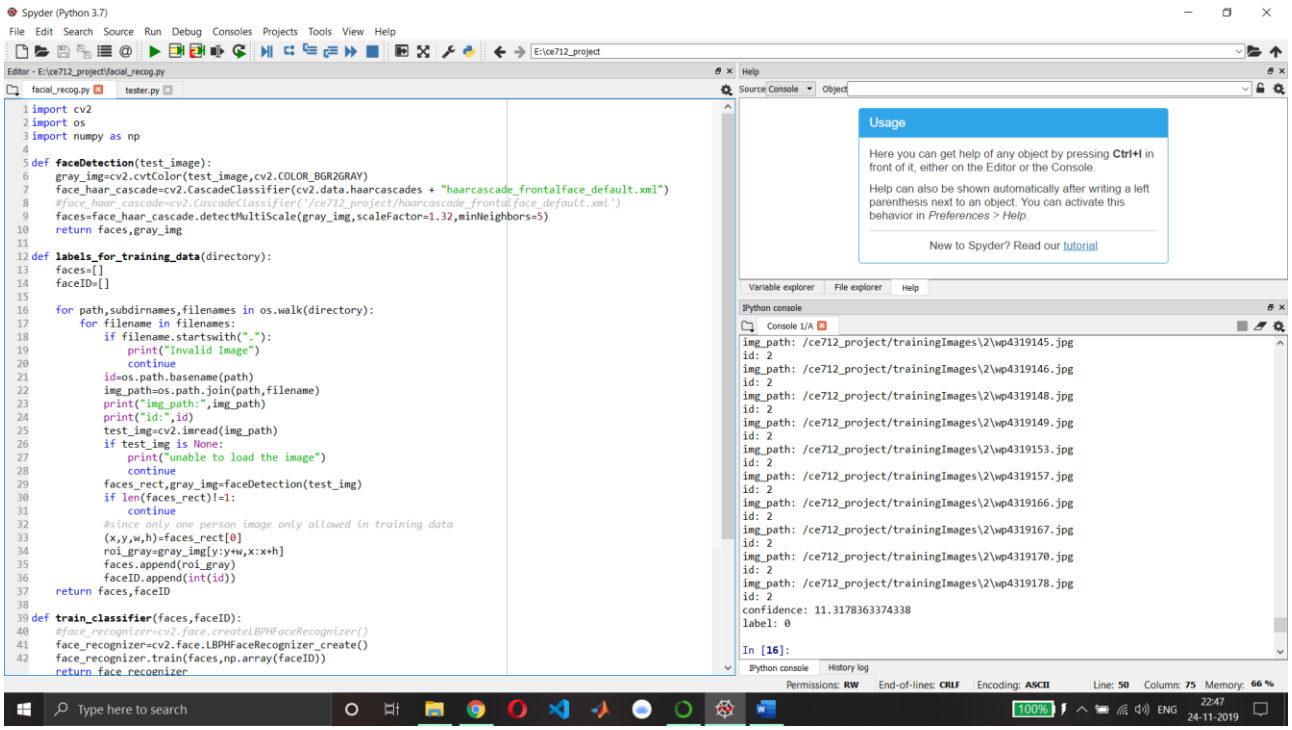
Facial Recognition

Kanishk Jadhav (170040004) | | **CE-712 COURSE PROJECT** | | Jay Thakkar (170040006)

Introduction: Detecting the face in the test image and later recognizing which class image belongs with. The probability of a test image of being related to a particular class in training data is calculated by Machine Learning Techniques or Statistical Methods. We use the method of extracting Histogram of Oriented Gradients (HOG) features and use a ECOC classifier to create a model from the training data in MATLAB and method of Linear Binary Pattern Histogram (LBPH) in the Python

A] Using OpenCV python

Insync with Code and Logic Flow:



For, face_recog.py:

Initially in the ***faceDetection*** function we are passing the image for which we want to find the results. The faceDetection function detects the faces (can be multiple) in the particular test image and returns the array of coordinates of detected faces. In faceDetection function we are also converting the test image into greyscale so that the feature extraction becomes easy in the greyscale image. In this function we are using Haar Cascade classifier which is basically an opencv tool trained to detect the faces with the help of a pre-developed XML file. This Haar Cascade derives its result from superimposing the positive and negative images. Scale factor= 1.32 means decrease the size of image by 32% because our classifier was trained to detect for certain fixed size only, so to increase the accuracy we rescale the image.

