Summer Internship

Under the guidance of

Prof. V. M. Gadre

Eigenfaces for Face Recognition using Python and OpenCV

May 2019 - July 2019

by

Shruti S. Patkar

170907094

Department of Electronics and Communication Engineering

Manipal Institute of Technology, Manipal, Karnataka

1 Acknowledgements

For the summer internship which lasted from May 2019 to July 2019, I wish to express my sincere thanks to Prof. V. M. Gadre for giving me this golden opportunity to work on the biometric detection project, and intern at IIT Bombay, by which I gained many skills. I also wish to deeply thank our project mentor Mr.Sagar G. Sandogkar for his time, and guidance. I am also thankful to the other members of the project namely Priyanka Ojha, Sanjay Shekhavat and Vishaldeep Singh Deora from CTAE Udaipur for their support. Last but not the least, I would like to thank the Electrical Engineering department of IIT Bombay for their facilities and use of their labs.

2 Introduction

- Eigenfaces is probably one of the simplest face recognition methods but is also a rather old technique, the idea of using eigenfaces for recognition was developed by Sirovich and Kirby and used by Matthew Turk and Alex Pentland in face classification. It is simple and works quite well, and this simplicity also makes it a good way to understand how face recognition/dimensionality reduction etc works.

- This approach revolves around linear and matrix algebra. To recap the definition of eigenvalues and eigenvectors. λ is said to be an eigenvalue of matrix A if

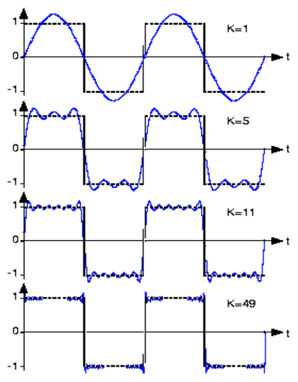
$$A\underline{x} = \lambda\underline{x}$$

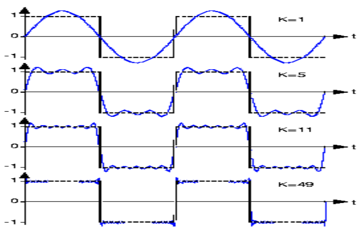
  for $\underline{x} \neq \underline{0}$. and such $\underline{x}$ is said to be an eigenvector of A corresponding to the eigenvalue λ

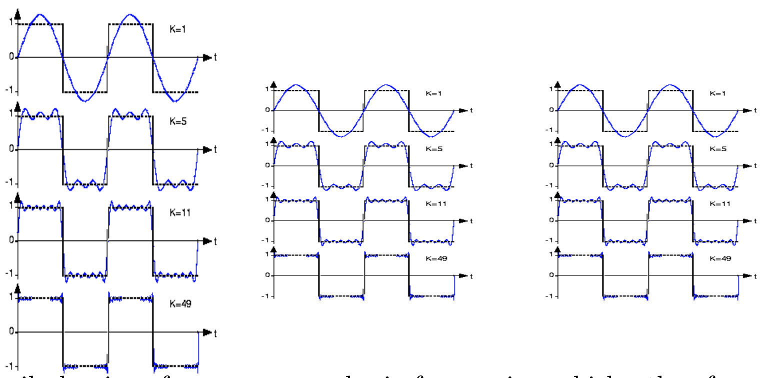
- The project has been coded in python with the use of OpenCV (Computer Vision) and Numpy modules in implementation, and the OS (Operating Systems) module to search for files.

- Representing a signal as a linear combination of some basis functions is a great idea that is evident in Fourier Series and Wavelet Transforms of various types.

- An example that illustrates the Fourier series is:





