ZAKIR HUSAIN COLLEGE OF ENGINEERING AND TECHNOLOGY
ALIGARH MUSLIM UNIVERSITY, ALIGARH

Project Report

On

"Face Detection and Recognition using **OpenCV**"

Submitted to: DIGINIQUE TECHLABS

Submitted by: Rajneesh Sharma
Btech (Computer Engineering)
2nd year
Z.H. College of Engineering and
Technology, Aligarh,202002

Face Detection and Recognition model using OpenCV

Acknowledgement

I have taken efforts in this project. First of all I am thankful to Bipul Shahi Sir for his exceptional teaching due to which I have got such understanding in the particular topic- Machine learning, Data Analytics, with python. His way of teaching was quite brilliant.

I have tried my best to complete this project. Also I have researched a lot about this topic (that I have chosen as project) and learned a lot about this topic.I have referred some websites and blogs to complete this project.

Table of contents

Objective

The purpose of this project is to detect and recognize the human faces using Open CV. I could have used tensorflow and deep learning (neural networks) but here I have done this project using OpenCV and LBPH Face Recognizer for detecting multiple faces and recognizing multiple faces. It is a quite accurate model but still I feel its accuracy can be increased if we use the deep learning/neural network model.

We can either use the inbuilt webcam or an external one.

Technologies Used

Operating System: Windows

Tools Used: Anaconda

Scripting language: Python3

Libraries Used

• **Pillow** - Python Imaging Library is a free library for the Python programming language that adds support for opening, manipulating, and saving many different image file formats.

• **Matplotlib** - Matplotlib is a plotting library for the Python programming language and its numerical mathematics extension Numpy.

• **Pandas** – Pandas is a popular Python package for data science, it offers expressive and flexible data structures that make data manipulation and analysis easy.

• **Opencv** - OpenCV is a library of Python bindings designed to solve computer vision problems.

• **Numpy** - NumPy is a library for the Python programming language, adding support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays.

4.Introduction

4.1 Artificial Intelligence

The term artificial intelligence was coined in 1956, but AI has become more popular today thanks to increased data volumes, advanced algorithms, and improvements in computing power and storage.

Early AI research in the 1950s explored topics like problem solving and symbolic methods. In the 1960s, the US Department of Defence took interest in this type of work and began training computers to mimic basic human reasoning. For example, the Defence Advanced Research Projects Agency (DARPA) completed street mapping projects in the 1970s. And DARPA produced intelligent personal assistants in 2003, long before Siri, Alexa or Cortana were household names.

This early work paved the way for the automation and formal reasoning that we see in computers today, including decision support systems and smart search systems that can be designed to complement and augment human abilities.

While Hollywood movies and science fiction novels depict AI as human-like robots that take over the world, the current evolution of AI technologies isn’t that scary – or quite that smart. Instead, AI has evolved to provide many specific benefits in every industry. Keep reading for modern examples of artificial intelligence in health care, retail and more.

Advantages of A.I. are as follows:

1) AI automates repetitive learning and discovery through data.
2) AI adds intelligence to existing products.
3) AI adapts through progressive learning algorithms to let the data do the programming.
4) AI analyzes more and deeper data using neural networks that have many hidden layers.
5) AI gets the most out of data.

4.2 Machine Learning

Because of new computing technologies, machine learning today is not like machine learning of the past. It was born from pattern recognition and the theory that computers can learn without being programmed to perform specific tasks; researchers interested in artificial intelligence wanted to see if computers could learn from data. The iterative aspect of machine learning is important because as models are exposed to new data, they are able to independently adapt. They learn from previous computations to produce reliable, repeatable decisions and results. It's a science that's not new – but one that has gained fresh momentum.

While many machine learning algorithms have been around for a long time, the ability to automatically apply complex mathematical calculations to big data – over and over, faster and faster – is a recent development.

Machine learning algorithms are often categorized as supervised or unsupervised.

