PROJECT REPORT

On

Emotion Recognition using Facial Expression (ERFE)

Submitted by

Vivek Singhal (171500395)
Divyansh Arya (171500103)
Ashutosh Anand (171500064)
Vineet Rathore (171500382)

Under the supervision of

Dr. Suresh Raikwar

Department of Computer Engineering & Applications

Institute of Engineering & Technology

GLA University
Mathura- 281406, INDIA

2019

Acknowledgement

We thank the almighty for giving us the courage and perseverance in completing the project. This project itself is acknowledgements for all those people who have given us their heartfelt co-operation in making this project a grand success.

We thank GLA University for providing us resources and support. We extend our sincere thanks to **Dr. Suresh Raikwar**, Assistant Professor, GLA University, Mathura for providing valuable guidance at every stage of this project work. We are profoundly grateful towards the unmatched services rendered by him.

Last but not least, we would like to express our deep sense of gratitude and earnest thanks giving to our dear parents for their moral support and heartfelt cooperation in doing the main project.

Abstract

The human face plays a prodigious role for automatic recognition of emotion. This project helps to identify the interaction between human and computer for some real application like driver state surveillance, personalized learning, health monitoring etc. Most reported facial emotion recognition systems, however, are not fully considered subject-independent dynamic features, so they are not robust enough for real life recognition tasks with subject (human face) variation, head movement and illumination change. In this article we have tried to design an automated framework for emotion detection using facial expression. For human-computer interaction facial expression makes a platform for non-verbal communication. The emotions are effectively changeable happenings that are evoked as a result of impelling force. So in real life application, detection of emotion is very challenging task. Facial expression recognition system requires to overcome the human face having multiple variability such as color, orientation, expression, posture and texture so on. In our project we have taken frame from live streaming and processed it using “shape_predictor_68_face_landmarks.dat” file of dlib library. To detect the emotion from it, we have taken coordinates of each point, distance of these points from the mean point and angle of these distance line from normalized line. Finally to determine facial expressions separately, the processed feature vector is channeled through the already learned pattern classifiers.

Contents

1. Introduction

1.1 Overview

This project has mainly concentrated on the creation of smart framework with the inherent capabilities of drawing the inference for emotion detection from facial expressions. Recently, the notion of emotion recognition is attaining mostly the researcher's mind in the area of exploration on smart system and interaction between human and computer. Based on facial attributes the facial expression recognition can be classified one of the six well known fundamental emotions: sadness, disgust, happiness, fear, anger and surprise.

Emotion recognition is useful to make smooth communication between human & computer interaction. The recognition of human emotion can have wide applications in heterogeneous field. The applications are mainly based on the man and machine interaction, patient surveilling, inspecting for antisocial motives etc. Even we can recognize emotion for customers by analyzing their response on seeing certain commodity or advertisement or immediately after getting a message and based on the response from the customers, the resource hub can improve their strategies.

The first aim of this work is to incorporate anatomical grip for emotion recognition. Facial behavior is represented using “shape_predictor_68_face_landmarks.dat” file of dlib library. This file provides the coordinates of 68 landmarks on the given face. In the project, we use these 68 landmarks to get a normalized form of happy face and neutral face and then train the model on this normalized form

1.2 Motivation and Objective

The technology is rapidly improving and the surroundings as well. The world has become digital and in this digital world, images are a part of life.

Various organization and companies are trying to find new ways to gather information from these images to improve their products and services. As we can see, face detection and face recognition are already implemented in our devices. But what if a organization could also be able to recognize the emotion on its customer’s face. Obviously it would help them to get customer feedback in real time.

Many traffic accidents happen daily due to the driver’s drowsiness. A good drive state surveillance system can alert the driver before the accident could happen.

If a device could tell a visually impaired person the emotions on the other persons’ faces, it would be really helpful.

So our objective is to fulfill these market as well as social needs so we can give a model which will be really helpful to the companies for their feedback system and it will also be a guide to needy persons like blind person.

2. Dataset

The ERFE has used CK+ dataset, which is renowned in the field of emotion classification. The details of CK+ are given below:

2.1 CK+ Dataset:

2.1.1 Description:

The database includes approximately 2000 image sequences from over 200 subjects.

