# FaceRecognition

## Imports and Functions Documentation

Amer Ashraf Zakaria

# IMPORTS

**OpenCV**

Is a library of python binding designed to solve computer vision problems.

**OpenCV makes NumPy**

Highly optimized library for numerical operations with MATLAB-style syntax.

**NumPy**

-Library which provide multidimensional array object. Nd-Array object encapsulate n-dimensional array of homogeneous data types with many operations being performed in compiled code for performance.

-All OpenCV array are converted to and from NumPy array.

**OS**

Provides a way that enables us to capture anything from the operation system we use, Such as: Read, OS.Path, Open() .

**PIL**

Python imaging library that adds supports for opening, manipulating, and saving images.

**CV2**

All packages in OpenCV library have the same namespace which is CV2.

**Import NumPy as np**

An alias for the namespace will be created, else access all the modules and all the functions in the NumPy module.

# Functions

**vid_cam = cv2.VideoCapture(0)**

**-**Set the video source to the default webcam.

-Related to opening and closing the webcam.

**face_detector = cv2.CascadeClassifier('haarcascade_frontalface_default.xml')**

**-Haarcascade Classifier**

Object identification method.

-Here we work on face recognition.

-We need a lot of positive images of faces to train classifier.

-We need to extract features from it.

**-'haarcascade_frontalface_default.xml'**

File that are stored in OpenCV which contains pre-trained classifiers for the face.

We use frontalFace which focus on the face, it can be changed to eye to change to eye recognition with totally different features.

**Count = 0**

-Counter to count how many photos are taken.

**_, image_frame = vid_cam.read()**

-Captures frame from the video.

-Returns a Boolean (True/False), if the frame is read correctly, it returns true.

**gray = cv2.cvtColor(image_frame, cv2.COLOR_BGR2GRAY)**

-Convert the colour of the frame it took from the video using last function.

**-Parameters**

**image_frame→** The variable name of the frame it took.

**cv2.COLOR_BGR2GRAY**

-"Flag" Determines the type of conversion which here is the colour.

-BGR2GRAY→ From coloured to Gray.

**faces = face_detector.detectMultiScale(gray, 1.3, 5)**

-Detect objects of different sizes in the input image. The detected objects return in a list of rectangles.

-gray→ Variable name for the image after conversion to gray colour.

-1.3→ Scale Factor: Parameter specifying how much the image size is reduced at each image scale. **Default "1.3".**

-5→ MinNeighbours(): Parameter specifying how many neighbours each candidate rectangle should have to retain it. Reducing it should reduce false positive.

-The detection algorithm uses a moving window to detect objects, it defines how many objects are detected near the current one before it declares the face is found.

**for (x,y,w,h) in faces**

-This loop returns four values for the rectangle

X: x-axis

Y: y-axis

“**The location of the rectangle”**

W: width

H: height

**cv2.rectangle(image_frame, (x,y), (x+w,y+h), (255,0,0), 2)**

-image_frame: the image itself.

-(x,y): the top left coordinate.

-(x+w,y+h): bottom right coordinate.

-(255,0,0): the color of the rectangle.

-2: thickness of the rectangle.

-takes photos, crop them with the upper features.

**count += 1**

-increment sample face image

-counter for number of image that has been taken.

**cv2.imwrite("dataset/User." + str(face_id) + '.' + str(count) + ".jpg", gray[y:y+h,x:x+w])**

-Saves the image taken in a as a JPG file in the dataset file with the ID number and the number of the counter it was count on.

-with grayscale.

**cv2.imshow('frame', image_frame)**

-Displays our image in a window it will show automatically fits to the image size.

'frame': Window name.

Image_frame: The image that was saved.

**cv2.waitKey(100) & 0xFF == ord('q')**

**- cv2.waitKey(100) :** wait for 100 milliseconds to exit.

**-0xFF == ord('q'):** Or press 'q' to exit.

**if count>100**

-If count is 100 stop as well.

**cam.release()**

-Stop the camera.

**cv2.destroyAllWindows()**

-close all the windows the app opened.

**recognizer = cv2.face.LBPHFaceRecognizer_create()**

-Not to look at the image as whole, try to find its local structure by comparing each pixel with the neighbouring pixel.

-LBPH→ Local binary patterns histogram

-Each pixel has a value, when the value of pixels increase, it sees the neighbour's pixels value, and compare them.

- "like a histogram diagram".

**def getImagesAndLabels(path)**

-To get the path of the dataset folder.

-Inside the function:

-Load the trainer images from dataset folder

-capture faces and ids from training images from training images.

-put them in samples and return them

**imagePaths = [os.path.join(path,f) for f in os.listdir(path)]**

-This will get all file path, the path of each image in the dataset folder.

**faceSamples=[]**

**ids = []**

-List of faces and IDs to store Faces and IDs (Array).

**PIL_img = Image.open(imagePath).convert('L')**

-Loading the image from its path and converting it to grayscale.

**img_numpy = np.array(PIL_img,'uint8')**

-Converting the image we loaded with the previous function to NumPy array.

**id = int(os.path.split(imagePath)[-1].split(".")[1])**

-To get the ID we split the image path and took the first from the last part (which is '-1' in Python) and that is the name of the image file.

-we save the file name in our previous program as "User.ID.SampleNumber", so if we split by '.', we get three tokens int the list (user, ID, SampleNumber).

-So to get the ID we will choose the first index (index start from zero).

**for (x,y,w,h) in faces**

-Now we are using the detector to extract the faces and append them in the face sample list with the ID.

**faceSamples.append(img_numpy[y:y+h,x:x+w])**

-Append() : Adds a single element to the end of the list

-Common error: does not return the new list, just modifies the original.

-Adding images to image sample.

-[y:y+h,x:x+w] : To crop the image by this feature.

**return faceSamples,ids**

-Return the value of face sample and IDs

**faces,ids = getImagesAndLabels('dataset')**

-In the end by calling the function we find the data from file called from file called dataset.

**recognizer.train(faces, np.array(ids))**

-Train the model by using the face samples and IDs.

**recognizer.save('trainer/trainer.yml')**

-Save the model in the file called trainer with the name trainner which has the extension .yml .

**recognizer.read('trainer/trainer.yml')**

-read from the file we created in the previous function.

**font = cv2.FONT_HERSHEY_SIMPLEX**

-The font of the text that will be on top of the rectangle of recognition.

**Id = recognizer.predict(gray[y:y+h,x:x+w])**

-The recognizer predicts the user ID and confidence of the prediction respectively.

**cv2.putText(im, str(Id), (x,y-40), font, 2, (255,255,255), 3)**

-we write the prediction User ID in the screen above the face, which is (x,y-40) .

-im: the image

-str(id): the name it has predicted

-(x,y-40): the location it is going to put the name at.

-font: the font it will be written in.

-2: Font scale factor that is multiplied by the font-specific base size.

-(255,255,255): the colour of the word.

-the thickness of the word.

**cv2.imshow('im',im)**

-Display the video frame with the bounded rectangle.

-Display an image.

**cv2.imread('a.jpeg',0)**

-The image should be in the working directory or a full path of image should be given.

-0: is the grayscale.