- Supervised machine learning algorithms can apply what has been learned in the past to new data using labeled examples to predict future events. Starting from the analysis of a known training dataset, the learning algorithm produces an inferred function to make predictions about the output values. The system is able to provide targets for any new input after sufficient training. The learning algorithm can also compare its output with the correct, intended output and find errors in order to modify the model accordingly.
- In contrast, unsupervised machine learning algorithms are used when the information used to train is neither classified nor labeled. Unsupervised learning studies how systems can infer a function to describe a hidden structure from unlabeled data. The system doesn't figure out the right output, but it explores the data and can draw inferences from datasets to describe hidden structures from unlabeled data.
- Semi-supervised machine learning algorithms fall somewhere in between supervised and unsupervised learning, since they use both labeled and unlabeled data for training – typically a small amount of labeled data and a large amount of unlabeled data. The systems that use this method are able to considerably improve learning accuracy. Usually, semi-supervised learning is chosen when the acquired labeled data requires skilled and relevant resources in

order to train it / learn from it. Otherwise, acquiring unlabeled data generally doesn’t require additional resources.

- Reinforcement machine learning algorithms is a learning method that interacts with its environment by producing actions and discovers errors or rewards. Trial and error search and delayed reward are the most relevant characteristics of reinforcement learning. This method allows machines and software agents to automatically determine the ideal behaviour within a specific context in order to maximize its performance. Simple reward feedback is required for the agent to learn which action is best; this is known as the reinforcement signal.

4.3 Deep Learning

Deep learning achieves recognition accuracy at higher levels than ever before. This helps consumer electronics meet user expectations, and it is crucial for safety-critical applications like driverless cars. Recent advances in deep learning have improved to the point where deep learning outperforms humans in some tasks like classifying objects in images.

While deep learning was first theorized in the 1980s, there are two main reasons it has only recently become useful:

1. Deep learning requires large amounts of labeled data. For example, driverless car development requires millions of images and thousands of hours of video.

Deep learning requires substantial computing power. High-performance GPUs have a parallel architecture that is efficient for deep learning. When combined with clusters or cloud computing, this enables development teams to reduce training time for a deep learning network from weeks to hours or less.

5. Installing Anaconda and Setting Environment

For installing Anaconda 2.0 in windows and setting up the environment you can follow the below mentioned link:

https://docs.anaconda.com/anaconda/install/windows/

After that you need to install the following libraries (in case not previously installed)

1.OpenCV: For installing OpenCV just open the Anaconda prompt and type any one of the command

- *conda install -c conda-forge opencv*
- *pip install opencv-python (if pip is installed)*

2.Pillow: For installing Pillow type the following command in anaconda prompt.

- *pip install pillow (if pip is installed)*
- *conda install pillow*

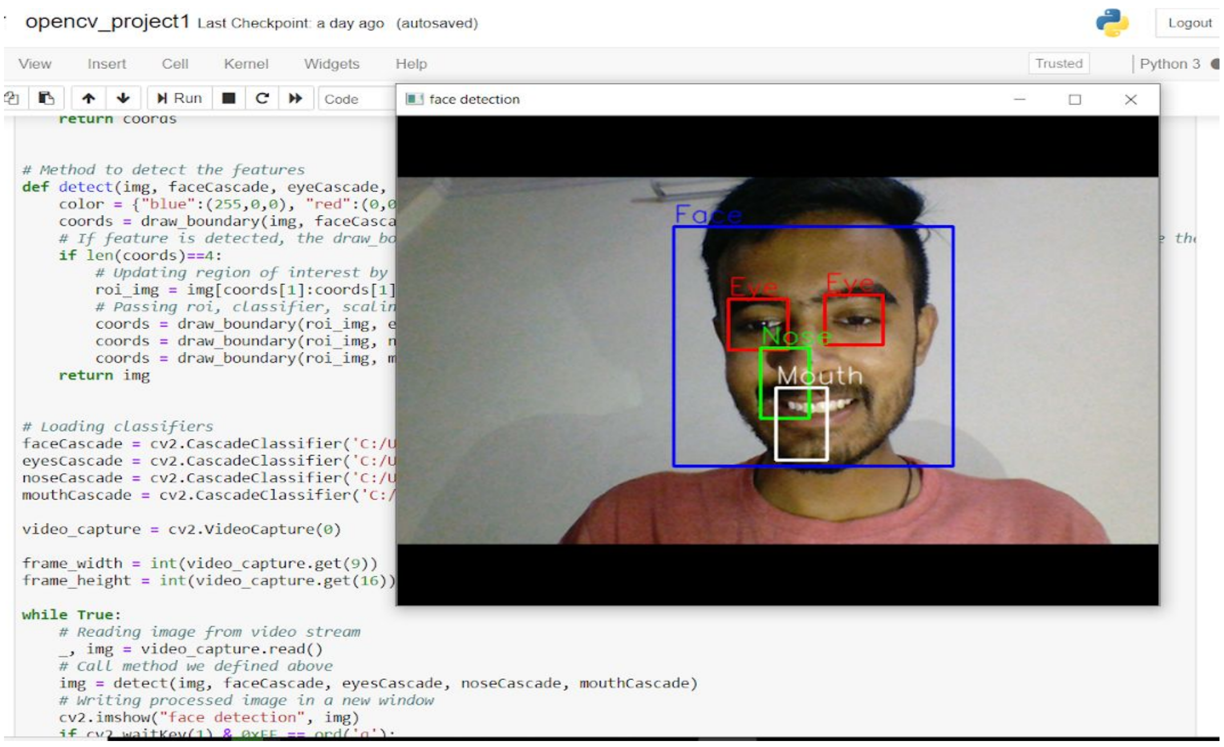
or else follow the below link:
https://pypi.python.org/packages/3.4/P/Pillow/Pillow-2.5.3.win-amd64-py3.4.exe#md5=6ee659d7b945e826a07c53c15578424f

