Emotion Recognition Using Audio and Video

Human being display their emotions using facial expressions. For humans it is very easy to recognize those emotions but for computers it is very challenging. Facial expressions vary from person to person. Brightness, contrast and resolution of every random image is different. This is why recognizing facial expressions is very difficult. Facial expression recognition is an active research area. In this project, we worked on recognition of seven basic human emotions. These emotions are angry, disgust, fear, happy, sad, surprise and neutral.

The model involves generally 3 steps training, testing and validation and after completion of these steps we input our audio or video to predict emotion in the input.

FER2013 Dataset-

The data consists of 48x48 pixel grayscale images of faces. The faces have been automatically registered so that the face is more or less centered and occupies about the same amount of space in each image. The task is to categorize each face based on the emotion shown in the facial expression into one of seven categories (**0=Angry, 1=Disgust, 2=Fear, 3=Happy, 4=Sad, 5=Surprise, 6=Neutral**).

The training set consists of 28,709 examples. The public test set consists of 3,589 examples.

Firstly we have done emotion recognition using video in which we used basic CNN algorithm , trained and tested with the help of fer2013 dataset to predict emotion. Below you can see the code and explanation of the video part.

Emotion recognition using video has been completed.

Now coming to emotion recognition using the audio part , RAVDESS dataset (https://www.kaggle.com/uwrfkaggler/ravdess-emotional-speech-audio) will be used in this part. We have started studying RNN and research papers of deep speech to implement this part.

And then we will try to sum up both parts and predict emotion using audio and video both.

Import required libraries:

Import the required libraries for building the network. The code for importing the libraries is written below.

```
import pandas as pd
import  sys,os
import numpy as np import keras
from  keras.models import  Sequential
from  keras.layers.core  import  Dropout
from  keras.layers  import  Dense,  Conv2D,  MaxPooling2D  ,  Flatten
from  tensorflow.keras.layers.experimental  import  preprocessing
from  keras.optimizers  import  SGD
import cv2 import np_utils
from  keras.preprocessing  import  image
from  keras.preprocessing.image  import  ImageDataGenerator
```

After importing all the libraries we have to import dataset which is to be used,we can give the directory in small brackets where we have stored the dataset as shown in the code below.

```
emotion_data  =  pd.read_csv('C:\\Users\\HP\\Desktop\\fer2013.csv')
```

We will then create a different list for testing and training image pixels.
After this we will check if pixels belong to the training set or testing set and we will append to the training or testing list respectively.The code for this is written below.

```
X_train = [] y_train = [] X_test
= [] y_test = []
for  index,  row  in  emotion_data.iterrows():
      k =  row['pixels'].split(" ")

     if  row['Usage']  ==  'Training':

          X_train.append(np.array(k))
          y_train.append(row['emotion'])
     elif  row['Usage']  ==  'PublicTest':
          X_test.append(np.array(k))
          y_test.append(row['emotion'])
X_train  =  np.array(X_train)
y_train  =  np.array(y_train)
X_test  =  np.array(X_test)
y_test  =  np.array(y_test) #print(X_train.shape)
#print(X_test.shape) #print(y_test.shape)
#print(y_test.shape)

X_train  =  X_train.reshape(X_train.shape[0],  48,  48,  1)

X_test  =  X_test.reshape(X_test.shape[0],  48,  48,  1)
#print(X_train.shape)
y_train=  keras.utils.to_categorical(y_train,  num_classes=7)
```

