CS 5590/490:Python and Deep Learning

FACE RECOGNITION and DETECTION

Project Report Submitted on 10th December 2018

Group Members:

Kamal Tej Veerapaneni

Vinay Maturi

Harish Chandra Jyoshi

Akhila Atluri

Video URL: https://youtu.be/8JGa_EbLvPI

School of Computing & Engineering

University of Missouri - Kansas City

Contents:

Contributors:

This report contains entire documentation that is required for the project. Project was done by Kamal Tej Veerapaneni,Vinay Maturi,Harish Chandra Jyoshi, Akhila Atluri currently pursuing Python/Deep Learning Course

Scope:

Uses live face recognition to recognize each individual face by utilizing video and image processing to provide inputs to the system. Uses cascade trained files to detect the face area and the emotion in it.

Objectives:

Face Recognition:

-Input as an Image must be given using Webcam

-Train the image using a Recognizer

-Compare the image to live face recognition

Face Detection:

-Use trained cascade files as inputs to compare

-Recognize faces

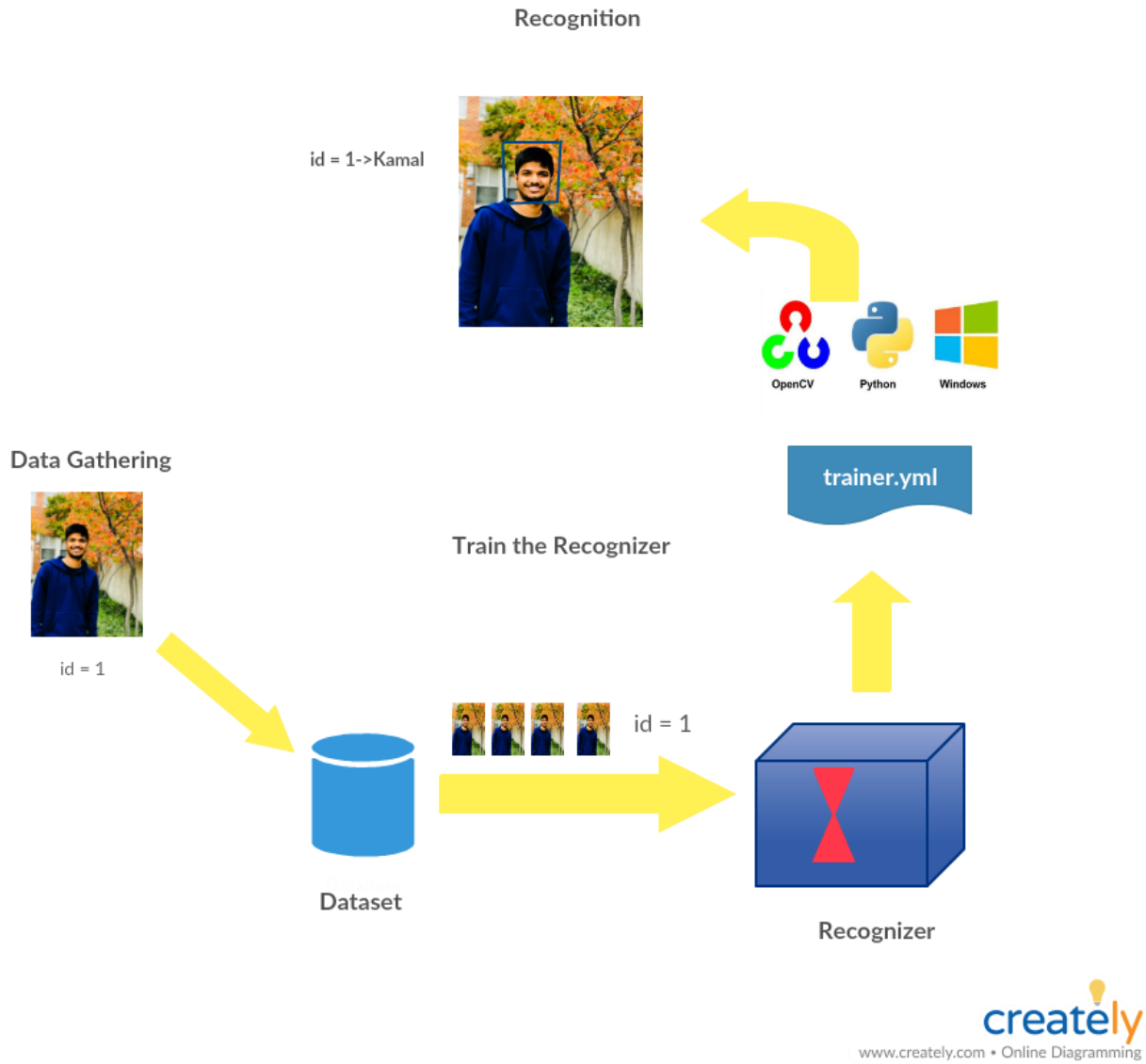
-Recognize emotions

Methodology:

Phase 1: Face Recognition:

- Create a directory for storing the images,each image is stored with id
- Now take all the data from dataset and train it using OpenCV Recognizer- **Local Binary Patterns Histograms (LBPH)**

- This is done by specific OpenCV function and the result is stored in .yml file
- Finally we capture a fresh image on our camera and if the person had his face trained before,our recognizer will make a prediction returning id and index,showing how confidently recognizer made the match



OpenCV face recognition Methods:

- **Eigenfaces**
- **Fisherfaces**
- **Local Binary Patterns Histograms (LBPH)**

Every one of the three techniques play out the acknowledgment by contrasting the face with be perceived with some preparation set of known countenances. In the preparation set, we supply the calculation faces and instruct it to which individual they have a place. At the point when the calculation is requested to perceive some obscure face, it utilizes the preparation set to make the acknowledgment

Why do we use LBPH rather than Eigenfaces/Fisher faces

Each of the three aforementioned methods uses the training set a bit differently.

Eigen Faces /Fisher Faces	Local Binary Patterns Histogram
Finds a mathematical description of the most dominant features of the training set as a whole	LBPH analyzes each face in the training set separately and independently.
Looks at the dataset as a whole	Simple, in the sense that we characterize each image in the dataset locally and when a new unknown image is provided, we perform the same analysis on it and compare the result to each of the images in the dataset
Not so great under same conditions	Works better in different environments and light conditions

Softwares and Libraries:

- OpenCV: If NumPy's main goal is large, efficient, multi-dimensional array representations, then, by far, the main goal of OpenCV is real-time image processing.
- Numpy: We can express images as multi-dimensional arrays, by using NumPy's built-in high-level mathematical functions, we can quickly perform numerical analysis on an image.

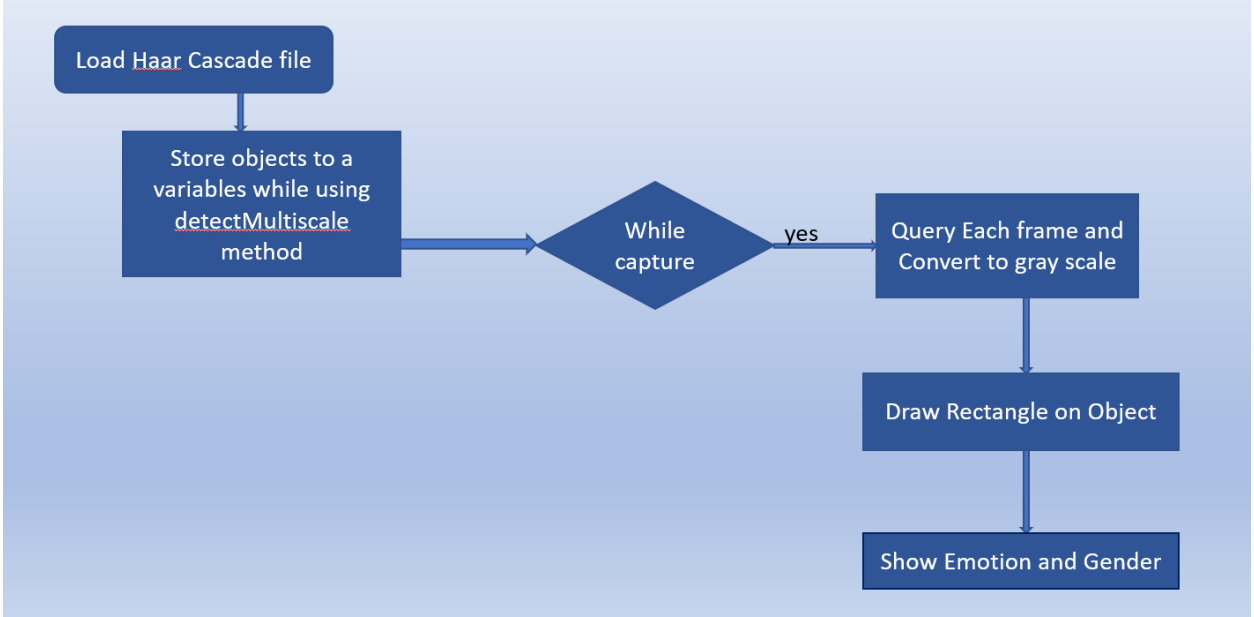
- Pillow: Image module provides a class with same name which is used to represent a PIL image. Also provides a number of factory functions to load images from files

OpenCV was designed for computational efficiency and with a strong focus on real-time applications. So, it's perfect for real-time face recognition using a camera.

