DIP PROJECT REPORT

**FACIAL EMOTION RECOGNITION**

**GROUP 21**

**Group Members’ Details**

Hardik Bhati                 19UCS024.

Abhishek Tiwari         19UCS028.

Mansi Agarwal            19UCS091.

Rishabh Sahu               19UCS129.

Hemang Goyal             19UCS193.

**INTRODUCTION**

Through this project the primary objective of our group is to create a face expression detection application which can pick up an image, detect the facial expression of the face present in it and display the detected face emotion as output.

**OBJECTIVE**

Through this project our group has demonstrated the use of Digital Image Processing concepts like Image Denoising using Median Filter and Contrast Stretching by manipulating intensity values of the image, and the ability to use OpenCV library in python. We also used the matplotlib library to display graphical information related to the images. This project also uses some concepts of machine learning.

**SUMMARY OF DIGITAL IMAGE PROCESSING TECHNIQUES USED IN THE PROJECT**

We have used two techniques of digital image processing in our project:

1. Image Denoising using Median filter

2. Contrast Stretching of Image by manipulation of intensity values.

**SEQUENCE OF IMPLEMENTATION OF THE PROJECT**

Our application works in a series of steps that are enlisted below:

1.Image Selection

2 Image Enhancement

3.Face Detection

4. Facial Emotion Recognition

**IMAGE SELECTION**

The application helps the user to select some locally stored image with the help of image picker window. After the user selects the image through this image picker window the application gets the path of the locally stored image on which it will detect the face expression.

The file dialog that appears and helps user to select the image is imported from python’s tkinter library. Image\_c variable stores the image stored in a NumPy array.

**Code snippets related to image selection**

import tkinter

from tkinter import filedialog

image\_c=image\_picker()

def image\_picker():

#This place will hold code to pick up locally stored image file, at least. Png and .jpeg should be compatible

#Call the file picker to pick up an image file

root = tkinter.Tk()

root.withdraw()

#Get image path in text form

img\_path = filedialog.askopenfilename()

#Call imread function with the image location obtained above to apply image processing on it

img1=cv2.imread(img\_path)

#Create resizable window to display the image

cv2.namedWindow("Window1", cv2.WINDOW\_NORMAL)

cv2. setWindowProperty ('Window1', cv2. WND\_PROP\_FULLSCREEN, cv2. WINDOW\_FULLSCREEN)

#Display image in the resizable window defined above to display unmodified image

cv2.imshow("Window1", img1)