Subjects in the available portion of the database were 97 university students enrolled in introductory psychology classes. They ranged in age from 18 to 30 years. Sixty-five percent were female, 15 percent were African-American, and three percent were Asian or Latino. One of the cameras was located directly in front of the subject, and the other was positioned 30 degrees to the right of the subject. Only image data from the frontal camera are available at this time. Subjects were instructed by an experimenter to perform a series of 23 facial displays that included single action units (e.g. lip corners pulled obliquely) and combinations of action units (e.g. inner and outer brows raised).

Image sequences from neutral to target display were digitized into 640 by 480 or 490 pixel arrays with 8-bit precision for grayscale values. The images are available in png and jpg. Images are labeled using their corresponding VITC. The final frame of each image sequence was coded using FACS (Facial Action Coding System) which describes subject's expression in terms of action units (AUs).

2.1.2 Details of each feature:

2.1.2.1 The Landmarks (Landmarks.zip) - All sequences are AAM tracked with 68points landmarks for each image.

2.1.2.2 The FACS coded files (FACS_labels.zip) - for each sequence (593) there is only 1 FACS file, which is the last frame (the peak frame). Each line of the file corresponds to a specific AU and then the intensity. An example is given below.

2.1.2.3 The Emotion coded files (Emotion_labels.zip) - ONLY 327 of the 593 sequences have emotion sequences. This is because these are the only ones the fit the prototypic definition. Like the FACS files, there is only 1 Emotion file for each sequence which is the last frame (the peak frame). There should be only one entry and the number will range from 0-7 (i.e. 0=neutral, 1=anger, 2=contempt, 3=disgust, 4=fear, 5=happy, 6=sadness, 7=surprise). N.B there is only 327 files- IF THERE IS NO FILE IT MEANS THAT THERE IS NO EMOTION LABEL (sorry to be explicit but this will avoid confusion).

2.2 Our Own Dataset:

2.2.1 Description:

We have used only a portion of images of CK+ dataset. There are 2000 images in CK+ dataset as mentioned above but we have used only 125 images out of them because our main objective is to detect happy and neutral face only. So we have used 66 images for neutral faces and 59 images of happy faces.

2.2.2 Details of Each Feature:

We have not used features provided by CK+ dataset. We have extracted features from the images on our own. We have used 68 landmarks technique to detect the movement. We have dropped the 0th landmark. In order to normalize, we have taken mean of x coordinate and y coordinate and then to remove ambiguity, we have added the distance of each dot from mean point and have also added the angle for slightly tilt faces. So for each dot there are 4 values: its x coordinate, its y coordinate, its distance from mean point and the angle it makes from normalized line. So in total there is 4x67=268 feature vector in our dataset.

