**SER- Speech Emotion Recognition – Code**

# Load the drive helper and mount

from google.colab import drive

# This will prompt for authorization

drive.mount('/content/drive')

cd drive/'My Drive'/'Colab Notebooks'/'Emotion Speech Recognition'/

#Now above cell output is our path in which is in current working directory

!ls

#Provides a way of using operating system dependent functionality

import os

#LibROSA provides the audio analysis

import librosa

#Need to implicity import from librosa

import librosa.display

#Import the audio playback widget

import IPython.display as ipd

from IPython.display import Image

#Enable plot in notebook

%pylab inline

%matplotlib inline

import matplotlib.pyplot as plt

#This are generally useful to have around

import numpy as np

import pandas as pd

#To build neural network and desired model

import keras

from keras.models import Model

from keras.models import Sequential

from keras.layers import Conv1D, MaxPooling1D #,AveragePooling1D

from keras.layers import Flatten, Dropout, Activation #Input,

from keras.layers import Dense #,Embedding

from keras.utils import np\_utils

from sklearn.preprocessing import LabelEncoder

data, sampling\_rate = librosa.load('Dataset/surprise/surprise002.wav')

# To play audio this in the colab notebook

ipd.Audio('Dataset/surprise/surprise002.wav')

plt.figure(figsize=(15,5))

librosa.display.waveplot(data, sr=sampling\_rate)

dataset\_path = os.path.abspath('./Dataset')

destination\_path = os.path.abspath('./')

# To shuffle the dataset records

randomize = True

# for splitting the dataset into training  and testing datasets

split = 0.8

# Number of samples per sec e.g 16KHz

sampling\_rate = 20000

emotions=["anger","disgust","fear","happy","neutral","sad","surprise"]

#loading dataframes using dataset module

from utils import dataset

df, train\_df, test\_df = dataset.create\_and\_load\_meta\_csv\_df(dataset\_path, destination\_path, randomize, split)

print('Dataset samples :', len(df),"\nTraining Samples : ", len(train\_df),"\nTesting Samples :", len(test\_df))

df.head()

print("Actual Audio : ", df['path'][0])

print("Labels : ", df['label'][0])

unique\_labels = train\_df.label.unique()

unique\_labels.sort()

print("unique labels in emotion dataset : ")

print(\*unique\_labels, sep=', ')

unique\_labels\_counts = train\_df.label.value\_counts(sort=False)

print("\n\nCount of unique labels in Emotion dataset : ")

print(\*unique\_labels\_counts, sep=', ')

#Histogram of the classes

plt.bar(unique\_labels, unique\_labels\_counts, align= 'center', width=0.6, color='c')

plt.xlabel('Number of labels', fontsize=16)

plt.xticks(unique\_labels)

plt.ylabel('Count of each labels', fontsize=16)

plt.title('Histogram of labels', fontsize=16)

plt.show()

from utils.feature\_extraction import get\_features\_dataframe

from utils.feature\_extraction import get\_audio\_features

trainfeature, trainlabel= get\_features\_dataframe(train\_df, sampling\_rate)

testfeature, testlabel= get\_features\_dataframe(test\_df, sampling\_rate)

#I have ran the above two lines to get featured dataframe and store it into pickle file for later purpose.it takes 30 mins approx to generate features.

trainfeatures = pd.read\_pickle('./features\_dataframe/trainfeatures')

trainlabel = pd.read\_pickle('./features\_dataframe/trainlabel')

testfeatures = pd.read\_pickle('./features\_dataframe/testfeatures')

testlabel = pd.read\_pickle('./features\_dataframe/testlabel')

trainfeatures.shape

trainfeatures = trainfeatures.fillna(0)

testfeatures = testfeatures.fillna(0)

# By using .ravel(): converting 1d to 2d e.g.(512,1) -> (512,).To prevent DataConversionWarning

x\_train = np.array(trainfeatures)

y\_train = np.array(trainlabel).ravel()

x\_test = np.array(testfeatures)

y\_test = np.array(testlabel).ravel()

y\_train[:5]

# One-hot encoding

lb = LabelEncoder()

y\_train = np\_utils.to\_categorical(lb.fit\_transform(y\_train))

y\_test = np\_utils.to\_categorical(lb.fit\_transform(y\_test))

y\_train[:5]

x\_traincnn =np.expand\_dims(x\_train, axis=2)

x\_testcnn= np.expand\_dims(x\_test, axis=2)

x\_traincnn.shape

from tensorflow.keras import optimizers

optimizers.RMSprop()

model = Sequential()

model.add(Conv1D(256, 5,padding='same',

                 input\_shape=(x\_traincnn.shape[1],x\_traincnn.shape[2])))

model.add(Activation('relu'))

model.add(Conv1D(128, 5,padding='same'))

model.add(Activation('relu'))

model.add(Dropout(0.1))

model.add(MaxPooling1D(pool\_size=(8)))

model.add(Conv1D(128, 5,padding='same',))

model.add(Activation('relu'))

model.add(Conv1D(128, 5,padding='same',))

model.add(Activation('relu'))

model.add(Flatten())

model.add(Dense(y\_train.shape[1]))

model.add(Activation('softmax'))

opt = keras.optimizers.RMSprop(lr=0.00001, decay=1e-6)

model.summary()

model.compile(loss='categorical\_crossentropy', optimizer=opt,metrics=['accuracy'])

cnnhistory=model.fit(x\_traincnn, y\_train, batch\_size=16, epochs=400, validation\_data=(x\_testcnn, y\_test))

plt.plot(cnnhistory.history['loss'])

plt.plot(cnnhistory.history['val\_loss'])

plt.title('model loss')

plt.ylabel('loss')

plt.xlabel('epoch')

plt.legend(['train', 'test'], loc='upper left')

plt.show()

model\_name = 'Emotion\_Voice\_Detection\_Model.h5'

save\_dir = os.path.join(os.getcwd(), 'saved\_models')

# Save model and weights

if not os.path.isdir(save\_dir):

    os.makedirs(save\_dir)

model\_path = os.path.join(save\_dir, model\_name)

model.save(model\_path)

print('Saved trained model at %s ' % model\_path)

import json

model\_json = model.to\_json()

with open("model.json", "w") as json\_file:

    json\_file.write(model\_json)

# loading json and creating model

from keras.models import model\_from\_json

json\_file = open('model.json', 'r')

loaded\_model\_json = json\_file.read()

json\_file.close()

loaded\_model = model\_from\_json(loaded\_model\_json)

# load weights into new model

loaded\_model.load\_weights("saved\_models/Emotion\_Voice\_Detection\_Model.h5")

print("Loaded model from disk")

# evaluate loaded model on test data

loaded\_model.compile(loss='categorical\_crossentropy', optimizer=opt, metrics=['accuracy'])

score = loaded\_model.evaluate(x\_testcnn, y\_test, verbose=0)

print("%s: %.2f%%" % (loaded\_model.metrics\_names[1], score[1]\*100))

preds = loaded\_model.predict(x\_testcnn,

                         batch\_size=32,

                         verbose=1)

preds

preds1=preds.argmax(axis=1)

preds1

abc = preds1.astype(int).flatten()

predictions = (lb.inverse\_transform((abc)))

preddf = pd.DataFrame({'predictedvalues': predictions})

preddf[:10]

actual=y\_test.argmax(axis=1)

abc123 = actual.astype(int).flatten()

actualvalues = (lb.inverse\_transform((abc123)))

actualdf = pd.DataFrame({'actualvalues': actualvalues})

actualdf[:10]

finaldf = actualdf.join(preddf)

finaldf[170:180]

finaldf.groupby('actualvalues').count()

finaldf.groupby('predictedvalues').count()

finaldf.to\_csv('Predictions.csv', index=False)

demo\_audio\_path = './demo\_audio2.wav'

demo\_mfcc, demo\_pitch, demo\_mag, demo\_chrom = get\_audio\_features(demo\_audio\_path,sampling\_rate)

mfcc = pd.Series(demo\_mfcc)

pit = pd.Series(demo\_pitch)

mag = pd.Series(demo\_mag)