#Code to close the windows

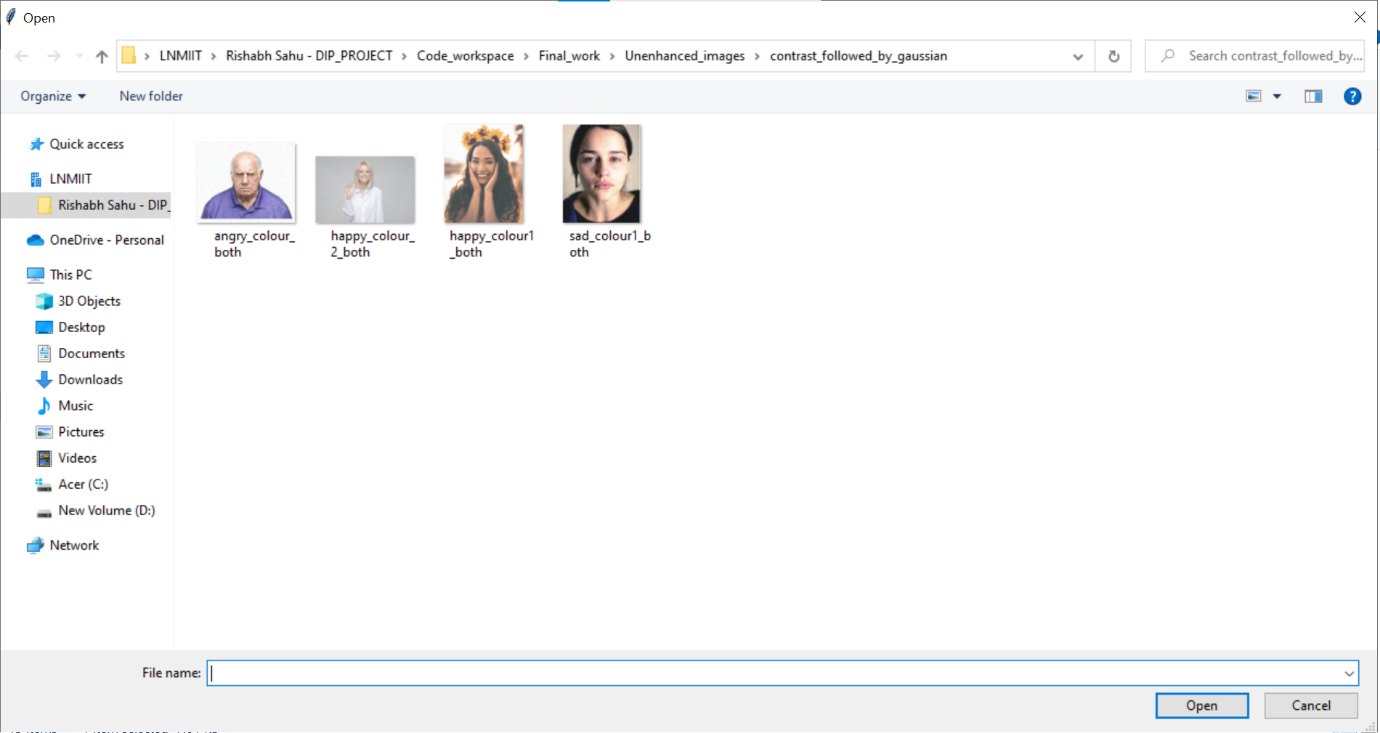
cv2.waitKey()

cv2.destroyAllWindows()

return img1

**Image selection code snippet ends here**

The image given below shows the image picker window which appears and gives the user the option to select the image.



**IMAGE ENHANCEMENT**

We have used two Image enhancement techniques that work one after the other to enhance the image. This will help the other functions, face detection and emotion recognition to have face with clear details and therefore make it easier to detect and display output.

**Code snippets related to image denoising**

median\_blur\_img=median\_blur\_func(image\_c)

#Apply median blurring filter on image in 5 x 5 neighbourhood

def median\_blur\_func(image):

    dst = cv2.medianBlur(image,5)

    return dst

#Reading the original image, here 1 implies that image is read as color

    image\_c=image\_picker()

    #currently median blur with 5 as parameter (specified in the defined function) works best

    median\_blur\_img=median\_blur\_func(image\_c)

    #generate matplotlib plot for result of denoising the image

    fig1,axs1 = plt.subplots()

    axs1.axis('off')

    axs1.set\_title ('Image after Denoising (Median filter)')

    plt.tight\_layout()

    figManager = plt.get\_current\_fig\_manager()

    figManager.full\_screen\_toggle()

    fig1.canvas.toolbar.pack\_forget()

    axs1.imshow(cv2.cvtColor(median\_blur\_img, cv2.COLOR\_BGR2RGB),aspect='auto')

    #close displayed plot after pressing button

    plt.waitforbuttonpress(0)

    plt.close()

**Image denoising code snippet ends here**



Image denoising result for the case of Angry face image.



Image denoising result for the case of happy face image.

As visible in these results the images after denoising are slightly blurred from the original image and the noise in the image has been reduced by application of Median filter.

**Contrast Stretching**

To apply contrast stretching we first converted the image to YUV format. Here we are interested in the Y channel which is related to the brightness and intensity values in the image. Then we picked up the maximum and minimum intensity values of the overall image and used these minimum and maximum values to normalise the intensities to 0,1 range, it was then multiplied to 255 to stretch the intensity values to full range till 255. After this the image was again converted to RGB format and displayed using matplotlib subplots along with the corresponding histograms.