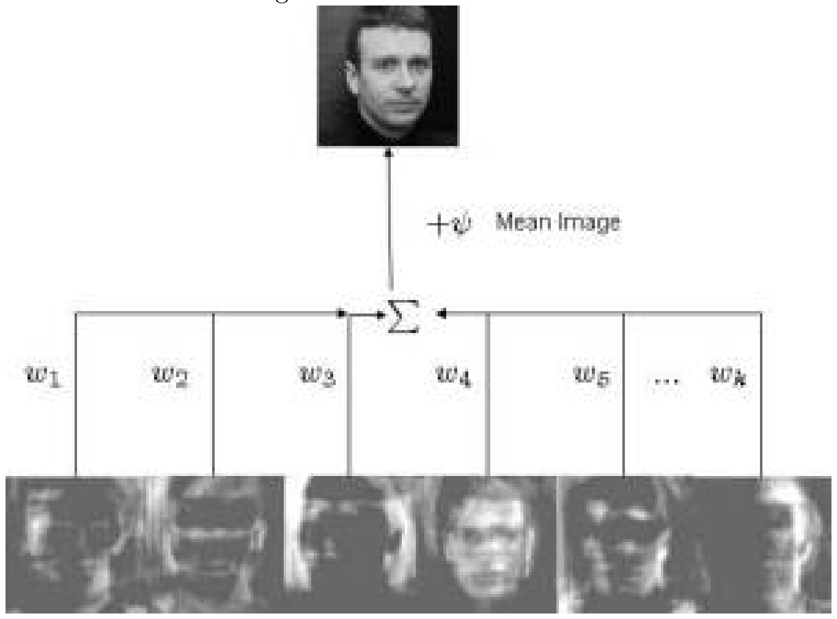


Similarly eigen-faces are some basis faces using which other faces can be expressed (hopefully well-enough / approximately).

$$\Phi_i = \sum_{j=1}^{K} w_j \mathbf{u_j}$$

Notation being used here is Φ_i to represent i^{th} face. Remember that we subtract mean from it so as to .... The coefficients of linear combinations are w_j for $j = 1 \dots K$. Finally $\mathbf{u_j}$ are the vectors representing eigenfaces.

Eigenfaces are obtained from training data. The weights (coefficients in linear combination) can be used as a sort of low dimensional representation. This can be useful for recognition.

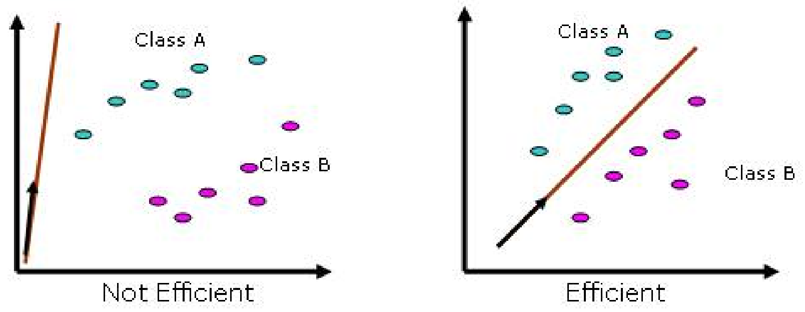


Here in the above example taken from Shubhendu Trivedi's blog, a face from training database was expressed as a linear combination of eigenfaces plus the mean face. Please note that in the figure the ghost like basis images (also called as eigenfaces, we'll see why they are called so) are not in order of their importance.

3 Prerequisites

We assume faces are in proper position and angle.

The principal components (or eigenvectors) are basically directions in which it is data has significant variation.



There are as many eigen-faces as the the number of images in the training set. We approximate using a small number of most significant eigen-faces.

Each image is $N \times N$ matrix and is represented using column vector with $N^2 \times 1$ entries.

4 Algorithm

- 1. Iterate over the folders/files in the image database and choose and build M training images $I_1, I_2 \ldots I_M$. The images should be appropriately cropped.

- 2. For $i = 1, \ldots, K$, training image I_i is represented using column vector Γ_i.

$$I_i = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1N} \\ a_{21} & a_{22} & \cdots & a_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ a_{N1} & a_{N2} & \cdots & a_{NN} \end{bmatrix}_{N \times N} \xrightarrow{\text{concatenation}} \begin{bmatrix} a_{11} \\ \vdots \\ a_{1N} \\ \vdots \\ a_{2N} \\ \vdots \\ a_{NN} \end{bmatrix}_{N^2 \times 1} = \Gamma_i$$

- 3. Compute mean face Ψ.

$$\Psi = \frac{1}{M} \sum_{i=1}^{M} \Gamma_i$$

- 4. By subtracting the mean face, we remove common information from the face vectors. This gives vectors Φ_i for $i = 1, \ldots, K$.

$$\Phi_i = \Gamma_i - \Psi$$

- 5. Covariance matrix C is then calculated using numpy library methods:

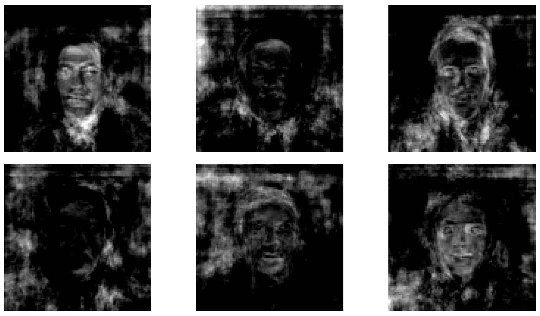
$$C = AA^T, \text{where } A = [\Phi_1, \Phi_2 \ldots \Phi_M]$$

- 6. Instead of eigenvectors $\mathbf{u_i}$ of AA^T, eigenvectors $\mathbf{v_i}$ of matrix A^TA are calculated.

  It can be shown that $\mathbf{u_i} = A\mathbf{v_i}$. Using this idea we get eigenvectors of AA^T corresponding to M most significant eigenvalues. Note that $M \ll N^2$.

  Also (verify / normalize) $||\mathbf{u_i}|| = 1$.

- These are some of the faces I got after training on the LFW dataset



- 7. Choose best K among these eigenvectors using some suitable criterion.

5 Explanation of Python Code

We import several python packages

- ”os” for file/folder system exploration and manipulation
- ”numpy” for matrix and linear algebra
- ”cv2” for opencv
- ”PIL” for python imaging library
- ”matplotlib” for plotting

- The variable ”path_name” stores the name of the folder under which the image dataset is stored.
- Using ”os.listdir” command the subfolders of ”data” folder are listed and
- then using ”sorted” command, we obtain a sorted list of image folders. It is stored in the python list ”folders”

```
import os
import numpy as np
import cv2
from PIL import Image
from matplotlib import pyplot as plt
from matplotlib.image import imread

path_name = "data\lfw"
folders = sorted(os.listdir(path_name))
```

- Sorting faces into test and train dictionaries
- The keys of those dictionaries are names of the people i.e. folders.

```
face_train_dict = {}
face_test_dict = {}
```

- For each folder inside folders (i.e. the sorted list of folders),
- we list the names of the files, and (using os.path.join)
- prepare the absolute path-names for the files in the folder.
- First half of this list of files are associated with the folder in the training dictionary.
- Second half of this list of files are associated with the folder in the testing dictionary.

```
for folder in folders:
    files = [os.path.join(path_name , folder, path) \
            for path in \
            os.listdir(os.path.join(path_name , folder))]
    if len(files) == 1:
        face_test_dict[folder] = files
    else:
        face_train_dict[folder] = files[:int(len(files)/2)]
        face_test_dict[folder] = files[int(len(files)/2):]
```

- width and height are defined as 100,
- so image will be 100 X 100
- We resize and equalize the image histogram to make the images workable,
- also convert them to greyscale

```
wh = 100
for face in face_train_dict:
    for i in range(len(face_train_dict[face])):
        img_file =  face_train_dict[face][i]
        img = cv2.imread(img_file)
        img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
        img = cv2.resize(img, (wh, wh), \
                  interpolation = cv2.INTER_AREA)
        img = cv2.equalizeHist(img)
        face_train_dict[face][i] = img
        #cv2.imshow('image', img)
```

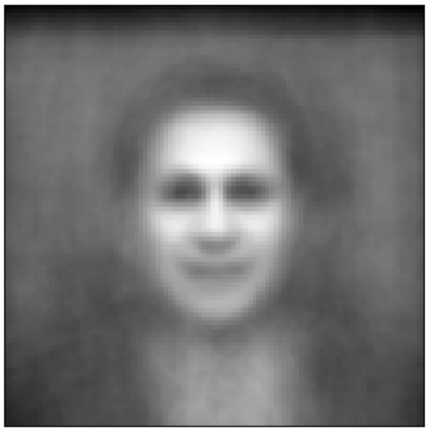
- For each of the faces used for training,
- We use numpy methods for obtaining the 2-d array data of the 2-d image,
- and then use the "flatten" method on this numpy 2-d array object to obtain 1-d vector.
- We collect all these 1-d vectors in the list "face_vec_list".

```
face_vec_list = []

for face in face_train_dict:
    for i in range(len(face_train_dict[face])):
        face_eq = face_train_dict[face][i]
        face_eq = np.array(face_eq)
        #print (np.shape(face_eq))
        face_vec = face_eq.flatten()
        #print (face_vec)
        face_vec_list.append(face_vec)
```