C = pd.Series(demo\_chrom)

demo\_audio\_features = pd.concat([mfcc,pit,mag,C],ignore\_index=True)

demo\_audio\_features= np.expand\_dims(demo\_audio\_features, axis=0)

demo\_audio\_features= np.expand\_dims(demo\_audio\_features, axis=2)

demo\_audio\_features.shape

livepreds = loaded\_model.predict(demo\_audio\_features,

                         batch\_size=32,

                         verbose=1)

livepreds

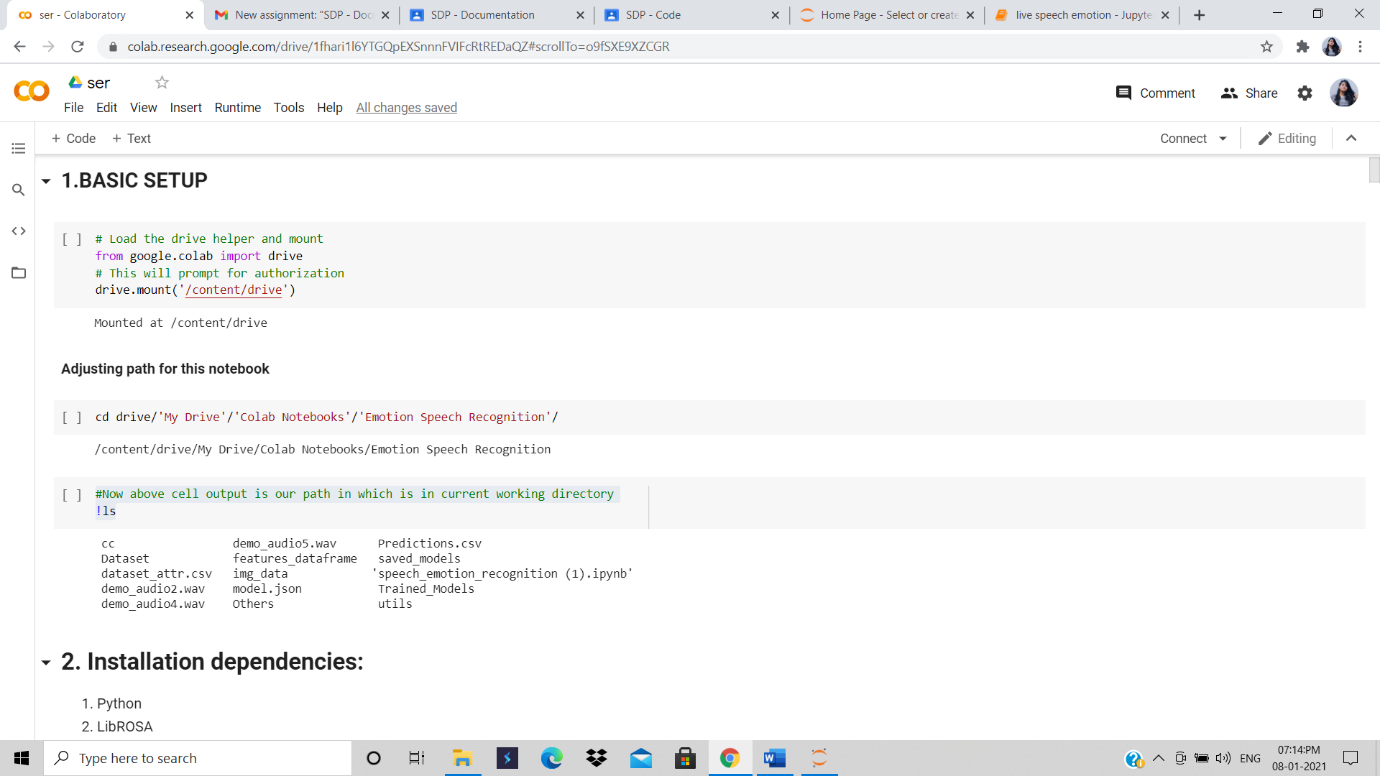
# emotions=["anger","disgust","fear","happy","neutral", "sad", "surprise"]

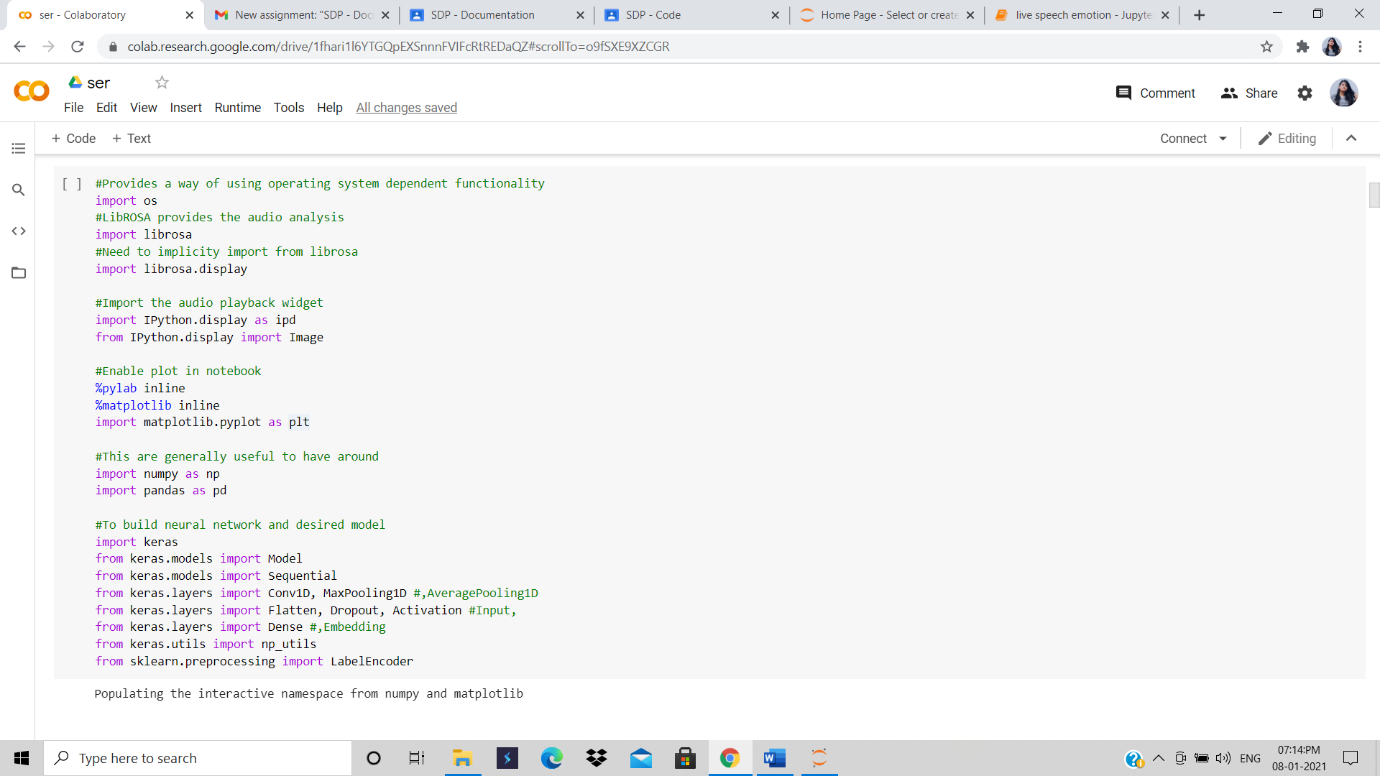
index = livepreds.argmax(axis=1).item()