**Code snippets for Contrast stretching part**

#Convert to YUV channel

image\_yuv = cv2.cvtColor(median\_blur\_img, cv2.COLOR\_BGR2YUV)

# Loop over the Y channel and apply Min-Max Contrasting

min = numpy.min(image\_yuv[:,:,0])

max = numpy.max(image\_yuv[:,:,0])

for i in range(image\_yuv.shape[0]):

for j in range(image\_yuv.shape[1]):

image\_yuv[:,:,0][i,j] = 255\*(image\_yuv[:,:,0][i,j]-min)/(max-min)

# Convert the YUV image back to RGB format

image\_c\_cs = cv2.cvtColor(image\_yuv, cv2.COLOR\_YUV2BGR)

fig, axs = plt.subplots(2, 2)

    axs[0, 0].set\_title('Original Image histogram') #histogram of image after denoising

    axs[1, 0].axis('off')

    #Plotting histogram of image before contrast stretching

    axs[0, 0].hist(image\_c.flatten(),bins\_c)

    #Displaying image before applying contrast stretching.

    axs[1, 0].imshow(cv2.cvtColor(median\_blur\_img, cv2.COLOR\_BGR2RGB),aspect='auto')

    #Plotting histogram of image after contrast stretching

    axs[0, 1].hist(image\_c\_cs.flatten(),bins\_c\_cs)

    axs[0, 1].set\_title('Histogram after Contrast Stretching')

    axs[1, 1].axis('off')

    #Displaying image after applying contrast stretching.

    axs[1, 1].imshow(cv2.cvtColor(image\_c\_cs, cv2.COLOR\_BGR2RGB),aspect='auto')

    plt.tight\_layout()

    figManager = plt.get\_current\_fig\_manager()

    figManager.full\_screen\_toggle()

    plt.waitforbuttonpress(0)

    plt.close()

    #generate matplotlib subplots

    fig3,axs3 = plt.subplots()

    axs3.axis('off')

    axs3.set\_title('Image after Contrast Stretching')

    plt.tight\_layout()

    figManager = plt.get\_current\_fig\_manager()

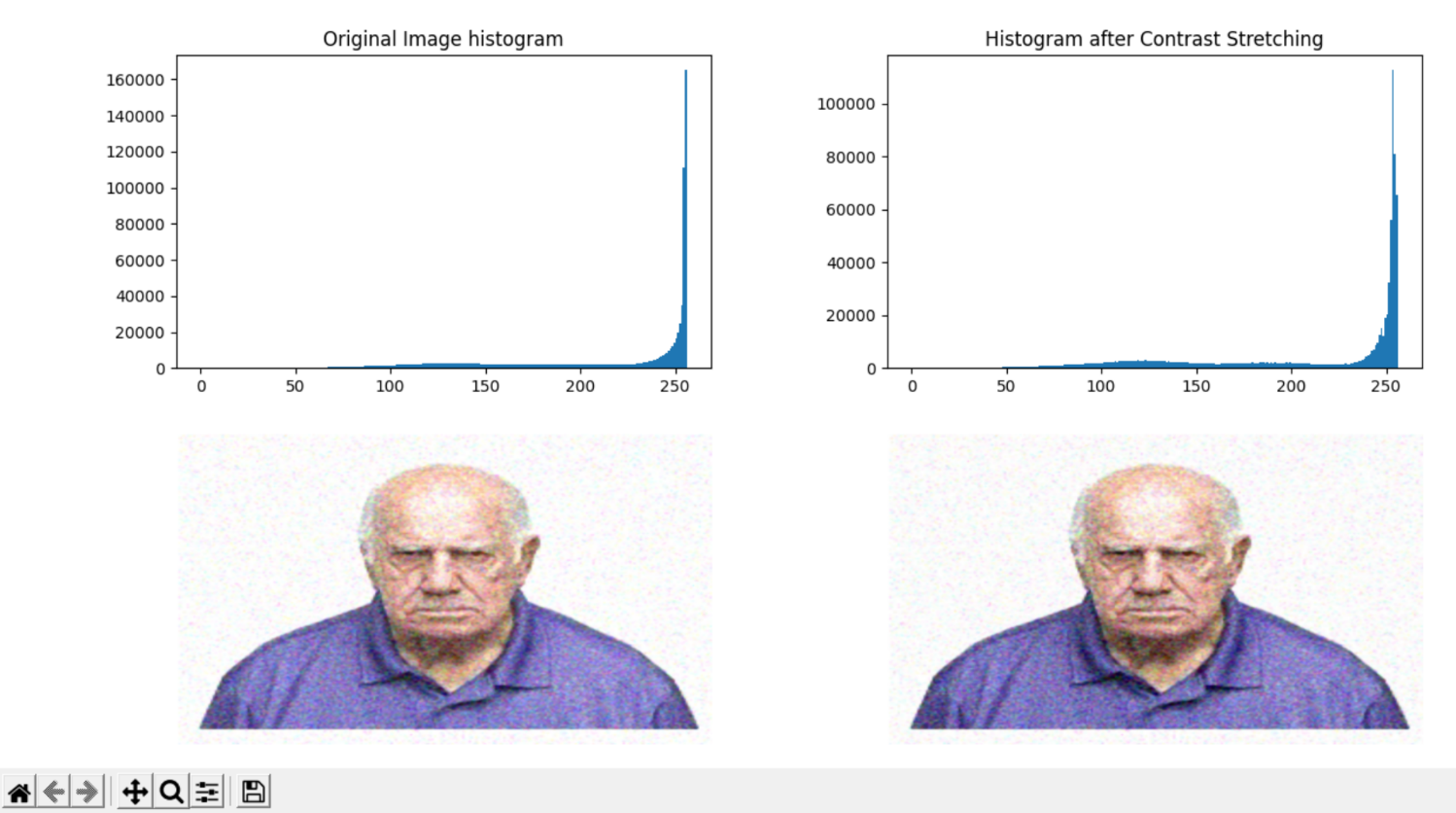
    figManager.full\_screen\_toggle()