- Next we convert the list face_vec_list into a numpy array.
- It will be a 2-d array. This can be checked using "shape" method in numpy.

```
fva = np.array(face_vec_list)
orig_fva = np.array( fva )
s = np.shape( fva)
#print (s)
```

- Now we compute average face.
- First initialize it to all zeros, using "zeros" method of numpy.
- We iterate over all the training faces of all people
- and add them all up, using "numpy.add" method
- and then using "numpy.divide" method compute the average.



-

```
lens  = 0
average_face_vector = np.zeros(10000)
for i in face_train_dict:
    for j in face_train_dict[i]:
        s = j.flatten()
        lens = lens + 1
        average_face_vector = np.add(average_face_vector,s)
average_face_vector = \
        np.divide(average_face_vector,lens).flatten()
```

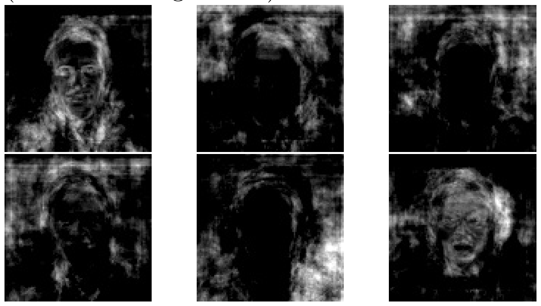
- The following can be un-commented while debugging.

```
#print("Size of average face \
#                 vector:",average_face_vector.shape)
#print("Average face vector:",average_face_vector)
#print("Average face:")
#plt.imshow(average_face_vector.reshape(100, 100), \
#        cmap='gray')
#plotting the average face generated
#plt.tick_params(labelleft='off', labelbottom='off', \
#  bottom='off',top='off',right='off',left='off', \  #
#  which='both')
#removing the labels from the plot
#plt.show()
```

- Eigenfaces are calculated with the help of the linalg library provided by numpy

```
mean_fva = fva.mean(axis = 0)

fva = fva - mean_fva
fvat = fva.T
#t = np.shape(fvat)
#print(t)
Cov_mat = np.dot(fva, fvat)
a = np.shape(Cov_mat)
#print (a)
print ("done")

eigval, eigvec=np.linalg.eig(Cov_mat)

#print ("eigvals are " , eigval)
list_eigvals = []
for x in np.nditer(eigval):
    x_abs = abs(x)
    list_eigvals.append(x_abs)
eigvals = np.array(list_eigvals)
ind  = np.argsort(eigvals)
sort_eigvals = np.sort(eigvals)
eigveclist = eigvec.tolist()
#print (eigvec)
a = ind[-1]
eigfac = np.array(eigveclist[a])
```

(Some more eigenfaces)