index

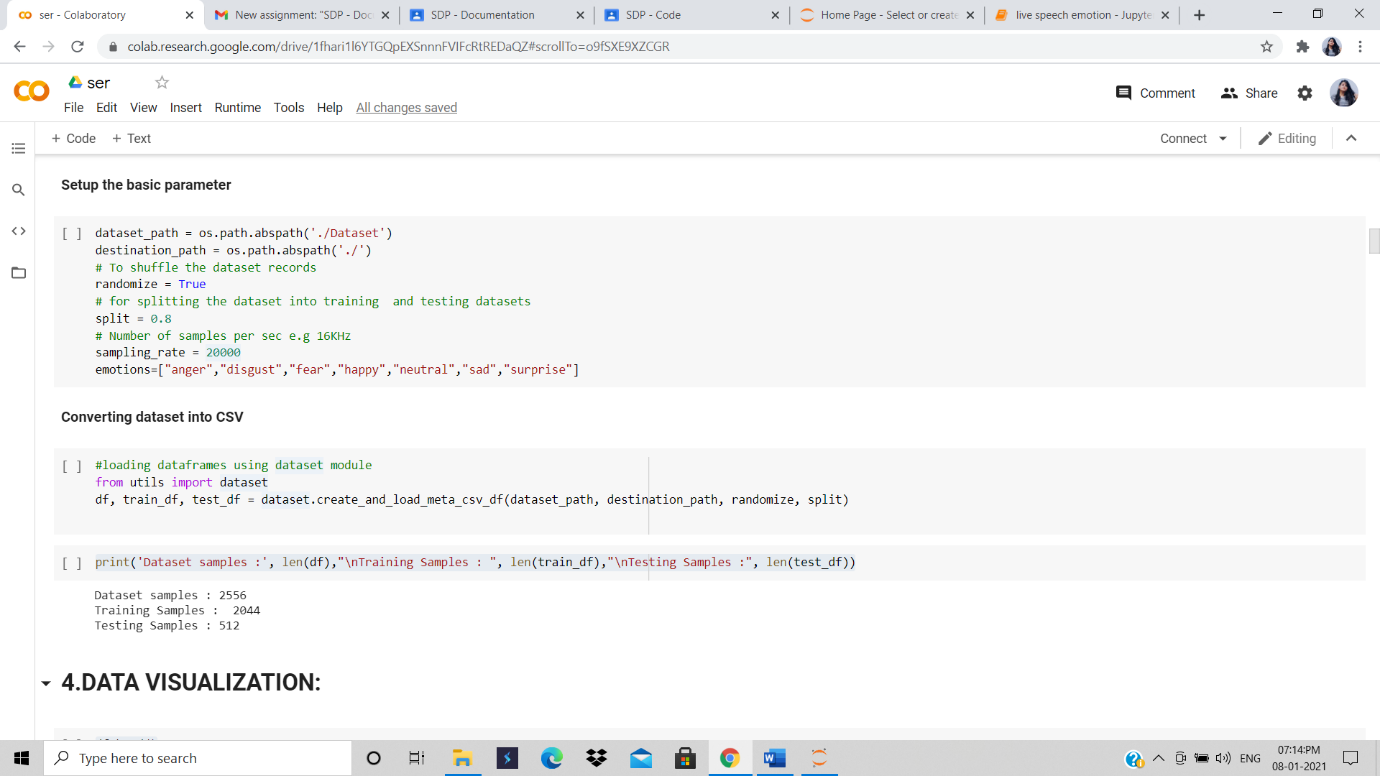
emotions[index]

