PROJECT REPORT

FACE RECOGNITION AND EXPRESSION RECOGNITION

Submitted by

Name	Registration Number
Arjun Rao Rachakonda	11903860

INT 246

Under the Guidance of

Dr Sagar Pande

School of Computer Science and Engineering

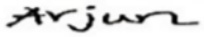
LOVELY PROFESSIONAL UNIVERSITY

DECLARATION

I hereby declare that the project work entitled "**FACE RECOGNITION AND EXPRESSION RECOGNITION**" is an authentic record of our own work carried out as requirements of Project for the award of **B.Tech** degree in **Computer Science and Engineering** from Lovely Professional University, Phagwara, under the guidance of **Dr Sagar Pande**, during August to November 2021. All the information furnished in this project report is based on our own intensive work and is genuine.

Arjun Rao Rachakonda

11903860



16th Nov, 2021

CERTIFICATE

This is to certify that the declaration statement made by this group of students is correct to the best of my knowledge and belief. They have completed this Project under my guidance and supervision. The present work is the result of their original investigation, effort and study. No part of the work has ever been submitted for any other degree at any University. The Project is fit for the submission and partial fulfilment of the conditions for the award of **B.Tech** degree in **Computer Science and Engineering** from Lovely Professional University, Phagwara.

Dr Sagar Pande

Professor

School of Computer Science and Engineering,
Lovely Professional University,
Phagwara, Punjab.

Date: 20/11/2021

ACKNOWLEDGEMENT

It is a great pleasure to have the opportunity to extend our heartfelt gratitude to everyone who helped us throughout the course of this project. I am profoundly grateful to our supervisor **Dr Sagar Pande**, Professor, Lovely Professional University, Department of Computer Science, for his expert guidance, continuous encouragement, constant motivation, support and ever willingness to spare time from his otherwise busy schedule for the project's progress reviews. His continuous inspiration has made us complete this project and achieve its target.

TABLE OF CONTENTS

1. Introduction

Face recognition is a task of pattern recognition that is specifically performed on faces. In other words, it can be described as classifying a face either known or unknown by comparing a face with stored known individuals in the database. It is also desirable to have a system that has the ability of learning to recognize unknown faces. People have a good ability to recognize and distinguish between faces but recognizing human face automatically by computer is very difficult. The main goal of face recognition technology is to match a given face image against the stored database of images. Face recognition technique uses several other disciplines such as image processing, computer vision, pattern recognition, neural networks and psychology. With the current perceived world security situations, governments as well as businesses require reliable methods to accurately identify individuals, without overly infringing on rights to privacy or requiring significant compliance on the part of the individual being recognized and Expression Recognition is the most effective form of non-verbal communication and it provides intimation about emotional state, mindset and intention. Facial expressions not only can change the flow of conversation but also provides the listeners a way to communicate a wealth of information to the speaker without even uttering a single word. When the facial expression does not match with the spoken words, then the information passes on by the face gets more power in interpreting the information [1].

1.1. Framework for Face Recognition

Face recognition is a technique that takes the image of a person (query image) and compares it with the previously recorded images in the database. This is done by comparing the invariant features obtained from the techniques that capture the representative variability of the faces or the structure, the shape and the face attributes like distance between the eye centres and nose, upper outlines of the eyes, width of eyebrows, etc. Face recognition has the benefit of being a passive, non-intrusive system to verify personal identity in a natural and friendly way. The main benefit of this technique over other biometric approaches is that the face

images can be taken from a distance even without the knowledge of the individual being observed as might be required in identifying the presence of the criminals in a bank or government offices, etc [1].

1.2. Applications of Face Recognition System

It has become one of the most active research areas especially in recent years as it has a variety of wide applications in the areas [2]:

- Public security
- Law enforcement and commerce
- Credit card verification
- Criminal identification
- Access control
- Human-computer intelligent interaction
- Digital libraries and information security

1.3. Challenges in the Field of Face Recognition

The challenges associated with face recognition can be attributed to the following factors [2]:

- Presence or absence of structural components: Facial features such as beards, moustaches, and glasses may or may not be present and there is a great deal of variability among these components including shape, colour and size.
- Pose: The images of a face vary due to the relative camera-face pose (frontal, tilted, profile, upside down).
- Facial expression and emotions: The appearance of faces is directly affected by a person's facial expression and emotions.
- Occlusion: Faces may be partially occluded by other objects. For an example, in an image with a group of people, some faces may partially occlude other faces (face identification).
- Image orientation: Face images directly vary for different rotations about the camera's optical axis.

- ❖ Imaging conditions: When the image is formed, factors such as lightning and camera characteristics affect the appearance of a face.
- ❖ Age: Images taken after one- or two-year's gap may not match with the images in database.

1.4. Why Use the Face for Recognition?

Biometric based techniques have emerged as the most promising option for recognizing individuals in recent years, instead of authenticating people and granting them access to physical and virtual domains based on passwords, PINs, smart cards, plastic cards, tokens, keys and so forth, these methods examine an individual's physiological or behavioural characteristics in order to determine or ascertain his identity. Passwords and PINs are hard to remember and can be stolen or guessed; cards, tokens, keys and the like can be misplaced, forgotten, purloined or duplicated; magnetic cards can become corrupted and unreadable. However, an individual's biological traits cannot be misplaced, forgotten, stolen or forged. Biometric based technologies include identification based on physiological characteristics (such as face, fingerprints, finger geometry, hand geometry, hand veins, palm, iris, retina, ear and voice) and behavioural traits (such as signature and keystroke dynamics) [3].

2. Problem Statement

In Image-based face recognition. We were given a picture, we'd like to know if there is any person inside, where his/her face locates at, and who he/she is. Towards this goal, we generally separate the face recognition procedure into three steps [4]:

1) Face Detection.
2) Feature Extraction.
3) Face Recognition.

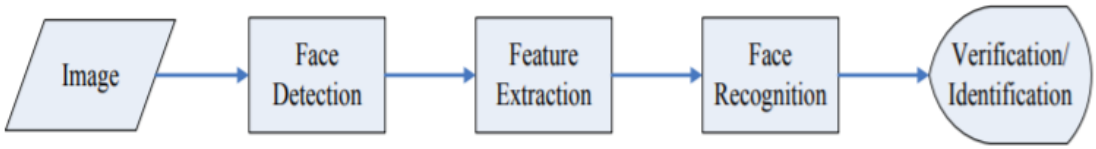


Figure 1: Face Recognition Procedure

1. **Face Detection**: The main function of this step is to determine (1) whether human faces appear in a given image, and (2) where these faces are located at. The expected outputs of this step are patches containing each face in the input image. In order to make further face recognition system more robust and easier to design, face alignment is performed to justify the scales and orientations of these patches. Besides serving as the pre-processing for face recognition, face detection could be used for region-of-interest detection, retargeting, video and image classification, etc [4].

2. **Feature Extraction**: After the face detection step, human-face patches are extracted from images. Directly using these patches for face recognition have some disadvantages, first, each patch usually contains over 1000 pixels, which are too large to build a robust recognition system. Second, face patches may be taken from different camera alignments, with different face expressions, illuminations, and may suffer from occlusion and clutter. To overcome these drawbacks, feature extractions are performed to do information packing, dimension reduction, salience extraction, and noise cleaning. After this step, a face patch is usually

transformed into a vector with fixed dimension or a set of fiducial points and their corresponding locations. In some literatures, feature extraction is either included in face detection or face recognition [4].

3. **Face Recognition**: After formulizing the representation of each face, the last step is to recognize the identities of these faces. In order to achieve automatic recognition, a face database is required to build. For each person, several images are taken and their features are extracted and stored in the database. Then when an input face image comes in, we perform face detection and feature extraction, and compare its feature to each face class stored in the database. There have been many researches and algorithms proposed to deal with this classification problem, and we'll discuss them in later sections. There are two general applications of face recognition, one is called identification and another one is called verification. Face identification means given a face image, we want the system to tell who he / she is or the most probable identification; while in face verification, given a face image and a guess of the identification, we want the system to tell true or false about the guess [4].

4. **Expression Recognition**: To Detect Human emotions and intentions which are expressed through facial expressions and deriving an efficient and effective feature is the fundamental component of facial expression system. Face recognition is important for the interpretation of facial expressions in applications such as intelligent, man-machine interface and communication, intelligent visual surveillance, teleconference and real-time animation from live motion images. The facial expressions are useful for efficient interaction Most research and system in facial expression recognition are limited to four basic expressions (Sad, Anger, surprise, Neutral). It is found that it is insufficient to describe all facial expressions and these expressions are categorized based on facial actions. Detecting face and recognizing the facial expression is a very complicated task when it is a vital to pay attention to primary components like: face configuration, orientation, location where the face is set [3].