6 Haar Cascade based Face Detection

```
import numpy as np
import cv2
import matplotlib.pyplot as plt
import os
import cv2
haar_cascade_face = cv2.CascadeClassifier(  \
        'haarcascades/haarcascade_frontalface_alt.xml')

image_filenames = {}
for path,dirs,files in os.walk("data_small"):
    for file in files :
        if ( file.endswith('jpg') ) :
            #image_filenames.append( path+ "\\" +file )
            tokens = path.split("\\")
            key = tokens[-1]
            image_filenames[key]= path + "\\" +file
print( image_filenames )
#exit()


images = {}
cropped_faces = []

def convertToRGB(image):
    return cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

for name in image_filenames :
    images[name] = cv2.imread( image_filenames[name] )
    test_image = cv2.imread(image_filenames[name])

    # Converting to grayscale as opencv expects
    # detector takes in input gray scale images
    test_image_gray = cv2.cvtColor(test_image,  \
            cv2.COLOR_BGR2GRAY)

    # Displaying grayscale image
    cv2.waitKey()
    #cv2.imshow( 'person',test_image_gray)

    faces_rects = \
        haar_cascade_face.detectMultiScale(   \
            test_image_gray, scaleFactor = 1.2,  \
                minNeighbors = 5);

    # Let us print the no. of faces found
    print('Faces found: ', len(faces_rects))

# Our next step is to loop over all the co-ordinates
# it returned and draw rectangles around them using
# Open CV.We will be drawing a green rectangle with thicknessof 1
# In[148]:

    #cropped_faces = []
    for (x,y,w,h) in faces_rects:
        a = x//1 + w//2 -50
        b = y//1 + h//2 - 50
        c = x//1 + w//2 + 50
        d = y//1 + h//2 + 50
        cv2.rectangle(test_image, (a, b),  \
                ( c, d) , (1, 1, 1), 1)
        test_image = convertToRGB(test_image)
        crop_img = test_image[a:c, b:d]
        #cv2.imshow("cropped_face", crop_img)
        cropped_faces.append(crop_img)

    test_image = convertToRGB(test_image)
    #cv2.imshow("person", test_image)

equ = []

print (len( cropped_faces ))

face_vec_list = []

for face in cropped_faces :
    face_gray = cv2.cvtColor(face, cv2.COLOR_BGR2GRAY)
    dimensions = face_gray.shape
    face_eq = cv2.equalizeHist(face_gray)
    face_vec = face_eq.flatten()
    face_list_vec = list(face_vec)
    face_vec_list.append( face_list_vec)

fva = np.array(face_vec_list)
orig_fva = np.array( fva )
s = np.shape( fva)
print (s)
mean_fva = fva.mean(axis = 0)


fva = fva - mean_fva
fvat = fva.T
#t = np.shape(fvat)
#print(t)
Cov_mat = np.dot(fva, fvat)
a = np.shape(Cov_mat)
print (a)

eigval, eigvec=np.linalg.eig(Cov_mat)

print ("eigvals are " , eigval)
input( "proceed ? press any key " )

list_eigvals = []

for x in np.nditer(eigval):
    x_abs = abs(x)
    list_eigvals.append(x_abs)

eigvals = np.array(list_eigvals)

ind  = np.argsort(eigvals)

sort_eigvals = np.sort(eigvals)

eigveclist = eigvec.tolist()
print (eigvec)
a = ind[-1]
eigfac = np.array(eigveclist[a])

eigenfaces = []



for x in range (-2,-12,-1):
    a=ind[x]
    eigfac = np.vstack((eigfac,np.array(eigveclist[a])))

eigfac_correct = eigfac.T

#print (eigfac_correct)

eigenfac = np.dot(fvat,eigfac_correct)

for i in range( eigenfac.shape[1] )  :
    nrm =  np.linalg.norm ( eigenfac[:,i] )
    print ("norm is ", nrm )
    eigenfac[ : ,i ] = eigenfac[:,i] /  nrm


s = np.shape(eigenfac)
#print(s)
print ("Bark")
#work_eig = eigenfac.T

implist = []
print ("dog")



for g in range (0 , s[1]):
    f = eigenfac[:,g]
    print ( "l2-norm of  f is ", np.linalg.norm( f ) )

    m = f.reshape(100,100)
    implist.append(m)
    print ("Bear")

for f in face_vec_list :
    f_approx = np.dot ( np.array( f)-mean_fva , \
            np.dot( eigenfac , eigenfac.T ) ) + mean_fva

    cv2.imshow( "image" , (np.array(f) ).reshape(100,100) )
    cv2.waitKey()

    cv2.imshow( "image" , \
        ((np.array(f_approx ))/255.0).reshape(100,100))
    cv2.waitKey()
    yesorno = input("proceed (y/n) ? ")
    if ( yesorno == 'n' ) :
        exit()
```

```
for i in range (s[1]):
    cv2.imshow("image", implist[i])
    cv2.waitKey()
```

- At first glance, this program would appear to be quite similar to the one we saw previously.
- This code makes use of haar cascade for face detction, so it is quite useful if one does not know if the picture that's being tested or trained is a face or not. Our cascade classifier could easily discern the two.
- It should be duly noted that we crop faces from the images after they're detected, as opposed to simply resizing them with OpenCV as we did in the other example. This could lead to a number of issues, such as centering. So we have tried to make sure that the 100 x 100 image covers as much of the face as possible and is centred, as you can see in this snippet. (Original imagr is 250 x 250)