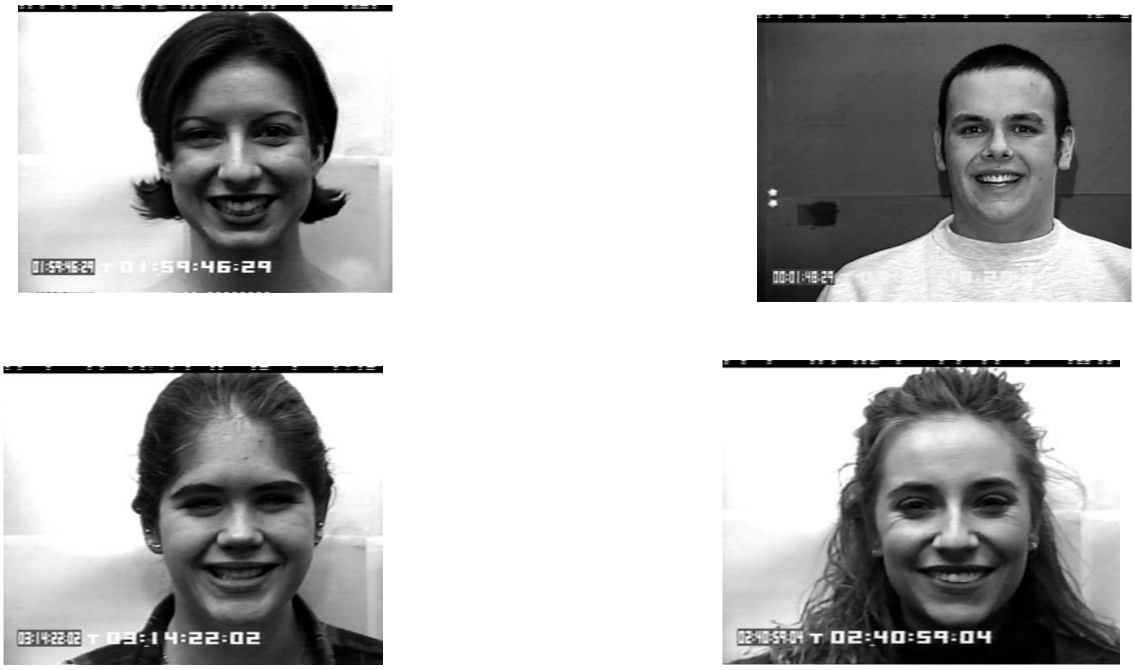


Fig 2.1 The selected happy faces from CK+ dataset

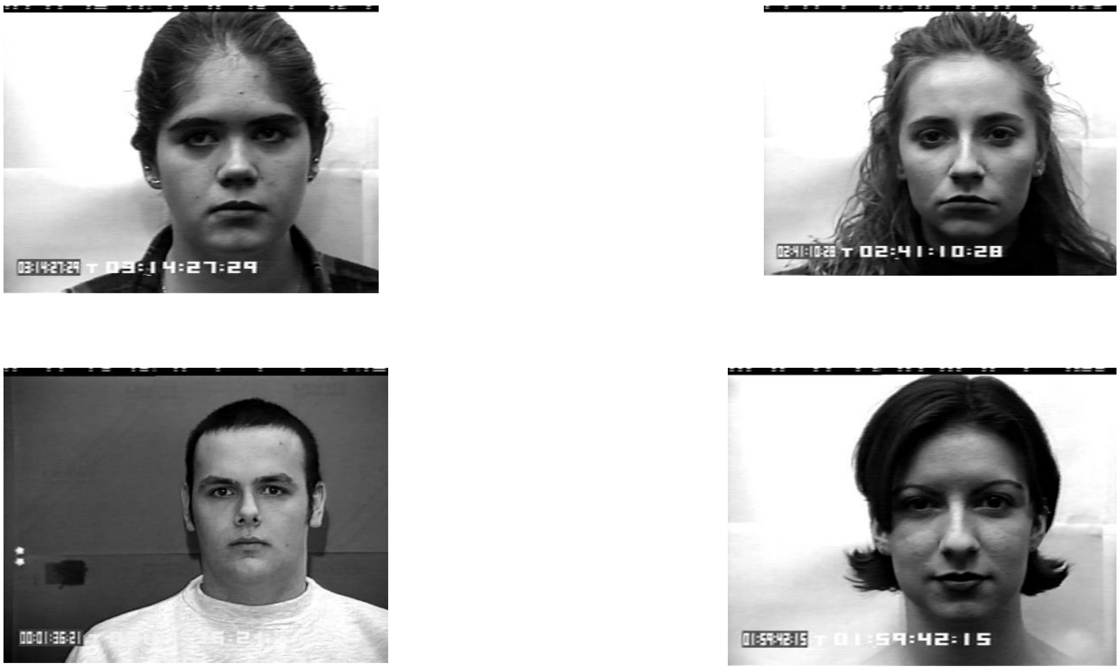


Fig 2.2 The selected neutral faces from CK+ dataset

3. Classification methods

In this section, the existing classification methods are presented. However, the Decision Tree is used in ERFE.

3.1 KNN (K-Nearest Neighbors):

KNN is the non-parametric method or classifier used for classification as well as regression problems. This is the lazy or late learning classification algorithm where all of the computations are derived until the last stage of classification, as well as this, is the instance-based learning algorithms where the approximation takes place locally. Being simplest and easiest to implement there is no explicit training phase earlier and the algorithm does not perform any generalization of training data.