Phase 2: Face Detection:

Cascades Play an important role in detecting a face and emotions in it

- Cascades are not actually neural networks ,they are deep series of filters not series of layers
- OpenCV cascades are not based on any neural network libraries so we don't need tensor flow
- **Haar cascades:** Fast, but less accurate. Can be a pain to tune parameters
- **Emotion Cascades**: Trained data taken from images
- **Gender Cascades**: Trained data taken from images



Here we used Fisher face model. Why FisherFace??

First of all, it is superior to utilizing Eigenface recognizer. Eigenfaces are the eigenvectors related to the biggest. It utilizes integer values in its prediction. This value is the related eigenvalue of the eigenvector.

In the interim, Fisherface utilizes Linear Derived Analysis (LDA) to decide the vector portrayal. It produces float value in the prediction. This likewise implies the outcome is better contrasted with Eigenface.

Libraries:

- OpenCV
- Numpy
- Glob : Basically, it has two functions that either return a list or an iterator of files in a directory using shell pattern matching

CODE SNIPPETS

Data Gathering: We will simply create a dataset, where we will store for each id, a group of photos in gray with the portion that was used for face detecting.

In the project directory we save the haarcascade.xml file

```
# Importing the libraries
import cv2
import os
cam = cv2.VideoCapture(0)
cam.set(3, 640) # set video width
cam.set(4, 480) # set video height
face_detector = cv2.CascadeClassifier('C:\\Users\kamal\PycharmProjects\Webcam-Face-Detect-master\haarcascade_frontalface_default.xml')
# For each person, enter one numeric face id
face_id = input('\n enter user id end press <return> ==>  ')
print("\n [INFO] Initializing face capture. Look the camera and wait ...")
# Initialize individual sampling face count
count = 0
while(True):
    ret, img = cam.read()
    img = cv2.flip(img, 1) # flip video image vertically
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    faces = face_detector.detectMultiScale(gray, 1.3, 5)
    for (x,y,w,h) in faces:
        cv2.rectangle(img, (x,y), (x+w,y+h), (255,0,0), 2)
        count += 1
        # Save the captured image into the datasets folder
        cv2.imwrite("C:\\Users\kamal\PycharmProjects\Webcam-Face-Detect-master\datasets/user." + str(face_id) + '.' + str(count) + ".jpg", gray[y:y+h,x:x+w])
        cv2.imshow('image', img)
    k = cv2.waitKey(100) & 0xff # Press 'ESC' for exiting video
    if k == 27:
        break
    elif count >= 30: # Take 30 face sample and stop video
         break
# Do a bit of cleanup
print("\n [INFO] Exiting Program and cleanup stuff")
cam.release()
cv2.destroyAllWindows()
```

Here we add, an "input command" to capture a user id, that should be an integer number (1, 2, 3, ….. etc)

```
face_id = input('\n enter user id end press <return> ==>  ')
```

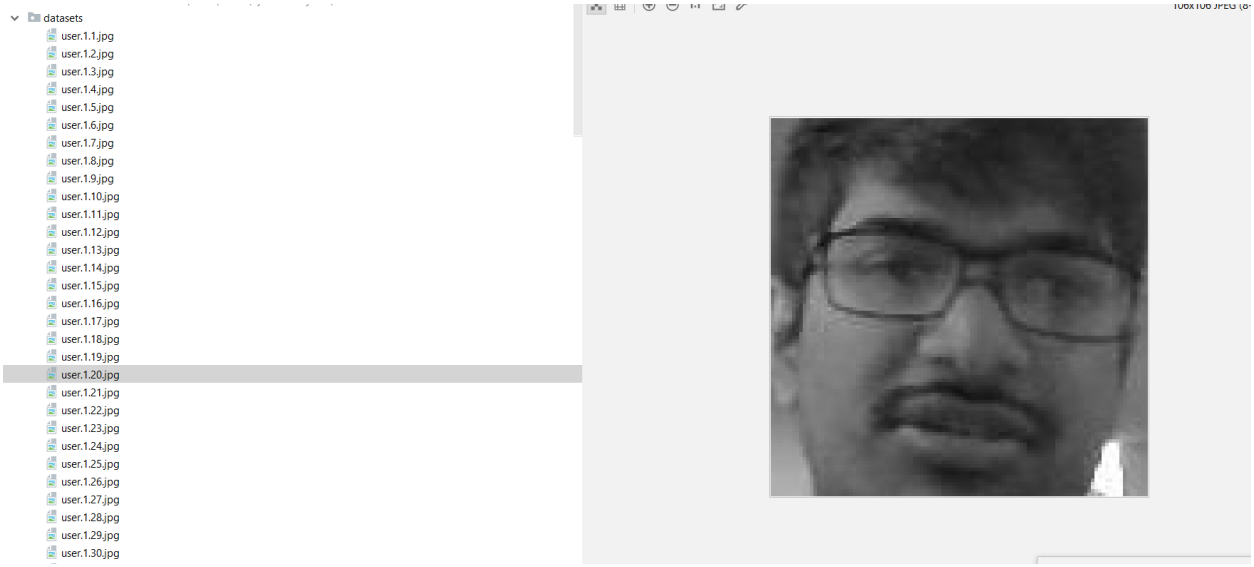
And for each one of the captured frames, we should save it as a file on a "dataset" directory:

```
cv2.imwrite("C:\\Users\kamal\PycharmProjects\Webcam-Face-Detect-master\datasets/user." + str(face_id) + '.' + str(count) + ".jpg", gray[y:y+h,x:x+w])
```