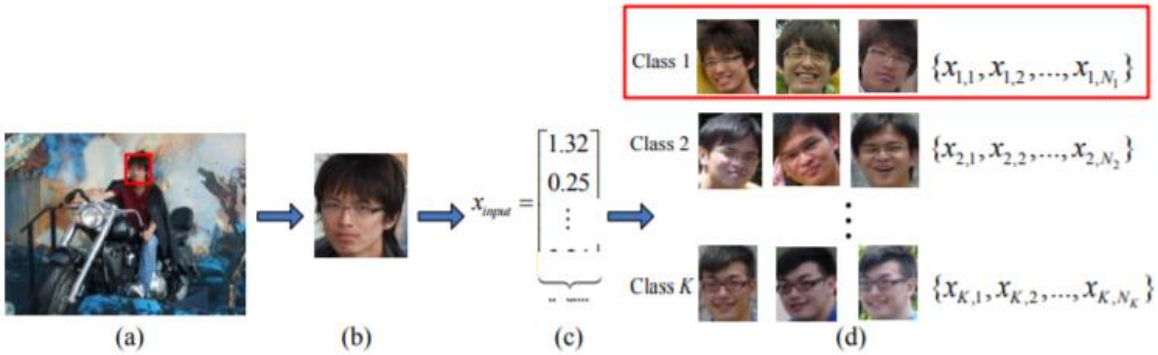


Figure 2: An Example of how the three steps work on an input image.

a) The input image and the result of face detection (the red rectangle face).
b) The extracted face patches.
c) The feature vector after feature extraction.
d) Comparing the input vector with the stored vectors in the database by classification techniques and determine the most probable class (the red rectangle face).

Here I express each face patch as a d-dimensional vector, the vector $x_{m,n}$ as the n_{th} vector in m_{th} class, and N_k as the number of faces stored in the k_{th} class [4].

2.1. Objectives

1. To develop a facial expression recognition system.
2. To experiment machine learning algorithm in computer vision fields.
3. To detect emotion thus facilitating Intelligent Human-Computer Interaction.

2.2. Scope and Applications

The scope of this system is to tackle with the problems that can arise in day-to-day life [3].

Some of the Applications are:

1. The system can be used to detect and track a user's state of mind.
2. The system can be used in mini-marts, shopping centre to view the feedback of the customers to enhance the business.

3. The system can be installed at busy places like airport, railway station or bus station for detecting human faces and facial expressions of each person. If there are any faces that appeared suspicious like angry or fearful, the system might set an internal alarm.
4. The system can also be used for educational purpose such as one can get feedback on how the student is reacting during the class.
5. This system can be used for lie detection amongst criminal suspects during interrogation.
6. This system can help people in emotion related -research to improve the processing of emotion data.
7. Clever marketing is feasible using emotional knowledge of a person which can be identified by this system.

3. Existing System

3.1. Introduction

To build a face recognition system, first, I had performed face detection, extract face embeddings from each face using deep learning, train a face recognition model on the embeddings, and then finally recognize faces in both images and video streams with OpenCV.

3.2. Existing Software

To perform face recognition using the **OpenCV** library. I had used OpenCV to facilitate face recognition, But, OpenCV itself is not responsible for identifying faces. I have applied deep learning and OpenCV together to [5]:

1. Detect faces.
2. Compute 128-d face embeddings to quantify a face.
3. Train a Support Vector Machine (SVM) on top of the embeddings.
4. Recognize faces in images and video streams

All of these tasks are accomplished with OpenCV, enabling us to obtain a “pure” OpenCV face recognition pipeline.

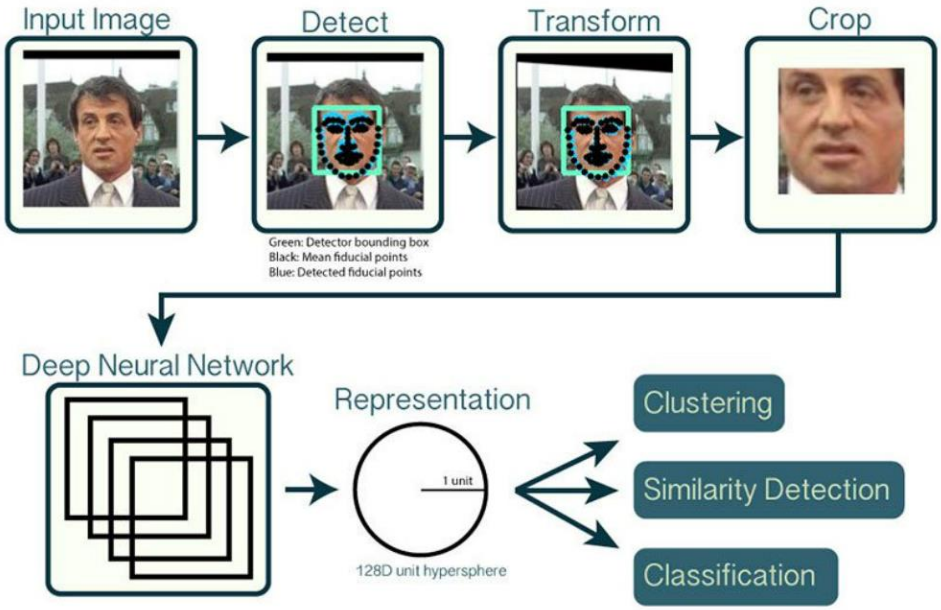


Figure 3: OpenCV face recognition pipeline

To build a OpenCV face recognition pipeline, I had applied deep learning in two key steps [5]:

1. To apply face detection, which detects the presence and location of a face in an image, but does not identify it.

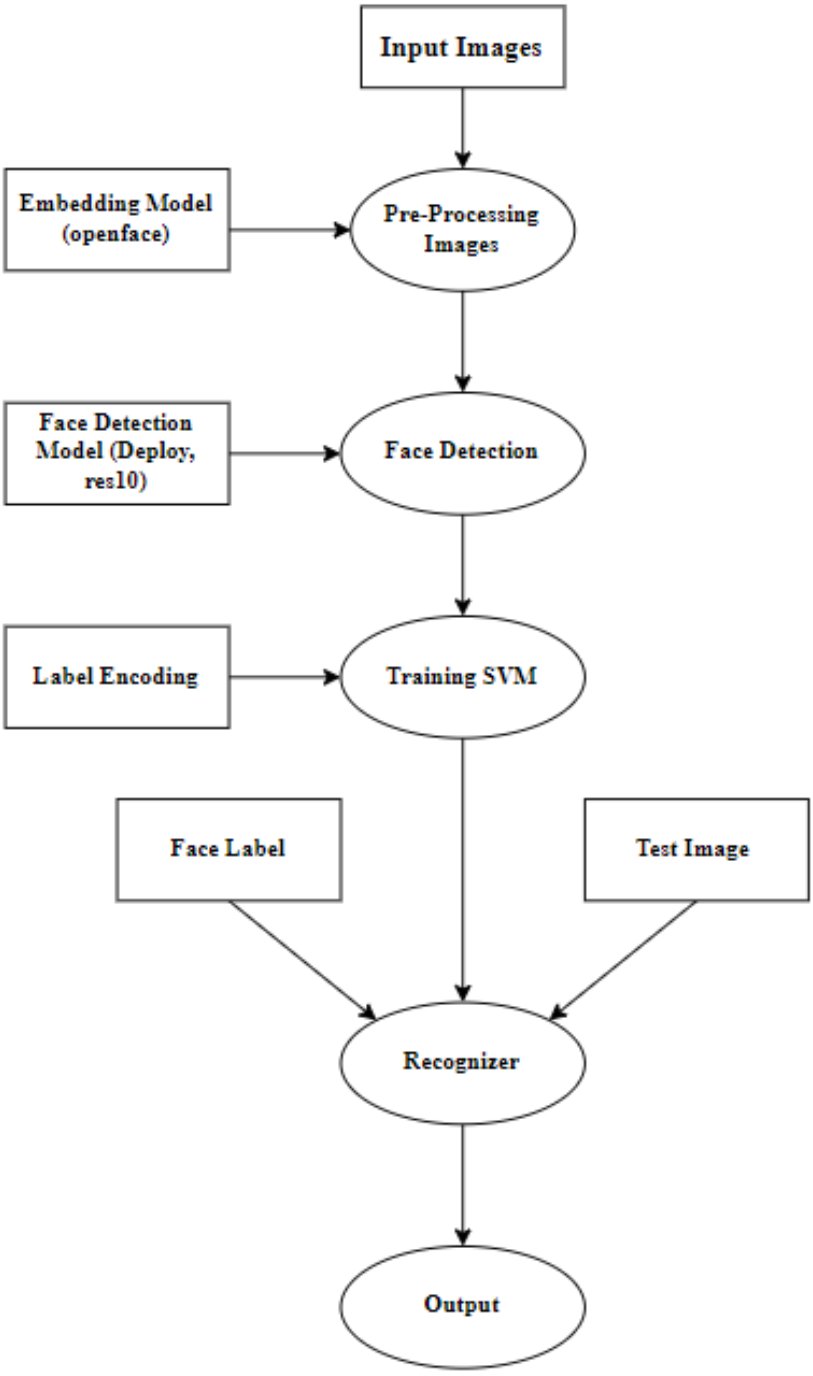
2. To extract the 128-d feature vectors (called “embeddings”) that quantify each face in an image.

This model is responsible for actually quantifying each face in an image is from:

OpenFace: The project which is a Python and Torch implementation of face recognition with deep learning.

FaceNet: A Unified Embedding for Face Recognition and Clustering.

3.3. Data Flow Diagram (DFD) for System



3.4. Novelty in the System

In this deep learning model, I had used (very likely) never seen the faces I was about to pass through it, the model will still be able to compute embeddings from each face ideally, these face embeddings will be sufficiently different such that I can train a "standard" machine learning classifier (**SVM**, SGD classifier, Random Forest, etc.) on top of the face embeddings, and therefore obtain OpenCV face recognition pipeline.