    fig3.canvas.toolbar.pack\_forget()

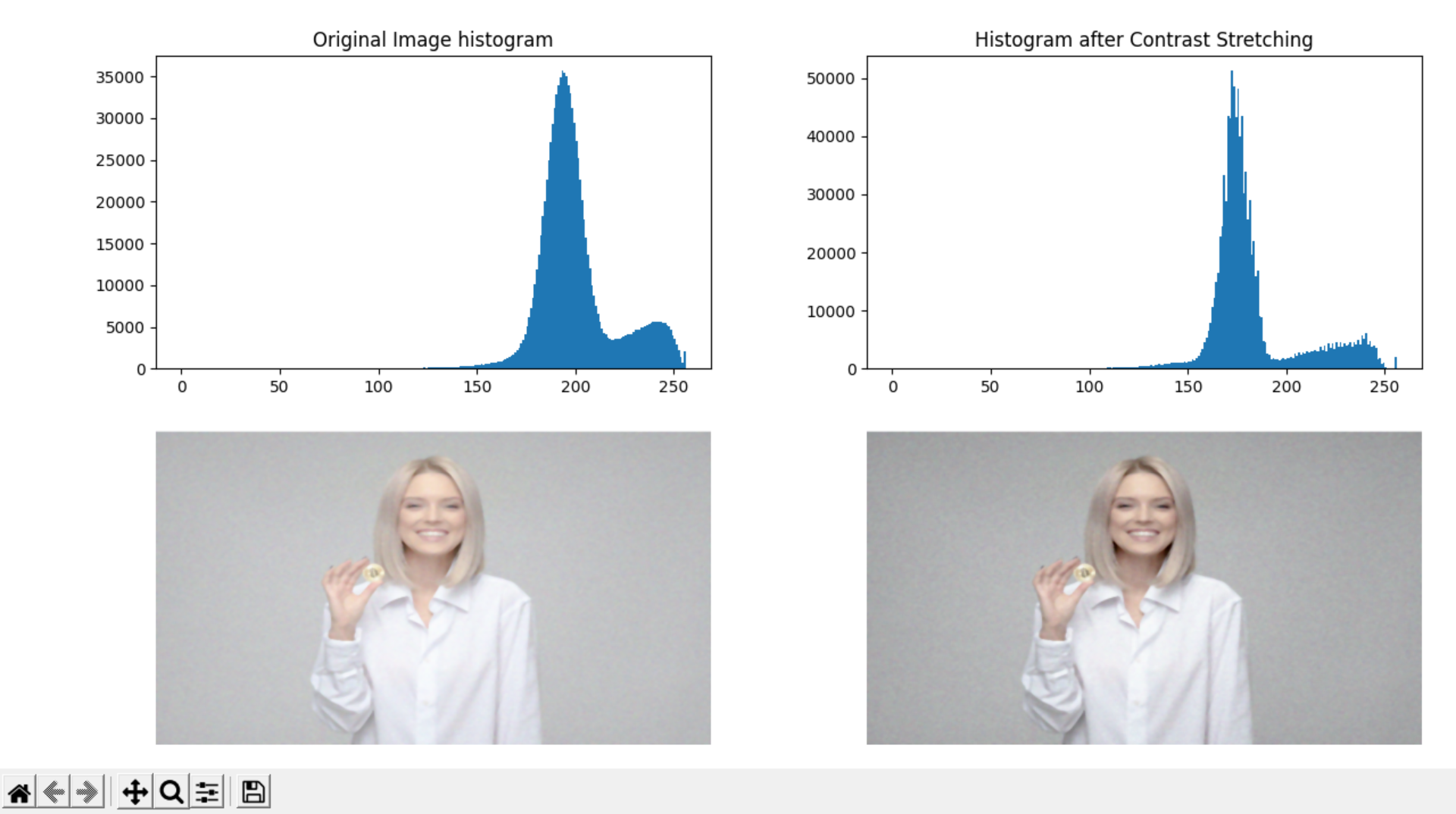
    axs3.imshow(cv2.cvtColor(image\_c\_cs, cv2.COLOR\_BGR2RGB),aspect='auto')