Now it's time to design the CNN model for emotion detection with different layers. We start with the initialization of the model followed by batch normalization layer and then different convents layers with ReLu as an activation function, max pool layers, and dropouts to do learning efficiently.

```
model = Sequential()
#1st 1ayer
model.add(Conv2D(64,(5,5),activation='relu',input_shape=(48,48,1)))
preprocessing.RandomFlip(mode='horizontal')
preprocessing.RandomRotation(factor=0.05)
model.add(MaxPooling2D(pool_size=(5,5),strides=(2,2)))
#2nd 1ayer
model.add(Conv2D(128,(3,3),activation='relu'))
model.add(Conv2D(128,(3,3),activation='relu'))
model.add(MaxPooling2D(pool_size=(3,3),strides=(2,2)))
#3rd 1ayer
 model.add(Conv2D(256,(3,3),activation='relu'))
model.add(Conv2D(256,(3,3),activation='relu'))
model.add(MaxPooling2D(pool_size=(3,3),strides=(2,2)))

model.add(Flatten())

model.add(Dense(1024,activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(1024,activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(7,activation='softmax')) gen =
ImageDataGenerator()
train_gen = gen.flow(X_train,y_train,batch_size=512)
gen1= ImageDataGenerator()
test_gen=gen1.flow(X_test,y_test,batch_size=512)
```

We have also compiled the model using Adam as an optimizer, loss as categorical cross-entropy, and metrics as accuracy as shown in the below code.

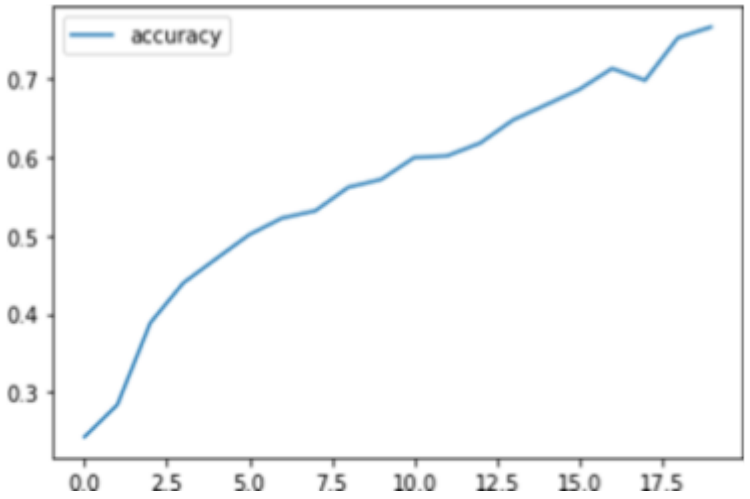
```
model.
 ←compile(optimizer='Adam',loss='categorical_crossentropy',metrics=['accuracy'])
```

After compiling the model we then fit the data for training and validation.
Here we are taking batch_size of 512 and epochs =20,we can tune them according to our need.

```
history=model.fit(train_gen,validation_data=test_gen,batch_size=512,epochs = 20)
```

```
Epoch 1/20
57/57 [==============================] - 333s 6s/step - loss: 3.2164 - accuracy:
0.2267 - val_loss: 1.8244 - val_accuracy: 0.2014
Epoch 2/20

57/57 [==============================] - 317s 6s/step - loss: 1.7918 - accuracy:

0.2594 - val_loss: 1.6372 - val_accuracy: 0.3625 Epoch 3/20
57/57 [==============================] - 307s 5s/step - loss: 1.5928 - accuracy:
0.3768 - val_loss: 1.5994 - val_accuracy: 0.3859
Epoch 4/20

57/57 [==============================] - 313s 5s/step - loss: 1.4850 - accuracy:

0.4268 - val_loss: 1.4373 - val_accuracy: 0.4544 Epoch 5/20

57/57 [==============================] - 325s 6s/step - loss: 1.4126 - accuracy:

0.4621 - val_loss: 1.3738 - val_accuracy: 0.4820 Epoch 6/20

57/57 [==============================] - 324s 6s/step - loss: 1.3196 - accuracy:

0.5014 - val_loss: 1.3005 - val_accuracy: 0.5093 Epoch 7/20

57/57 [==============================] - 323s 6s/step - loss: 1.2455 - accuracy:

0.5280 - val_loss: 1.3039 - val_accuracy: 0.4937 Epoch 8/20
57/57 [==============================] - 317s 6s/step - loss: 1.2677 - accuracy:
0.5156 - val_loss: 1.2738 - val_accuracy: 0.5177 Epoch 9/20

57/57 [==============================] - 292s 5s/step - loss: 1.1523 - accuracy:

0.5674 - val_loss: 1.3120 - val_accuracy: 0.5018 Epoch 10/20

57/57 [==============================] - 293s 5s/step - loss: 1.1477 - accuracy:

0.5685 - val_loss: 1.2689 - val_accuracy: 0.5185 Epoch 11/20

57/57 [==============================] - 292s 5s/step - loss: 1.0679 - accuracy:

0.5958 - val_loss: 1.2960 - val_accuracy: 0.5283 Epoch
12/20

57/57 [==============================] - 294s 5s/step - loss: 1.0363 - accuracy:

0.6144 - val_loss: 1.2462 - val_accuracy: 0.5358 Epoch
13/20
57/57 [==============================] - 307s 5s/step - loss: 0.9905 - accuracy:
0.6239 - val_loss: 1.2570 - val_accuracy: 0.5311 Epoch
14/20

57/57 [==============================] - 298s 5s/step - loss: 0.9414 - accuracy:

0.6512 - val_loss: 1.2798 - val_accuracy: 0.5489 Epoch
15/20

57/57 [==============================] - 294s 5s/step - loss: 0.8686 - accuracy:

0.6748 - val_loss: 1.2689 - val_accuracy: 0.5369 Epoch
16/20

57/57 [==============================] - 519s 9s/step - loss: 0.8699 - accuracy:

0.6773 - val_loss: 1.3989 - val_accuracy: 0.5294
```