6.Face Detection

Face detection and recognition is technology which is used to identify a person from a video or photo source. In the 1960s face recognition was introduced by Woodrow Wilson Bledsoe. Bledsoe developed a device that could classify photos of faces by hand using what's known as a RAND tablet, a device that people could use to input horizontal and vertical coordinates on a grid using a pen like stylus that emitted electromagnetic pulses. Ever since then the recognition system is being improved and optimized constantly, the technology becomes gradually mature and is more and more widely used in human daily life. It has been used increasingly for forensics by law enforcement and military professionals. In fact, facial recognition system was used to help confirm the identity of Osama bin Laden after he was killed in a U.S. raid. The face recognition system is also being increasingly used in the mobiles for device security. In this paper, we propose a face detection and recognition system using python along with OpenCV package. This system contains three modules which are detection, training and recognition. Basically, the detection module detects the face which gets into the field of vision of the camera and saves the face in the form of an image in JPG format.

Then the training modules trains the system using Haar cascade algorithm which was proposed by Paul Viola and Michael Jones in their paper.
The very first thing I have done in this project is I have detected the human face using haar cascade file and after detecting face I have detected the features of the face using other classifier files made for detecting those features.
I have used four cascade classifier files each for detecting the different features of face.The cascade classifier consists of a number of stages, where each stage is a group of weak learners. These weak learners are simple classifiers called decision stumps. Each stage is trained using a method called boosting. Boosting provides the ability to train a highly accurate classifier by taking the weighted average of decisions made by the weak learners.
Finally, in the recognition module the principal components of the face from the new video are extracted. Then those features get compared with the list of elements stored during training and the ones with the best match is found and name of the person recognized is displayed. This monitoring system fulfills the basic needs of face detection and recognition system, also takes the cost into consideration to ensure the pervasive mode as economical as possible. Furthermore, it can also be combined with real-time analysis.

**Screenshot of the real time face detection.*


opencv_project1 Last Checkpoint: a day ago (autosaved)
Logout
View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code
face detection
Face
Eye
Eye
Nose
Mouth

```
    return coords


# Method to detect the features
def detect(img, faceCascade, eyeCascade,
    color = {"blue":(255,0,0), "red":(0,0
    coords = draw_boundary(img, faceCasca
    # If feature is detected, the draw_bo
    if len(coords)==4:
        # Updating region of interest by
        roi_img = img[coords[1]:coords[1]
        # Passing roi, classifier, scalin
        coords = draw_boundary(roi_img, e
        coords = draw_boundary(roi_img, n
        coords = draw_boundary(roi_img, m
    return img


# Loading classifiers
faceCascade = cv2.CascadeClassifier('C:/U
eyesCascade = cv2.CascadeClassifier('C:/U
noseCascade = cv2.CascadeClassifier('C:/U
mouthCascade = cv2.CascadeClassifier('C:/

video_capture = cv2.VideoCapture(0)

frame_width = int(video_capture.get(9))
frame_height = int(video_capture.get(16))

while True:
    # Reading image from video stream
    _, img = video_capture.read()
    # Call method we defined above
    img = detect(img, faceCascade, eyesCascade, noseCascade, mouthCascade)
    # Writing processed image in a new window
    cv2.imshow("face detection", img)
    if cv2.waitKey(1) & 0xFF == ord('q'):
```