Note that for saving the above file, you must have imported the library "os". Each file's name will follow the structure

User.face_id.count.jpg

Consider a User, say face_id=1, the 20th sample on dataset will be like



For my program, I am capturing 35 samples from each id. Changing it is possible on the last "elif". The number of samples is used to break the loop where the face samples are captured.

Trainer:

We must take all user data from our dataset and "trainer" the OpenCV Recognizer. This is done directly by a specific OpenCV function. The result will be a .yml file that will be saved on a "trainer/" directory

```
# Path for face image database
import cv2
import os
import numpy as np
path = 'C:\\Users\kamal\PycharmProjects\Webcam-Face-Detect-master\datasets'
recognizer = cv2.face.LBPHFaceRecognizer_create()
detector = cv2.CascadeClassifier("haarcascade_frontalface_default.xml");
# function to get the images and label data
def getImagesAndLabels(path):
    imagePaths = [os.path.join(path,f) for f in os.listdir(path)]
    faceSamples=[]
    ids = []
    for imagePath in imagePaths:
        from PIL import Image
        PIL_img = Image.open(imagePath).convert('L') # convert it to grayscale
        img_numpy = np.array(PIL_img,'uint8')
        id = int(os.path.split(imagePath)[-1].split(".")[1])
        faces = detector.detectMultiScale(img_numpy)
        for (x,y,w,h) in faces:
            faceSamples.append(img_numpy[y:y+h,x:x+w])
            ids.append(id)
    return faceSamples,ids
print ("\n [INFO] Training faces. It will take a few seconds. Wait ...")
faces,ids = getImagesAndLabels(path)
recognizer.train(faces, np.array(ids))
# Save the model into trainer/trainer.yml
recognizer.write('C:\\Users\kamal\PycharmProjects\Webcam-Face-Detect-master\\trainer/trainer.yml') # recognizer.save()
# Print the numer of faces trained and end program
print("\n [INFO] {0} faces trained. Exiting Program".format(len(np.unique(ids))))
```

Recognizer that we use, the LBPH (LOCAL BINARY PATTERNS HISTOGRAMS) Face Recognizer, included on OpenCV package.

```
recognizer = cv2.face.LBPHFaceRecognizer_create()
```

The function "getImagesAndLabels (path)", will take all photos on directory: "pictures/", returning 2 arrays: "Ids" and "faces". With those arrays as input, we will "train our recognizer":

```
recognizer.train(faces, np.array(ids))
```

As a result, a file named "trainer.yml" will be saved in the trainer directory that was previously created by us.

That's it! I included the last print statement where I displayed for confirmation, the number of User's faces we have trained.

Recognizer:

Here, we will capture a fresh face on our camera and if this person had his face captured and trained before, our recognizer will make a "prediction" returning its id and an index, shown how confident the recognizer is with this match.

```
# iniciate id counter
id = 0
# names related to ids: example ==> Marcelo: id=1,  etc
names = ['None', 'Kamal', 'yu', 'tejid', 'abc', 'W']
# Initialize and start realtime video capture
cam = cv2.VideoCapture(0)
cam.set(3, 640)  # set video widht
cam.set(4, 480)  # set video height
# Define min window size to be recognized as a face
minW = 0.1 * cam.get(3)
minH = 0.1 * cam.get(4)
while True:
    ret, img = cam.read()
    img = cv2.flip(img, 1)  # Flip vertically
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

    faces = faceCascade.detectMultiScale(
        gray,
        scaleFactor=1.2,
        minNeighbors=5,
        minSize=(int(minW), int(minH)),
    )
    for (x, y, w, h) in faces:
        cv2.rectangle(img, (x, y), (x + w, y + h), (0, 255, 0), 2)
        id, confidence = recognizer.predict(gray[y:y + h, x:x + w])
        # Check if confidence is less them 100 ==> "0" is perfect match
        if (confidence < 100):
            id = names[id]
            confidence = "  {0}%".format(round(100 - confidence))
        else:
            id = "unknown"
            confidence = "  {0}%".format(round(100 - confidence))

        cv2.putText(img, str(id), (x + 5, y - 5), font, 1, (255, 255, 255), 2)
        cv2.putText(img, str(confidence), (x + 5, y + h - 5), font, 1, (255, 255, 0), 1)

    cv2.imshow('camera', img)
    k = cv2.waitKey(10) & 0xff  # Press 'ESC' for exiting video
```