KNN Classifier does following calculation at the hidden side.

(i) Compute the distance metric between the test data point and all labeled data points.

(ii) Order the labeled data points in increasing order of distance metric.

(iii) Select the top K labeled data points and look at class labels.

(iv) Look for the class labels that majority of these K labeled data points have and assign it to test data points.

This distance can be calculated from the Euclidean distance, Mahalanobis distance, Hamming distance, etc. Select the parameter K based on the data. Parameter selection- Best choice of K depends on data. The larger value of K reduce the effect of noise on classification but makes the decision boundaries between classless distinct. The smaller value of K tends to be affected by noise with clear separation between classes.

Different distance functions used in KNN are-

i. Euclidean function
ii. Manhattan function

$$\sqrt[2]{\sum_{i=1}^{k}(x_i - y_i)^2}$$

Fig 3.1 Euclidean Distance

$$\sum_{i=1}^{k}|x_i - y_i|$$

Fig 3.2 Manhattan Distance

3.2 SVM Classifier:

A Support Vector Machine (SVM) is a discriminative classifier formally defined by a separating hyperplane. In other words, given labeled training data (*supervised learning*), the algorithm outputs an optimal hyperplane which categorizes new examples. In two dimentional space this hyperplane is a line dividing a plane in two parts where in each class lay in either side.

3.2.1. Kernel:

The learning of the hyperplane in linear SVM is done by transforming the problem using some linear algebra. This is where the kernel plays role.

For **linear kernel** the equation for prediction for a new input using the dot product between the input (x) and each support vector (xi) is calculated as follows:

f(x) = B(0) + sum(ai * (x,xi))

This is an equation that involves calculating the inner products of a new input vector (x) with all support vectors in training data. The coefficients B0 and ai (for each input) must be estimated from the training data by the learning algorithm.

3.3 Decision Tree Classifier :

Decision tree algorithm falls under the category of supervised learning. They can be used to solve both regression and classification problems. Decision tree uses the tree representation to solve the problem in which each leaf node corresponds to a class label and attributes are represented on the internal node of the tree. We can represent any boolean function on discrete attributes using the decision tree.

Types of decision tree is based on the type of target variable we have. It can be of two types:

i. **Categorical Variable Decision Tree:** Decision Tree which has binary target variable then it called as Binary Variable Decision Tree. Example:- In above scenario of student problem, where the target variable was “Student will play cricket or not” i.e. YES or NO.

ii. **Continuous Variable Decision Tree:** Decision Tree has continuous target variable then it is called as Continuous Variable Decision Tree.

As our project is based on classification problem, so the Decision Tree used in the project is Categorical Variable Decision Tree.

4. Testing

We applied KNN, SVM and Decision Tree Classifier for predicting our output. We train our model on all of these classifier. Let's have a look on performance measure(Precision, Recall, Accuracy) of each classifier on our dataset.

4.1 SVM (Kernel = linear):

First we used Linear SVM classifier. As we can see, the accuracy is 100% in case of SVM Classifier

Table 4.1 SVM Classifier Testing Result

S.No.	Precision	Recall	f1-score	support
0	1.00	1.00	1.00	11
1	1.00	1.00	1.00	2
micro avg	1.00	1.00	1.00	13
macro avg	1.00	1.00	1.00	13
weighted avg	1.00	1.00	1.00	13
Accuracy			1.0	13

4.2 KNN (K=3):

Then we used knn algorithm with number of neighbors = 3 as we have only 2 class. This didn't give quite good results.

Table 4.2 KNN Classifier testing Result

S.No.	Precision	Recall	f1-score	support
0	0.60	0.75	0.67	4
1	0.00	0.00	0.00	2
micro avg	0.50	0.50	0.50	6
macro avg	0.30	0.38	0.33	6
weighted avg	0.40	0.50	0.44	6
Accuracy			0.5	6

4.3 Decision Trees:

We moved further and tried to get good results with the help of decision tree classifier. As we can see, this classifier gives the 100% accuracy.