For more details on haar cascades and face detection you can visit the following link:
https://becominghuman.ai/face-detection-using-opencv-with-haar-cascade-classifiers-941dbb25177

We have used the **detectMultiscale** module from OpenCV. What this does is create a rectangle with coordinates (x,y,w,h) around the face detected in the image. This contains code parameters that are the most important to consider.

scaleFactor: The value indicates how much the image size is reduced at each image scale. A lower value uses a smaller step for downscaling. This allows the algorithm to detect the face. It has a value of x.y, where x and y are arbitrary values you can set.

minNeighbors: This parameter specifies how many "neighbors" each candidate rectangle should have. A higher value results in less detections but it detects higher quality in an image. You can use a value of X that specifies a finite number.

minSize: The minimum object size. By default it is (30,30). The smaller the face in the image, it is best to adjust the minSize value lower.

7.Face Recognition

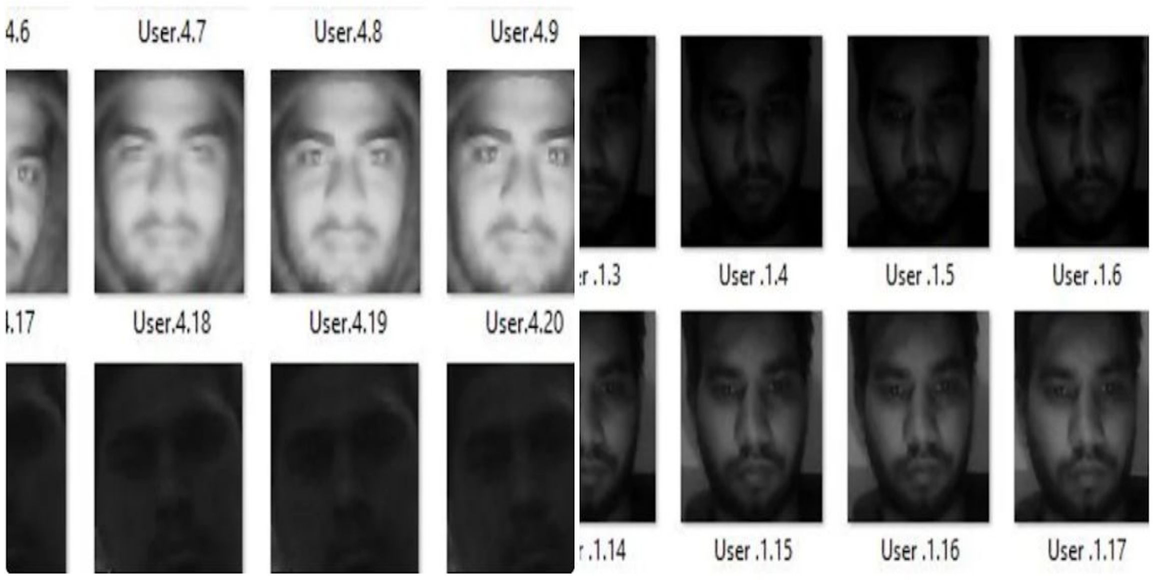
Human beings perform face recognition automatically every day and practically with no effort.Although it sounds like a very simple task for us, it has proven to be a complex task for a computer, as it has many variables that can impair the accuracy of the methods, for example: illumination variation, low resolution, occlusion, amongst other.

In computer science, face recognition is basically the task of recognizing a person based on its facial image. It has become very popular in the last two decades, mainly because of the new methods developed and the high quality of the current videos/cameras.