A new array is included here, so we will display "names", instead of numbered ids:

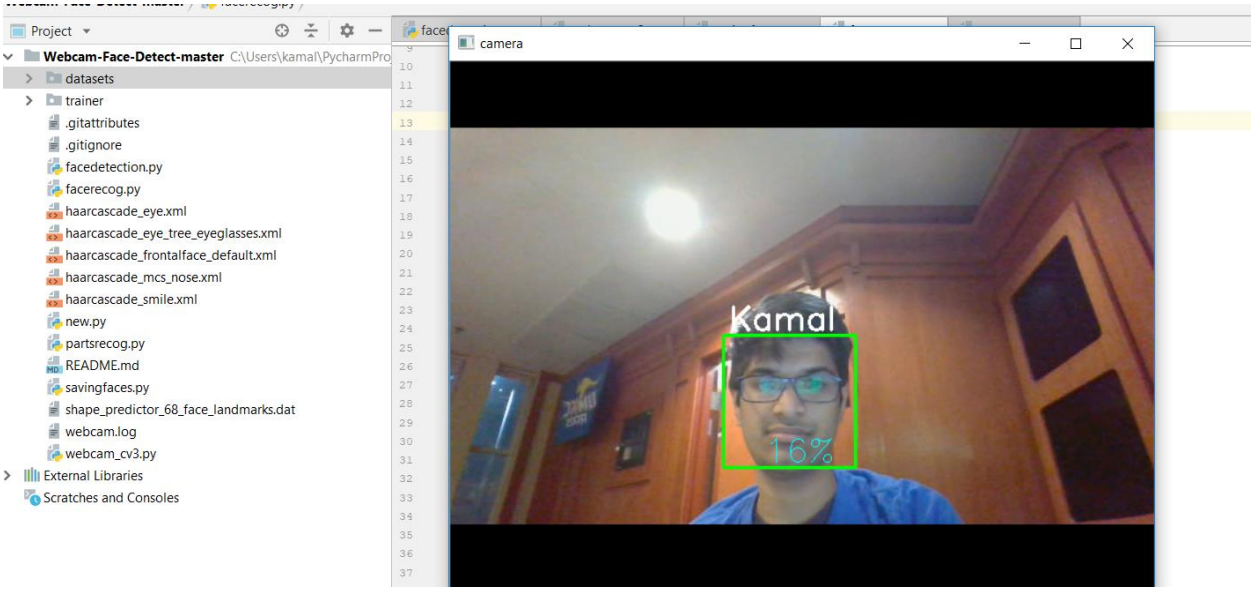
```
names = ['None', 'Kamal', 'yu', 'tejid', 'abc', 'W']
```

Now, we will be detecting a face, same as we did before with the haasCascade classifier. Having a detected face we can call the most important function in the above code

```
id, confidence = recognizer.predict(gray[y:y + h, x:x + w])
```

The recognizer.predict (), will take as a parameter a captured portion of the face to be analyzed and will return its probable owner, indicating its id and how much confidence the recognizer is in relation with this match.

If not, an "unknown" label is put on the face.



Evaluation:

Phase 1: Face Recognition

- Note: Confidence Index will return "zero" if it will be considered a perfect match
- And at last, if the recognizer could predict a face, we put a text over the image with the probable id and how much is the "probability" in % that the match is correct ("probability" = 100 - confidence index).
- We used LBPH algorithm for recognizing faces

Phase 2: Face Detection

Haar Cascade files are used to Detect the face region whereas emotion cascade and gender cascade for detecting emotions and gender