4. Product Analysis

4.1. Problem Planning

In planning phase study of reliable and effective algorithms is done. On the other hand, data were collected and were pre-processed for more fine and accurate results. Since huge amount of data were needed for better accuracy, I have collected the data surfing the internet. Since, I am new to this project I have decided to use local binary pattern algorithm for feature extraction and support vector machine for training the dataset. I had decided to implement these algorithms by using OpenCV framework.

4.2. Data Collection & Preparation

The dataset I had used contains Five people:

1) Myself
2) Virat Kohli
3) Narendra Modi
4) Prabhas
5) Unknown

Project has four directories in the root folder:

1) **dataset**: Contains face images organized into subfolders by name.
2) **images**: Contains three test images that I use to verify the operation of the model.
3) **face_detection_model**: Contains a pre-trained Caffe deep learning model provided by OpenCV to detect faces. This model detects and localizes faces in an image.
4) **output**: Contains my output pickle files. If you're working with your own dataset, you can store your output files here as well.
   The output files include:
   1) **embeddings.pickle**: A serialized facial embeddings file. Embeddings have been computed for every face in the dataset and are stored in this file.

2) **le.pickle** : To label encoder. Contains the name labels for the people that the model can recognize.
3) **recognizer.pickle** : Linear Support Vector Machine (SVM) model. This is a machine learning model rather than a deep learning model and it is responsible for actually recognizing faces.

- **The five files in the root directory are** [5]:

extract_embeddings.py: It is responsible for using a deep learning feature extractor to generate a 128-D vector describing a face. All faces in dataset will be passed through the neural network to generate embeddings.

openface_nn4.small2.v1.t7: A Torch deep learning model which produces the 128-D facial embeddings.

train_model.py: The Linear SVM model was trained by this script. The Model can detected faces, extracted embeddings, and fit The SVM model to the embeddings data.

recognize.py: Model can recognize faces in images. Can detected faces, extract embeddings, and query SVM model to determine who is in the image. To draw boxes around faces and annotate each box with a name.

recognize_video.py: This describe how to recognize who is in the frames of a video stream just as Model did on static images.

4.3. Feasibility Study

Feasibility study is necessary to determine if creating a new or improved system is friendly with the cost, benefits, operation, technology and time. Following feasibility study is given as below [3]:

4.3.1 Technical Feasibility

Technical feasibility study includes the hardware and software devices. The required technologies (python, Notepad++, anaconda) existed.

4.3.2. Operational Feasibility

Operational Feasibility is a measure of how well a proposed system solves the problem and takes advantage of the opportunities identified during scope definition. The following points were considered for the project's technical feasibility [3]:

- The system will detect and capture the image of face.
- The captured image is then (identified which category).

4.3.3. Economic Feasibility

The purpose of economic feasibility is to determine the positive economic benefits that include quantification and identification. The system is economically feasible due to availability of all requirements such as collection of data from [3]:

- Open Face
- Face Net

4.3.4. Schedule Feasibility

Schedule feasibility is a measure of how reasonable the project timetable is. The system is found schedule feasible because the system is designed in such a way that it will finish prescribed time [3].

5. Software Requirement Analysis

Requirement analysis is mainly categorized into two types:

5.1. Functional requirements

The functional requirements for a system describe what the system should do. Those requirements depend on the type of software being developed, the expected users of the software. These are statement of services the system should provide, how the system should react to particular inputs and how the system should behave in particular situation [3].

5.2. Non-Functional requirements

Non-functional requirements are requirements that are not directly concerned with the specified function delivered by the system. They may relate to emergent system properties such as reliability, response time and store occupancy. Some of the non-functional requirements related with this system are hereby below [3]:

a) Reliability

Reliability based on this system defines the evaluation result of the system, correct identification of the facial expressions and maximum evaluation rate of the facial expression recognition of any input images.

b) Ease of Use

The system is simple, user friendly so any can use this system without any difficulties.

6. Design

6.1. System Design

System design shows the overall design of system. In this section we discuss in detail the design aspects of the system [3]:

6.1.1. System Diagram:

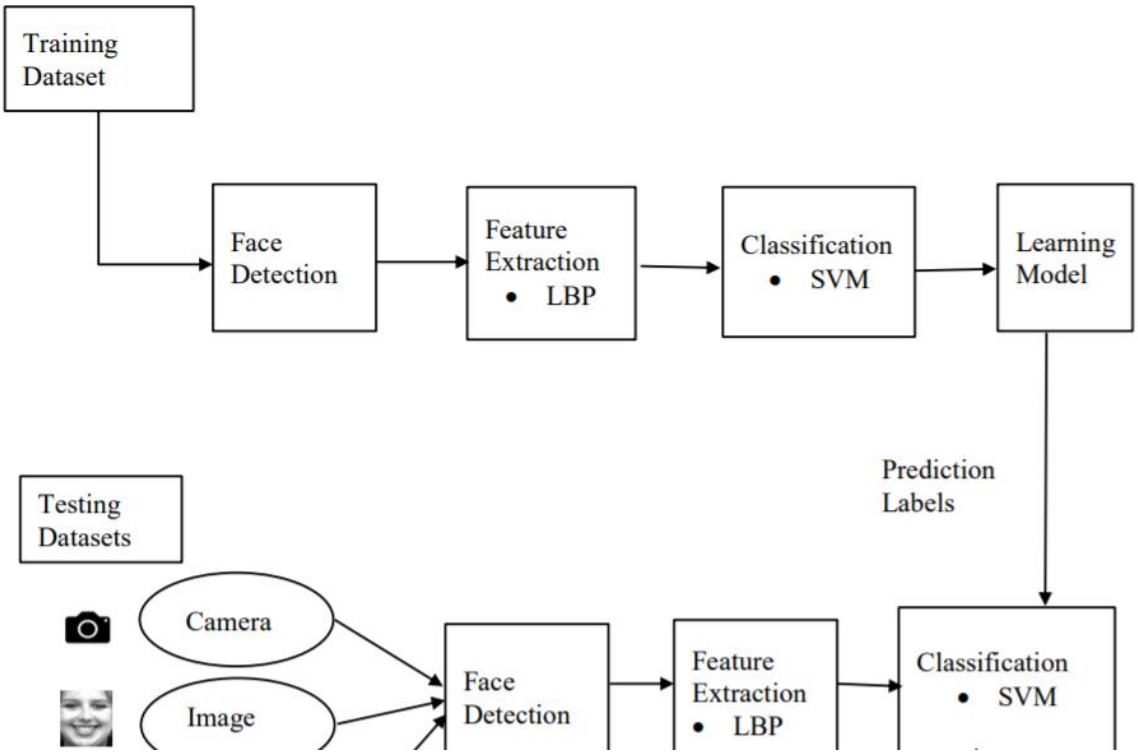


Figure 4: System Diagram

6.2. Flowcharts

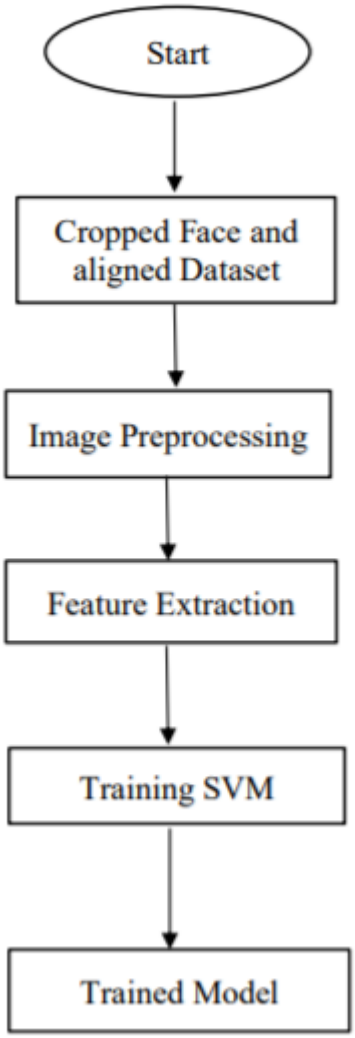


Figure 5: System Flowchart

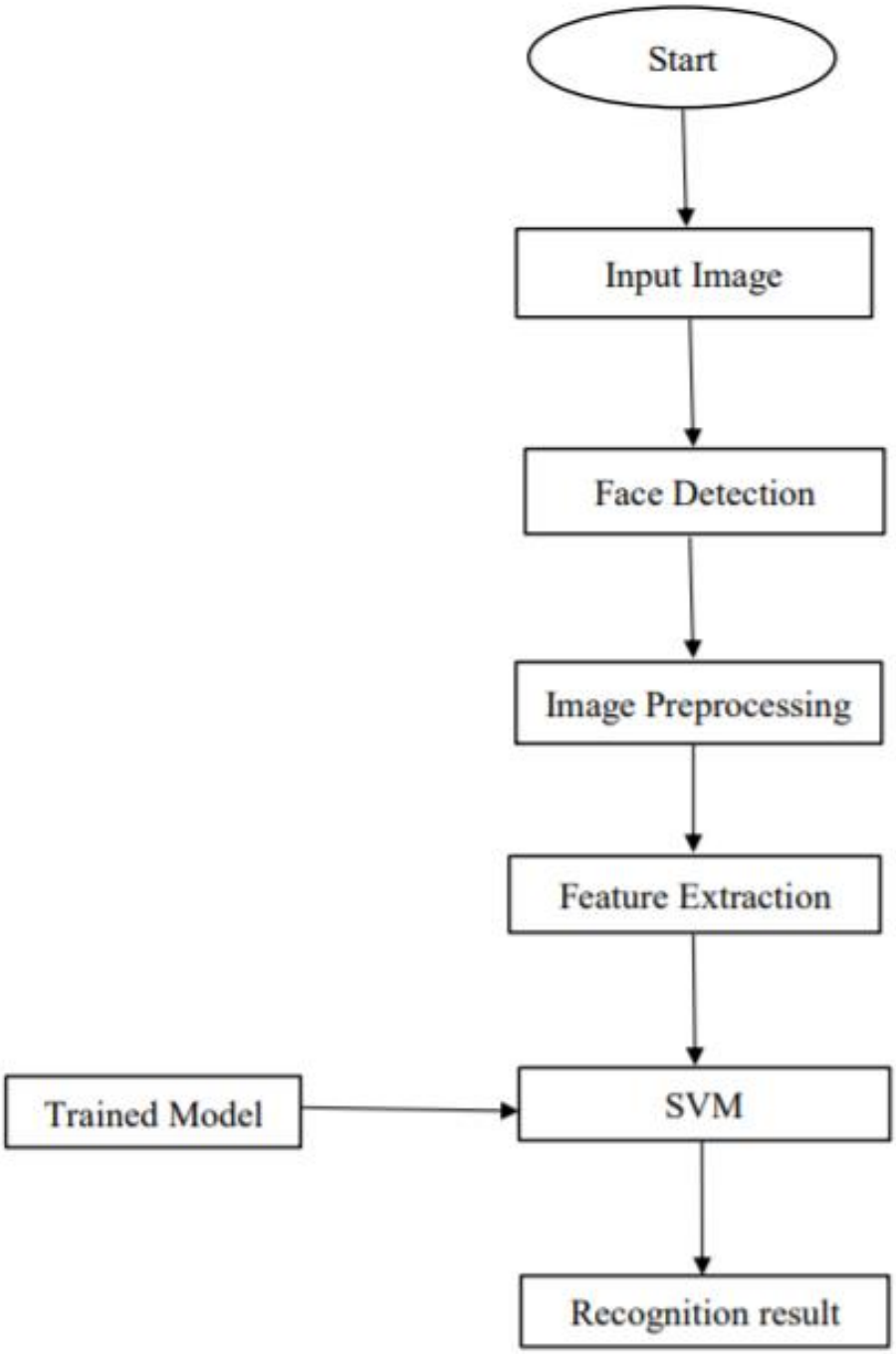


Figure 6: Flowchart of Testing/Prediction

7. Testing

7.1. Programming Tools

Python

- Python is a general-purpose interpreted, interactive, object-oriented, and high-level programming language. Python source code is also available under the GNU General Public License (GPL) [6].

- Python is a high-level, interpreted, interactive and object-oriented scripting language. Python is designed to be highly readable. It uses English keywords frequently where as other languages use punctuation, and it has fewer syntactical constructions than other languages [6].

- Face Recognition with Python is the latest trend in Machine Learning techniques. OpenCV, the most popular library for computer vision, provides bindings for Python. OpenCV uses machine learning algorithms to search for faces within a picture [6].

Notepad++

- Notepad++ is a text and source code editor for use with Microsoft Windows. It supports tabbed editing, which allows working with multiple open files in a single window. The product's name comes from the C increment operator [7].

- Notepad++ is distributed as free software. At first the project was hosted on SourceForge.net, from where it has been downloaded over 28 million times, and twice won the Source Forge Community Choice Award for Best Developer Tool [7].

Framework

OpenCV

- OpenCV is the most popular library for computer vision. Originally written in C/C++, it now provides bindings for Python [8].
- OpenCV uses machine learning algorithms to search for faces within a picture. Because faces are so complicated, there isn't one simple test that will tell you if it found a face or not. Instead, there are thousands of small patterns and features that must be matched [8].
- The algorithms break the task of identifying the face into thousands of smaller, bite-sized tasks, each of which is easy to solve. These tasks are also called classifiers [8].
- The library has more than 2500 optimized algorithms, which includes a comprehensive set of both classic and state-of-the-art computer vision and machine learning algorithms.
- These algorithms can be used to detect and recognize faces, identify objects, classify human actions in videos, track camera movements, track moving objects, extract 3D models of objects, produce 3D point clouds from stereo cameras, stitch images together to produce a high resolution image of an entire scene, find similar images from an image database, remove red eyes from images taken using flash, follow eye movements, recognize scenery and establish markers to overlay it with augmented reality, etc. It has C++, C, Python, Java and MATLAB interfaces and supports Windows, Linux, Android and Mac OS [8].

7.2. System testing

System testing was done by giving different training and testing datasets. This test was done to evaluate whether the system was predicting accurate result or not. During the phase of the development of the system our system was tested time and again. The series of testing conducted are as follows [3]:

7.2.1. Unit Testing

In unit testing, I had designed the whole system in modularized pattern and each module was tested. Till now We can get the accurate output from the individual module. Now I worked on the same module [3].

7.2.2. Integration Testing

After constructing individual modules all the modules were merged and a complete system was made. Then the system was tested whether the prediction given by training dataset to testing set was correct or not. I tried to meet the accuracy as higher as much as I can get. After spending a couple of days in integration testing the average accuracy of our system was 91% [3].

7.3. Testing the project

7.3.1. Alpha testing

Alpha testing is the first stage of software engineering which is considered as a simulated or actual operational testing done by the individual member of this project. Alpha testing is conducted by the me, in context of our project [3].

7.3.2. Beta Testing

Beta testing comes continuously after alpha testing which is considered as a form of external user acceptance testing. The beta version of the program is developed to and provided to limited audience. This is the final test process in the case of this project. In this system the beta-testing is done by our colleagues and the project supervisor [3].

8. Implementation

8.1. Extract embeddings from face dataset (step 1)

- Create the **extract_embeddings.py** file then:

```
3  # import the necessary packages
4  from imutils import paths
5  import numpy as np
6  import argparse
7  import imutils
8  import pickle
9  import cv2
10 import os
11
12 # construct the argument parser and parse the arguments
13 ap = argparse.ArgumentParser()
14 ap.add_argument("-i", "--dataset", required=True,
15     help="path to input directory of faces + images")
16 ap.add_argument("-e", "--embeddings", required=True,
17     help="path to output serialized db of facial embeddings")
18 ap.add_argument("-d", "--detector", required=True,
19     help="path to OpenCV's deep learning face detector")
20 ap.add_argument("-m", "--embedding-model", required=True,
21     help="path to OpenCV's deep learning face embedding model")
22 ap.add_argument("-c", "--confidence", type=float, default=0.5,
23     help="minimum probability to filter weak detections")
24 args = vars(ap.parse_args())
```

- imported the required packages on Lines 3-10. We need to have OpenCV and imutils installed.
- Imutils, OpenCv packages can be installed with pip:

$ pip install imutils

$ pip install opencv-python

➢ **imutils**

A series of convenience functions to make basic image processing functions such as translation, rotation, resizing, skeletonization, and displaying Matplotlib images easier with OpenCV [5].

- we process command line arguments:

--dataset:

The path to input dataset of face images.

--embeddings:

The path to output embeddings file. The Script will compute face embeddings which we'll serialize to disk.

--detector:

Path to OpenCV's Caffe-based deep learning face detector used to actually localize the faces in the images.

--embedding-model:

Path to the OpenCV deep learning Torch embedding model. This model allows to extract a 128-D facial embedding vector.

➢ We've imported packages and parsed command line arguments, now we load the face detector and embedder from disk:

```
26  # load our serialized face detector from disk
27  print("loading face detector...")
28  protoPath = os.path.sep.join([args["detector"], "deploy.prototxt"])
29  modelPath = os.path.sep.join([args["detector"],
30      "res10_300x300_ssd_iter_140000.caffemodel"])
31  detector = cv2.dnn.readNetFromCaffe(protoPath, modelPath)
32
33  # load our serialized face embedding model from disk
34  print("loading face recognizer...")
35  embedder = cv2.dnn.readNetFromTorch(args["embedding_model"])
```

- we load the face detector and embedder:

detector: Loaded via Lines 28-31. We're using a Caffe based DL face detector to localize faces in an image.

embedder: Loaded on Line 35. This model is Torch-based and is responsible for extracting facial embeddings via deep learning feature extraction.

We can Notice that cv2.dnn functions to load the two separate models.

- To grab the image paths and perform initializations:

```
37  # grab the paths to the input images in our dataset
38  print("quantifying faces...")
39  imagePaths = list(paths.list_images(args["dataset"]))
40
41  # initialize our lists of extracted facial embeddings and
42  # corresponding people names
43  knownEmbeddings = []
44  knownNames = []
45
46  # initialize the total number of faces processed
47  total = 0
```

- The imagePaths list, built on Line 39, contains the path to each image in the dataset. I've made this easy via my imutils function, paths.list_images .

- embeddings and corresponding names will be held in two lists: knownEmbeddings and knownNames (Lines 43 and 44).

- I had also kept track of how many faces we've processed via a variable called total (Line 47).

- To loop over the image path I had used this loop responsible for extracting embeddings from faces found in each image:

```
49  # loop over the image paths
50  for (i, imagePath) in enumerate(imagePaths):
51      # extract the person name from the image path
52      print("processing image {}/{}".format(i + 1,
53          len(imagePaths)))
54      name = imagePath.split(os.path.sep)[-2]
55
56      # load the image, resize it to have a width of 600 pixels (while
57      # maintaining the aspect ratio), and then grab the image
58      # dimensions
59      image = cv2.imread(imagePath)
60      image = imutils.resize(image, width=600)
61      (h, w) = image.shape[:2]
```

- We begin looping over imagePaths on Line 50.
- we extract the name of the person from the path (Line 54).