Fig 4.3 Decision tree Classifier testing result

S.No.	Precision	Recall	f1-score	support
0	1.00	1.00	1.00	4
1	1.00	1.00	1.00	2
micro avg	1.00	1.00	1.00	6
macro avg	1.00	1.00	1.00	6
weighted avg	1.00	1.00	1.00	6
Accuracy			1.0	6

On comparing of all three classifier, we find that Linear SVM and Decision Tree Classifier gave the best accuracy which is 100% but when we tested both of the classifier on real time emotion capturing, we found that in case of SVM Classifier, the camera feed was not stable; the frames were fluctuating. But in the case of Decision Tree Classifier, it gave very good results.

That’s why we will be using Decision tree classifier for our model.

4.4 Limitations:

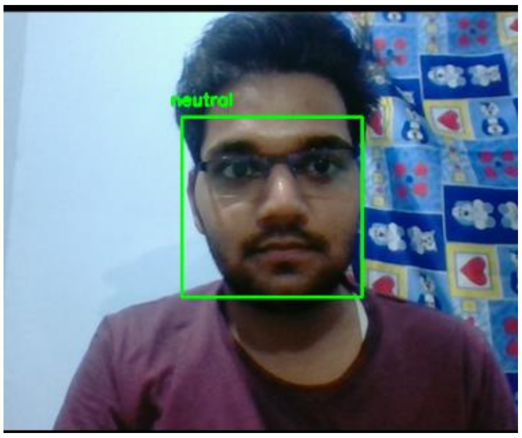
i. The model doesn’t work on distance more that 1m meter.

ii. The model doesn’t recognize micro-expressions. So it needs clear smile on the face to detect the happiness.

iii. We still haven’t tried it on any hardware or as an application.

5. Real Time Testing

This project, ERFE, also provides good result on the real time emotion recognition as you can see below:



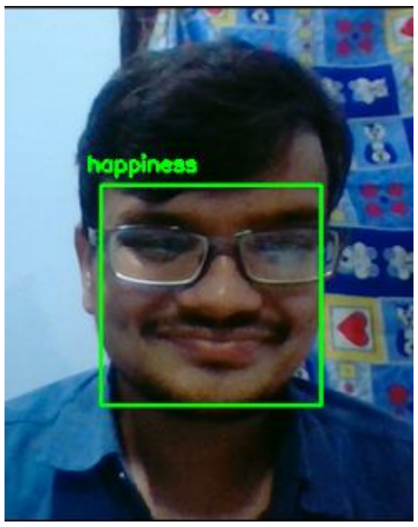


Fig 5.1 Neutral Emotion Detection

Fig 5.2 Happy Face Detection

6. Conclusion

6.1 Applications:

i. The model can be used in measuring the satisfaction level of an organization. It will detect the happy and non-happy faces with the camera and then it can give the total amount of happy and non-happy persons.

ii. Another application of this project is for the visually impaired person. A visually challenged person can’t see the person he/she is talking to and hence cannot know the expression on the other person’s face. This model can be implemented into a hardware device which will detect the emotion on other person’s face and then with the help of a earphone or buds, it can tell the visually impaired person the emotion via voice.

iii. Last but not the least application can be in home robots. Today IOT has become a giant and it is rapidly enhancing and we are seeing the various application of it like Sony Home Robot. It is a robot which works on human touch. We can improve these kinds of robots by embedding this model into their hardware and then the robot can detect the emotion on its owner’s face and can act accordingly which will make it more realistic.

6.2 Future Scope:

i. This model can be improved by using larger dataset.

ii. This model recognizes only two emotions: happiness and neutral. More emotions can be added by adding images related to that emotion.

iii. Facial electromyography can be used to detect electrical impulse generated by muscles to recognize miniature expressions.