    plt.waitforbuttonpress(0)

**Code snippet for Contrast stretching ends here**



Contrast stretching result for angry image



Contrast stretching result for case of happy image

As it is visible in the histograms the range of lowermost pixel intensities and highest pixel intensities has shifted to wider range as well as the overall distribution of pixels has changed. For the Angry case lowermost pixel intensity that was earlier around 60 is now below 50. For the Happy case lowermost intensity changed from around125 to around 100.Highermost intensity values show minor change due to already being closer to maximum value 255.

**FACE DETECTION**

We used the Face detection function which used computer vision technology and helps to locate/visualize human faces in digital images.

This technique is a specific use case of object detection technology that deals with detecting instances of semantic objects of a certain class (such as humans, buildings, or cars) in digital images and videos. With the advent of technology, face detection has gained a lot of importance especially in fields like photography, security, and marketing.

So, for face detection first the program converts the given image into grey-scale image (as open-cv stores work on colour images in a different format than the usual RGB format). And then the program calls the functions to detect the faces (using Haar Classifier model).

And then after detecting the face again the program converts the image again into its original form (RGB).

After detection of the face, the program gets the coordinates of the rectangle in which the face is present. This will help to find a short region in which the emotion recognition part will work, saving time and overhead to detect the emotion from the face.

Face detection is performed by using classifiers. A classifier is an algorithm that decides whether a given image is positive(face) or negative (not a face). A classifier needs to be trained on thousands of images with and without faces. OpenCV already has pre-trained face detection classifiers, which can readily be used in a program:

Haar Classifier. We have used haar classifier for detection.

Basics of the working of haar classifier

After the tremendous amount of training data (in the form of images) is fed into the system while training the classifier, the classifier begins by extracting Haar features from each image. Haar Features are kind of convolution kernels which primarily detect whether a suitable feature is present on an image or not.

These Haar Features are like windows and are placed upon images to compute a single feature.

Thus, when the feature window moves over the part that matches pattern of a part of face for example eyes, it will calculate a single value.