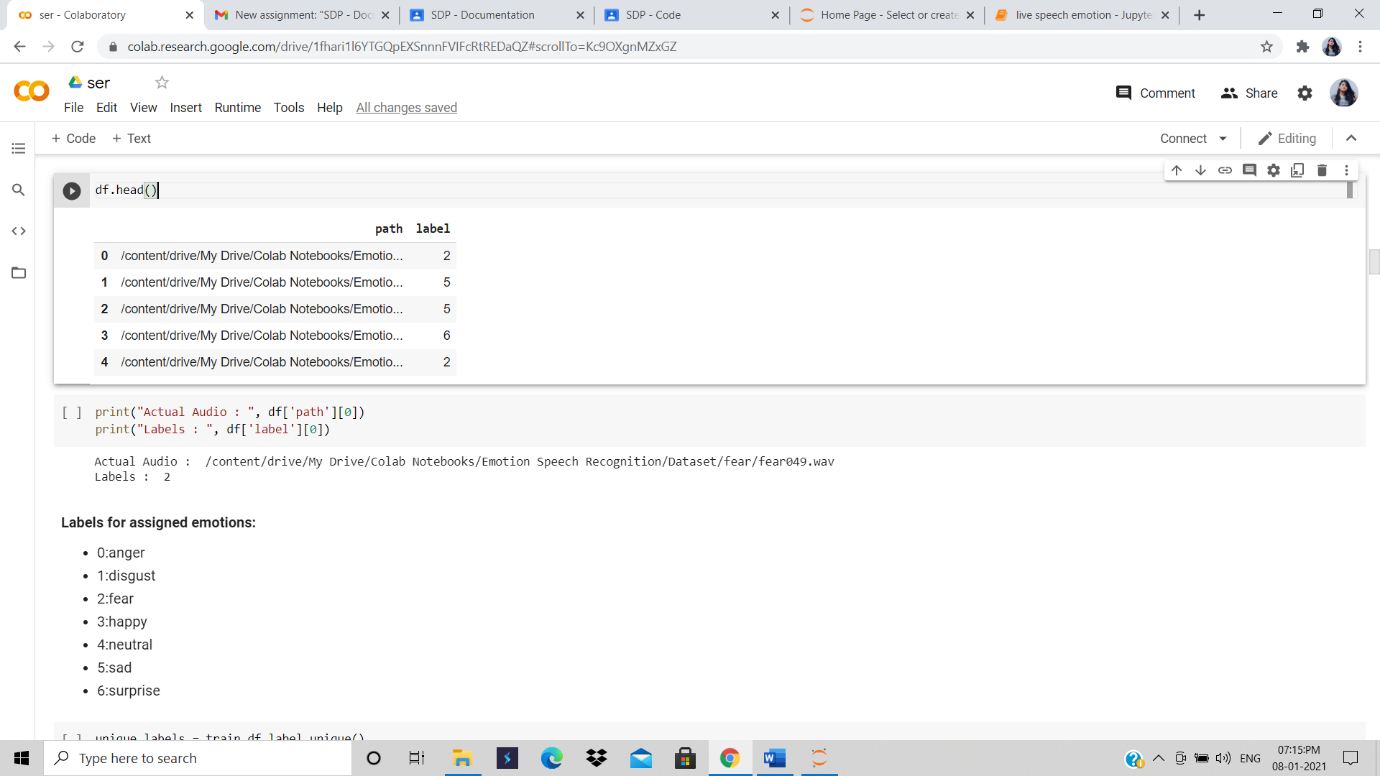
**Screenshots**

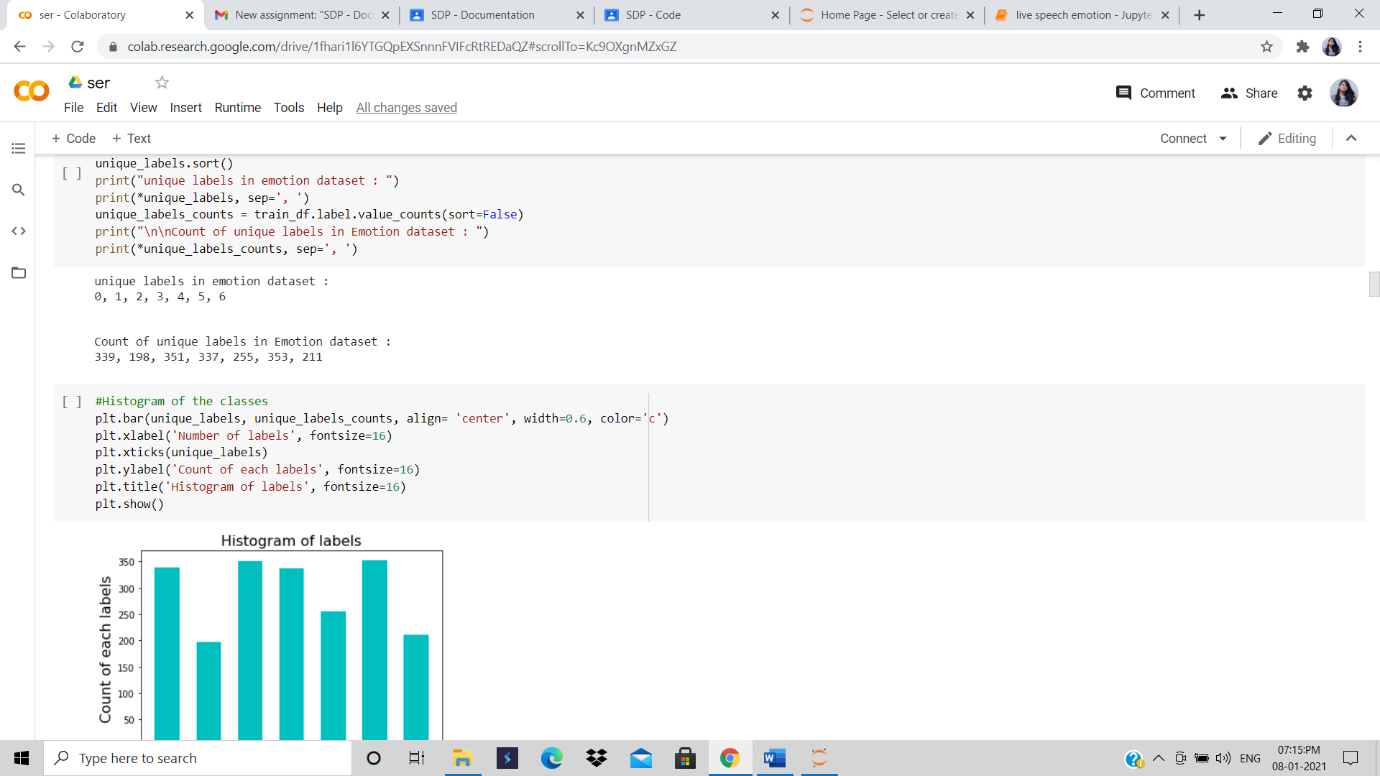


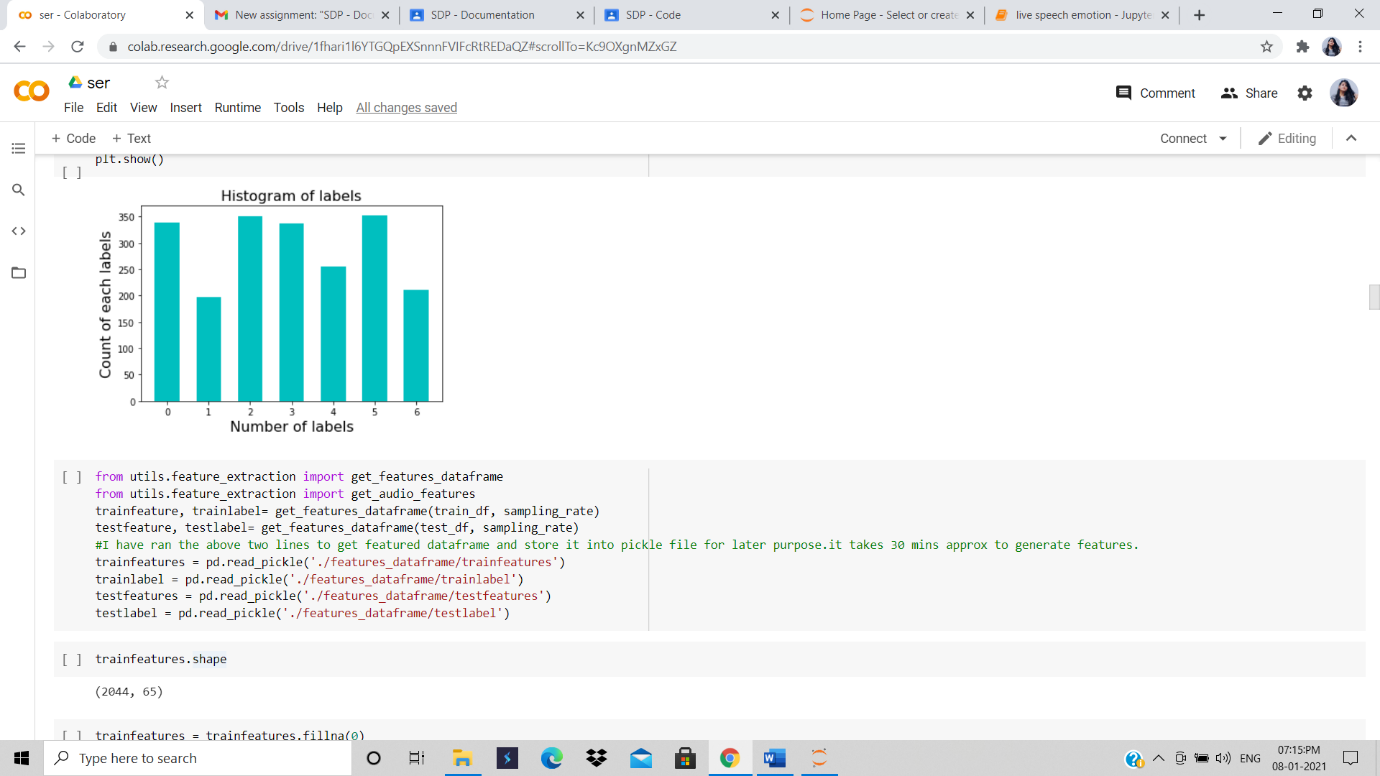


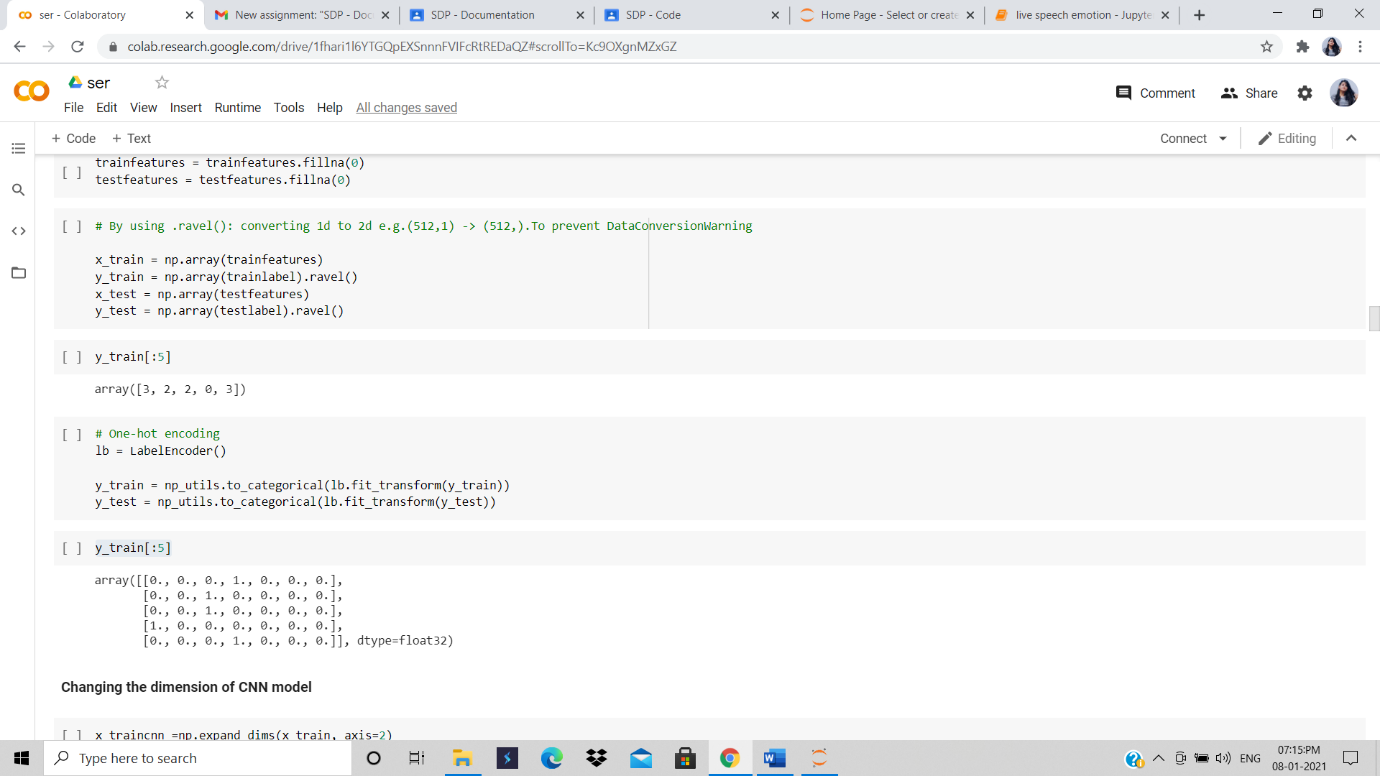


