**CMR INSTITUTE OF TECHNOLOGY**

1. #132, AECS LAYOUT, IT PARK ROAD, KUNDALAHALLI,
2. BANGALORE-560037

# DEPARTMENT OF Computer SCIENCE AND ENGINEERING

****

**A Mini Project Report**

**On**

**“Face Recognition using Python” (Machine Learning)**

**Submitted in Partial fulfillment of the Requirements for the 5TH Semester of the Degree of**

**Bachelor of Engineering**

**In**

**Computer Science & Engineering**

**By**

**CHETHAN B S**

**(1CR17CS032)**

**ADITYA AV**

**(1CR17CS006)**

**CHANDAN N**

**(1CR17CS031)**

**Under the guidance**

**Of**

**Rubini**

**Asst Professor, Dept of CSE**

…………………… ……………………………… ……………………..

**Reviewer 1**: RUBINI  **Reviewer 2:** POONAM TIJARE **Reviewer 3:** P N Singh

**ABSTRACT**

The face is one of the easiest ways to distinguish the individual identity of each other. Face recognition is a personal identification system that uses personal characteristics of a person to identify the person's identity. Human face recognition procedure basically consists of two phases, namely face detection, where this process takes place very rapidly in humans, except under conditions where the object is located at a short distance away, the next is the introduction, which recognize a face as individuals. Stage is then replicated and developed as a model for facial image recognition (face recognition) is one of the much-studied biometrics technology and developed by experts. There are two kinds of methods that are currently popular in developed face recognition pattern namely, Eigenface method and Fisherface method. Facial image recognition Eigenface method is based on the reduction of face dimensional space using Principal Component Analysis (PCA) for facial features.

**INTRODUCTION**

Face recognition is the task of identifying an already detected object as a known or unknown face. Often the problem of face recognition is confused with the problem of face detection. Face Recognition on the other hand is to decide if the "face" is someone known, or unknown, using for this purpose a database of faces in order to validate this input face.

Face detection involves separating image windows into two classes; one containing faces (tarning the background (clutter). It is difficult because although commonalities exist between faces, they can vary considerably in terms of age, skin colour and facial expression. The problem is further complicated by differing lighting conditions, image qualities and geometries, as well as the possibility of partial occlusion and disguise.

**CHAPTER 1**

**FACE RECOGNITION:**

DIFFERENT APPROACHES OF FACE RECOGNITION:

There are two predominant approaches to the face recognition problem:

1. Geometric (feature based)
2. photometric (view based).

**Popular recognition algorithms include:**

1. Principal Component Analysis using Eigenfaces (PCA)
2. Linear Discriminate Analysis
3. Elastic Bunch Graph Matching using the Fisherface algorithm
4. LBPH Recognizer

**OpenCV provides the following three face recognizers:**

1. Eigenface recognizer
2. Fisherface recognizer
3. LBPH face recognizer

**FACE DETECTION:**

Face detection involves separating image windows into two classes; one containing faces (tarning the background (clutter).

It is difficult because although commonalities exist between faces, they can vary considerably in terms of age, skin colour and facial expression. The problem is further complicated by differing lighting conditions, image qualities and geometries, as well as the possibility of partial occlusion and disguise.

**The face detection system can be divided into the following steps:-**

1. **Pre-Processing:**

To reduce the variability in the faces, the images are processed before they are fed into the network. All positive examples that is the face images are obtained by cropping the input images. All the cropped images are then corrected for lighting through standard algorithms.

1. **Classification:**

Neural networks are implemented to classify the images as faces or nonfaces by training on these examples. We use both our implementation of the neural network and the OpenCV for this task. Different network configurations are experimented with to optimize the results.

1. **Localization:**

The trained neural network is then used to search for faces in an image and if present localize them in a bounding box.