Here are the codes to print the accuracy plot.



We now serialize the model to JSON and save the model weight in hd5 file So that we can make use of this file to make predictions rather than training The network again.Code for this is written below.

```
fer_json = model.to_json()
with open("fer.json", "w") as json_file: json_file.write(fer_json)
model.save_weights("fer.h5")
```

Testing the model in Real-time using OpenCV and WebCam

Till now we have train our model on our dataset.Now we will test the model that we build for emotion detection in real-time using OpenCV and webcam. To do so we will write a python script. We will use the Jupyter notebook in our local system to make use of a webcam. You can use other IDEs as well. First, we will install a few libraries that are required. Use the below code to import those all.

[20]:
```
from keras.models import load_model
from time import sleep
#from keras.models import model_from_jason
from keras.preprocessing.image import img_to_array
from keras.preprocessing import image
import cv2
import numpy as np

#model = model_from_json(open("fer.json", "r").read())
#load weights
model.load_weights('fer.h5')
```

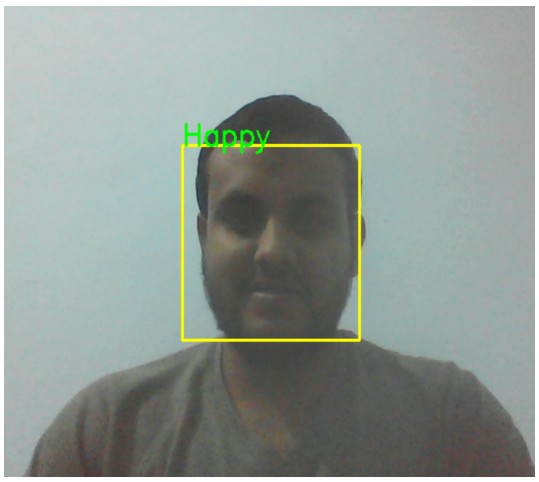
After importing all the required libraries we will load the model weights that we saved earlier after training. Use the below code to load your saved model. After importing the model weights we have imported a haar cascade file that is designed by open cv to detect the frontal face.

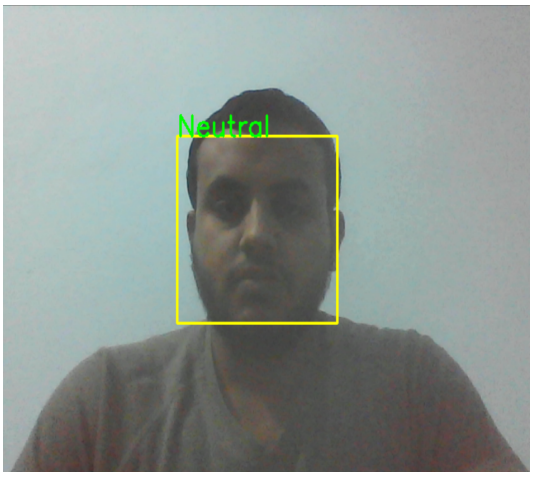
```
face_classifier = cv2.CascadeClassifier(r"C:
↪\Users\HP\Desktop\Emotion-recognition-master\haarcascade_files\haarcascade_frontalface_defaul
↪xml")
```