This value will then be compared to some threshold and if it passes that it will conclude that there is an edge here or some positive feature

**Code snippets related to face detection**

def detect\_faces(cascade, test\_image, scaleFactor = 1.1):

# create a copy of the image to prevent any changes to the original one.

image\_copy = test\_image.copy()

#Convert the test image to gray scale as OpenCV face detector expects gray images

gray\_image = cv2.cvtColor(image\_copy, cv2.COLOR\_BGR2GRAY)

# Applying the haar classifier to detect faces, function returns coordinates of corners in array form

faces\_rect = cascade.detectMultiScale(gray\_image, scaleFactor=scaleFactor, minNeighbors=5)

centre\_x=(faces\_rect[0][0]+faces\_rect[0][2])/2

centre\_y=(faces\_rect[0][1]+faces\_rect[0][3])/2

#display green rectangle over detected face

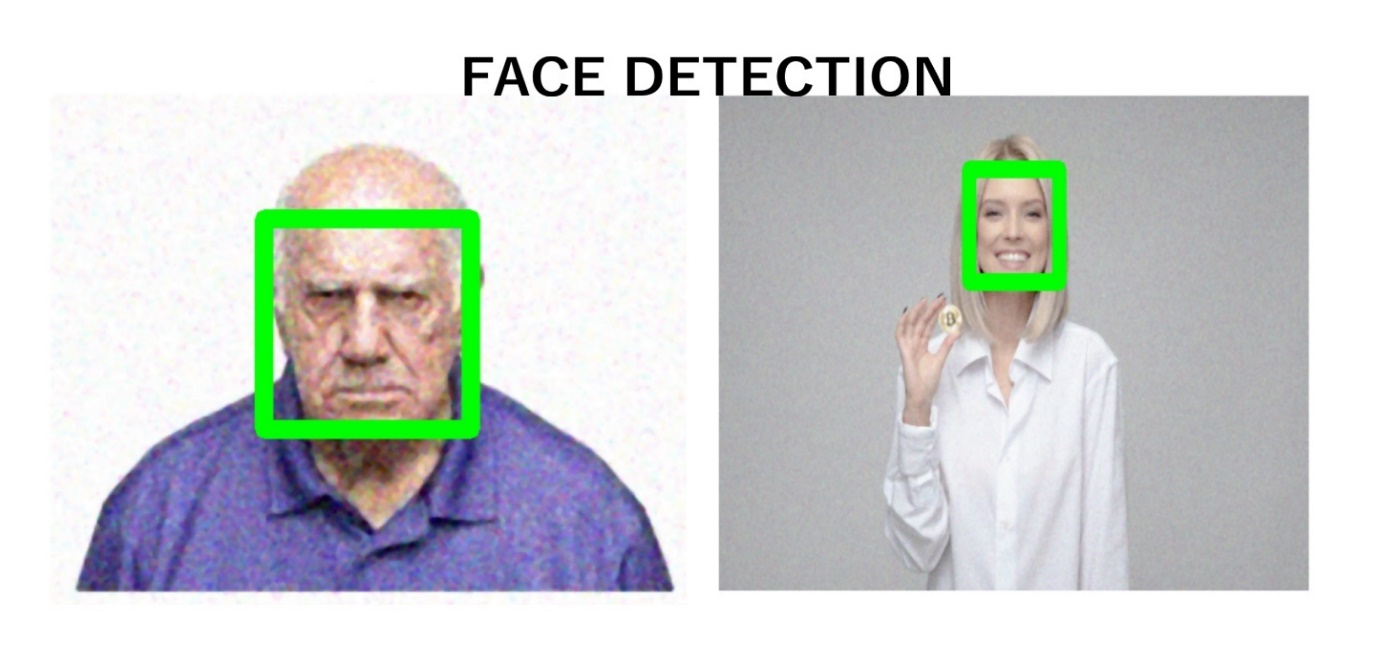
for (x, y, w, h) in faces\_rect:

cv2.rectangle(image\_copy, (x, y), (x+w, y+h), (0, 255, 0), 15)

#return face with rectangle and coordinates of detected face

return image\_copy,faces\_rect

**Face detection code snippets end here.**



Face detection result for both the cases

**FACE EXPRESSION RECOGNITION**

The face expression recognition part finally comes into picture. It uses the coordinates of face enclosing rectangle to operate in that region and find the emotion from the face in the image. We have used a pretrained model ("model\_v6\_23. hdf5") in our code. We use the predict () method of load\_model class which belongs to keras.models library to predict the emotions of the detected face.