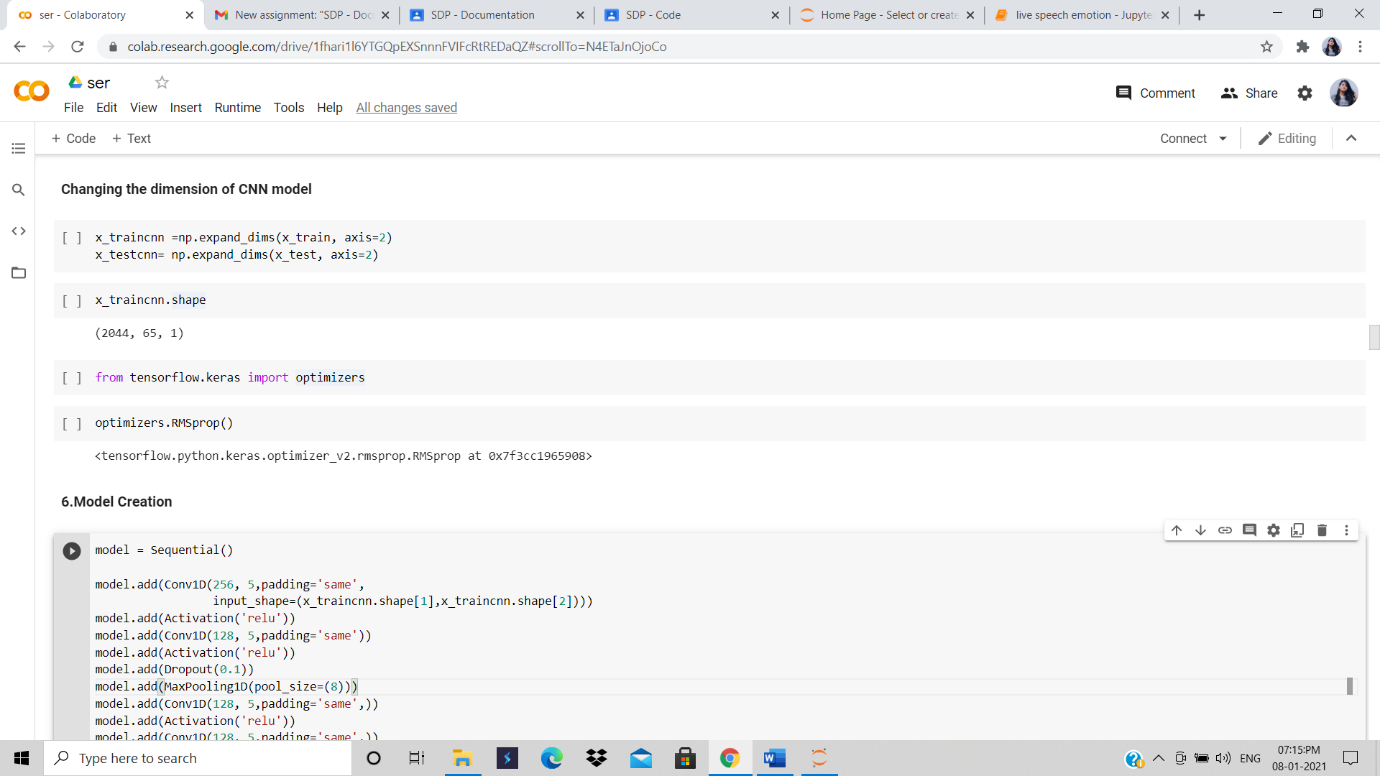




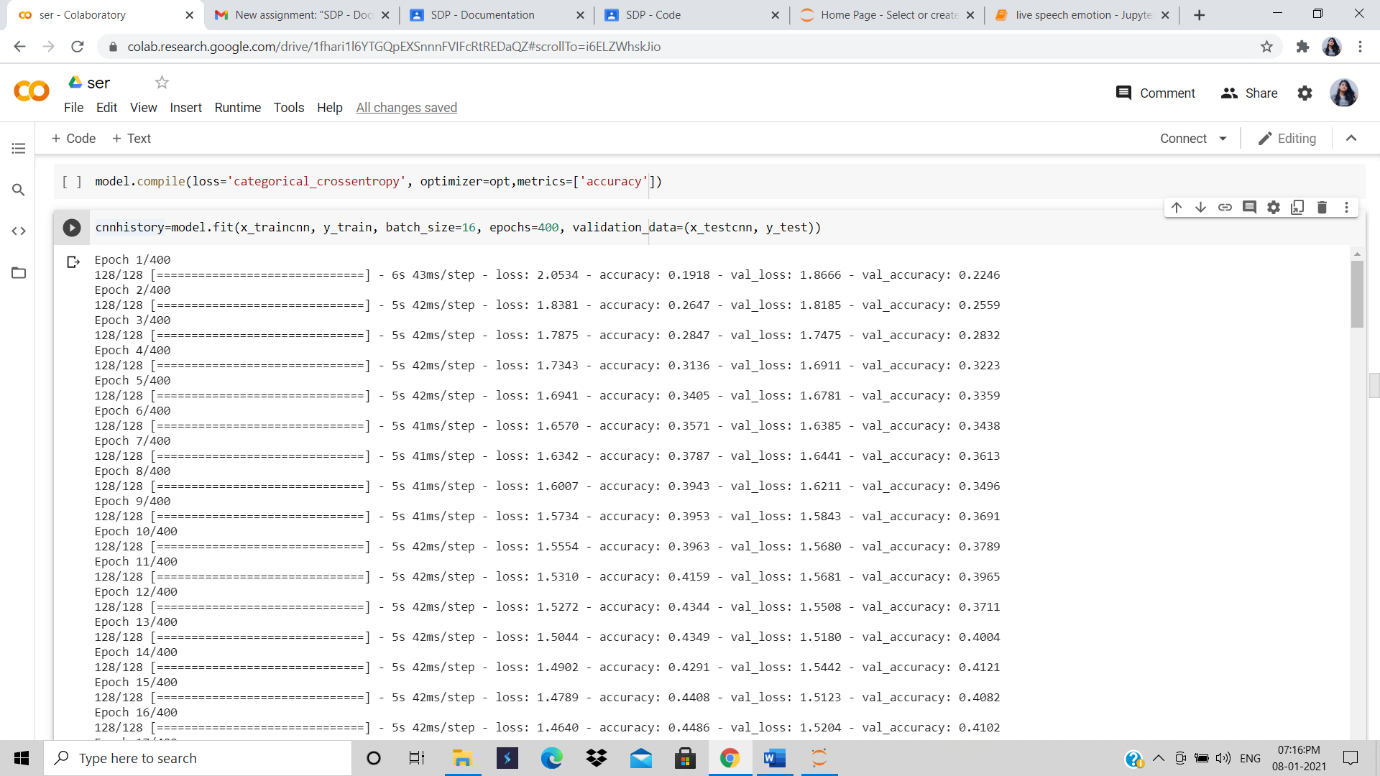


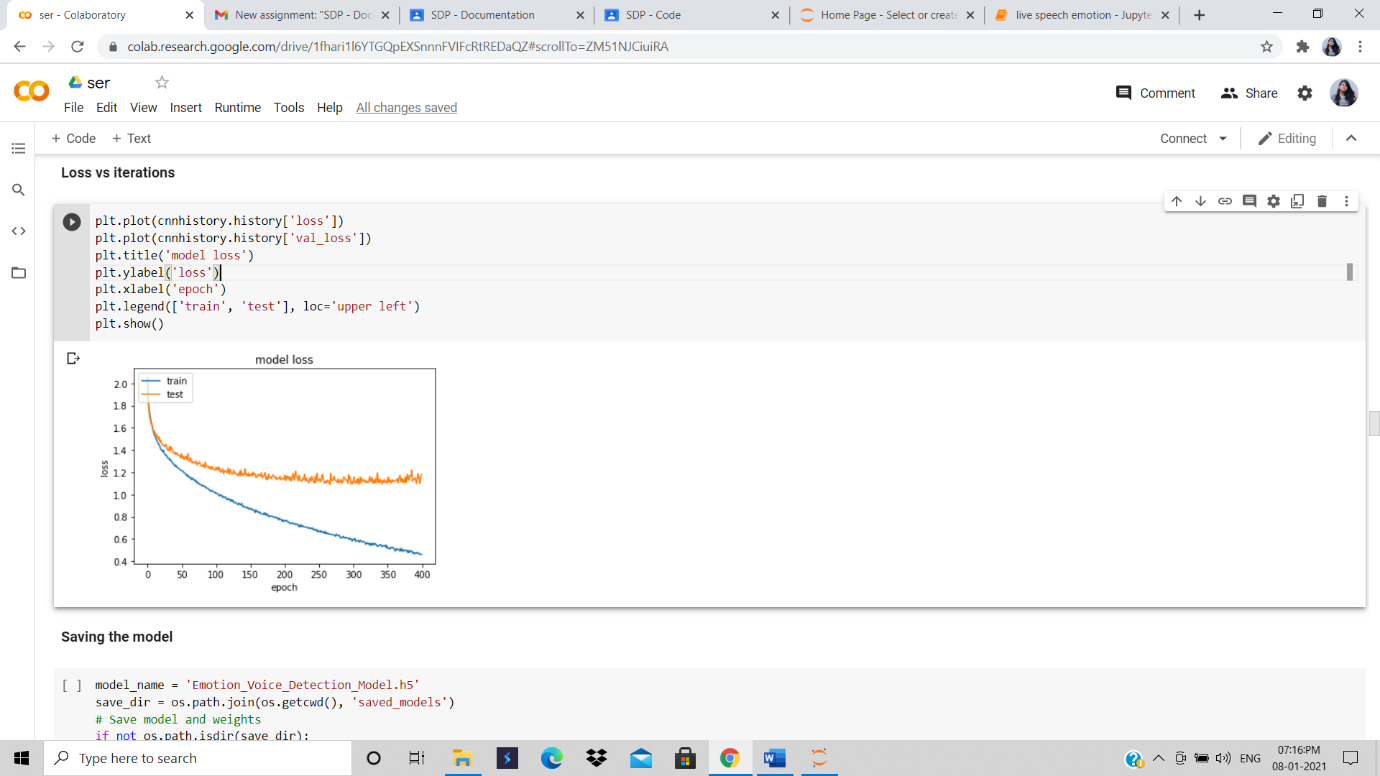


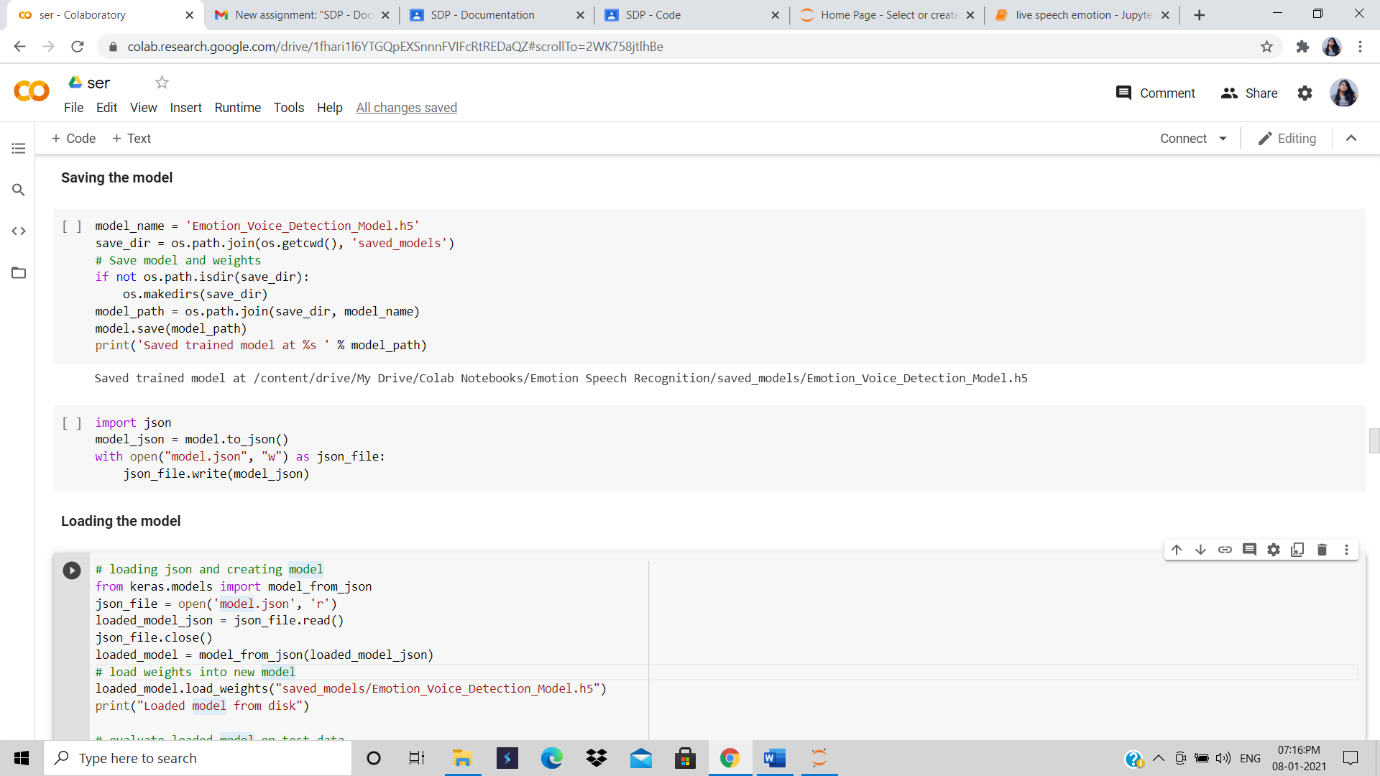