- Finally, I wrap up the above code block by loading the image and resize it to a known width (Lines 59 and 60).

➢ To detect and localize faces:

```
63      # construct a blob from the image
64      imageBlob = cv2.dnn.blobFromImage(
65          cv2.resize(image, (300, 300)), 1.0, (300, 300),
66          (104.0, 177.0, 123.0), swapRB=False, crop=False)
67
68      # apply OpenCV's deep learning-based face detector to localize
69      # faces in the input image
70      detector.setInput(imageBlob)
71      detections = detector.forward()
```

- On Lines 64-66, we construct a blob.
- From there we detect faces in the image by passing the imageBlob through the detector network (Lines 70 and 71).

➢ To process the detections:

```
73      # ensure at least one face was found
74      if len(detections) > 0:
75          # we're making the assumption that each image has only ONE
76          # face, so find the bounding box with the largest probability
77          i = np.argmax(detections[0, 0, :, 2])
78          confidence = detections[0, 0, i, 2]
79
80          # ensure that the detection with the largest probability also
81          # means our minimum probability test (thus helping filter out
82          # weak detections)
83          if confidence > args["confidence"]:
84              # compute the (x, y)-coordinates of the bounding box for
85              # the face
86              box = detections[0, 0, i, 3:7] * np.array([w, h, w, h])
87              (startX, startY, endX, endY) = box.astype("int")
88
89              # extract the face ROI and grab the ROI dimensions
90              face = image[startY:endY, startX:endX]
91              (fH, fW) = face.shape[:2]
92
93              # ensure the face width and height are sufficiently large
94              if fW < 20 or fH < 20:
95                  continue
```

- The detections list contains probabilities and coordinates to localize faces in an image.
- Assuming we have at least one detection, we had proceeded into the body of the if-statement (Line 74).

- We make the assumption that there is only one face in the image, so we extract the detection with the highest confidence and check to make sure that the confidence meets the minimum probability threshold used to filter out weak detections (Lines 77-83).
- Assuming we've met that threshold, we extract the face ROI and grab/check dimensions to make sure the face ROI is sufficiently large (Lines 86-95).

➢ To take advantage of embedder CNN and extract the face embeddings:

```
97              # construct a blob for the face ROI, then pass the blob
98              # through our face embedding model to obtain the 128-d
99              # quantification of the face
100             faceBlob = cv2.dnn.blobFromImage(face, 1.0 / 255,
101                 (96, 96), (0, 0, 0), swapRB=True, crop=False)
102             embedder.setInput(faceBlob)
103             vec = embedder.forward()
104
105             # add the name of the person + corresponding face
106             # embedding to their respective lists
107             knownNames.append(name)
108             knownEmbeddings.append(vec.flatten())
109             total += 1
110
```

- To construct another blob, this time from the face ROI (not the whole image as I did before) on Lines 100 and 101.

- Subsequently, we pass the faceBlob through the embedder CNN (Lines 102 and 103). This generates a 128-D vector (vec) which describes the face. I had leveraged this data to recognize new faces via machine learning.

- And then we simply add the name and embedding vec to knownNames and knownEmbeddings, respectively (Lines 107 and 109).

- We also can't forget about the variable I set to track the total number of faces either we go ahead and increment the value on Line 109.

- I continued this process of looping over images, detecting faces, and extracting face embeddings for each and every image in dataset.

➢ When the loop finishes, Now we are left with to dump the data to disk:

```
111    # dump the facial embeddings + names to disk
112    print("[INFO] serializing {} encodings...".format(total))
113    data = {"embeddings": knownEmbeddings, "names": knownNames}
114    f = open(args["embeddings"], "wb")
115    f.write(pickle.dumps(data))
116    f.close()
```

- We add the name and embed data to a dictionary and then serialize the data in a pickle file on Lines 112-116.
- Now, we're ready to extract embeddings by running the script.

➢ By opening up a terminal and execute the following command to compute the face embeddings with OpenCV:

$ python extract_embeddings.py --dataset dataset --embeddings output/embed.pickle --detector face_detection_model --embedding-model openface_nn4.small2.v1.t7

```
processing image 82/117
processing image 83/117
processing image 84/117
processing image 85/117
processing image 86/117
processing image 87/117
processing image 88/117
processing image 89/117
processing image 90/117
processing image 91/117
processing image 92/117
processing image 93/117
processing image 94/117
processing image 95/117
processing image 96/117
processing image 97/117
processing image 98/117
processing image 99/117
processing image 100/117
processing image 101/117
processing image 102/117
processing image 103/117
processing image 104/117
processing image 105/117
processing image 106/117
processing image 107/117
processing image 108/117
processing image 109/117
processing image 110/117
processing image 111/117
processing image 112/117
processing image 113/117
processing image 114/117
processing image 115/117
processing image 116/117
processing image 117/117
[INFO] serializing 111 encodings...
```

- We can see that we have extracted 111 face embeddings.

8.2. Train face recognition model (Step 2)

- we have extracted 128-d embeddings for each face. Now we need to train a "standard" machine learning model (such as an SVM, k-NN classifier, Random Forest, etc.) on top of the embeddings.

- Now we have to built a powerful classifier on top of the embeddings. By training the embedding with SVM.

  - Create the train_model.py file then:

```
3  # import the necessary packages
4  from sklearn.preprocessing import LabelEncoder
5  from sklearn.svm import SVC
6  import argparse
7  import pickle
8
9  # construct the argument parser and parse the arguments
10 ap = argparse.ArgumentParser()
11 ap.add_argument("-e", "--embeddings", required=True,
12     help="path to serialized db of facial embeddings")
13 ap.add_argument("-r", "--recognizer", required=True,
14     help="path to output model trained to recognize faces")
15 ap.add_argument("-l", "--le", required=True,
16     help="path to output label encoder")
17 args = vars(ap.parse_args())
```

  - We need scikit-learn, a machine learning library, installed in environment prior to running this script. We can install it via pip:

$ pip install scikit-learn

- imported packages and modules on Lines 4-7. I had used scikit-learn's implementation of Support Vector Machines (SVM), a common machine learning model.

- I had parse the command line arguments:

--embeddings: The path to the serialized embeddings (we exported it by running the previous extract_embeddings.py script).

--recognizer: This will be output model that recognizes faces. It is based on SVM. We'll be saving it so we can use it in the next two recognition scripts.

--le: label encoder output file path. We'll serialize label encoder to disk so that we can use it and the recognizer model in image/video face recognition scripts.

- Each of these arguments is required.

- To load the facial embeddings and encode labels:

```
19  # load the face embeddings
20  print("[INFO] loading face embeddings...")
21  data = pickle.loads(open(args["embeddings"], "rb").read())
22
23  # encode the labels
24  print("[INFO] encoding labels...")
25  le = LabelEncoder()
26  labels = le.fit_transform(data["names"])
```

- we load embeddings from **Step 1** on Line 21. We won't be generating any embeddings in this model training script — I had used the embeddings previously to generate and serialize.

- Then I initialized scikit-learn **Label Encoder** and encode name labels (Lines 25 and 26).

- To train SVM model for recognizing faces we used:

```
28  # train the model used to accept the 128-d embeddings of the face and
29  # then produce the actual face recognition
30  print("[INFO] training model...")
31  recognizer = SVC(C=1.0, kernel="linear", probability=True)
32  recognizer.fit(data["embeddings"], labels)
```

- On Line 31 we initialize SVM model, and on Line 32 we fit the model (also known as "training the model").
- we used a Linear Support Vector Machine (SVM) but we can also try experimenting with other machine learning models.
- After training the model we output the model and label encoder to disk as pickle files.

```
34  # write the actual face recognition model to disk
35  f = open(args["recognizer"], "wb")
36  f.write(pickle.dumps(recognizer))
37  f.close()
38
39  # write the label encoder to disk
40  f = open(args["le"], "wb")
41  f.write(pickle.dumps(le))
42  f.close()
```

- We had written two pickle files to disk in this block for the face recognizer model and the label encoder.

- Now, we have finished coding train_model.py. To apply it to extracted face embeddings we use command:

$ python train_model.py --embeddings output/embed.pickle --recognizer output/recognizer.pickle --le output/label.pickle

```
(base) C:\Users\racha\Workspace\INT 246\Project>python train_model.py --embeddings output/embed.pickle
--recognizer output/recognizer.pickle --le output/label.pickle
[INFO] loading face embeddings...
[INFO] encoding labels...
[INFO] training model...
```

- We can see that SVM had been trained on the embeddings and both the (1) SVM itself and (2) the label encoding had been written to disk, enabling us to apply them to input images and video [5].

8.3. Recognize faces with OpenCV (Step 3)

- Create the **recognize.py** file then

```
5   # import the necessary packages
6   import numpy as np
7   import argparse
8   import imutils
9   import pickle
10  import cv2
11  import os
12
13  # construct the argument parser and parse the arguments
14  ap = argparse.ArgumentParser()
15  ap.add_argument("-i", "--image", required=True,
16      help="path to input image")
17  ap.add_argument("-d", "--detector", required=True,
18      help="path to OpenCV's deep learning face detector")
19  ap.add_argument("-m", "--embedding-model", required=True,
20      help="path to OpenCV's deep learning face embedding model")
21  ap.add_argument("-r", "--recognizer", required=True,
22      help="path to model trained to recognize faces")
23  ap.add_argument("-l", "--le", required=True,
24      help="path to label encoder")
25  ap.add_argument("-c", "--confidence", type=float, default=0.5,
26      help="minimum probability to filter weak detections")
27  args = vars(ap.parse_args())
```

- We import required packages on Lines 6-11.
- Now, the command line arguments are parsed on Lines 14-27:

  --image: The path to the input image. We had attempted to recognize the faces in this image.