Something about the model-

Model is prepared using an open source data set — [Face Emotion Recognition (FER](https://github.com/npinto/fer2013)) from [Kaggle](https://www.kaggle.com/) and a 6 layered CNN(Convolutional neural network model) was built in Keras to detect emotions. The emotions can be classified into 7 classes — happy, sad, fear, disgust, angry, neutral and surprise. The model was able to achieve 61.3% of validation accuracy. ​ More image augmentation and hyperparameter tuning could be used to further increase the accuracy of model.

**Code snippets related to emotion recognition part**

#Call the function to recognise the emotion expressed by the face. and store it in a string

emotion=recognise\_emotion(location,faces)

 emostr='Emotion Detected :' + emotion

    fig5,axs5 = plt.subplots()

    axs5.axis('off')

    axs5.set\_title(emostr)

    plt.tight\_layout()

    figManager = plt.get\_current\_fig\_manager()

    figManager.full\_screen\_toggle()

    fig5.canvas.toolbar.pack\_forget()

    #Displaying detected emotion

    axs5.imshow(cv2.cvtColor(faces, cv2.COLOR\_BGR2RGB),aspect='auto')

    plt.waitforbuttonpress(0)

    plt.close()

    input('Press ENTER to exit')

def recognise\_emotion(locations,image):

#Using coordinates obtained from face detect to specify region of face to emotion detector

x,y,w,h = locations[0]

top=y

right=x+w

bottom=y+h

left=x

f\_image = image[top:bottom, left:right]

face\_image= f\_image

#Load the model trained for detecting emotions of a face

model = load\_model("model\_v6\_23.hdf5")

#resizing the image

face\_image = cv2.resize(face\_image, (48,48))

face\_image = cv2.cvtColor(face\_image, cv2.COLOR\_BGR2GRAY)

face\_image = numpy.reshape(face\_image, [1, face\_image.shape[0], face\_image.shape[1], 1])

#Creating a dictionary with emotions as keys and their numeric representations as values.

emotion\_dict= {'Angry': 0, 'Sad': 5, 'Neutral': 4, 'Disgust': 1, 'Surprise': 6, 'Fear': 2, 'Happy': 3}

predicted\_class = numpy.argmax(model.predict(face\_image))