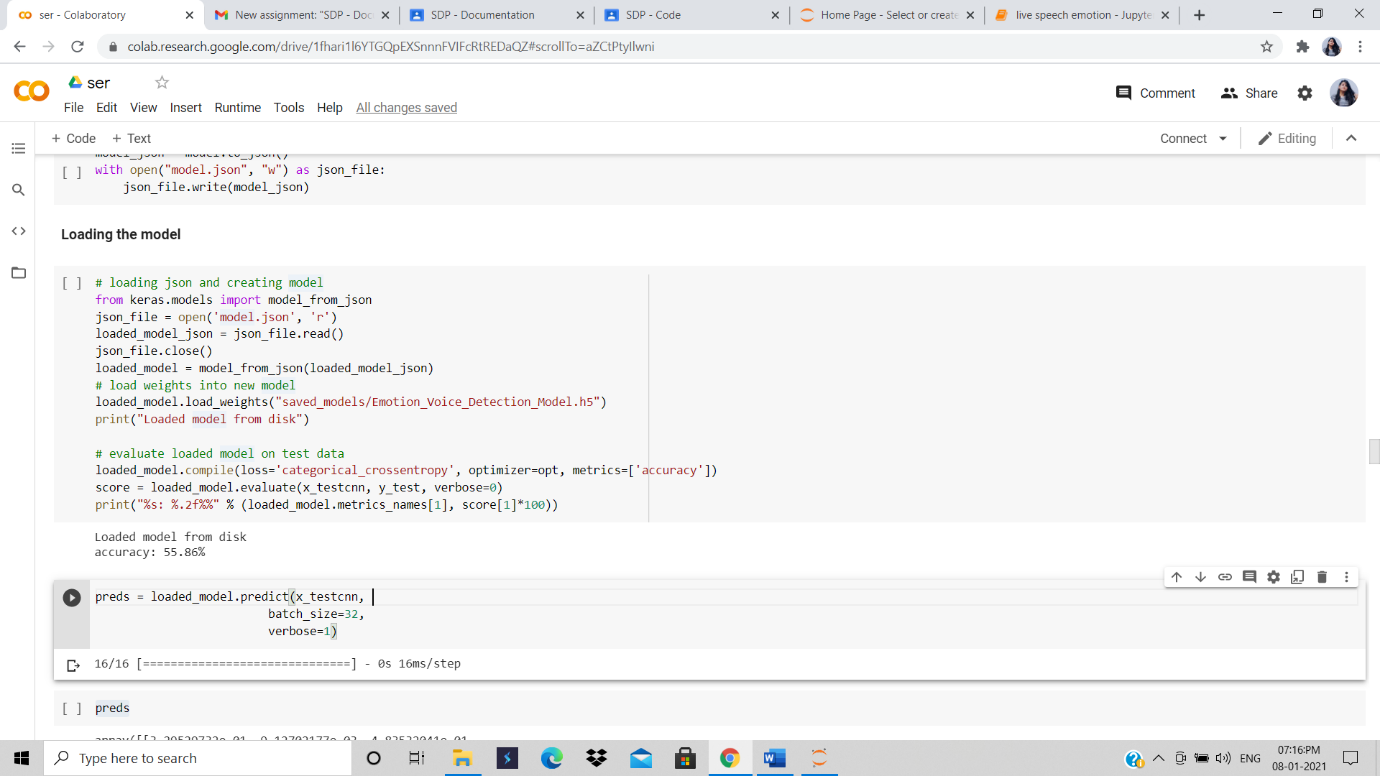


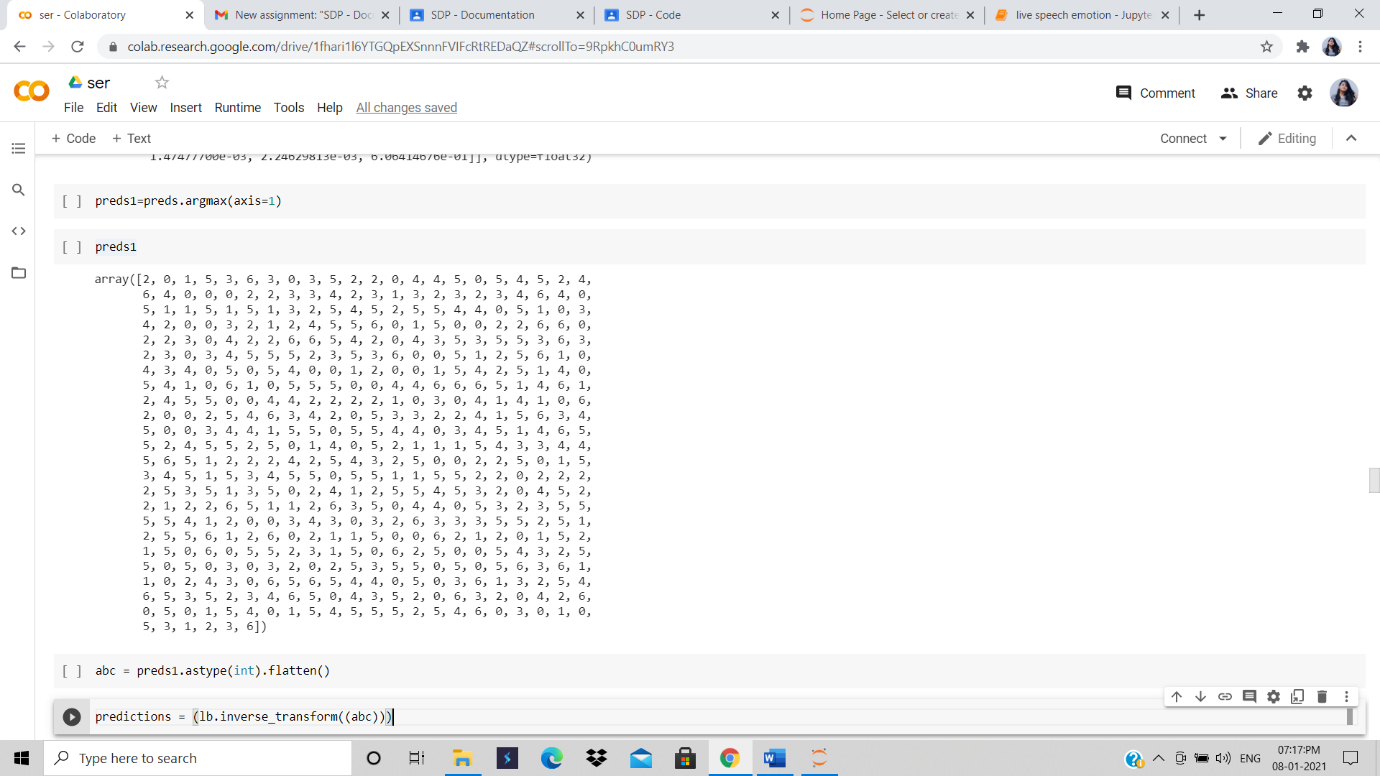


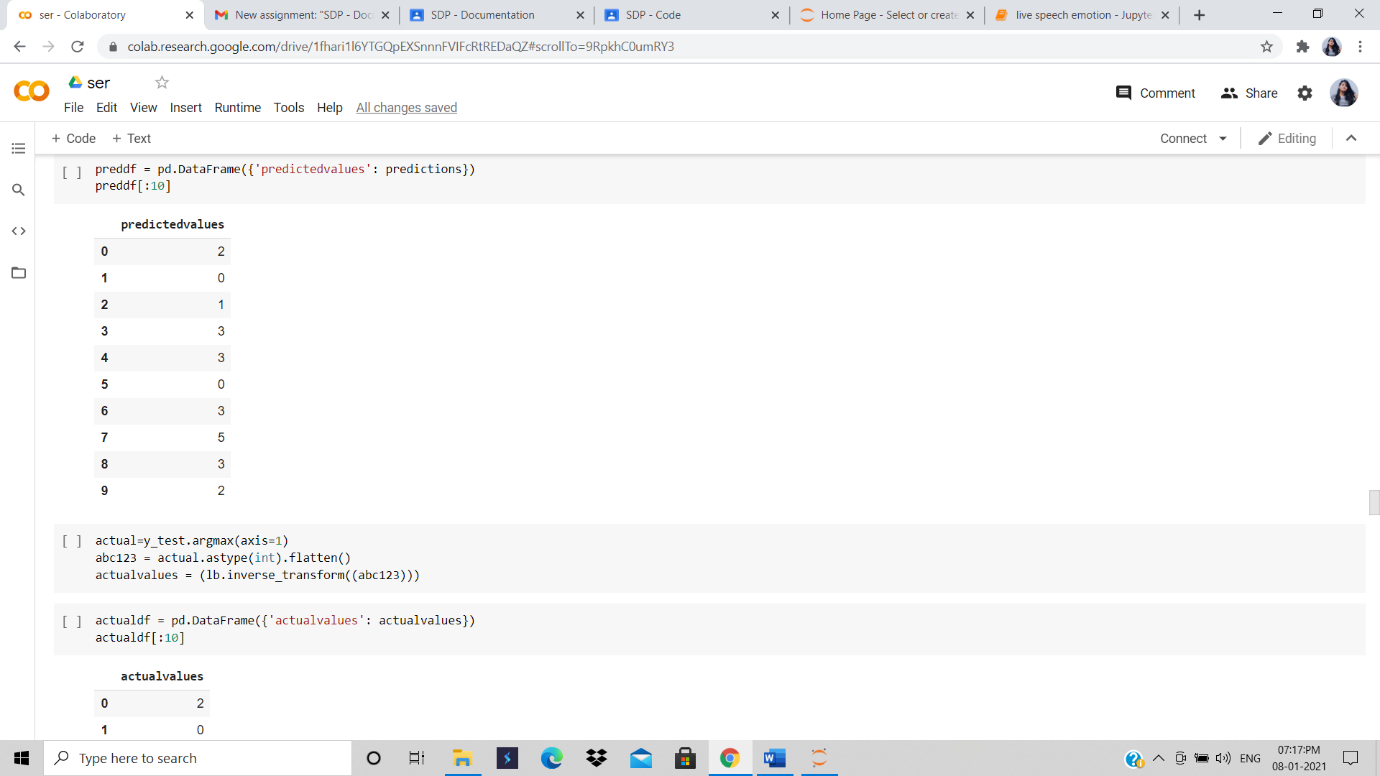






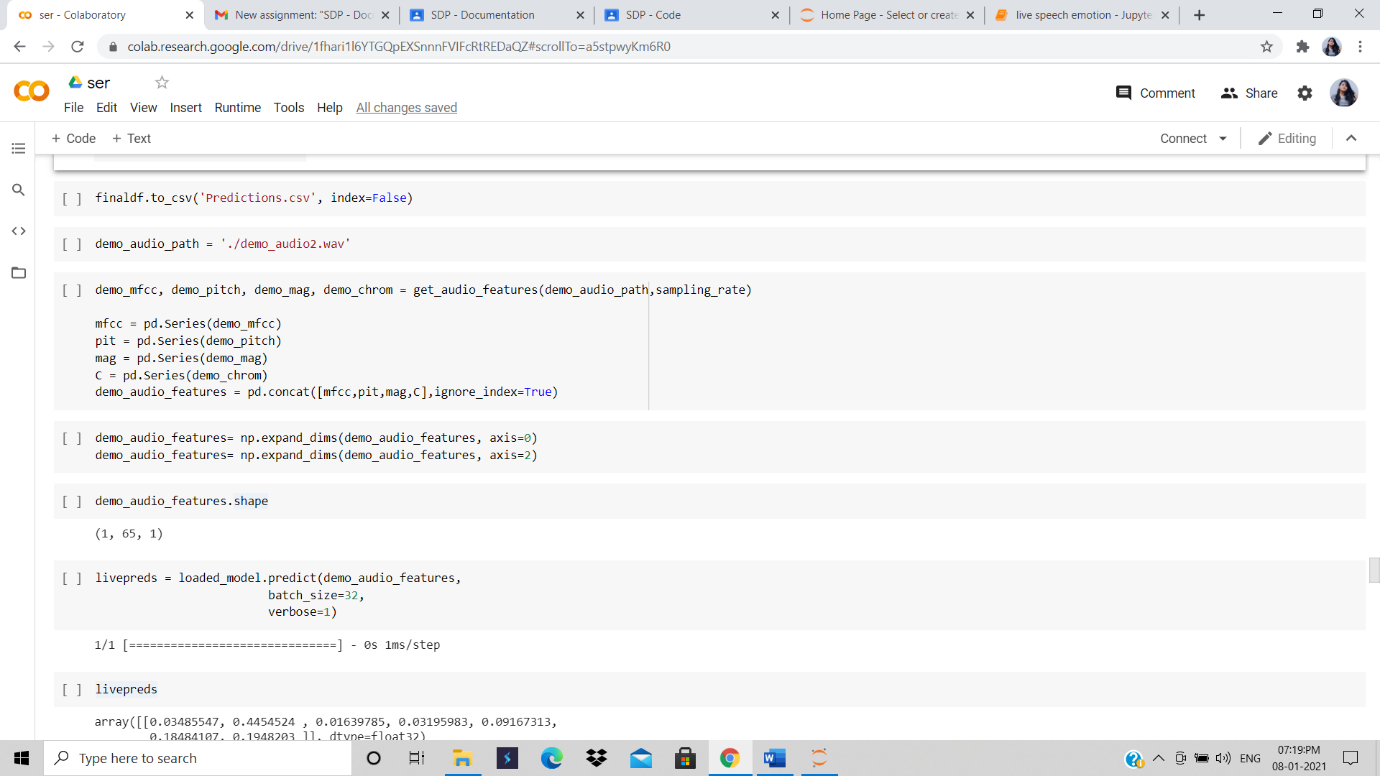