For Face recognition I followed the particular steps:

1. Generate Datasets
2. Training the data
3. Recognizing the person/Testing

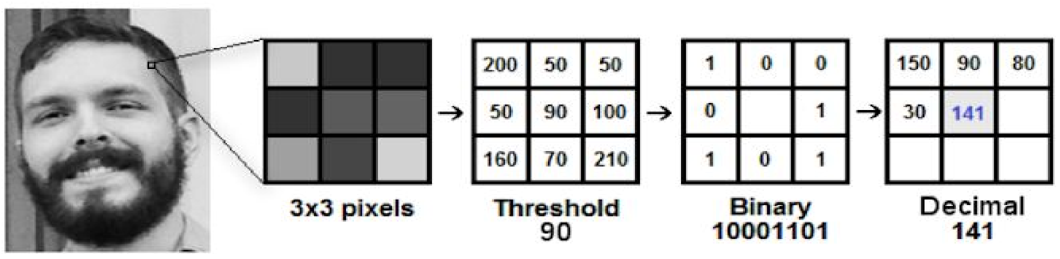
1.Generating Dataset: First of all we will create the dataset. The dataset that contains multiple images of a single user so that the training of data can be more efficient and better. As you can see in the code that I have created a function that captures many pictures of a person for better training of data.Also we will convert the images captured to grey color so that we can have the better classification. This kind of dataset will be created but we are not saving the grey images,instead we are saving normal images but training them after converting them to grey.



2. Training the model: Now, we need to train the algorithm. To do so, we need to use a dataset with the facial images of the people we want to recognize. We need to also set an ID (it may be a number or the name of the person) for each image, so the algorithm will use this information to recognize an input image and give you an output. Images of the same person must have the same ID.
The more images used in training the better. Normally a lot of images are used for training a face recognizer so that it can learn different looks of the same person, for example with glasses, without glasses, laughing, sad, happy, crying, with beard, without beard etc.
I am using OpenCV's **LBP** face detector. I convert the image to grayscale because most operations in OpenCV are performed in gray scale, then I load LBP face detector using cv2.CascadeClassifier class. After that I use **cv2.CascadeClassifier** class' **detectMultiScale** method to detect all the faces in the image. From detected faces I only pick the first face because in one image there will be only one face (under the assumption that there will be only one prominent face). As faces returned by detectMultiScale method are actually rectangles (x, y, width, height) and not actual faces images so we have to extract face image area from the main image. So I extract face area from gray image and return both the face image area and face rectangle.

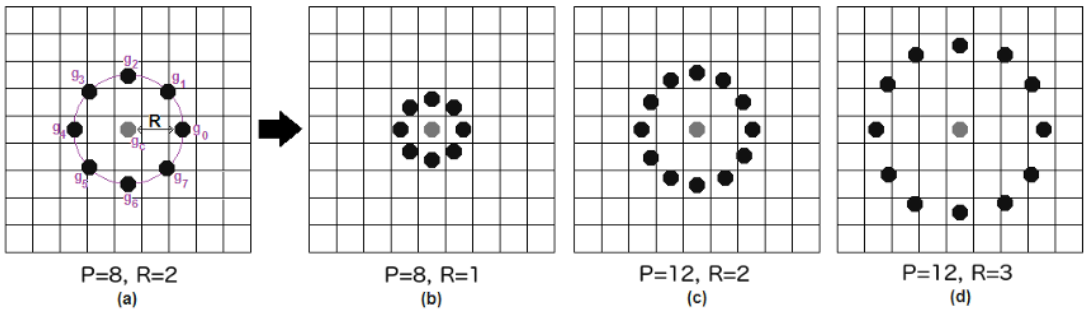
Applying the LBP operation: The first computational step of the LBPH is to create an intermediate image that describes the original image in a better way, by highlighting the facial characteristics. To do so, the algorithm uses a concept of a sliding window, based on the parameters **radius** and **neighbors**.