  --detector: The path to OpenCV's deep learning face detector. We used for this model to detect where in the image the face ROIs are.

  --embedding-model: The path to OpenCV's deep learning face embedding model. We used for this model to extract the 128-D face embedding from the face ROI that we have feed the data into the recognizer.

  --recognizer: The path to recognizer model. We trained SVM recognizer in Step 2. This is what actually determine who a face is.

  --le: The path to label encoder. This contains face labels such as 'Arjun', 'Virat Kohli', 'Narendra Modi', 'Prabhas' and 'Unknown'.

--confidence: The optional threshold to filter weak face detections.

- It is important for us to know the difference between the two deep learning models and the SVM model.
- To handle imports and command line arguments, we load the three models from disk into memory with:

```
29  # load our serialized face detector from disk
30  print("[INFO] loading face detector...")
31  protoPath = os.path.sep.join([args["detector"], "deploy.prototxt"])
32  modelPath = os.path.sep.join([args["detector"],
33      "res10_300x300_ssd_iter_140000.caffemodel"])
34  detector = cv2.dnn.readNetFromCaffe(protoPath, modelPath)
35
36  # load our serialized face embedding model from disk
37  print("[INFO] loading face recognizer...")
38  embedder = cv2.dnn.readNetFromTorch(args["embedding_model"])
39
40  # load the actual face recognition model along with the label encoder
41  recognizer = pickle.loads(open(args["recognizer"], "rb").read())
42  le = pickle.loads(open(args["le"], "rb").read())
```

- We load three models in this block. Those are:

1) **detector**: A pre-trained Caffe DL model to detect where in the image the faces are (Lines 31-34).
2) **embedder**: A pre-trained Torch DL model to calculate 128-D face embeddings (Line 38).
3) **recognizer**: Linear SVM face recognition model (Line 41). We trained this model in Step 2.

- Both 1 & 2 are pre-trained meaning that they are provided by OpenCV, they are buried in the OpenCV project on GitHub.
- We also loaded label encoder which holds the names of the people model can recognize (Line 42).

- Now, to load test image and detect faces we used:

```
44  # load the image, resize it to have a width of 600 pixels (while
45  # maintaining the aspect ratio), and then grab the image dimensions
46  image = cv2.imread(args["image"])
47  image = imutils.resize(image, width=600)
48  (h, w) = image.shape[:2]
49
50  # construct a blob from the image
51  imageBlob = cv2.dnn.blobFromImage(
52      cv2.resize(image, (300, 300)), 1.0, (300, 300),
53      (104.0, 177.0, 123.0), swapRB=False, crop=False)
54
55  # apply OpenCV's deep learning-based face detector to localize
56  # faces in the input image
57  detector.setInput(imageBlob)
58  detections = detector.forward()
```

- To Load the image into memory and construct a blob (Lines 46-53).
- To Localize faces in the image via the detector (Lines 57 and 58).

- For new detections, to recognize faces in the image. But for we have to filter weak detections and extract the face ROI:

```
60  # loop over the detections
61  for i in range(0, detections.shape[2]):
62      # extract the confidence (i.e., probability) associated with the
63      # prediction
64      confidence = detections[0, 0, i, 2]
65
66      # filter out weak detections
67      if confidence > args["confidence"]:
68          # compute the (x, y)-coordinates of the bounding box for the
69          # face
70          box = detections[0, 0, i, 3:7] * np.array([w, h, w, h])
71          (startX, startY, endX, endY) = box.astype("int")
72
73          # extract the face ROI
74          face = image[startY:endY, startX:endX]
75          (fH, fW) = face.shape[:2]
76
77          # ensure the face width and height are sufficiently large
78          if fW < 20 or fH < 20:
79              continue
```

- We recognize this block from Step 1.
- We loop over the detections on Line 61 and extract the confidence of each on Line 64.

- Then we compared the Confidence to the minimum probability detection threshold contained in command line args dictionary, ensuring that the computed probability is larger than the minimum probability (Line 67).

- We extract the face Region of Image (ROI) (Lines 70-74) and ensure its spatial dimensions are sufficiently large (Lines 78 and 79).

➢ To Recognize the name of the face ROI we used:

```
81          # construct a blob for the face ROI, then pass the blob
82          # through our face embedding model to obtain the 128-d
83          # quantification of the face
84          faceBlob = cv2.dnn.blobFromImage(face, 1.0 / 255, (96, 96),
85              (0, 0, 0), swapRB=True, crop=False)
86          embedder.setInput(faceBlob)
87          vec = embedder.forward()
88
89          # perform classification to recognize the face
90          preds = recognizer.predict_proba(vec)[0]
91          j = np.argmax(preds)
92          proba = preds[j]
93          name = le.classes_[j]
```

- we construct a faceBlob (from the face ROI) and pass it through the embedder to generate a 128-D vector which describes the face (Lines 84-87).

- Then, we pass the vec through SVM recognizer model (Line 90), the result of which is predictions for who is in the face ROI.

- We take the highest probability index (Line 91) and query label encoder to find the name (Line 93). In between, I extract the probability on Line 92.

- We can further filter out weak face recognitions by applying an additional threshold test on the probability. For example, inserting if proba < T (where T is a variable we define) can provide an additional layer of filtering to ensure there are less false-positive face recognitions.

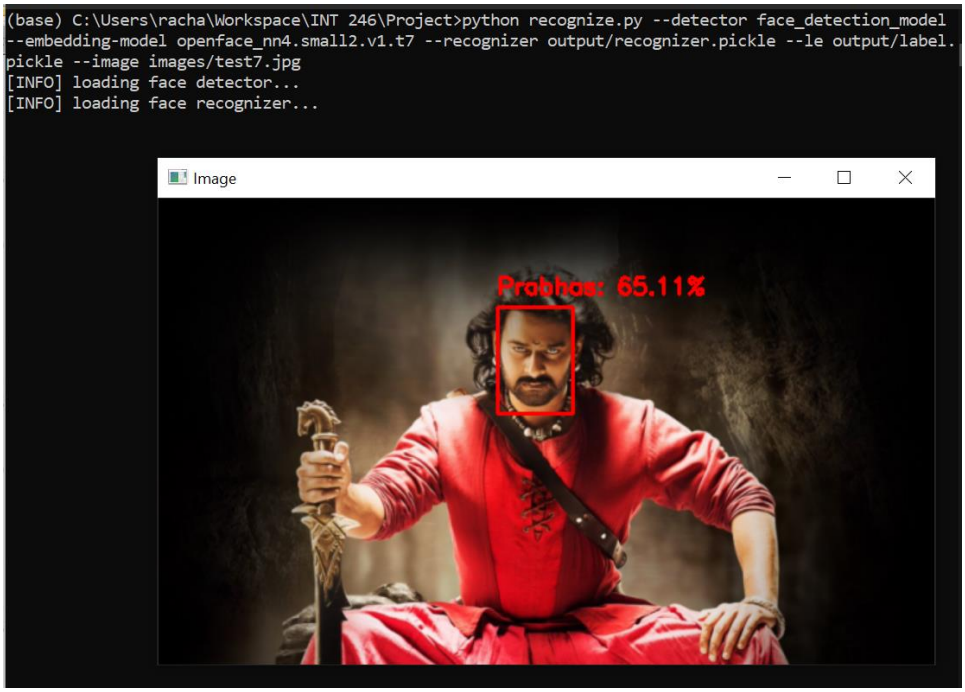
➢ Now, to display OpenCV face recognition results:

```
95              # draw the bounding box of the face along with the associated
96              # probability
97              text = "{}: {:.2f}%".format(name, proba * 100)
98              y = startY - 10 if startY - 10 > 10 else startY + 10
99              cv2.rectangle(image, (startX, startY), (endX, endY),
100                 (0, 0, 255), 2)
101             cv2.putText(image, text, (startX, y),
102                 cv2.FONT_HERSHEY_SIMPLEX, 0.60, (0, 0, 255), 2)
103
104 # show the output image
105 cv2.imshow("Image", image)
106 cv2.waitKey(0)
```

- For every face we recognize in the loop (including the “unknown”) people:
  1. We construct a text string containing the name and probability on Line 97.
  2. Then we draw a rectangle around the face and place the text above the box (Lines 98-102).
  3. Finally, we visualize the results on the screen until a key is pressed (Lines 105 and 106).