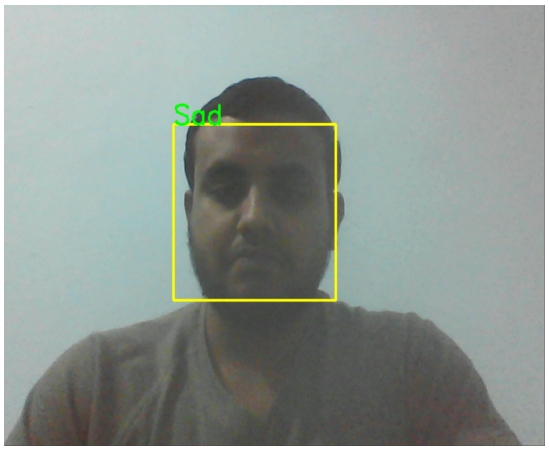
After importing the haar cascade file we will have written a code to detect faces and classify the desired emotions. We have assigned the labels that will be different emotions like angry, happy, sad, surprise, neutral. As soon as you run the code a new window will pop up and your webcam will turn on. It will then detect the face of the person, draw a bounding box over the detected person, and then convert the RGB image into grayscale & classify it in real-time. Please refer to the below code for the same and sample outputs that are shown in the images. To stop the code you need to press ‘q’.

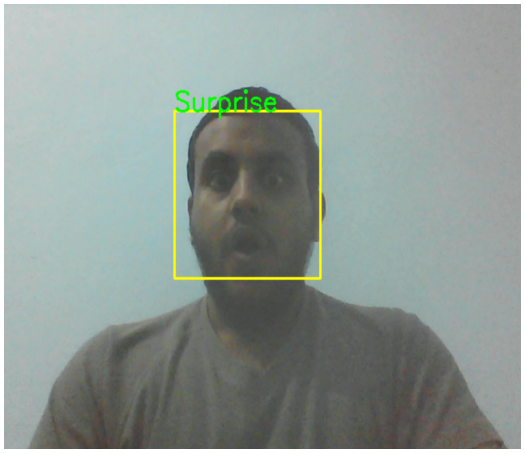
```
emotion_labels = ['Angry','Disgust','Fear','Happy','Sad', 'Surprise', 'Neutral']

cap = cv2.VideoCapture(0)



while True:
    _, frame = cap.read()
    labels = []
    gray = cv2.cvtColor(frame,cv2.COLOR_BGR2GRAY)
    faces = face_classifier.detectMultiScale(gray)

    for (x,y,w,h) in faces:
        cv2.rectangle(frame,(x,y),(x+w,y+h),(0,255,255),2)
        roi_gray = gray[y:y+h,x:x+w]
        roi_gray = cv2.resize(roi_gray,(48,48),interpolation=cv2.INTER_AREA)
```

```
        if np.sum([roi_gray])!=0:
            roi = roi_gray.astype('float')/255.0
            roi = img_to_array(roi)
            roi = np.expand_dims(roi,axis=0)

            prediction =model.predict(roi)[0]
#             print(prediction)
            label=emotion_labels[prediction.argmax()]
            label_position = (x,y)
            cv2.putText(frame,label,label_position,cv2.
 ↪FONT_HERSHEY_SIMPLEX,1,(0,255,0),2)
        else:
            cv2.putText(frame,'No Faces',(30,80),cv2.
 ↪FONT_HERSHEY_SIMPLEX,1,(0,255,0),2)
    cv2.imshow('Emotion Detector',frame)
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

cap.release()
cv2.destroyAllWindows()
```


Happy


Neutral


Sad


Surprise