The image below shows this procedure:



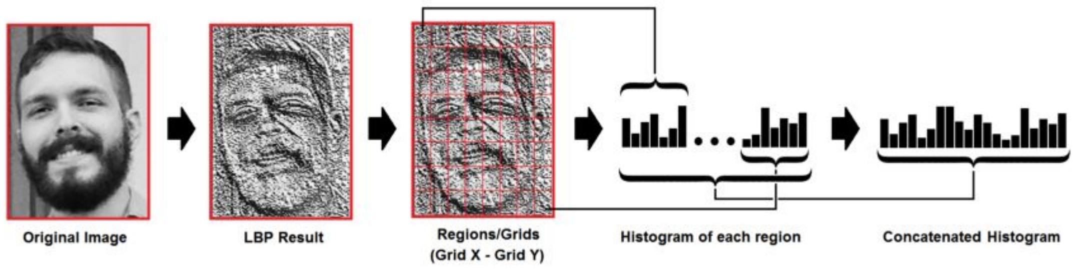
Based on the image above, let's break it into several small steps so we can understand it easily:

- Suppose we have a facial image in grayscale.
- We can get part of this image as a window of 3x3 pixels.
- It can also be represented as a 3x3 matrix containing the intensity of each pixel (0~255).
- Then, we need to take the central value of the matrix to be used as the threshold.
- This value will be used to define the new values from the 8 neighbors.
- For each neighbor of the central value (threshold), we set a new binary value. We set 1 for values equal or higher than the threshold and 0 for values lower than the threshold.
- Now, the matrix will contain only binary values (ignoring the central value). We need to concatenate each binary value from each position from the matrix line by line into a new binary value (e.g. 10001101). Note: some authors use other approaches to concatenate the binary values (e.g. clockwise direction), but the final result will be the same.
- Then, we convert this binary value to a decimal value and set it to the central value of the matrix, which is actually a pixel from the original image.
- At the end of this procedure (LBP procedure), we have a new image which represents better the characteristics of the original image.
- **Note**: The LBP procedure was expanded to use a different number of radius and neighbors, it is called Circular LBP.



It can be done by using **bilinear interpolation**. If some data point is between the pixels, it uses the values from the 4 nearest pixels (2x2) to estimate the value of the new data point.

Extracting the Histograms: Now, using the image generated in the last step, we can use the **Grid X** and **Grid Y** parameters to divide the image into multiple grids, as can be seen in the following image:



Based on the image above, we can extract the histogram of each region as follows:

- As we have an image in grayscale, each histogram (from each grid) will contain only 256 positions (0~255) representing the occurrences of each pixel intensity.
- Then, we need to concatenate each histogram to create a new and bigger histogram. Supposing we have 8x8 grids, we will have 8x8x256=16.384 positions in the final histogram. The final histogram represents the characteristics of the image original image.

The LBPH algorithm is pretty much it.

Performing the face recognition: In this step, the algorithm is already trained. Each histogram created is used to represent each image from the training dataset. So, given an input image, we perform the steps again for this new image and creates a histogram which represents the image.

- So to find the image that matches the input image we just need to compare two histograms and return the image with the closest histogram.
- We can use various approaches to compare the histograms (calculate the distance between two histograms), for example: **euclidean distance**, **chi-square**, **absolute value**, etc. In this example, we can use the Euclidean distance (which is quite known) based on the following formula:

$$D = \sqrt{\sum_{i=1}^{n}(hist1_i - hist2_i)^2}$$

- So the algorithm output is the ID from the image with the closest histogram. The algorithm should also return the calculated distance, which can be used as a '**confidence**' measurement. **Note**: don't be fooled about the 'confidence' name, as lower confidences are better because it means the distance between the two histograms is closer.
- We can then use a threshold and the 'confidence' to automatically estimate if the algorithm has correctly recognized the image. We can assume that the algorithm has successfully recognized if the confidence is lower than the threshold defined.