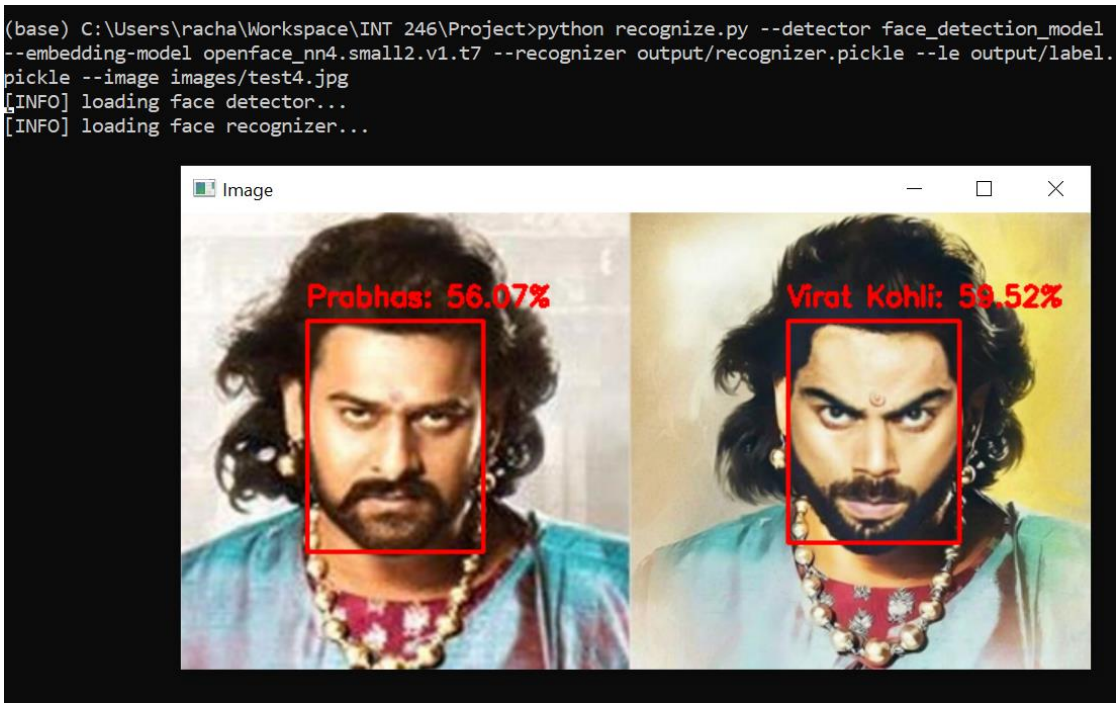
➢ To recognize faces in images with OpenCV, we apply OpenCV face recognition pipeline to provide images (or our own dataset + test images)

- By opening up a terminal and execute the following command we get recognized output:

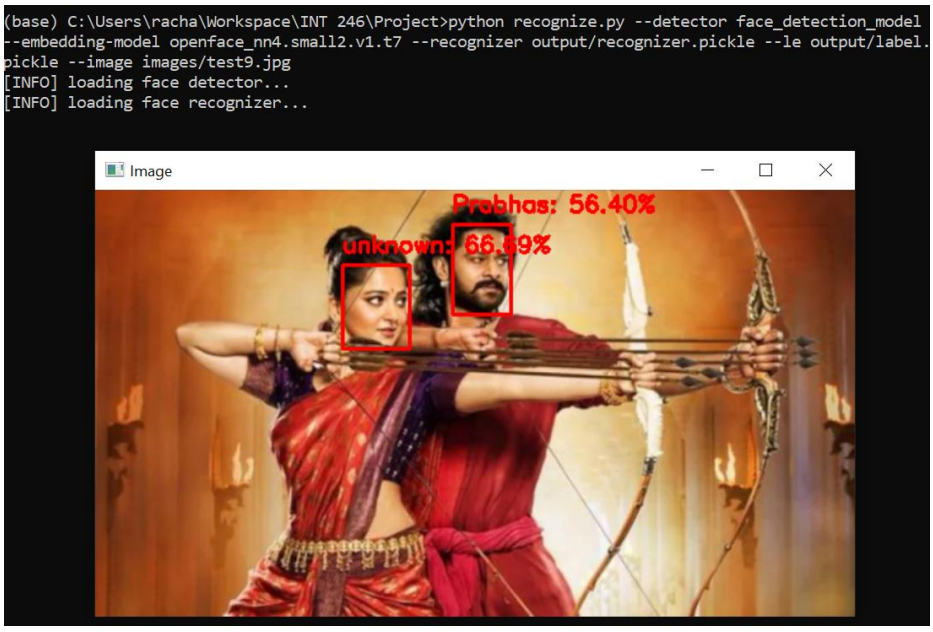


- To test with another image containing two different faces, we just change the test image number in command line and the command is,

$ python recognize.py --detector face_detection_model --embedding-model openface_nn4.small2.v1.t7 --recognizer output/recognizer.pickle --le output/label.pickle --image images/test4.jpg



- when model is unable to recognize the actual face it label them as Unknown for e.g;.



8.4. Recognize faces in video streams (step 4)

The actual OpenCV pipeline itself is near identical to recognizing faces in images, But only a few updates for Face Recognition in video Stream.

➢ Create the recognize_video.py file then,

```
5  # import the necessary packages
6  from imutils.video import VideoStream
7  from imutils.video import FPS
8  import numpy as np
9  import argparse
10 import imutils
11 import pickle
12 import time
13 import cv2
14 import os
15
16 # construct the argument parser and parse the arguments
17 ap = argparse.ArgumentParser()
18 ap.add_argument("-d", "--detector", required=True,
19     help="path to OpenCV's deep learning face detector")
20 ap.add_argument("-m", "--embedding-model", required=True,
21     help="path to OpenCV's deep learning face embedding model")
22 ap.add_argument("-r", "--recognizer", required=True,
23     help="path to model trained to recognize faces")
24 ap.add_argument("-l", "--le", required=True,
25     help="path to label encoder")
26 ap.add_argument("-c", "--confidence", type=float, default=0.5,
27     help="minimum probability to filter weak detections")
28 args = vars(ap.parse_args())
```

- The imports are the same as the Step 3 section above, except for Lines 6 and 7 where we used the **imutils.video** module. We use **VideoStream** to capture frames from the camera and FPS to calculate frames per second statistics [5].

- The command line arguments are also the same except we aren't passing a path to a static image via the command line. Rather, we use grab a reference to the webcam and then process the video. Refer to Step 3 if needed to review the arguments.

➢ The three models and label encoder are loaded with:

```
30  # load our serialized face detector from disk
31  print("[INFO] loading face detector...")
32  protoPath = os.path.sep.join([args["detector"], "deploy.prototxt"])
33  modelPath = os.path.sep.join([args["detector"],
34      "res10_300x300_ssd_iter_140000.caffemodel"])
35  detector = cv2.dnn.readNetFromCaffe(protoPath, modelPath)
36
37  # load our serialized face embedding model from disk
38  print("[INFO] loading face recognizer...")
39  embedder = cv2.dnn.readNetFromTorch(args["embedding_model"])
40
41  # load the actual face recognition model along with the label encoder
42  recognizer = pickle.loads(open(args["recognizer"], "rb").read())
43  le = pickle.loads(open(args["le"], "rb").read())
```

- To load face detector, face embedder model, face recognizer model (Linear SVM), and label encoder.
- Again, refer to Step 3 if needed to review about the three models or label encoder.

➢ To initialize video stream and begin processing frames:

```
45  # initialize the video stream, then allow the camera sensor to warm up
46  print("[INFO] starting video stream...")
47  vs = VideoStream(src=0).start()
48  time.sleep(2.0)
49  rows = 480
50  cols = 640
51
52  # start the FPS throughput estimator
53  fps = FPS().start()
54
55  # loop over frames from the video file stream
56  while True:
57      # grab the frame from the threaded video stream
58      frame = vs.read()
59
60      # resize the frame to have a width of 600 pixels (while
61      # maintaining the aspect ratio), and then grab the image
62      # dimensions
63      frame = imutils.resize(frame, width=640)
64      (h, w) = frame.shape[:2]
65
66      # construct a blob from the image
67      imageBlob = cv2.dnn.blobFromImage(
68          cv2.resize(frame, (300, 300)), 1.0, (300, 300),
69          (104.0, 177.0, 123.0), swapRB=False, crop=False)
70
71      # apply OpenCV's deep learning-based face detector to localize
72      # faces in the input image
73      detector.setInput(imageBlob)
74      detections = detector.forward()
```

- **VideoStream** object is initialized and started on Line 47. We wait for the camera sensor to warm up on Line 48.

- We also initialize frames per second counter (Line 53) and begin looping over frames on Line 50. We grab a frame from the webcam on Line 57.

- Here onwards everything is the same as Step 3. We resize the frame (Line 63) and then we construct a blob from the frame + detect where the faces are (Lines 67-74).

➢ To process the detections, we use:

```
76      # loop over the detections
77      for i in range(0, detections.shape[2]):
78          # extract the confidence (i.e., probability) associated with
79          # the prediction
80          confidence = detections[0, 0, i, 2]
81
82          # filter out weak detections
83          if confidence > args["confidence"]:
84              # compute the (x, y)-coordinates of the bounding box for
85              # the face
86              box = detections[0, 0, i, 3:7] * np.array([w, h, w, h])
87              (startX, startY, endX, endY) = box.astype("int")
88
89              # extract the face ROI
90              face = frame[startY:endY, startX:endX]
91              (fH, fW) = face.shape[:2]
92
93              # ensure the face width and height are sufficiently large
94              if fW < 20 or fH < 20:
95                  continue
```

- we begin looping over detections and filter out weak ones (Lines 77-83). Then we extract the face ROI as well as ensure the spatial dimensions are sufficiently large enough for the next steps (Lines 90-95).