We trained Emotion and Gender data from ghent university dataset and Georgia Tech Dataset and the links for these are given at the end of the report.

```
ESC = 27

def start_webcam(model_emotion,model_gender, window_size, window_name='live', update_time=50):
    cv2.namedWindow(window_name, WINDOW_NORMAL)
    if window_size:
        width, height = window_size
        cv2.resizeWindow(window_name, width, height)

    video_feed = cv2.VideoCapture(0)
    video_feed.set(3, width)
    video_feed.set(4, height)
    read_value, webcam_image = video_feed.read()

    delay = 0
    init = True
    while read_value:
        read_value, webcam_image = video_feed.read()
        for normalized_face, (x, y, w, h) in find_faces(webcam_image):
          if init or delay == 0:
            init = False
            emotion_prediction = model_emotion.predict(normalized_face)
            gender_prediction = model_gender.predict(normalized_face)
          if (gender_prediction[0] == 0):
              cv2.rectangle(webcam_image, (x,y), (x+w, y+h), (0,0,255), 2)
          else:
              cv2.rectangle(webcam_image, (x,y), (x+w, y+h), (255,0,0), 2)
          cv2.putText(webcam_image, emotions[emotion_prediction[0]], (x,y-10), cv2.FONT_HERSHEY_SCRIPT_COMPLEX, 1.5, (255,0,0), 2)
          cv2.putText(webcam_image, gender[gender_prediction[0]], (10,20), cv2.FONT_HERSHEY_SCRIPT_COMPLEX, 1,
                      (255, 0, 0), 2)

        delay += 1
        delay %= 20
        cv2.imshow(window_name, webcam_image)
```

Below we can observe the emotions and genders we mentioned in the code that will be displayed as output when we do real time face detection

```
if __name__ == '__main__':
    emotions = ["afraid", "angry", "disgusted", "happy", "neutral", "sad", "surprised"]
    gender   = ["male","female"]
    # Loading the trained models of emotion
    fisher_face_emo = cv2.face.FisherFaceRecognizer_create()
    fisher_face_emo.read('C:\\Users\kamal\PycharmProjects\Webcam-Face-Detect-master\emotion_classifier_model.xml')

#loading the trained models of gender
    fisher_face_gen = cv2.face.FisherFaceRecognizer_create()
    fisher_face_gen.read('C:\\Users\kamal\PycharmProjects\Webcam-Face-Detect-master\gender_classifier_model.xml')

    # starting the model to predict
    choice = input("starting your webcam?(y/n) ")
    if (choice == 'y'):
        window_name = "Facifier Webcam (press ESC to exit)"
        begin_webcame(fisher_face_emo, fisher_face_gen, win_size=(1280, 720), win_name=window_name, updatetime=15)

    else:
        print("Invalid input, exiting program.")
```

Here we detect the faces that are used above emotion and gender classification

```
import cv2

faceCascade = cv2.CascadeClassifier('C:\\Users\kamal\PycharmProjects\Webcam-Face-Detect-master\haarcascade_frontalface_default.xm

def faces_find(image):
    coordinates = locate_faces(image)
    cropped_faces = [image[y:y + h, x:x + w] for (x, y, w, h) in coordinates]
    normalized_faces = [normalize_face(face) for face in cropped_faces]
    return zip(normalized_faces, coordinates)

def normalize_face(face):
    face = cv2.cvtColor(face, cv2.COLOR_BGR2GRAY)
    face = cv2.resize(face, (350, 350))

    return face;

def locate_faces(image):
    faces = faceCascade.detectMultiScale(
        image,
        scaleFactor=1.1,
        minNeighbors=15,
        minSize=(70, 70)
    )

    return faces
```

Emotion Training: Here we train the dataset required for emotion classification

First we load the dataset which we intend to train

```
#loading the dataset for training
file = glob.glob("C:\\Users\kamal\PycharmProjects\Webcam-Face-Detect-master\data\\raw_emotion\\{0}\\*" .format(emotion))
random.shuffle(file) #shuffling the data
train = file[:int(len(file) * training_size)]
```

Next we write it to an xml file

```
#Writing the trained dataset
    fisherface.write('C:\\Users\kamal\PycharmProjects\Webcam-Face-Detect-master\emotion_classifier_model.xml')
```