Conclusions

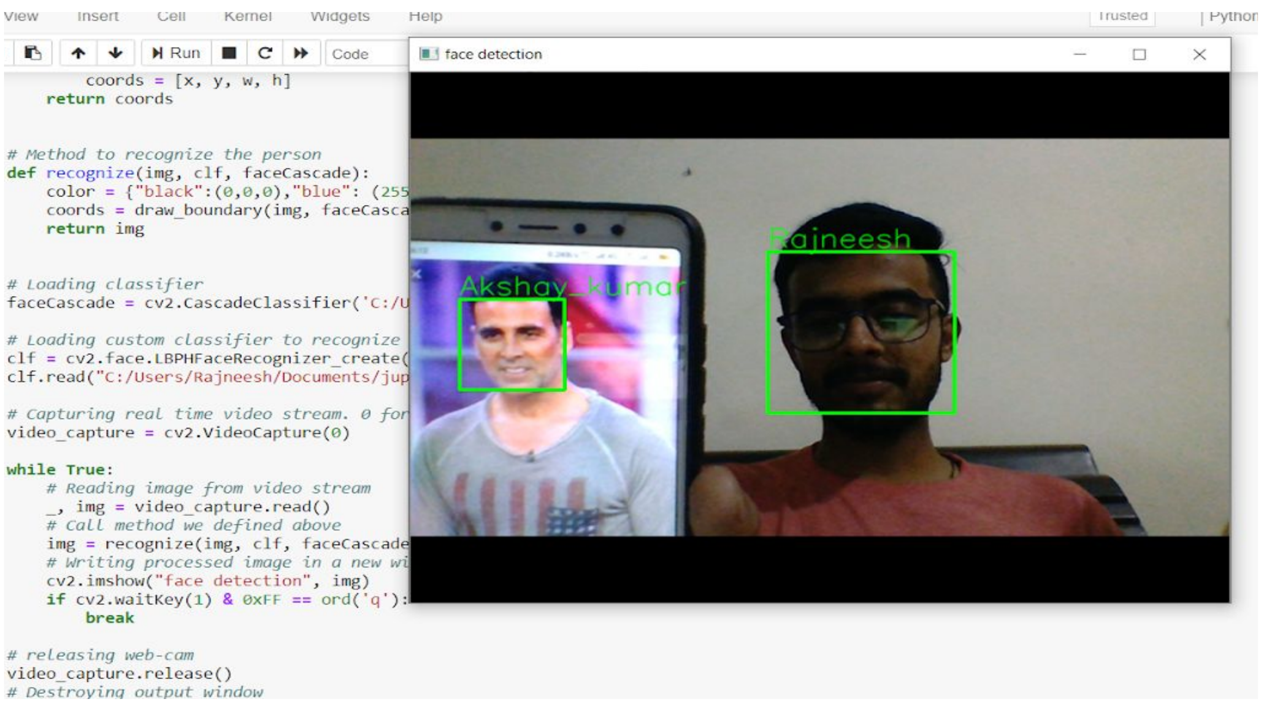
- LBPH is one of the easiest face recognition algorithms.
- It can represent local features in the images.
- It is possible to get great results (mainly in a controlled environment).

- It is robust against monotonic gray scale transformations.
- It is provided by the OpenCV library (Open Source Computer Vision Library).

