Facial Recognition Using LBPH

Final Project

1st Abdul Hadi Khan
Student - CS Department
University of Texas at Dallas
Dallas, USA
abdulhadi.khan@utdallas.edu

2nd Dr. Anurag Nagar
Professor - CS Department
University of Texas at Dallas
Dallas, USA
anurag.nagar@utdallas.edu

3rd Jincheng Li
Teacher's Assistant - CS Department
University of Texas at Dallas
Dallas, USA
jxl170011@utdallas.edu

***Abstract*—Facial recognition is fast becoming a necessity for day to day lives. It is no longer just used by law enforcement officials but is being used by normal people like us. The objective of this document is to provide a detailed description of facial recognition tool that we have built for the course project. It will describe the Local Binary Pattern Histogram model and how it works for facial recognition. Then we will go through the implementation and discuss the algorithm/technique for Local Binary Pattern Histogram. We will describe the data-set that we have used and how to build data-sets for train and test. Finally we will go through the results and accuracy of the model.**

***Index Terms*—machine learning, CNN, algorithm, deep learning, machine translation**

I. Introduction

Facial recognition has quite recently gained a lot of attention among the computer vision fraternity. More and more applications are being built for regular people that involve, in some form or the other, image recognition and facial recognition.
The reason for the rise in interest in facial recognition has to do with the increase in the security concerns for persons. It is fast becoming impossible for people to maintain data security when their data is staggered across multiple applications/domains/geographical boundaries. In an increasingly connected world, hacking has become quite an easy affair. Therefore, there is a need for adding that extra layer of non-binary protection that is still difficult to hack; i.e our physical features.
Whenever we use our cameras to unlock our phones, we are basically using facial recognition to protect our data from being accessed without permission. In order to make this work, especially for mobile devices, learning models need to be robust and less computationally intensive. We have selected "Image and video captioning with deep neural networks" as our project topic. We will be using this to implement a facial recognition application.
There are many good machine learning models for facial recognition such as; Eigenfaces, Fisherfaces, Speed Up Robust Features etc. All these models have different ways of learning how to distinguish and classify images. However, we have decided to use the Local Binary Pattern Histogram model for facial recognition. This is considered to be the most simple to understand. However, it is very efficient in facial recognition where in it label pixels of the image by thresholding the nearest neighbours of each pixel. The flow of our facial happens as follows:

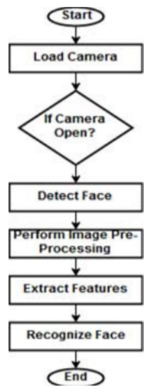


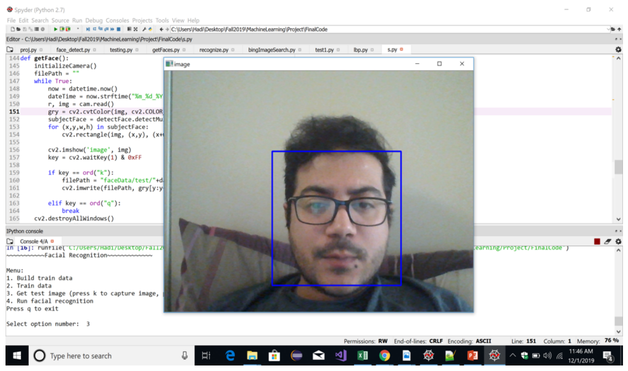
Fig. 1. Facial detection using Haar Cascades

II. Libraries Used

The main libraries that we have used to the project are as follows:
OpenCV 3.2.0, this library was specially useful is interfacing with the camera and capturing images from video streams. Furthermore, we used this library for helping us with the Face Detection part of this project. Since we were pressed on time, we decided to make custom model for Facial recognition but use OpenCV for Facial detection (explained later).
The other python library we used was Numpy 1.11.3, this was mainly used to create array of histograms so that they are easily accessible.
We also used the SciPy library for the Procrustes formula, to basically calculate the similarity or disparity between given data points (mentioned later). There were other libraries as well for procedures that were essential to the saving of the training, test data sets; however, they will not be mentioned in this document.