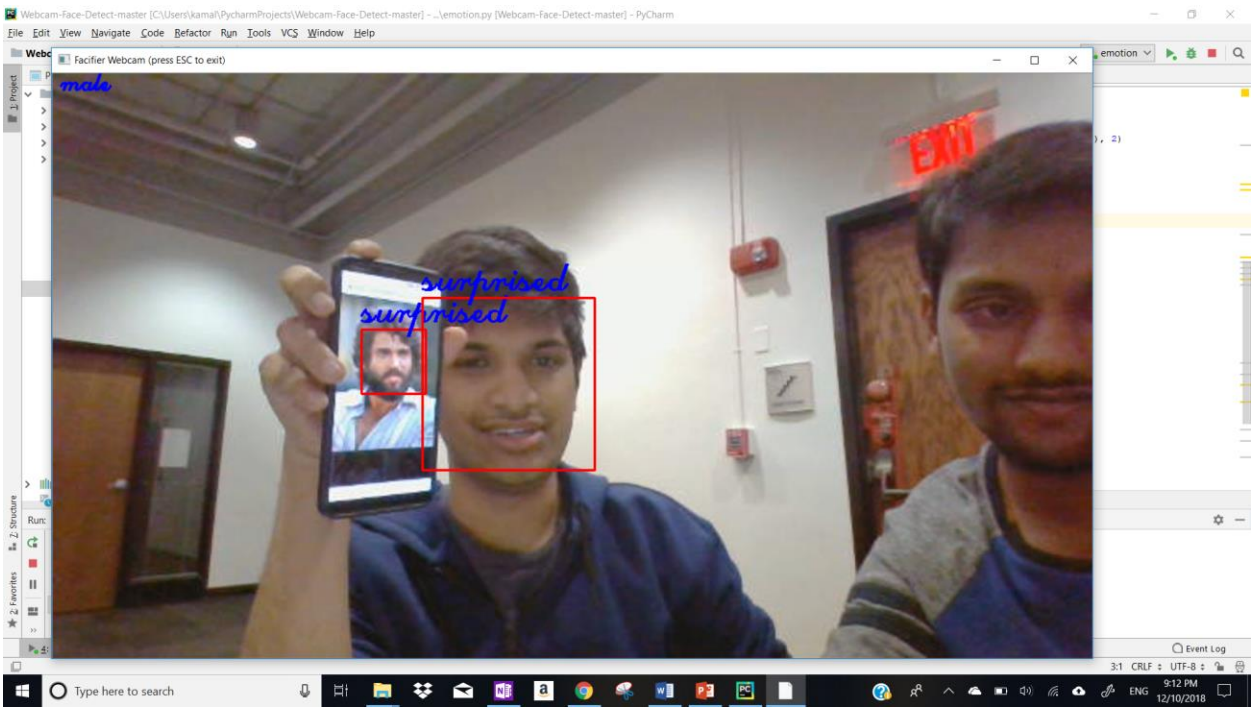
Gender Training: Here we train the dataset required for gender classification

First we load the dataset which we intend to train

```
file = glob.glob("E:\\cropped_faces\\{0}\\*" .format(gender))
```

Next we write it to an xml file

```
#writing the training data
    fisherface.write('D:\\models\gender_classifier_model.xml')
```