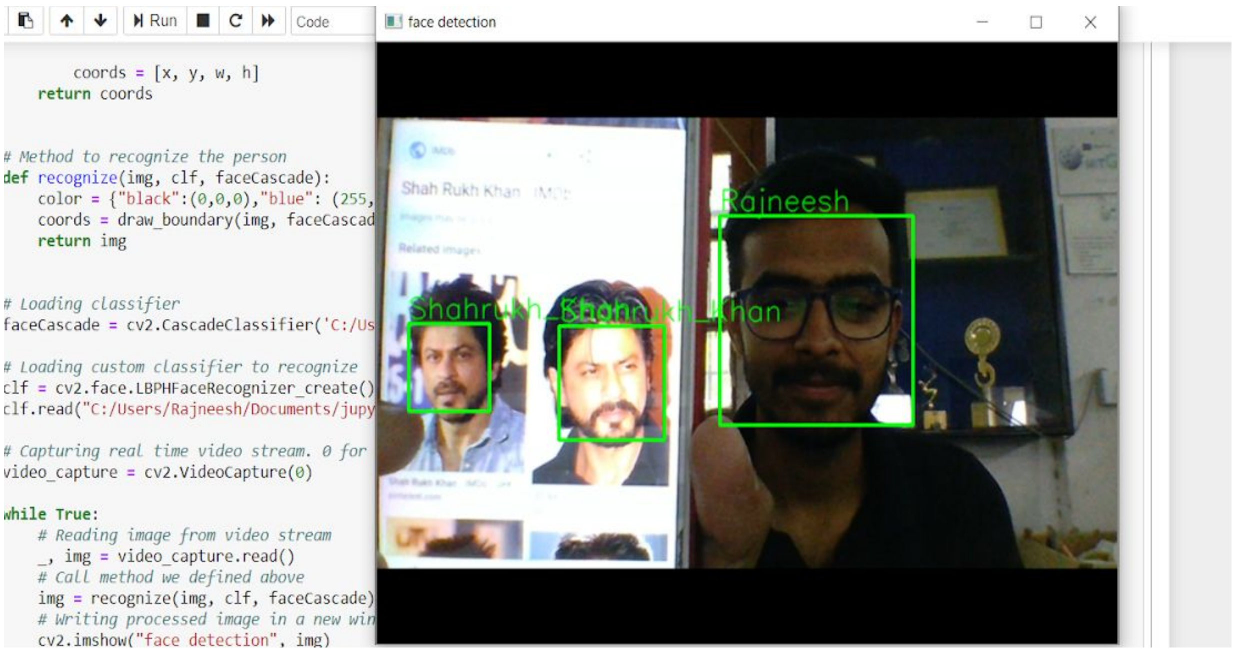
So in this way we created the classifier file and made it a **.yml** file so that it becomes easy for us to read that file.

3. Face Recognition: After all this now we are ready to run our recognizer file. Now when we run the recognizer file , BOOM we will see the name of the person on the screen for which model has been trained.Each person has been assigned an id, so when the classifier file matches the detected person it passes the id and according to the id, the name of the person splashes on the screen.
The Model has been trained for 5 people:

1. Rajneesh Sharma(that's me)
2. Virat Kohli
3. Shahrukh Khan
4. Akshay Kumar
5. Chris Evans (aka Captain America)



```
        coords = [x, y, w, h]
    return coords


# Method to recognize the person
def recognize(img, clf, faceCascade):
    color = {"black":(0,0,0),"blue": (255
    coords = draw_boundary(img, faceCasca
    return img


# Loading classifier
faceCascade = cv2.CascadeClassifier('C:/U

# Loading custom classifier to recognize
clf = cv2.face.LBPHFaceRecognizer_create(
clf.read("C:/Users/Rajneesh/Documents/jup

# Capturing real time video stream. 0 for
video_capture = cv2.VideoCapture(0)

while True:
    # Reading image from video stream
    _, img = video_capture.read()
    # Call method we defined above
    img = recognize(img, clf, faceCascade
    # Writing processed image in a new wi
    cv2.imshow("face detection", img)
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

# releasing web-cam
video_capture.release()
# Destroying output window
```



```
        coords = [x, y, w, h]
    return coords


# Method to recognize the person
def recognize(img, clf, faceCascade):
    color = {"black":(0,0,0),"blue": (255,
    coords = draw_boundary(img, faceCascad
    return img


# Loading classifier
faceCascade = cv2.CascadeClassifier('C:/Us

# Loading custom classifier to recognize
clf = cv2.face.LBPHFaceRecognizer_create()
clf.read("C:/Users/Rajneesh/Documents/jupy

# Capturing real time video stream. 0 for
video_capture = cv2.VideoCapture(0)

while True:
    # Reading image from video stream
    _, img = video_capture.read()
    # Call method we defined above
    img = recognize(img, clf, faceCascade)
    # Writing processed image in a new win
    cv2.imshow("face detection", img)
```

After seeing the screenshots we came to the conclusion that our model is working fine. Although it's not that perfect but still quite considerable. All the files are uploaded on my Github profile including the haar cascade files, image dataset and all codes.Kindly visit the profile for the files.

My Github link : *https://github.com/rajneesh44/Face_Recognition*

8. CONCLUSION

Hence, after lots of efforts and research I have completed this project and looking forward to pick more projects in future based on the same concept but make them with better accuracy.
Thanks to Bipul Sir again for his excellent teaching.

* * * * * * * *