**CHAPTER 2**

**(OpenCV)**

**OpenCV ( Open Computer Vision ):**

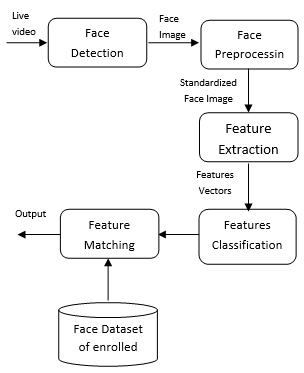
The ability to see and perceive the world comes naturally to us humans. It’s second nature for us to gather informatiofrom our surroundings through the gift of vision and perception.

**Features of OpenCV:**

1. Changing Color Spaces
2. Reading, Writing and Displaying Images
3. Resizing Images
4. Image Rotation
5. Image Translation
6. Simple Image Thresholding
7. Adaptive Thresholding
8. Image Segmentation (Watershed Algorithm)
9. Bitwise Operations
10. Edge Detection
11. Image Filtering Image
12. Contours
13. Scale Invariant Feature Transform (SIFT)
14. Speeded-Up Robust Features (SURF)
15. Feature Matching
16. Face Detection

Here, we are using Face Detection, Recognition and Feature Matching for implementing the model.

The below chart will show the steps involved in recognizing a face as known or unknown:



**CHAPTER 3**

DESIGN AND IMPLEMENTATION

**Classifier.py**

# Python code to classify the images

import numpy as np

from PIL import Image

import os, cv2

# Method to train custom classifier to recognize face

def train\_classifer(data\_dir):

# Read all the images in custom data-set

path = [os.path.join(data\_dir, f) for f in os.listdir(data\_dir)]

faces = []

ids = []

# Store images in a numpy format and ids of the user on the same index in imageNp and id lists

for image in path:

img = Image.open(image).convert('L')

imageNp = np.array(img, 'uint8')

id = int(os.path.split(image)[1].split(".")[1])

faces.append(imageNp)

ids.append(id)

ids = np.array(ids)

# Train and save classifier

clf = cv2.face.LBPHFaceRecognizer\_create()

clf.train(faces, ids)

clf.write("classifier.xml")

train\_classifer("data")

**collect\_training\_data.py**

#Python code to generate data for our model

import cv2

# Method to generate dataset to recognize a person

def generate\_dataset(img, id, img\_id):

cv2.imwrite("data/user."+str(id)+"."+str(img\_id)+".jpg", img)

# Method to draw boundary around the detected feature

def draw\_boundary(img, classifier, scaleFactor, minNeighbors, color, text):

# Converting image to gray-scale

gray\_img = cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY)

features = classifier.detectMultiScale(gray\_img, scaleFactor, minNeighbors)

coords = []

# drawing rectangle around the feature and labeling it

for (x, y, w, h) in features:

cv2.rectangle(img, (x,y), (x+w, y+h), color, 2)

cv2.putText(img, text, (x, y-4), cv2.FONT\_HERSHEY\_SIMPLEX, 0.8, color, 1, cv2.LINE\_AA)

coords = [x, y, w, h]

return coords

# Method to detect the features

def detect(img, faceCascade, img\_id):

color = {"blue":(255,0,0), "red":(0,0,255), "green":(0,255,0), "white":(255,255,255)}

coords = draw\_boundary(img, faceCascade, 1.1, 10, color['blue'], "Face")

if len(coords)==4:

# Updating region of interest by cropping image

roi\_img = img[coords[1]:coords[1]+coords[3], coords[0]:coords[0]+coords[2]]

# Assign unique id to each user

user\_id = 1

# img\_id to make the name of each image unique

generate\_dataset(roi\_img, user\_id, img\_id

return img

# Loading classifiers

faceCascade = cv2.CascadeClassifier('haarcascade\_frontalface\_default.xml')

# Capturing real time video stream. 0 for built-in web-cams, 0 or -1 for external web-cams

video\_capture = cv2.VideoCapture(-1)

# Initialize img\_id with 0

img\_id = 0

while True:

if img\_id % 50 == 0:

print("Collected ", img\_id," images")

