CAMERA CHECKING:

camcheck.py > ...

```python
import cv2
import numpy as np

cap = cv2.VideoCapture(0)
while True:
    ret, frame = cap.read()
    cv2.imshow('DetectCam', frame)
    if(cv2.waitKey(1)==ord('q')):
        break

cap.release()
cv2.destroyAllWindows()
```

VideoCapture(0)- to capture the video. Basically it uses the inbuilt laptop webcam to display this video. (0) is used for one camera if more required increase the number.

cap.read()- this return a boolean value (True/False). If frame is read correctly, it will return True.

imshow()- it gives the frame a name, and second parameter is frame object.

waitKey()- it gives a continuous live video feed.

cap.release()- releases the software and hardware resources.

destroyAllWindows()- destroys the frame window created.

DATASET GATHERING:

datagather.py > ...

```python
import cv2
import os

cam = cv2.VideoCapture(0)
face_detector = cv2.CascadeClassifier('haarcascade_frontalface_default.xml')
face_id = input('\nEnter Id No.: ')
count = 0
while(True):
    ret, img = cam.read()
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    faces = face_detector.detectMultiScale(gray, 1.3, 5)
    for (x,y,w,h) in faces:
        cv2.rectangle(img, (x,y), (x+w,y+h), (255,0,0), 2)
        count += 1
        cv2.imwrite("dataset/User." + str(face_id) + '.' + str(count) + ".jpg", gray[y:y+h,x:x+w])
        cv2.imshow('CollectDatasets', img)
    if(cv2.waitKey(1)==ord('q')):
        break
    elif count >= 30:
        break
cam.release()
cv2.destroyAllWindows()
```

CascadeClassfier()- to load the haarcascade xml file.

cvtColor()- load the images and convert it to gray scale because generally photos are in RGB channel but openCV reads it as BGR channel. So, it is converted to gray scale containing only black and white.

detectMultiScale()- in this we will find features from the image.So, using the face_detector object we detect the features. First parameter is gray scale variable, second parameter is scaleFactor- how much the image size is reduced, last parameter is minNeighbours- how may candidate rectangles to retain. It returns 4 values (x,y,w,h).

rectangle()- based on the return values a rectangle is drawn around the face.

imwrite()- method to save the image to storage device. First parameter saving location path, second parameter in which format to save.

DATASET TRAINING:

facetrain.py > getImagesAndLabels

```python
import cv2
import numpy as np
from PIL import Image
import os

path = 'dataset'
recognizer = cv2.face.LBPHFaceRecognizer_create()
detector = cv2.CascadeClassifier("haarcascade_frontalface_default.xml")

def getImagesAndLabels(path):
    imagePaths = [os.path.join(path,f) for f in os.listdir(path)]
    faceSamples=[]
    ids = []
    for imagePath in imagePaths:
        PIL_img = Image.open(imagePath).convert('L')
        img_numpy = np.array(PIL_img,'uint8')
        id = int(os.path.split(imagePath)[-1].split(".")[1])
        faces = detector.detectMultiScale(img_numpy)
        for (x,y,w,h) in faces:
            faceSamples.append(img_numpy[y:y+h,x:x+w])
            ids.append(id)
    return faceSamples,ids
faces,ids = getImagesAndLabels(path)
recognizer.train(faces, np.array(ids))
recognizer.write('trainer.yml')
```

LBPHFaceRecognizer_create()- (LOCAL BINARY PATTERNS HISTOGRAMS) this is used for creating a recognizer. It labels the pixels of an image by thresholding the neighborhood of each pixel and considers the result as a binary number.

getImagesAndLabels(path)- it will take all the photos from the directory returning Ids and Faces as array.

imagePaths = [os.path.join(path,f) for f in os.listdir(path)]- used to append all the images files from the directory.

PIL_img = Image.open(imagePath).convert('L')- opens the image file and converts the grayscale image to PIL(Python Imaging Library) image.

img_numpy = np.array(PIL_img,'uint8')- converts the image to numpty array of pixels.

recognizer.train(faces, np.array(ids))- labels and ids of the image stored in vector form.

recognizer.write('trainer/trainer.yml')- stores all the values in an trainer.yml file.

