': 'opencv',
          'similarity_metric': 'cosine'}

In [17]: model_name = 'ArcFace' #using ArcFace model

In [18]: resp = DeepFace.verify(img1_path = img1_path, img2_path = img2_path, model_name = model_name) #image pair as input

         1/1 [==============================] - 1s 1s/step
         1/1 [==============================] - 0s 132ms/step

In [19]: resp #true image pair is same person. Facial verify function have O(1) time complexity
Out[19]: {'verified': True,
          'distance': 0.5028225364868218,
          'threshold': 0.68,
          'model': 'ArcFace',
          'detector_backend': 'opencv',
          'similarity_metric': 'cosine'}

In [20]: df = DeepFace.find(img_path = 'source.jpg', db_path = 'deepface/deepface/tests/dataset')

         WARNING: Representations for images in  deepface/deepface/tests/dataset  folder were previously stored in  representations_
         vgg_face.pkl . If you added new instances after this file creation, then please delete this file and call find function aga
         in. It will create it again.
         There are  5  representations found in  representations_vgg_face.pkl
         1/1 [==============================] - 1s 540ms/step
         find function lasts  3.117649793624878  seconds

In [21]: df.head #find function have On time complexity. where n is the number of instances in database
Out[21]: <bound method NDFrame.head of                                           identity  VGG-Face_cosine
         0  deepface/deepface/tests/dataset/img6.jpg        0.222111
         1  deepface/deepface/tests/dataset/img1.jpg        0.258812
         2  deepface/deepface/tests/dataset/img2.jpg        0.294987
         3  deepface/deepface/tests/dataset/img4.jpg        0.295518>
```

Fig. 6. Face Recognition using python in Jupyter kernel

The above Figure 3 shows the pseudocode which was obtained from DeepFace repository [6]. The objective was to determine to verify two face images are the same or different. We first import the DeepFace model from the library. For each of the face image we input the exact image paths stored in the PC database. Plt.imshow is a function that uses OpenCV to extract the face image 1 and 2 from the picture and removes the background.

We use the function DeepFace.verify checks if the two face images are same or different. We compared the efficiency of two face recognition models using Facenet and ArcFace There is a false result in the Facenet model when comparing the two images. And we can see there is a threshold lower coefficient of 0.4 when using the Facenet model therefore it is more efficient.

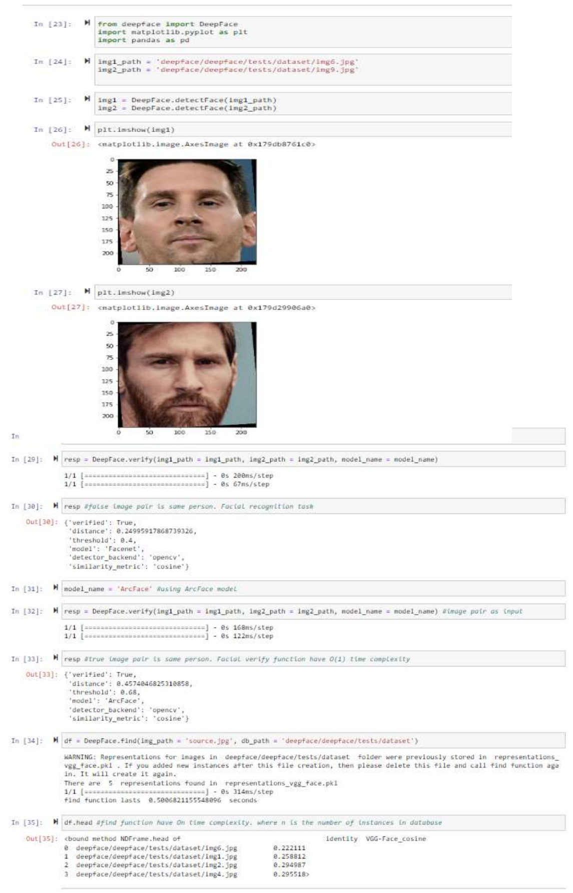
We can also represent the Facial emotions using DeepFace.analyze as shown in the Figure below:



```
In [22]:  DeepFace.analyze(img_path = img1_path) #input single image then make prediction for emotion, race
          Action: emotion:   0%|                                              | 0/4 [00:00<?, ?it/s]
          1/1 [==============================] - 0s 371ms/step
          Action: age:  25%|██████████                                | 1/4 [00:01<00:05,  1.93s/it]
          1/1 [==============================] - 1s 1s/step
          Action: gender:  50%|████████████████████                    | 2/4 [00:04<00:04,  2.05s/it]
          1/1 [==============================] - 1s 700ms/step
          Action: race:  75%|██████████████████████████████              | 3/4 [00:04<00:01,  1.47s/it]
          1/1 [==============================] - 1s 589ms/step
          Action: race: 100%|████████████████████████████████████████| 4/4 [00:05<00:00,  1.38s/it]
Out[22]: {'emotion': {'angry': 0.4507762182966601,
            'disgust': 0.00011465439912534812,
            'fear': 0.5452910297537463,
            'happy': 13.557467672673921,
            'sad': 1.4012989491998522,
            'surprise': 0.19575766057566765,
            'neutral': 83.84929203088832},
          'dominant_emotion': 'neutral',
          'region': {'x': 646, 'y': 87, 'w': 149, 'h': 149},
          'age': 28,
          'gender': 'Man',
          'race': {'asian': 12.82866303070343,
            'indian': 16.521531822864034,
            'black': 3.46913118292465,
            'white': 14.328295210396673,
            'middle eastern': 23.988759397120404,
            'latino hispanic': 28.863627179101368},
          'dominant_race': 'latino hispanic'}
```

Fig. 7. Facial characteristics for dataset 1 using DeepFace.analyze

The model makes a prediction for facial characteristics for emotion (happy, sad, etc..), race, age.To test the accuracy of the face recognition model, we can perform face recognition analysis to the second face image.

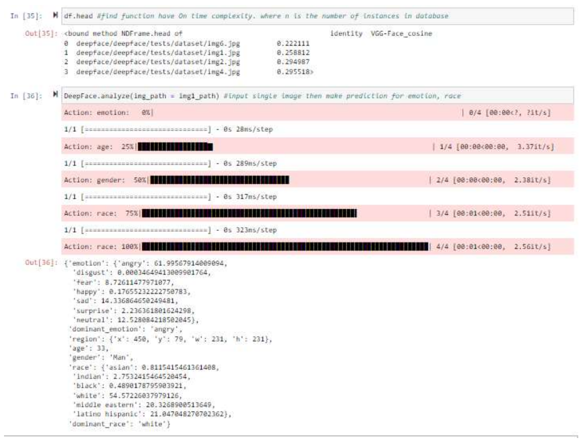


```
In [23]:  from deepface import DeepFace
          import matplotlib.pyplot as plt
          import pandas as pd

In [24]:  img1_path = 'deepface/deepface/tests/dataset/img6.jpg'
          img2_path = 'deepface/deepface/tests/dataset/img9.jpg'

In [25]:  img1 = DeepFace.detectFace(img1_path)
          img2 = DeepFace.detectFace(img2_path)

In [26]:  plt.imshow(img1)
Out[26]:  <matplotlib.image.AxesImage at 0x179db8761c0>

In [27]:  plt.imshow(img2)
Out[27]:  <matplotlib.image.AxesImage at 0x179d29906a0>

In

In [29]:  resp = DeepFace.verify(img1_path = img1_path, img2_path = img2_path, model_name = model_name)
          1/1 [==============================] - 0s 200ms/step
          1/1 [==============================] - 0s 67ms/step

In [30]:  resp #false image pair is same person. Facial recognition task
Out[30]:  {'verified': True,
           'distance': 0.24995917868739326,
           'threshold': 0.4,
           'model': 'Facenet',
           'detector_backend': 'opencv',
           'similarity_metric': 'cosine'}

In [31]:  model_name = 'ArcFace' #using ArcFace model

In [32]:  resp = DeepFace.verify(img1_path = img1_path, img2_path = img2_path, model_name = model_name) #image pair as input
          1/1 [==============================] - 0s 168ms/step
          1/1 [==============================] - 0s 122ms/step

In [33]:  resp #true image pair is same person. Facial verify function have O(1) time complexity
Out[33]:  {'verified': True,
           'distance': 0.4574046825310858,
           'threshold': 0.68,
           'model': 'ArcFace',
           'detector_backend': 'opencv',
           'similarity_metric': 'cosine'}

In [34]:  df = DeepFace.find(img_path = 'source.jpg', db_path = 'deepface/deepface/tests/dataset')
          WARNING: Representations for images in  deepface/deepface/tests/dataset  folder were previously stored in  representations_
          vgg_face.pkl . If you added new instances after this file creation, then please delete this file and call find function aga
          in. It will create it again.
          There are  5  representations found in  representations_vgg_face.pkl
          1/1 [==============================] - 0s 314ms/step
          find function lasts  0.5006821155548096  seconds

In [35]:  df.head #find function have On time complexity. where n is the number of instances in database
Out[35]:  <bound method NDFrame.head of                                         identity  VGG-Face_cosine
          0  deepface/deepface/tests/dataset/img6.jpg        0.222111
          1  deepface/deepface/tests/dataset/img1.jpg        0.258812
          2  deepface/deepface/tests/dataset/img2.jpg        0.294987
          3  deepface/deepface/tests/dataset/img4.jpg        0.295518>
```

Fig. 8. Face Recognition implementation for dataset 2

From the second dataset of images, we can see both the models: FaceNet and ArcFace was able to verify the two images represents the same person.

From using the DeepFace.analyze, we can showcase the facial emotions of the two images below:



```
In [35]:  df.head #find function have On time complexity. where n is the number of instances in database
Out[35]:  <bound method NDFrame.head of                                         identity  VGG-Face_cosine
          0  deepface/deepface/tests/dataset/img6.jpg        0.222111
          1  deepface/deepface/tests/dataset/img1.jpg        0.258812
          2  deepface/deepface/tests/dataset/img2.jpg        0.294987
          3  deepface/deepface/tests/dataset/img4.jpg        0.295518>

In [36]:  DeepFace.analyze(img_path = img1_path) #input single image then make prediction for emotion, race
          Action: emotion:   0%|                                              | 0/4 [00:00<?, ?it/s]
          1/1 [==============================] - 0s 28ms/step
          Action: age:  25%|██████████                                | 1/4 [00:00<00:00,  3.37it/s]
          1/1 [==============================] - 0s 289ms/step
          Action: gender:  50%|████████████████████                    | 2/4 [00:00<00:00,  2.38it/s]
          1/1 [==============================] - 0s 317ms/step
          Action: race:  75%|██████████████████████████████              | 3/4 [00:01<00:00,  2.51it/s]
          1/1 [==============================] - 0s 323ms/step
          Action: race: 100%|████████████████████████████████████████| 4/4 [00:01<00:00,  2.56it/s]
Out[36]: {'emotion': {'angry': 61.99567914009094,
            'disgust': 0.0003464941300990l764,
            'fear': 8.72611477971077,
            'happy': 0.17655232222750783,
            'sad': 14.336864650249481,
            'surprise': 2.236361801624298,
            'neutral': 12.528084218502045},
          'dominant_emotion': 'angry',
          'region': {'x': 450, 'y': 79, 'w': 231, 'h': 231},
          'age': 33,
          'gender': 'Man',
          'race': {'asian': 0.8115415461361408,
            'indian': 2.7532415464520454,
            'black': 0.4890178795903921,
            'white': 54.57226037979126,
            'middle eastern': 20.3268900513649,
            'latino hispanic': 21.047048270702362},
          'dominant_race': 'white'}
```

Fig. 9. Facial characteristics for dataset 2 using DeepFace.analyze

The results based on the above image showcase the different facial expressions, age, and race.

IV. Conclusion

In this report, we gave an overview of the Face detection and recognition methodology then we did some of our own implementation of Face recognition and Face detection. As we discussed in the paper, there can be mistakes from the system when detecting objects, faces and verifying similarity of faces. The neural network models system cannot be as accurate as human beings vision, since in our Haar cascade implementation there were false positive detections when th images have background noise. However, with fine tuning they are very fast and lightweight. For the face recognition we received a 75% accuracy (verified 3 of the 4 images, verified the second dataset of images) in verifying the same face images. The error could be due to the image quality and noise present in the image. We can use tools to remove noise from images which will present a greater accuracy rate in face detection and recognition. Based on the facial expression, we can extract the emotions of the face. These applications were able to be bought to light by using Open source software such as OpenCV library, the infrastructure for the face detection applications. The future of Artificial Intelligence based computer technology, Face Detection and Face Recognition technology looks bright. The source code for the Face Detection (Face Detection Haar Cascade – mkamil1.cpp) and Face Recognition (Face Recognition – mkamil1.py) implementation is attached in the Blackboard.

References

[1] N. Boyko, O. Basystiuk and N. Shakhovska, "Performance Evaluation and Comparison of Software for Face Recognition, based on Dlib and Opencv Library", IEEE, 2018. Available: https://ieeexplore.ieee.org/document/8478556.

[2] S. Jack, "Face detection using dlib HOG", *Medium*, 2022. [Online]. Available: https://medium.com/mlcrunch/face-detection-using-dlib-hog- 198414837945

[3] F. Felzenszwalb, R. B. Girshick, D. McAllester, and D. Ra- manan. Object detection with discriminatively trained part based models. IEEE Transactions on Pattern Analysis and Machine Intelligence, 32(9):1627– 1645, 2010.

[4] Girshick, J. Donahue, T. Darrell, and J. Malik. Rich fea- ture hierarchies for accurate object detection and semantic segmentation. In Computer Vision and Pattern Recognition (CVPR), 2014 IEEE Conference on, pages 580–587. IEEE, 2014.

Face Recognition and Face Detection code retrieved from:

[5] https://github.com/serengil/deepface

[6] https://www.computervision.zone/courses/opencv-cv/

Source code files (attached on Blackboard)

Face Detection– Face Detection Haar Cascade – mkamil1.cpp
Face Recognition - Face Recognition – mkamil1.py