7. Bibliography

[1] Emotion analysis using FACS Coding; https://ieeexplore.ieee.org/document/7831605

[2] A Complete Tutorial on Decision Tree; https://www.analyticsvidhya.com/blog/2015/01/decision-tree-simplified/2/

[3] Facial Landmarks detection; https://www.learnopencv.com/facial-landmark-detection/

8. Appendix

8.1 Testing.ipynb

```
import cv2
import glob
import random
import math
import numpy as np
import dlib
import itertools
from sklearn.svm import SVC
emotions = ["happy", "neutral"] #Emotion list
clahe = cv2.createCLAHE(clipLimit=2.0, tileGridSize=(8,8))
detector = dlib.get_frontal_face_detector()
predictor = dlib.shape_predictor("shape_predictor_68_face_landmarks.dat") #Or set
this to whatever you named the downloaded file
data = {} #Make dictionary for all values
#data['landmarks_vectorised'] = []

def get_files(emotion): #Define function to get file list, randomly shuffle it and split
80/20
    files = glob.glob("dataset\\%s\\*" %emotion)
    random.shuffle(files)
    training = files[:int(len(files)*0.8)] #get first 80% of file list
    prediction = files[-int(len(files)*0.2):] #get last 20% of file list
    return training, prediction


def get_landmarks(image):
    detections = detector(image, 1)
    for k,d in enumerate(detections): #For all detected face instances individually
        shape = predictor(image, d) #Draw Facial Landmarks with the predictor class
        xlist = []
        ylist = []
        for i in range(1,68): #Store X and Y coordinates in two lists
            xlist.append(float(shape.part(i).x))
            ylist.append(float(shape.part(i).y))
        xmean = np.mean(xlist)
        ymean = np.mean(ylist)
        xcentral = [(x-xmean) for x in xlist]
        ycentral = [(y-ymean) for y in ylist]
        landmarks_vectorised = []
        for x, y, w, z in zip(xcentral, ycentral, xlist, ylist):
            landmarks_vectorised.append(w)
            landmarks_vectorised.append(z)
            meannp = np.asarray((ymean,xmean))
            coornp = np.asarray((z,w))
            dist = np.linalg.norm(coornp-meannp)
            landmarks_vectorised.append(dist)
            landmarks_vectorised.append((math.atan2(y, x)*360)/(2*math.pi))
        data['landmarks_vectorised'] = landmarks_vectorised
    if len(detections) < 1:
        data['landmarks_vectorised'] = "error"



def make_sets():
    training_data = []
    training_labels = []
    prediction_data = []
    prediction_labels = []
    for emotion in emotions:
        print(" working on %s" %emotion)
        training, prediction = get_files(emotion)
        #Append data to training and prediction list, and generate labels 0-7
        for item in training:
            image = cv2.imread(item) #open image
            gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY) #convert to grayscale
            clahe_image = clahe.apply(gray)
            get_landmarks(clahe_image)
            if data['landmarks_vectorised'] == "error":
                print("no face detected on this one")
            else:
                training_data.append(data['landmarks_vectorised']) #append image array to
training data list
                training_labels.append(emotions.index(emotion))

        for item in prediction:
            image = cv2.imread(item)
            gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
            clahe_image = clahe.apply(gray)
            get_landmarks(clahe_image)
            if data['landmarks_vectorised'] == "error":
                print("no face detected on this one")
            else:
                prediction_data.append(data['landmarks_vectorised'])
                prediction_labels.append(emotions.index(emotion))
    return training_data, training_labels, prediction_data, prediction_labels

#splitting the dataset
training_data, training_labels, prediction_data, prediction_labels = make_sets()
npar_train = np.array(training_data) #Turn the training set into a numpy array for the
classifier
npar_pred = np.array(prediction_data)

#importing the confusion_matrix and classification_report for performance measure
from sklearn.metrics import classification_report, confusion_matrix
#SVM
```

```
#Set the classifier as a support vector machines with polynomial kernel
svm_clf = SVC(kernel='linear', probability=True, tol=1e-3)#, verbose = True)
svm_clf.fit(npar_train,training_labels)
print("accuracy")
print(svm_clf.score(npar_pred, prediction_labels))
svm_pred = svm_clf.predict(npar_pred)
#print("Confusion Matrix")
#print(confusion_matrix(prediction_labels, npar_pred))
print(classification_report(prediction_labels, svm_pred))
#KNN
from sklearn.neighbors import KNeighborsClassifier
knn_clf = KNeighborsClassifier(n_neighbors=3)
knn_clf.fit(npar_train, training_labels)
print("accuracy")
print(knn_clf.score(npar_pred, prediction_labels))
knn_pred = knn_clf.predict(npar_pred)
print(classification_report(prediction_labels, knn_pred))

#Decision Trees
from sklearn.tree import DecisionTreeClassifier
tree_clf = DecisionTreeClassifier(random_state=0)
tree_clf.fit(npar_train, training_labels)
print("accuracy")
print(tree_clf.score(npar_pred, prediction_labels))
tree_pred = tree_clf.predict(npar_pred)
print(classification_report(prediction_labels, tree_pred))
```