➢ To perform OpenCV face recognition:

```
97                  # construct a blob for the face ROI, then pass the blob
98                  # through our face embedding model to obtain the 128-d
99                  # quantification of the face
100                 faceBlob = cv2.dnn.blobFromImage(face, 1.0 / 255,
101                     (96, 96), (0, 0, 0), swapRB=True, crop=False)
102                 embedder.setInput(faceBlob)
103                 vec = embedder.forward()
104
105                 # perform classification to recognize the face
106                 preds = recognizer.predict_proba(vec)[0]
107                 j = np.argmax(preds)
108                 proba = preds[j]
109                 name = le.classes_[j]
110
111                 # draw the bounding box of the face along with the
112                 # associated probability
113                 text = "{}: {:.2f}%".format(name, proba * 100)
114                 y = startY - 10 if startY - 10 > 10 else startY + 10
115                 cv2.rectangle(frame, (startX, startY), (endX, endY),
116                     (0, 0, 255), 2)
117                 cv2.putText(frame, text, (startX, y),
118                     cv2.FONT_HERSHEY_SIMPLEX, 0.65, (0, 0, 255), 2)
119
120             # update the FPS counter
121             fps.update()
```

- We Construct the faceBlob (Lines 100 and 101) and calculate the facial embeddings via deep learning (Lines 102 and 103).

- We Recognize the most-likely name of the face while calculating the probability (Line 106-109).

- Draw a bounding box around the face and the person's name + probability (Lines 113 -118).

- With fps counter is updated on Line 121.

➢ To display the results and clean up pop-up window:

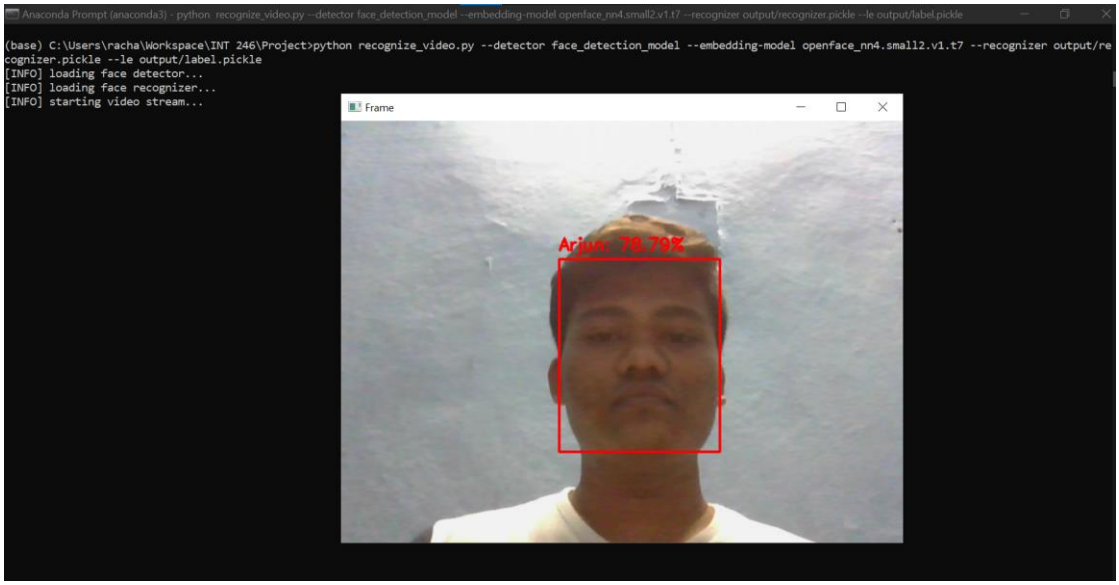
```
122         # show the output frame
123         cv2.imshow("Frame", frame)
124
125         key = cv2.waitKey(1) & 0xFF
126
127         # if the `q` key was pressed, break from the loop
128         if key == ord("q"):
129             break
130
131     # stop the timer and display FPS information
132     fps.stop()
133     print("[INFO] elasped time: {:.2f}".format(fps.elapsed()))
134     print("[INFO] approx. FPS: {:.2f}".format(fps.fps()))
135
136     # do a bit of cleanup
137     cv2.destroyAllWindows()
138     vs.stop()
```

- To close out the script, we us Display the annotated frame (Line 123) and wait for the “q” key to be pressed at which point we break out of the loop (Lines 125-129).

- To Stop fps counter and print statistics in the terminal (Lines 132-134).

- To Clean-up by closing windows and releasing pointers (Lines 137 and 138).

- To execute OpenCV face recognition pipeline on a video stream, open up a terminal and execute the following command:
  $ python recognize_video.py --detector face_detection_model --embedding-model openface_nn4.small2.v1.t7 --recognizer output/recognizer.pickle --le output/label.pickle



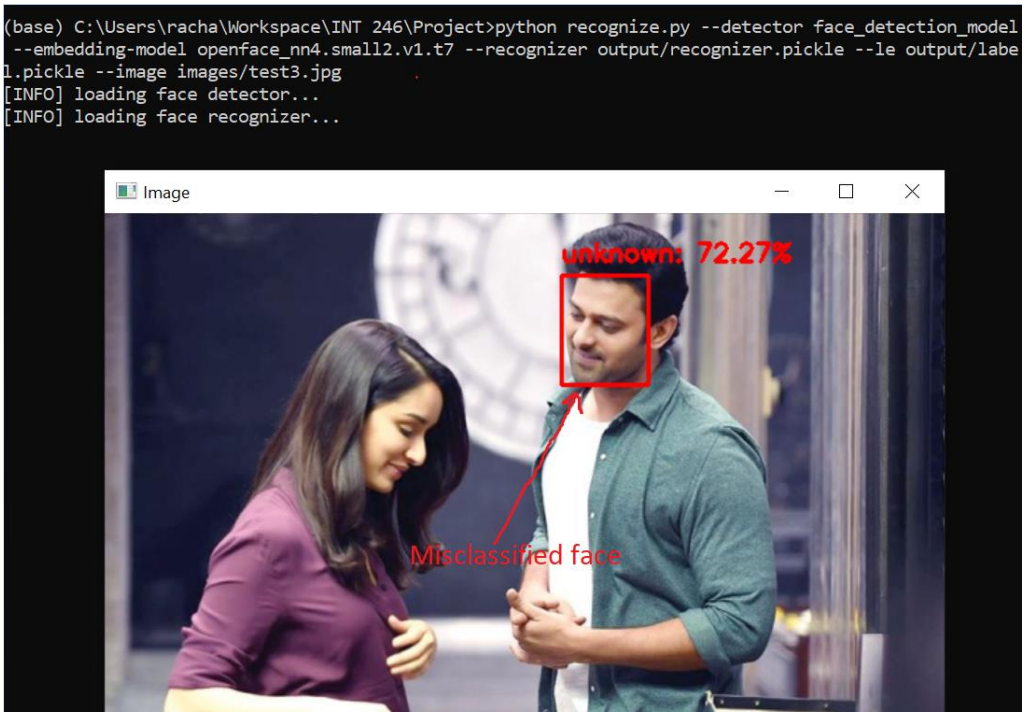
- We can see that my face was identified correctly.
- We can also find the frames per second (FPS) statistics when the window cleaned up.

```
(base) C:\Users\racha\Workspace\INT 246\Project>python recognize_video.py --detector
face_detection_model --embedding-model openface_nn4.small2.v1.t7 --recognizer output/
recognizer.pickle --le output/label.pickle
[INFO] loading face detector...
[INFO] loading face recognizer...
[INFO] starting video stream...
[INFO] elasped time: 76.50
[INFO] approx. FPS: 34.00
```

9. Project Legacy

9.1. Current Status of the project

➢ **Drawbacks, limitations, and how to obtain higher face recognition accuracy**



- In this situation where OpenCV does not recognize a face correctly.

9.2. Remaining Areas of concern

To improve OpenCV face recognition accuracy of face recognition pipeline there are few methods [5].

1. Perform face alignment

- The face recognition model OpenCV uses to compute the 128-d face embeddings comes from the OpenFace project.
- The OpenFace model will perform better on faces that have been aligned.

- Face alignment is the process of:
  1. Identifying the geometric structure of faces in images.
  2. Attempting to obtain a canonical alignment of the face based on translation, rotation, and scale.

2. Tune your hyperparameters

- By tuning hyperparameters on whatever machine learning model we are using (i.e., the model trained on top of the extracted face embeddings) [5].

- we used a Linear SVM; however, we did not tune the C value, which is typically the most important value of an SVM to tune [5].

- The C value is a “strictness” parameter and controls how much we want to avoid misclassifying each data point in the training set [5].

- Larger values of C will be stricter and try harder to classify every input data point correctly, even at the risk of overfitting [5].

- Smaller values of C will be “softer”, allowing some misclassifications in the training data, but ideally generalizing better to testing data [5].

10. Source Code

- **Github Link**:
  https://github.com/Arjun144/Face-Recognition.git

- **Dataset and Code:**
  https://drive.google.com/drive/folders/1mEf333IKHEFyaDqEAcsPRhmn9qXUAIqR?usp=sharing

11. Bibliography

[1] I.Michael Revina, W.R. SamEmmanuel, Human Face Expression Recognition Techniques

[2] Liyanage C De Silva, Prahlad Vadakkepat, Sinagapore: Research Gate, July 2008

[3] Mr. Ashok Kumar Pant, Nepal: Academia, September 2016

[4] Wei-Lun Chao, GICE, National Taiwan University, Recognition Survey

[5] OpenCv Face Recognition

[6] Face Recognition Python

[7] Notepad++

[8] OpenCV