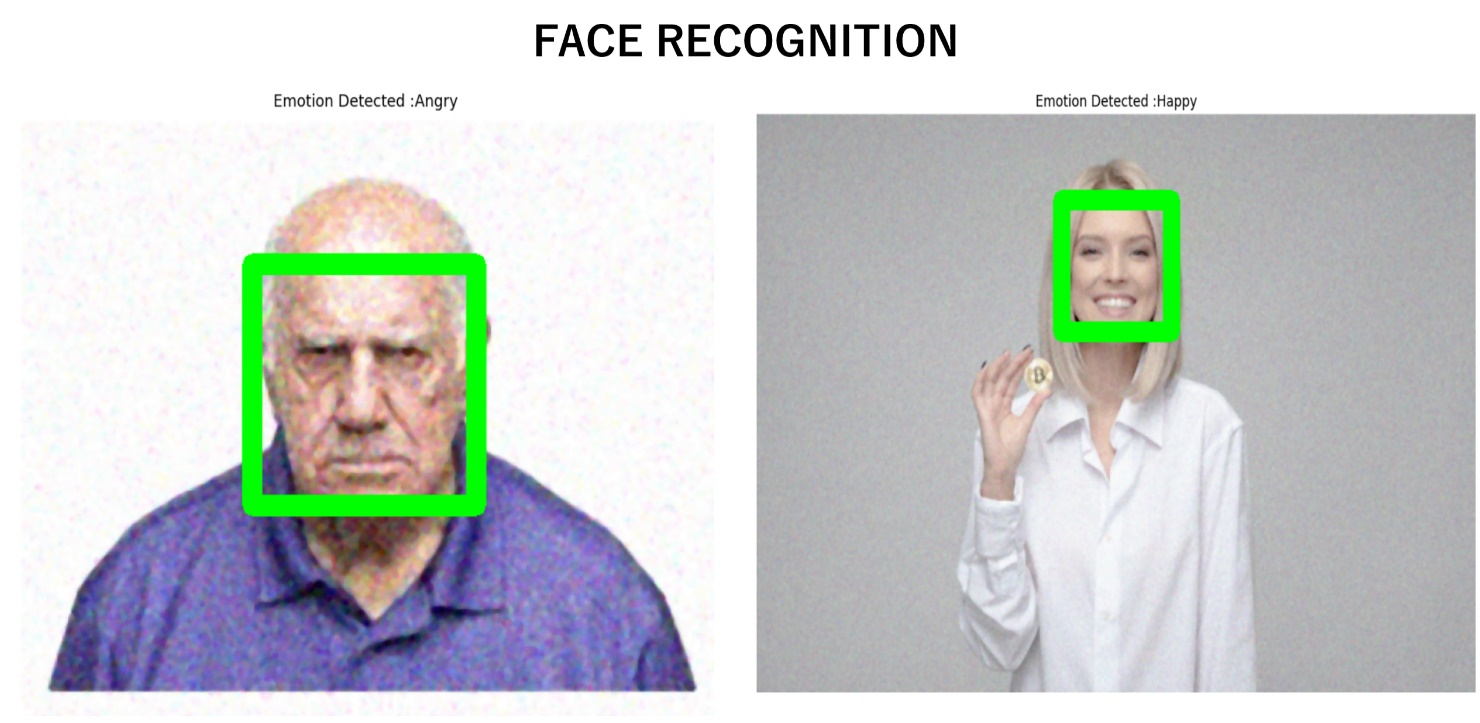
label\_map = dict((v,k) for k,v in emotion\_dict.items())

#Storing the emotion detected in predicted\_label variable.

predicted\_label = label\_map[predicted\_class]

return predicted\_label

**Emotion recognition code snippets end here**



Emotion recognition output for both the cases

**Ending remarks**

Thus, our group was able to learn Digital Image Processing concepts, python programming concepts and create an application which can recognise emotions from the face present in an image. This was possible because of efforts of our group members, our will for learning and our teamwork spirit.

Face emotion recognition can be used to get feedback from peoples facial expressions after they try a product , the facial expression will get converted to textual information and can be used to find whether the product was useful to them.

**ACKNOWLEDGEMENT**

We would like to express our heartfelt gratitude to our course professor, Dr. Ram Prakash Sharma, for giving us this opportunity which has helped us in expanding our knowledge and thinking capabilities.

We are also grateful to him for all his help and guidance that he provided and made it easy for us to complete this project. We also extend our gratitude to everyone else who helped us in completing this project.

**References**

1. Dwivedi, P. (2019, April 4). *Face Detection, Recognition and Emotion Detection in 8 lines of code!* Medium. https://towardsdatascience.com/face-detection-recognition-and-emotion-detection-in-8-lines-of-code-b2ce32d4d5de

*2. Use the dialog box to open an image file and display the picture - Programmer Sought*. (n.d.). ProgrammerSought. Retrieved March 1, 2021, from <https://programmersought.com/article/80784886101/>

3. *OpenCV: Smoothing Images*. (n.d.). OpenCV. Retrieved March 1, 2021, from <https://docs.opencv.org/master/d4/d13/tutorial_py_filtering.html>

4. Visualization with Matplotlib | Python Data Science Handbook (jakevdp.github.io)

https://jakevdp.github.io/PythonDataScienceHandbook/04.00-introduction-to-matplotlib.html

5. Face Detection with Python using OpenCV https://www.datacamp.com/community/tutorials/face-detection-python-opencv