III. Face Detection

Facial recognition is broken down into two part. The first part being Facial Detection; wherein given an image the model has to decide what part of the image is represented by the face of an actual person. We are using a pre-trained classifier known as, Haar Cascade classifier for detecting a face within the image. It has the ability to detect multiple faces within any given image. The **detectMultiScale** function uses three parameters for detecting faces; image, scaleFactor and minNeighbours. The faces that it detects are represented as a list of coordinates where each face is located in the image.



```
def getFace():
    initializeCamera()
    filePath = ""
    while True:
        now = datetime.now()
        dateTime = now.strftime("%m_%d_%Y
        r, img = cam.read()
        gry = cv2.cvtColor(img, cv2.COLOR
        subjectFace = detectFace.detectMu
        for (x,y,w,h) in subjectFace:
            cv2.rectangle(img, (x,y), (x+

        cv2.imshow('image', img)
        key = cv2.waitKey(1) & 0xFF

        if key == ord("k"):
            filePath = "faceData/test/"+d
            cv2.imwrite(filePath, gry[y:y

        elif key == ord("q"):
            break
    cv2.destroyAllWindows()
```

```
~~~~~~~~~~~~Facial Recognition~~~~~~~~~~~~~

Menu:
1. Build train data
2. Train data
3. Get test image (press k to capture image,
4. Run facial recognition
Press q to exit