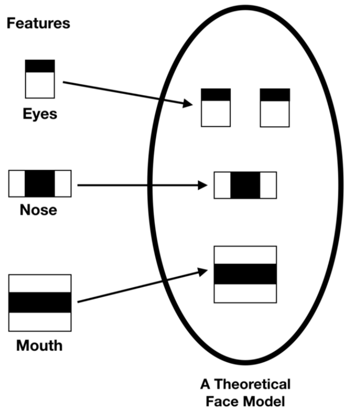
```
    for (x,y,w,h) in faces_rects:
        a = x//1 + w//2 -50
        b = y//1 + h//2 - 50
        c = x//1 + w//2 + 50
        d = y//1 + h//2 + 50
        cv2.rectangle(test_image, (a, b),  \
                ( c, d) , (1, 1, 1), 1)
        test_image = convertToRGB(test_image)
        crop_img = test_image[a:c, b:d]
        #cv2.imshow("cropped_face", crop_img)
        cropped_faces.append(crop_img)

    test_image = convertToRGB(test_image)
```

- The OS module is a very versatile module for traversing the filescape and directories and so on. We have made use of it in both of our examples, however in different ways. The command we use here is os.walk, which gives us a tuple with files, folders and directories. Although for our purpose, os.sorted would have been better (Which is used in the previous code) as all the images were on the same level and were kept in their respective folder.

```
for path,dirs,files in os.walk("data_small"):
    for file in files :
        if ( file.endswith('jpg') ) :
            #image_filenames.append( path+ "\\" +file )
            tokens = path.split("\\")
            key = tokens[-1]
            image_filenames[key]= path + "\\" +file
print( image_filenames )
```

- Object Detection using Haar feature-based cascade classifiers is an effective object detection method proposed by Paul Viola and Michael Jones in their paper, "Rapid Object Detection using a Boosted Cascade of Simple Features" in 2001.
- It is a machine learning based approach where a cascade function is trained from a lot of positive and negative images. It is then used to detect objects in other images.
- OpenCV uses two types of face detection classifiers, LBP (Local Binary Pattern) and Haar Cascades. We shall be speaking of the latter.
- A Haar Cascade is based on "Haar Wavelets" which is defined as:

  A sequence of rescaled "square-shaped" functions which together form a wavelet family or basis.

  Initially, the algorithm needs a lot of positive images (faces) and negative images (without faces) to train the classifier. Then we need to extract features from it. For this, Haar features shown in the below image are used. Each feature is a single value obtained by subtracting sum of pixels under the white rectangle from sum of pixels under the black rectangle.
- 

- In the above picture, we can see some weak classifiers grouped together. (The final classifier is a weighted sum of these weak classifiers. It is called weak because it alone can't classify the image, but together with others forms a strong classifier.)
- We have used the detectMultiscale module from OpenCV. This creates a rectangle with coordinates (x,y,w,h) around the face detected in the image, contains code parameters that are important to consider.

- scaleFactor: The value indicates how much the image size is reduced at each image scale. A lower value uses a smaller step for downscaling. This allows the algorithm to detect the face. It has a value of x.y, where x and y are arbitrary values you can set.
- minNeighbors: This parameter specifies how many “neighbors” each candidate rectangle should have. A higher value results in less detections but it detects higher quality in an image. You can use a value of X that specifies a finite number.
- minSize: The minimum object size. By default it is (30,30). The smaller the face in the image, it is best to adjust the minSize value lower.
-
```
haar_cascade_face.detectMultiScale(
            test_image_gray, scaleFactor = 1.2,
                minNeighbors = 5);
```
- (Snippet from the code as an example for Haar frontal face classifier)

7 Adaboost

- AdaBoost, short for Adaptive Boosting, is a machine learning meta-algorithm formulated by Yoav Freund and Robert Schapire
- It focuses on classification problems and aims to convert a set of weak classifiers into a strong one. The final equation for classification can be represented as
- $F(x) = sign(\sum_{m=1}^{M} \theta_m f_m(x)),$

$$F(x) = sign(\sum_{m=1}^{M} \theta_m f_m(x))$$

- where f_m stands for the m^{th} weak classifier and θ_m is the corresponding weight. It is exactly the weighted combination of M weak classifiers. The whole procedure of the AdaBoost algorithm can be summarized as follow.
- Given a data set containing n points, where -1 denotes the negative class while 1 represents the positive one. Initialize the weight for each data point as:
- $w(x_i, y_i) = \frac{1}{n}, i = 1, ..., n.$