\_, img = video\_capture.read()

img = detect(img, faceCascade, img\_id)

cv2.imshow("face detection", img)

img\_id += 1

if cv2.waitKey(1) & 0xFF == ord('q'):

break

video\_capture.release()

# Destroying output window

cv2.destroyAllWindows()

**face\_detection.py**

import cv2

# Method to draw boundary around the detected feature

def draw\_boundary(img, classifier, scaleFactor, minNeighbors, color, text):

# Converting image to gray-scale

gray\_img = cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY)

# detecting features in gray-scale image, returns coordinates, width and height of features

features = classifier.detectMultiScale(gray\_img, scaleFactor, minNeighbors)

coords = []

# drawing rectangle around the feature and labeling it

for (x, y, w, h) in features:

cv2.rectangle(img, (x,y), (x+w, y+h), color, 2)

cv2.putText(img, text, (x, y-4), cv2.FONT\_HERSHEY\_SIMPLEX, 0.8, color, 1, cv2.LINE\_AA)

coords = [x, y, w, h]

return coords

# Method to detect the features

def detect(img, faceCascade, eyeCascade, noseCascade, mouthCascade):

color = {"blue":(255,0,0), "red":(0,0,255), "green":(0,255,0), "white":(255,255,255)}

coords = draw\_boundary(img, faceCascade, 1.1, 10, color['blue'], "Face")

# If feature is detected, the draw\_boundary method will return the x,y coordinates and width and height of rectangle else the length of coords will be 0

if len(coords)==4:

# Updating region of interest by cropping image

roi\_img = img[coords[1]:coords[1]+coords[3], coords[0]:coords[0]+coords[2]]

# Passing roi, classifier, scaling factor, Minimum neighbours, color, label text

coords = draw\_boundary(roi\_img, eyeCascade, 1.1, 12, color['red'], "Eye")

coords = draw\_boundary(roi\_img, noseCascade, 1.1, 4, color['green'], "Nose")

coords = draw\_boundary(roi\_img, mouthCascade, 1.1, 20, color['white'], "Mouth")

return img

# Loading classifiers

faceCascade = cv2.CascadeClassifier('haarcascade\_frontalface\_default.xml')

eyesCascade = cv2.CascadeClassifier('haarcascade\_eye.xml')

noseCascade = cv2.CascadeClassifier('Nariz.xml')

mouthCascade = cv2.CascadeClassifier('Mouth.xml')

# Capturing real time video stream. 0 for built-in web-cams, 0 or -1 for external web-cams

video\_capture = cv2.VideoCapture(-1)

while True:

# Reading image from video stream

\_, img = video\_capture.read()

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'):

break

video\_capture.release()

# Destroying output window

cv2.destroyAllWindows()

**recognize.py**

import cv2

def draw\_boundary(img, classifier, scaleFactor, minNeighbors, color, text, clf):

# Converting image to gray-scale

gray\_img = cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY)

# detecting features in gray-scale image, returns coordinates, width and height of features

features = classifier.detectMultiScale(gray\_img, scaleFactor, minNeighbors)

coords = []

# drawing rectangle around the feature and labeling it

for (x, y, w, h) in features:

cv2.rectangle(img, (x,y), (x+w, y+h), color, 2)

# Predicting the id of the user

id, \_ = clf.predict(gray\_img[y:y+h, x:x+w])

# Check for id of user and label the rectangle accordingly

if id==1:

cv2.putText(img, "Ali", (x, y-4), cv2.FONT\_HERSHEY\_SIMPLEX, 0.8, color, 1, cv2.LINE\_AA)

coords = [x, y, w, h]

return coords

# Method to recognize the person

def recognize(img, clf, faceCascade):

color = {"blue": (255, 0, 0), "red": (0, 0, 255), "green": (0, 255, 0), "white": (255, 255, 255)}

coords = draw\_boundary(img, faceCascade, 1.1, 10, color["white"], "Face", clf)

return img

# Loading classifier

faceCascade = cv2.CascadeClassifier('haarcascade\_frontalface\_default.xml')