Mineighbors tells whether the face detected is true positive or negative on the basis of the neighbouring points.

Next, we construct the ***labels_for_training_data*** function. In this function we pass the directory of the training data. Accordingly using the folder name of classes (for eg 0,1,2) we label the training data and also each class folder must contain a same persons training data. We use the OS library of python for accessing the directories, sub-directories and paths. So, here we label all the images of all the classes using our function. Images in training data which are null or having 2 or more than 2 faces are ignored by us. Also, we call faceDetection function on each and every training image and return the coordinates of detected face and its converted greyscale image. Later we crop the grey image according to the coordinates of face detected and call it is as our region of interest. This new grey region of interest is appended than into "faces" array for each training image and label of each image is appended into "faceID" array for each training image. We return the faces and faceID arrays from this function

In the ***train classifier*** function, we pass the faces and faceID arrays of training data images. Later we use the cv library package functions to train our model. Here we use Local Binary Pattern Histogram (LBPH) to do the statistical analysis of each class (i.e. person). It uses the faces and faceID arrays to train the data using train function of OpenCV library. Here we return facerecognizer variable which later can be use predict the class of the input/test image. In LBPH, a pixel is selected in an image to analyse and surrounding 8 pixels around it are also selected simultaneously. After selecting the block of 9 pixels, the central pixel greyscale value is than compared with the neighbouring pixel and their new values are set accordingly. If the initial neighbour pixel value is greater than central pixel value then the new neighbour pixel value becomes 1 and if it is less than the central pixel value it becomes 0. This neighbouring number are later arranged in single series with staring from top left corner in clockwise. From this we obtain a binary number from above process and later convert this binary number to a decimal number. This decimal number will be the new value of the central pixel. Repeat the process for all the pixels in the image and calculate the histogram of the image. Average out the histogram of all training data sets into a single histogram. Later when test image is passed find the distance between the histogram of test image and training data sets histogram and accordingly find a class.

draw_rect function draws the border around the faces detected in given test or input image

put_text function puts text around the coordinates of the faces detected in given test or input image with custom input text.

FOR test.py:

We import the required packages from the respective libraries. We load an input image which is to be classified. Then we use faceDetection function on the input image to get the coordinates of detected face in input image and the converted grey image.

Next, we label the training data images using the labels_for_training_data function. Then we train our model or classifier using train classifier function

Initially we crop the faces detected in grey images. For each face of test/input image, we compare it with all of our training image faces using the predict function of OpenCV package. This return us the with the label and confidence value for each face in test/input image. Lower the confidence value, higher is it rate of accuracy. Confidence value and actual confidence of predicting are inversely proportional. Then using the predicted label, we find the predicted name from dictionary defined above for names: labels. For confidence value < 40, we discard the criteria for classification into that particular class. We use the function draw_rect and put_text later on the test/input image.

Resizing the image ensures that it is completely visible on our screens. Later imshow we display the image with the labelled name and rectangle box drawn around the face.

Results/Conclusions: Using Linear Binary Pattern Histogram method, the accuracy of classifying or rather recognizing the image increases even there is low lighting and other distortions. It is robust against monotonic grey scale transformations. Also, our model is not reliable when the number of training images are very less. It may start predicting wrongly with high confidence value. Hence, we have detected and recognized the faces given for an input/test image.