$$w(x_i, y_i) = \frac{1}{n}, \quad i = 1, \ldots, n$$

- For iteration m=1,. . . ,M:
- Fit weak classifiers to the data set and select the one with the lowest weighted classification error:
- Calculate the weight for the m^{th} weak classifier:
- For any classifier with accuracy higher than 50 percent, the weight is positive. The more accurate the classifier, the larger the weight. While for the classifer with less than 50 percent accuracy, the weight is negative. It means that we combine its prediction by flipping the sign. For example, we can turn a classifier with 40 percent accuracy into 60 percent accuracy by flipping the sign of the prediction. Thus even the classifier performs worse than random guessing, it still contributes to the final prediction. We only don't want any classifier with exact 50 percent accuracy, which doesn't add any information and thus contributes nothing to the final prediction.
- Update the weight for each data point as:
- $$w_{m+1}(x_i, y_i) = \frac{w_m(x_i, y_i)exp[-\theta_m y_i f_m(x_i)]}{Z_m},$$

  $$w_{m+1}(x_i, y_i) = \frac{w_m(x_i, y_i)exp(-\theta_m y_i f_m(x_i))}{Z_m}$$

  where Z_m is a normalization factor that ensures the sum of all instance weights is equal to 1. If a misclassified case is from a positive weighted classifier, the "exp" term in the numerator would be always larger than 1 (y*f is always -1, thetam is positive). Thus misclassified cases would be updated with larger weights after an iteration. The same logic applies to the negative weighted classifiers. The only difference is that the original correct classifications would become misclassifications after flipping the sign. After M iteration we can get the final prediction by summing up the weighted prediction of each classifier.

8 Conclusion

The Eigenfaces algorithm proposed by Viola-Jones has been successfully implemented on the LFW dataset. We have also used Haar Cascade detection for faces. But it is evident based on the outcome that low-light conditions, or lighting conditions in general, and the angle of the camera can play havoc with the results. This claim is also supported by most literature.This is where scattering invariant networks come into the picture. Yet, there is much scope for improvement, which can be done in the future.

9 Future Aspects

Biometrics is promising and will play an even more important role in the society than it does today because they offer an approach for identity identification based on unique, reliable and stable personal characteristics which include improving the convenience and efficiency of routine access transactions, reducing fraud, and enhancing public safety and national security. Questions persist, however, about the effectiveness of biometric systems as security or surveillance mechanisms, their usability and manageability, appropriateness in widely varying contexts, social impacts, effects on privacy, and legal and policy implications.

10 Appendix

1 https://github.com/Ei5baer/Eigenfaces

2 https://github.com/Ei5baer/EigenfacesHaarcascades

11 Bibliography

1 P. Viola and M. Jones. ”Rapid object detection using a boosted cascade of simple features,”. Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001.

2 Matthew A. Turk and Alex P. Pentland. ”Face Recognition Using Eigenfaces”. Proceedings. 1991 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, Maui, HI, USA, 1991, pp. 586-591. doi: 10.1109/CVPR.1991.139758.

3 Joan Bruna and Stephane Mallat. ”Invariant Scattering Convolution Networks”. IEEE Trans. on PAMI, vol. 35, no. 8, pp. 1872-1886, Aug. 2013. https://arxiv.org/abs/1203.1513v2.

4 S. Minaee, A. Abdolrashidi and Y. Wang. ”Face recognition using scattering convolutional network”. 2017 IEEE Signal Processing in Medicine and Biology Symposium (SPMB), Philadelphia, PA, 2017, pp. 1-6. doi: 10.1109/SPMB.2017.8257025

5 Bruna J, Ph.D. thesis. ”Scattering Representations for Recognition”. École Polytech-nique, Palaiseau, France,Nov. 2012. https://www.di.ens.fr/data/publications/papers/phd_joan.pdf

6 “Adv. Digital Signal Processing - Multirate and wavelets”, Prof. Vikram M. Gadre, Dept of Electrical Engineering, IIT Bombay https://nptel.ac.in/courses/117101001/

7 Deep Learning Haar Cascade http://www.willberger.org/cascade-haar-explained/

8 Face Recognition with Eigenfaces https://pythonmachinelearning.pro/face-recognition-with-eigenfaces/

10 Face Detection using Haar Cascades https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_objdetect/py_face_detection/py_face_detection.html

11 Working of the Haar Cascade Classifier http://www.cs.utexas.edu/~grauman/courses/spring2011/slides/lecture21_facedet.pdf

12 L. Sifre, J. Anden, ScatNet https://www.di.ens.fr/data/software/scatnet/quickstart-image/