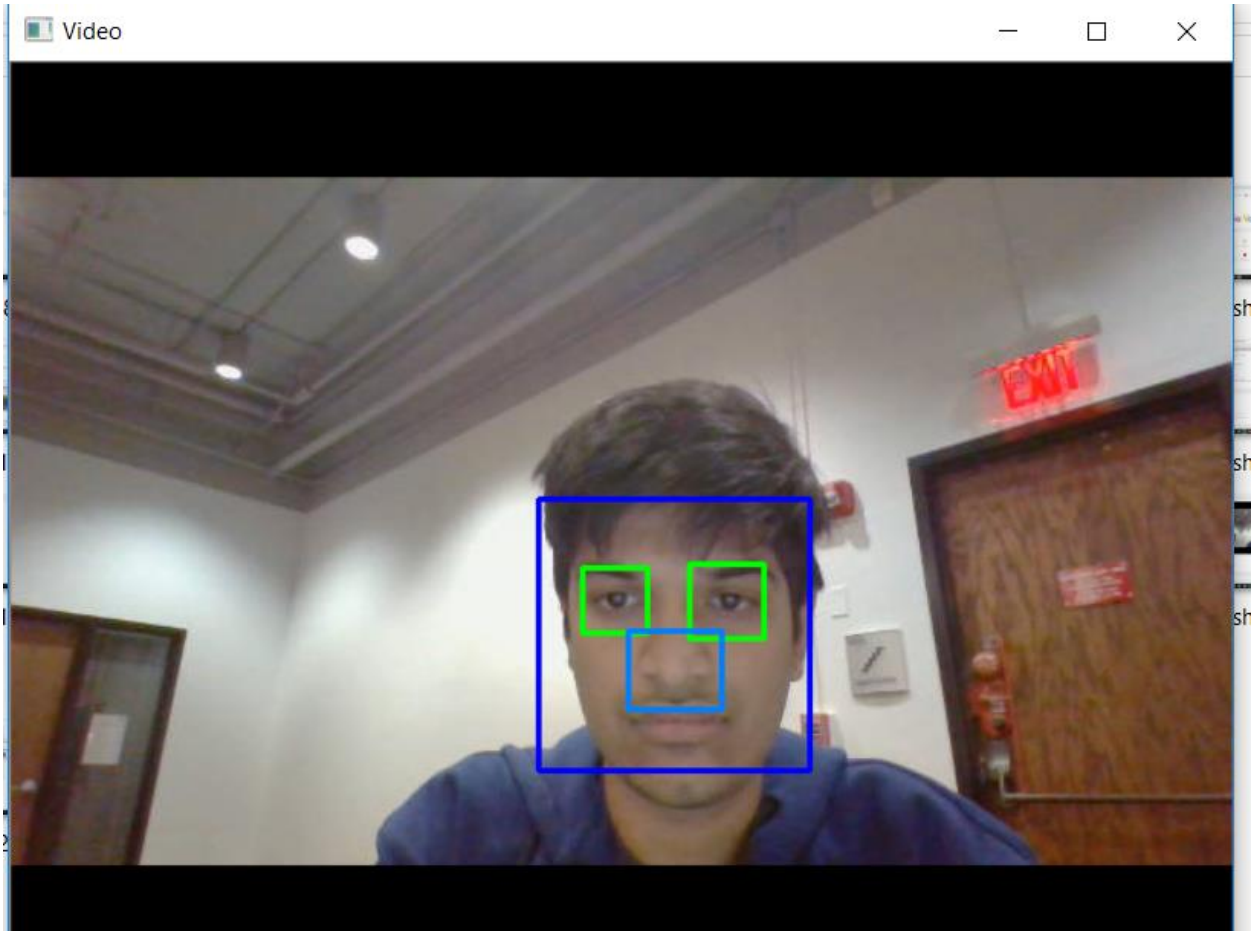


Extra features :

We also tried experimenting to find eyes and noses in images

```
 Cascades Loading
ace_cascade = cv2.CascadeClassifier('C:\\Users\kamal\PycharmProjects\Webcam-Face-Detect-master\haarcascade_frontalface_default.xml')
ye_cascade = cv2.CascadeClassifier('C:\\Users\kamal\PycharmProjects\Webcam-Face-Detect-master\haarcascade_eye.xml')
mile_cascade = cv2.CascadeClassifier('C:\\Users\kamal\PycharmProjects\Webcam-Face-Detect-master\haarcascade_smile.xml')
lasses_cascade = cv2.CascadeClassifier('C:\\Users\kamal\PycharmProjects\Webcam-Face-Detect-master\haarcascade_eye_tree_eyeglasses.xml')
ose_cascade = cv2.CascadeClassifier('C:\\Users\kamal\PycharmProjects\Webcam-Face-Detect-master\haarcascade_mcs_nose.xml')

 Defining a function that will do the detections
ef detect(gray, frame):
    faces = face_cascade.detectMultiScale(gray, 1.3, 5)
    for (x, y, w, h) in faces:
        cv2.rectangle(frame, (x, y), (x+w, y+h), (255, 0, 0), 2)
        roi_gray = gray[y:y+h, x:x+w]
        roi_color = frame[y:y+h, x:x+w]
        eyes = eye_cascade.detectMultiScale(roi_gray, 1.1, 22)
        for (ex, ey, ew, eh) in eyes:
            cv2.rectangle(roi_color, (ex, ey), (ex+ew, ey+eh), (0, 255, 0), 2)
        nose = nose_cascade.detectMultiScale(roi_gray, 1.4, 22)
        for (nx, ny, nw, nh) in nose:
            cv2.rectangle(roi_color, (nx, ny), (nx+nw, ny+nh), (255, 127, 0), 2)
        smiles = smile_cascade.detectMultiScale(roi_gray, 1.7, 22)
        for (sx, sy, sw, sh) in smiles:
            cv2.rectangle(roi_color, (sx, sy), (sx+sw, sy+sh), (0, 0, 255), 2)
            print("smiled")
    return frame
```



Conclusion:

During the project we learned how to work on Face recognition with confidence index by training the model using LBPH model which is better used for calculating probability for face match under different environments.

And for phase 2 that is for detecting face along with emotion and gender of a person is gone using haar cascade for face detection and training datasets for emotion and gender classification using Fisher faces

Links:

Github Link: https://github.com/Kamaltejveerapaneni/Face-Detection

Dataset Links:

- https://www.ugent.be/pp/ekgp/en/research/research-groups/panlab/kdef

- http://www.anefian.com/research/face_reco.htm

References:

- https://www.pyimagesearch.com/2018/09/24/opencv-face-recognition/
- https://www.udemy.com/computer-vision-a-z/learn/v4/overview
- https://data.vision.ee.ethz.ch/cvl/rrothe/imdb-wiki/
- https://docs.opencv.org/3.1.0/da/d60/tutorial_face_main.html
- http://vis-www.cs.umass.edu/lfw/#download
- https://docs.opencv.org/3.1.0/dc/da5/tutorial_py_drawing_functions.html