Experiment No. 7
To perform Face detection on Video
Name of Student :- 09_Amruta Poojary
Date of Performance: 6/9/2023
Date of Submission: 13/9/2023

Aim: To perform Face detection on Video

Objective: Performing face recognition Generating the data for face recognition Recognizing faces preparing the training data Loading the data and recognizing faces.

Theory:

Recognizing faces:

OpenCV 4 implements three different algorithms for recognizing faces: Eigenfaces, Fisherfaces, and Local Binary Pattern Histograms (LBPHs). Eigenfaces and Fisherfaces are derived from a more general-purpose algorithm called Principal Component Analysis (PCA). They take a set of classified observations (our face database, containing numerous samples per individual), train a model based on it, perform an analysis of face images (which may be face regions that we detected in an image or video), and determine two things: the subject's identity, and a measure of confidence that this identification is correct. The latter is commonly known as the confidence score.

Eigenfaces performs PCA, which identifies principal components of a certain set of observations (again, your face database), calculates the divergence of the current observation (the face being detected in an image or frame) compared to the dataset, and produces a value. The smaller the value, the smaller the difference between the face database and detected face; hence, a value of 0 is an exact match. Fisherfaces also derives from PCA and evolves the concept, applying more complex logic. While computationally more intensive, it tends to yield more accurate results than Eigenfaces.

LBPH instead divides a detected face into small cells and, for each cell, builds a histogram that describes whether the brightness of the image is increasing when comparing neighboring pixels in a given direction. This cell's histogram can be compared to the corresponding cells in the model, producing a measure of similarity. Of the face recognizers in OpenCV, the implementation of LBPH is the only one that allows the model sample faces and the detected faces to be of different shape and size. Hence, it is a convenient option, and the authors of this book find that its accuracy compares favorably to the other two options.

Loading the data and recognizing faces:

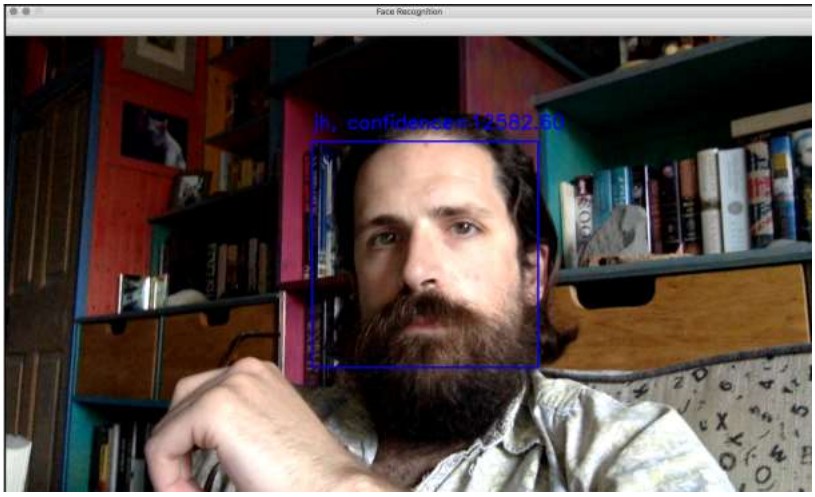
Regardless of our choice of face recognition algorithm, we can load the training images in the same way. Earlier, in the Generating the data for face recognition section, we generated training images and saved them in folders that were organized according to people's names or initials

Code:

```
import os

import cv2
import numpy

def read_images(path, image_size):
    names = []
    training_images, training_labels = [], []
    label = 0
    for dirname, subdirnames, filenames in os.walk(path):
        for subdirname in subdirnames:
            names.append(subdirname)
            subject_path = os.path.join(dirname, subdirname)
            for filename in os.listdir(subject_path):
                img = cv2.imread(os.path.join(subject_path, filename),
                                 cv2.IMREAD_GRAYSCALE)
                if img is None:
                    # The file cannot be loaded as an image.
                    # Skip it.
                    continue
                img = cv2.resize(img, image_size)
                training_images.append(img)
                training_labels.append(label)
            label += 1
    training_images = numpy.asarray(training_images, numpy.uint8)
    training_labels = numpy.asarray(training_labels, numpy.int32)
    return names, training_images, training_labels


path_to_training_images = '../data/at'
training_image_size = (200, 200)
names, training_images, training_labels = read_images(
    path_to_training_images, training_image_size)

model = cv2.face.EigenFaceRecognizer_create()
```

```
model.train(training_images, training_labels)

face_cascade = cv2.CascadeClassifier(
   f'{cv2.data.haarcascades}haarcascade_frontalface_default.xml')

camera = cv2.VideoCapture(0)
while (cv2.waitKey(1) == -1):
   success, frame = camera.read()
   if success:
      faces = face_cascade.detectMultiScale(frame, 1.3, 5)
      for (x, y, w, h) in faces:
         cv2.rectangle(frame, (x, y), (x+w, y+h), (255, 0, 0), 2)
         gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
         roi_gray = gray[x:x+w, y:y+h]
         if roi_gray.size == 0:
            # The ROI is empty. Maybe the face is at the image edge.
            # Skip it.
            continue
         roi_gray = cv2.resize(roi_gray, training_image_size)
         label, confidence = model.predict(roi_gray)
         text = '%s, confidence=%.2f' % (names[label], confidence)
         cv2.putText(frame, text, (x, y - 20),
                  cv2.FONT_HERSHEY_SIMPLEX, 1, (255, 0, 0), 2)
      cv2.imshow('Face Recognition', frame)
```

Output:



Conclusion:

Face detection on video is an important task in many computer vision or machine applications. OpenCV allows to perform face detection on video through eigenfaces. A set of eigenfaces can be generated by performing a mathematical process called principal component analysis (PCA) on a large set of images depicting different human faces. Informally, eigenfaces can be considered a set of "standardized face ingredients", derived from statistical analysis of many pictures of faces.