Face Recognition Attendance Management System

Requirements

Python 3.6

OpenCV for windows

* Source folder

Inside data folder you will see Haarcascade classifiers Xmls used to detect several objects

For example: (Eyes, eyes with sunglasses) and we will use the default one haarcascade\_frontalface\_default and we will copy and paste it into the project folder

* Build folder

**Description**

**Face Detection:**

detector=cv2.CascadeClassifier(harcascadePath)

to capture images using the webcam and process it and detect faces.

faces = detector.detectMultiScale(gray, 1.3, 5)

detect all faces on the current frame

on faces we will detect many faces and put int into rectangle

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

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

**Trainner.yml**

Get all samples that detected using face detection from dataset folder (TrainingImage)

For the trainer to recognize which id number for which face , so we will use os, cv2, PIL and numpy libraries and import them into our python code.

Now we will create the recognizer on TrainImages() using cv2.face\_LBPHFaceRecognizer

And create list of images and store Id and faces

Here we create numpy array pilImage=Image.open(imagePath).convert('L') and we convert the images to grayscale , we use numpy because OpenCV deals only with numpy arrays note the following :

imageNp=np.array(pilImage,'uint8') converting numerical array to numpy array

Id=int(os.path.split(imagePath)[-1].split(".")[1]) splitting list of images

faces=faceCascade.detectMultiScale(gray, 1.2,5)

* scaleFactor: Parameter specifying how much the image size is reduced at each image scale.
* minNeighbors: Parameter specifying how many neighbors each candidate rectangle should have to retain it

Now we have to create the detector

recognizer = cv2.face\_LBPHFaceRecognizer.create()

get the path of all the files in the folder

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

**LBPH Algorithm (Local Binary Patterns Histogram)**

* LBPH is one of the easiest face recognition algorithms.
* It can represent local features in the images.
* It is possible to get great results (mainly in a controlled environment).
* It is robust against monotonic gray scale transformations.
* It is provided by the OpenCV library (Open Source Computer Vision Library).

In computer science, face recognition is basically the task of recognizing a person based on its facial image. It has become very popular in the last two decades, mainly because of the new methods developed and the high quality of the current videos/cameras.

We have two concepts **face recognition** and **face detection**:

**Face Detection**: it has the objective of finding the faces (location and size) in an image and probably extract them to be used by the face recognition algorithm.

**Face Recognition**: with the facial images already extracted, cropped, resized and usually converted to grayscale, the face recognition algorithm is responsible for finding characteristics which best describe the image.

Let’s showing the LBPH method step-by-step.

**Local Binary Pattern** (LBP) is a simple yet very efficient texture operator which labels the pixels of an image by thresholding the neighborhood of each pixel and considers the result as a binary number.

1. **Parameters**: the LBPH uses 4 parameters:

* **Radius**: the radius is used to build the circular local binary pattern and represents the radius around the central pixel. It is usually set to 1.
* **Neighbors**: the number of sample points to build the circular local binary pattern. Keep in mind: the more sample points you include, the higher the computational cost. It is usually set to 8.
* **Grid X**: the number of cells in the horizontal direction. The more cells, the finer the grid, the higher the dimensionality of the resulting feature vector. It is usually set to 8.
* **Grid Y**: the number of cells in the vertical direction. The more cells, the finer the grid, the higher the dimensionality of the resulting feature vector. It is usually set to 8.

Don’t worry about the parameters right now, you will understand them after reading the next steps.

**2.** **Training the Algorithm**: First, we need to train the algorithm. To do so, we need to use a dataset with the facial images of the people we want to recognize. We need to also set an ID (it may be a number or the name of the person) for each image, so the algorithm will use this information to recognize an input image and give you an output. Images of the same person must have the same ID. With the training set already constructed, let’s see the LBPH computational steps.

**3. Applying the LBP operation**: The first computational step of the LBPH is to create an intermediate image that describes the original image in a better way, by highlighting the facial characteristics. To do so, the algorithm uses a concept of a sliding window, based on the parameter’s **radius** and **neighbours**

**4. Extracting the Histograms**: Now, using the image generated in the last step, we can use the **Grid X** and **Grid Y** parameters to divide the image into multiple grids

**5. Performing the face recognition**: In this step, the algorithm is already trained. Each histogram created is used to represent each image from the training dataset. So, given an input image, we perform the steps again for this new image and creates a histogram which represents the image.

* So, to find the image that matches the input image we just need to compare two histograms and return the image with the closest histogram.
* We can use various approaches to compare the histograms (calculate the distance between two histograms), for example: **euclidean distance**, **chi-square**, **absolute value**, etc.
* the algorithm output is the ID from the image with the closest histogram. The algorithm should also return the calculated distance, which can be used as a ‘**confidence**’ measurement. **Note**: don’t be fooled about the ‘confidence’ name, as lower confidences are better because it means the distance between the two histograms is closer.
* We can then use a threshold and the ‘confidence’ to automatically estimate if the algorithm has correctly recognized the image. We can assume that the algorithm has successfully recognized if the confidence is lower than the threshold defined.