Select option number:  3
```

Fig. 2. Facial detection using Haar Cascades

IV. Face Recognition

Local Binary Pattern (LBP) approach was introduced in 1994. LBP labels each pixel of the image by thresholding the neighborhood of each pixel. When we use the LBP with Histograms we are able to represent images in straightforward data vectors. Therefore, LBPH model uses 4 parameters; radius, Neighbors, Grid-X and Grid-Y. Radius is needed to construct the circular local binary pattern and to represent the neighbors around a central pixel that we get. We has set this to 1. Neighbors is the pixels we need to construct the circular local binary pattern. We have set this to 4. Grid X and Grid Y are the amount of divisions in the horizontal and vertical directions, respectively. We have set these to 4 as well. This is because we use the webcam from our laptop which has a resolution limit. Therefore, when we distribute the image in a large amount of grids we found that it will diminish the efficiency of recognition.
Now that we have an understanding of what the basic premise of the model is, we now need data. We have developed a routine that will open the camera and take images of the user until they decide to stop it. The routine will then add all these images to a directory which will be labeled as what the user wants it to be called.

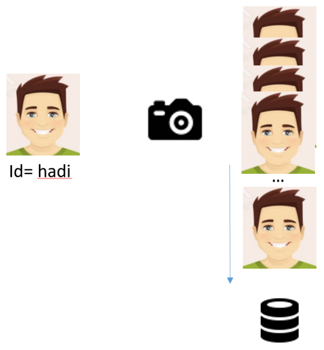


Fig. 3. Get Data for Training

Images of the the same user need to have the same label during the training phase, otherwise the model will not accurately classify the images for the testing phase.
Once we have the training data we need to train our model to get the vectors representing the histograms for each image with the associated labels. The first step of the LBPH model is to construct a staging images that will describe the actual image in a more meaningful manner. This means that it will represent the facial features of the face in the image. In order to do this we have used a sliding window based approach that employs the neighbor and radius parameters that we defined above.

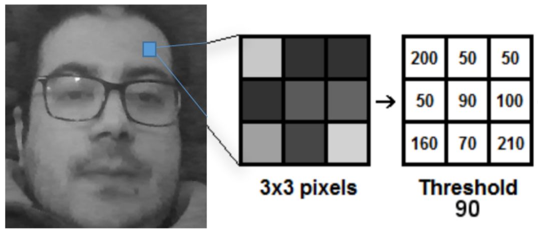


Fig. 4. Example of pixel thresholding

We will now set a new binary value for each neighbor of the threshold value. We set 1 for all the values that are either equal to or higher than the center point of the image. Furthermore, we will set a value of 0 for all values that are then lower than the threshold. Now we will combine all the binary values left-right or right-left to create one binary number and assign that to the central pixel and convert that into a decimal value. Similarly, we will use this central value to calculate the values for its neighbors. The resulting image will give us an image that better represents the features of the original image.

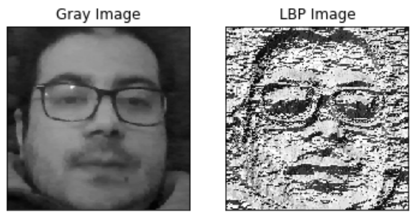


Fig. 5. Original Image to LBP Image

Once we have an image generated with pixel thresholding we will divide it into grids and create histograms. Then all histograms are concatenated together.
These concatenated histogram vectors will be saved in a json file with the associated labels attached as the id. Same image vectors will have the same id attached to them.

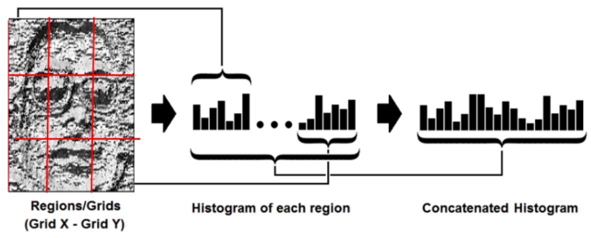


Fig. 6. Histogram construction

Our code will do this for all the label images in the directories and create a JSON file that will house the ”trained data”. Generally, the more data that we have the better the classification is going to be. Once all the training data, the histogram vectors, have been calculated, stored and trained, we will move on to the test phase of the project.

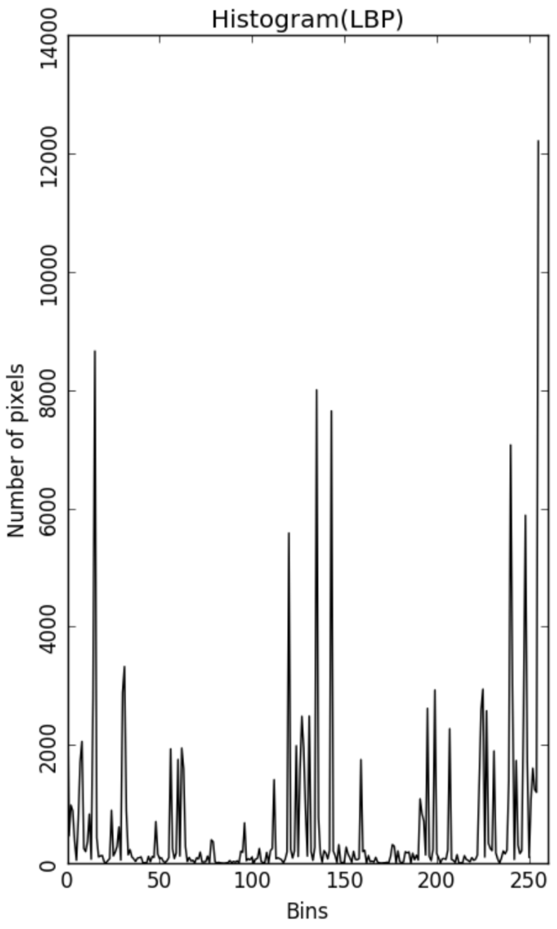


Fig. 7. Graphical Representation of Histograms

V. Classification Testing

The program gives users the option of getting fresh images as the test images one by one. The user will look at the screen and decide on whether they want the picture taken or not. Once the user takes the picture, the input image will go through the same process as the images in the training data set. Each image will go through pre-processing to detect the face through the Haar Cascade model. Once the face has been captured, the program will apply LBPH on the resulting face and create a resulting image of characteristic features. Then the program will calculate the histogram vector from the resultant LBP image. After its histogram has been calculated we will compare it with those in the trained data set.
The algorithm will return the calculated distance which will be used to define confidence. The lower the confidence value the closer the match. Initially we decided on using the Euclidean distance formula, given below:

$$D = \sqrt{\sum_{i=1}^{n}(hist1_i - hist2_i)^2}$$

Fig. 8. Euclidean Distance Formula

Later, we discovered the Procrustes analysis formula. It is one of the comparison metrics provided in the SciPy libraries in Python. Procrustes analysis is a similarity test done between two data sets.
Each input matrix is a set of points or vectors (the rows of the matrix). The dimension of the space is the number of columns of each matrix.
The equation uses vectors that are exactly of the same length. Our histogram data is precisely the same as the requirement for Procrustes. Each image will output the same length vector, depending on the number of grids numbers are selected in during training data and preparing the test data. The equation of Procrustes is as follows:

$$M^2 = \sum (data1 - data2)^2$$

Fig. 9. Procrustes Formula

The two parameters; namely, data1 and data2, are n-element array each. They represent the histogram vectors of the train and test images. The function return three values;
Firstly a standardized version of data1 array.
Secondly, the best fit representation of data2 array for data1 array.
Thirdly, the value of disparity (float) which is the measure of difference between the two input arrays, represented as M^2. The test image histogram will be matched against all the histograms from the training data. It will keep updating the best fit variable to reflect the closest match from the train data and capture the label. Once it gets the best match it will return the label and assign it to our test image. Thus, completing the classification. In case the best match is not found and the disparity value is greater than 0.1 or hundred percent, the model will classify it as an unknown image.

VI. Dataset

We on using using Georgia Tech face database. It contains the images of 50 people. For each subject on average, there are 8-9 images present. All people in the database are represented by 15 color JPEG images with cluttered background taken at resolution 640x480 pixels. The average size of the faces in these images is 150x150 pixels. The label files contain four integers that describe the coordinates of the face rectangles and a string (s1, ..., s50) indicating the identity of the face. Assuming that the upper left corner of the image has the coordinates (0,0), the numbers correspond to the x_left, y_top, x_right, y_bottom coordinates of the face rectangle.
The label files are named as follows:
lab001-lab015 correspond to files 01.jpg,...,15.jpg in s01
lab021-lab035 correspond to files 01.jpg,...,15.jpg in s02
....
However, we realized that using this data set and then doing a train and test split from the images available was not the most optimum way of training our model. This is also in some instances lead to over-fitting since most of the data sets were typically of the same instance and orientation. Therefore, in some situation it meant that after train, test split, we often had either the exact same of fairly similar images in either bucket. However, when we introduced pictures of the same subjects from different sources, the model was not classifying it correctly. Therefore, we decided to create data sets on our own. For us to be able to that easily was a little challenging. Firstly, we tried manually taking pictures of subjects. This was proving to be quite cumbersome.
Then we decided on using the Azure Image Search API. This meant that we could type in the name of any famous celebrity and get images of them. However, an issue with that was that the image was not always reflective of the celebrity in their natural state. There were a lot of instances where the celebrity would be under heavy costume and the API would still allow that to be mined. This was also throwing off classification as there was too much noise in the data to be able to classify correctly.
Finally, we decided on using the OpenCV library and getting the images of live subjects. This allowed us to get as much data as we could under many different circumstances such as lighting, proximity to the camera and different expressions. Furthermore, we did not have to worry about over fitting since we kept these images explicitly for training. Once the training was competed we would use OpenCV again to get a test image. This worked perfectly, we tried it on three data sets, hadi, faraz and bilal.

VII. Programming Language

We have implemented this project in Python programming language. This was decided on the basis of ease of programming complicated mathematical models. Furthermore, there is a huge community for Python which helps in getting the most up-to date information regarding packages and libraries available.

VIII. Conclusion and Future Scope

We have decided to make this into an android application. Furthermore, rather than just classifying still images we have decided to add classification of live stream data as well. This is allow us perform facial recognition on moving subjects further increasing the worth of this model.
The conclusions that we drew from this project was that this model has some room for improvements. Certain conditions have to be kept constant, for now, such as the still images as opposed to moving images for testing. However, this model showed a lot of promise as well. It was very helpful in classifying subjects faces that had been turned on the axis. While other classifiers were unable to perform well, LBPH was quite accurate in this regard.

IX. Team Members

Abdul Hadi Khan - axk180114

X. References

References

[1] https://towardsdatascience.com/face-recognition-how-lbph-works-90ec258c3d6b.

[2] https://docs.scipy.org/doc/scipy/reference/generated/scipy.spatial.procrustes

[3] https://www.instructables.com/id/Real-time-Face-Recognition-an-End-to-end-Project

[4] https://towardsdatascience.com/face-detection-in-2-minutes-using-opencv-python-90f89d7c0f81.

[5] https://docs.opencv.org/3.4/df/d25/classcv_1_1face_1_1LBPHFaceRecognizer.html.

[6] https://en.wikipedia.org/wiki/Local_binary_patterns