# Loading custom classifier to recognize

clf = cv2.face.LBPHFaceRecognizer\_create()

clf.read("classifier.yml")

# Capturing real time video stream. 0 for built-in web-cams, 0 or -1 for external web-cams

video\_capture = cv2.VideoCapture(-1)

while True:

\_, img = video\_capture.read()

# Call method we defined above

img = recognize(img, clf, faceCascade)

# Writing processed image in a new window

cv2.imshow("face detection", img)

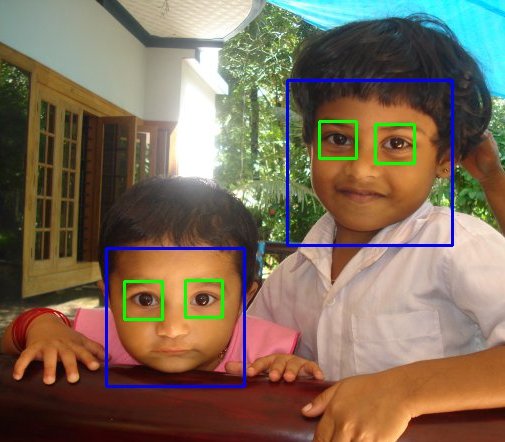
if cv2.waitKey(1) & 0xFF == ord('q'):

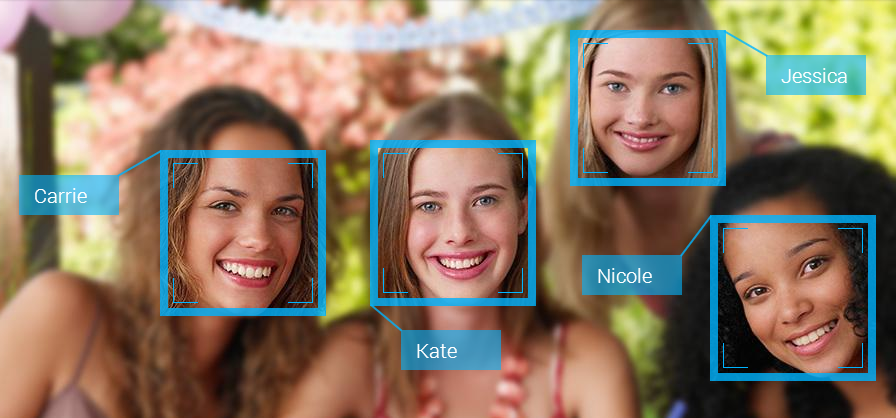
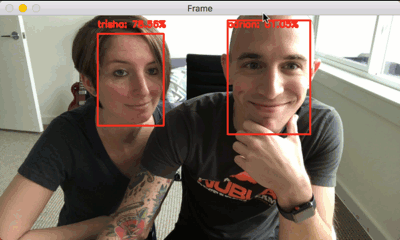
break

video\_capture.release()

# Destroying output window

cv2.destroyAllWindows()

OUTPUT SCREENSHOTS



CONCLUSION

The implemented fully automated face detection and recognition system (with an eye detection system) could be used for simple surveillance applications such as ATM user security, while the implemented manual face detection and automated recognition system is ideal of mugshot matching. Since controlled conditions are present when mugshots are gathered, the frontal view face recognition scheme should display a recognition accuracy far better than the results, which were obtained in this study, which was conducted under adverse conditions.

The automated vision systems implemented in this thesis did not even approach the performance, nor were they as robust as a human's innate face recognition system. However, they give an insight into what the future may hold in computer vision.

REFERENCES

#### Resources

1. **Link to source code, pre-trained classifiers:**

<https://github.com/chethan1999/Face-Recognition-using-ML.git>

1. **Online resource to understand OpenCV:**

<https://docs.opencv.org/2.4/doc/tutorials/tutorials.html>

<https://www.tutorialspoint.com/opencv/index.htm>

<https://www.youtube.com/watch?v=kdLM6AOd2vc>