8.2 Model.dat

```
import pickle
file=open(‘model.dat’,’wb’)
pickle.dump(clf,file)  #here clf refers to the Decision Tree Classifier
file.close()
```

8.3 LiveCapturing.ipynb

```
import cv2
from math import atan2,pi
import dlib
import numpy as np

import pickle
file=open('model.dat','rb')
clf=pickle.load(file)
file.close()

emotions = ["happy", "neutral"] #Emotion list
clahe = cv2.createCLAHE(clipLimit=2.0, tileGridSize=(8,8))
detector = dlib.get_frontal_face_detector()
predictor = dlib.shape_predictor("shape_predictor_68_face_landmarks.dat")
#data = {} #Make dictionary for all values
#data['landmarks_vectorised'] = []
def get_landmarks(image,d):
    shape = predictor(image, d) #Draw Facial Landmarks with the predictor class
    xlist = []
    ylist = []
    for i in range(1,68): #Store X and Y coordinates in two lists
        xlist.append(float(shape.part(i).x))
        ylist.append(float(shape.part(i).y))
    xmean = np.mean(xlist)
    ymean = np.mean(ylist)
    xcentral = [(x-xmean) for x in xlist]
    ycentral = [(y-ymean) for y in ylist]
    landmarks_vectorised = []
    for x, y, w, z in zip(xcentral, ycentral, xlist, ylist):
        landmarks_vectorised.append(w)
        landmarks_vectorised.append(z)
        meannp = np.asarray((ymean,xmean))
        coornp = np.asarray((z,w))
        dist = np.linalg.norm(coornp-meannp)
        landmarks_vectorised.append(dist)
        landmarks_vectorised.append((atan2(y, x)*360)/(2*pi))
    return landmarks_vectorised

def rect_to_bb(rect):
            # take a bounding predicted by dlib and convert it
            # to the format (x, y, w, h) as we would normally do
            # with OpenCV
            x = rect.left()
            y = rect.top()
            w = rect.right() - x
            h = rect.bottom() - y

            # return a tuple of (x, y, w, h)
            return (x, y, w, h)

#Set up some required objects
#Doing Live feeding
video_capture = cv2.VideoCapture(0) #Webcam object
while True:
    ret, frame = video_capture.read()
    cam_data = []
    cam_labels = []
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    clahe_image = clahe.apply(gray)
    faces = detector(gray, 1)
    if (len(faces))<1:
        print("No Face Detected")
    else:
        for (i, rect) in enumerate(faces):
            # determine the facial landmarks for the face region, then convert the facial
landmark (x, y)-coordinates to a NumPy
            # array
            # convert dlib's rectangle to a OpenCV-style bounding box
            # [i.e., (x, y, w, h)], then draw the face bounding box
            landmarks_vectorised = get_landmarks(clahe_image,rect)
            cam_arr = []
            cam_arr.append(np.array(landmarks_vectorised))
            emotion = clf.predict(cam_arr)
            (x, y, w, h) = rect_to_bb(rect)
            cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 255, 0), 2)
            # show the face number
            cv2.putText(frame, "{}".format(emotions[emotion[0]]), (x - 10, y -
10),cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 255, 0), 2)

            # loop over the (x, y)-coordinates for the facial landmarks
            # and draw them on the image

        cv2.imshow("Output", frame)
        k = cv2.waitKey(1)
        if k == ord('q'):
            break

video_capture.release()
cv2.destroyAllWindows()
```