FACE RECOGNITION:

facerecognition.py > ...

```python
import cv2
import numpy as np
import os

recognizer = cv2.face.LBPHFaceRecognizer_create()
recognizer.read('trainer.yml')
faceCascade = cv2.CascadeClassifier('haarcascade_frontalface_default.xml')
id = 0

names = ['None', 'Priyal', 'Heli', 'Simran', 'Nilesh Sir', 'W']
cam = cv2.VideoCapture(0)
# Define min window size to be recognized as a face
minW = 0.1*cam.get(3)
minH = 0.1*cam.get(4)
while True:
    ret, img =cam.read()
    gray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)

    faces = faceCascade.detectMultiScale(gray,scaleFactor = 1.2,minNeighbors = 5,minSize = (int(minW), int(minH)))
    for(x,y,w,h) in faces:
        cv2.rectangle(img, (x,y), (x+w,y+h), (0,255,0), 2)
        id, confidence = recognizer.predict(gray[y:y+h,x:x+w])
        # If confidence is less then 100 ==> "0" : perfect match
        if (confidence < 100):
            id = names[id]
            confidence = "  {0}%".format(round(100 - confidence))
        else:
            id = "unknown"
            confidence = "  {0}%".format(round(100 - confidence))
        cv2.putText(img,str(id),(x+5,y-5),cv2.FONT_HERSHEY_SIMPLEX,1,(255,255,255),2)
        cv2.putText(img,str(confidence),(x+5,y+h-5),cv2.FONT_HERSHEY_SIMPLEX,1,(255,255,0),1)
    cv2.imshow('Recognition',img)
    if(cv2.waitKey(1)==ord('q')):
        break
cam.release()
cv2.destroyAllWindows()
```

id, confidence = recognizer.predict(gray[y:y+h,x:x+w])- first parameter image to get a prediction from. It returns the id and confidence value. Less confidence value prefect match.

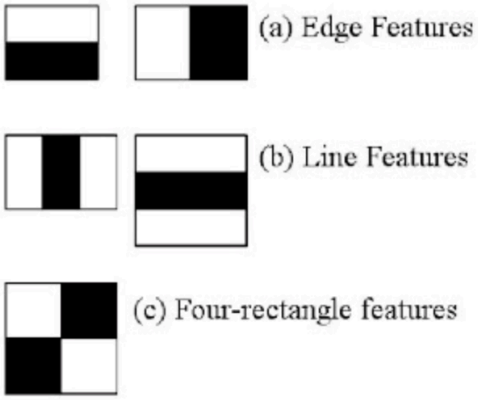
cv2.putText(img,str(id),(x+5,y-5),font,1,(255,255,255),2)- draw a text on the image. First parameter coordinates, second parameter font, third parameter font scale, fourth parameter colour, last parameter thickness.

Haar Cascade is a machine learning object detection algorithm used to identify objects in an image or video and based on the concept of features proposed by Paul Viola and Michael Jones.

The algorithm has four stages:

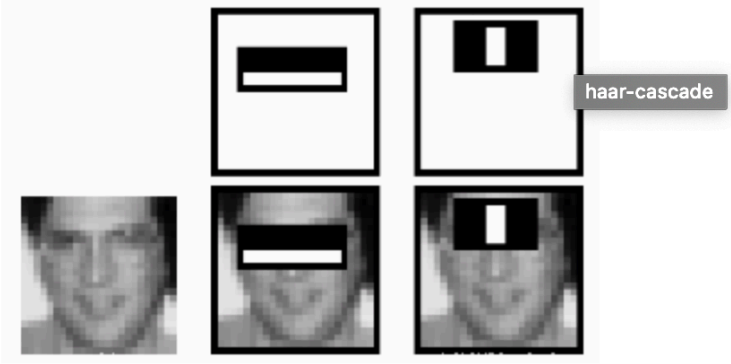
1. Haar Feature Selection-

A Haar feature considers adjacent rectangular regions at a specific location in a detection window, sums up the pixel intensities in each region and calculates the difference between these sums.



2. Creating Integral Images-

Top row shows two good features. The first feature selected seems to focus on the property that the region of the eyes is often darker than the region of the nose and cheeks. The second feature selected relies on the property that the eyes are darker than the bridge of the nose. But the same windows applying on cheeks or any other place is irrelevant.



3. Adaboost Training-

which both selects the best features and trains the classifiers that use them.

4. Cascading Classifiers- describe an object with sufficient accuracy