B] Using MATLAB (HOG feature):

Explanation of the code (MATLAB): -

The dataset used for this project is Yale Data set for facial recognition (link - http://vision.ucsd.edu/content/yale-face-database)

Functions used: -

1) imageSet() – This function is used to store the image locations of the images in the dataset. imageSet(folder,recursive) gives a vector containing the image sets found through a recursive search of the folder specified in the function parameter. This output vector is used throughout the program to use particular images of the dataset whenever required. This function is a part of the computer visions toolbox.
2) montage() – This function is used to create a montage of a set of images with the location as specified in the parameter. It roughly forms a square arrangement of images.
3) title() – This function is used to give a title to a figure(plot/images) which is given as output.
4) figure – This function is used to create plots or images for outputs.

5) for – This is a type of loop(iterative statement) used to access all the elements of a vector or an array.
6) partition() – We use this function to create a partition of a particular dataset. Here we use it partition our main dataset of images into a training set and a test set, keeping a 80% to 20% split meaning that the first 80% images from the images per person go to the training set and the last 20% images go to the test set for each subject.
7) extractHOGFeatures() - This function returns extracted HOG features from a truecolor or grayscale input image, I. The features are returned in a 1-by-*N* vector, where *N* is the HOG feature length. The returned features encode local shape information from regions within an image. You can use this information for many tasks including classification, detection, and tracking. The histogram of oriented gradients (HOG) is a feature descriptor used in computer vision and image processing for the purpose of object detection. The technique counts occurrences of gradient orientation in localized portions of an image. This method is similar to that of edge orientation histograms, scale-invariant feature transform descriptors, and shape contexts, but differs in that it is computed on a dense grid of uniformly spaced cells and uses overlapping local contrast normalization for improved accuracy.
8) subplot() – This function is used to divide the figure into sections/subplots so that multiple images/outputs can be expressed side by side.
9) zeros() – This function is used to create a vector/array of zeros of the specified length.
10) .Count – This is extension used behind dataset/imageSets to get the length/number of images present in that index of the array.
11) .ImageLocation – Used to access the images the subfiles present in the array(here faceDatabse) from the dataset.
12) .Description – Used to get the description of the particular image ,i.e, which subfile does that particular image belong to.
13) fitcecoc() – This function returns a full, trained, multiclass, error-correcting output codes (ECOC) model using the predictors in table Tbl and the class labels in Tbl.ResponseVarName. fitcecoc uses $K(K-1)/2$ binary support vector machine (SVM) models using the one-versus-one coding design, where *K* is the number of unique class labels (levels). Mdl is a Classification ECOC model.
14) predict() – This function returns the predicted label index and score for the input image. This supports parallel computing using multiple MATLAB® workers. Enable parallel computing using the Computer Vision System Toolbox Preferences dialog.
15) strcmp() – This function is used to compare 2 strings.
16) find() – This function is used to find the index at which lies a particular true value from an array.
17) imshow() – This function is used to display images.

Logic flow of the code: -

In the code we first import out dataset of images through the function imageSet() which creates a vector of the image sets present in the location given in the parameter. Then the images of a single person are displayed and a single image of all the subjects are displayed in montages to give the user an idea of the dataset being used. The dataset is then partitioned to form training and test datasets in a 80/20 split which means that out of the images of each person, 80% are put in the training set and 20% in the test set and this happens for all the people. We then store the Histogram of Oriented Gradient(HOG) features of every image in an array and assign a label to each set pointing to the image set it is a part of. From this training features and labels we create a classifier model using the function 'fitcecoc()' which is a part of the computer vision toolbox of MATLAB. This function uses deep learning approaches to create a classifier model from the training data provided. Thus, having the model we take query images from the test dataset created from the partition. We query every image in the test dataset by creating for loops. We extract the HOG features of the images and we pass the features and classifier model into the function 'predict()' which gives back the predicted image label which it predicts from the model created. Thus we get the image label in the training dataset with which the test image matches thus identifying the subject. We then calculate the index position of that subject's image set in the original dataset vector and display the query image and the matched image.

Result and Conclusion: -

From the output we can see that majority of the test images have been matched correctly to the same subject as that of the query image. From this method of using HOG features to compare the images in MATLAB we get a accuracy rate of 93.333%. The drawbacks of this method are that we can only use greyscale images for the dataset both training and test/query datasets.