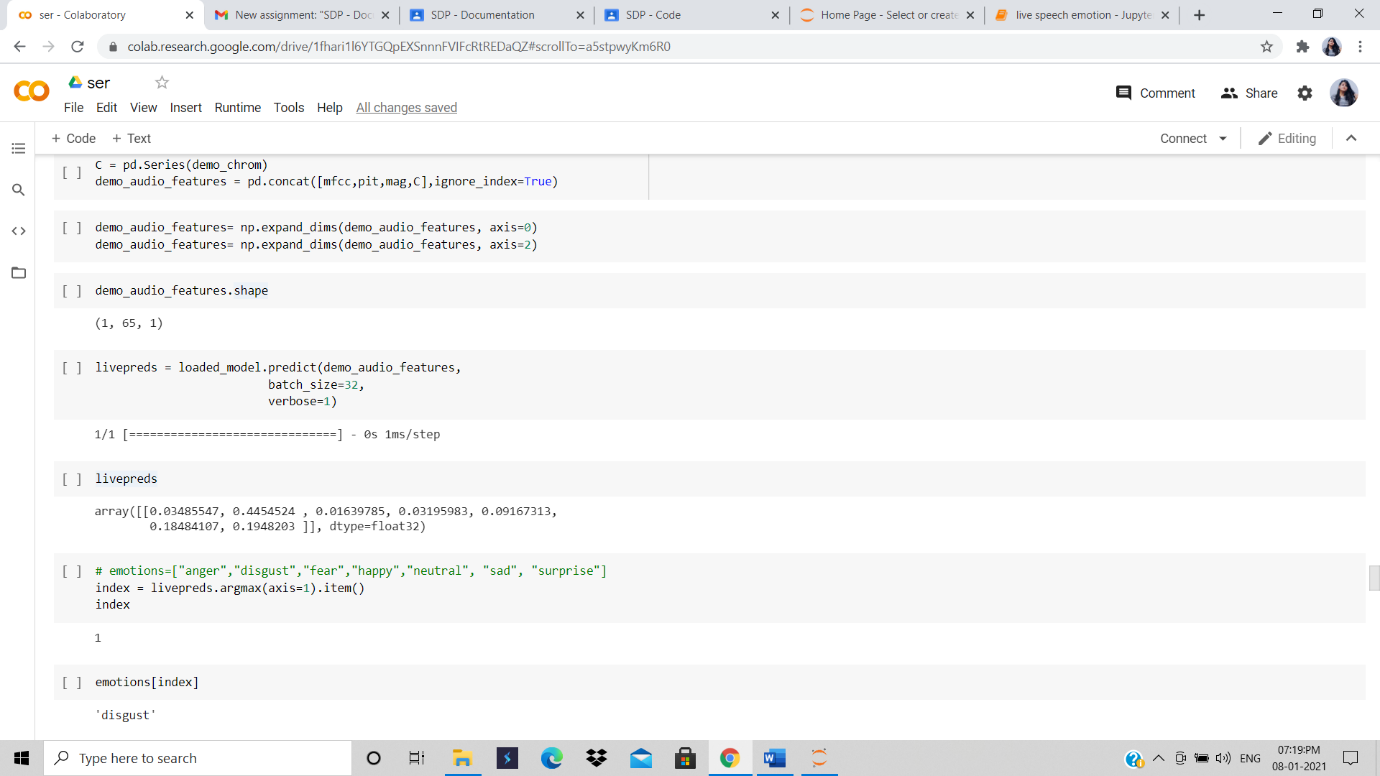












**Speech sentimental analysis- code**

import speech\_recognition as sr

def recognize\_speech\_from\_mic(recognizer, microphone):

# check that recognizer and microphone arguments are appropriate type

if not isinstance(recognizer, sr.Recognizer):

raise TypeError("`recognizer` must be `Recognizer` instance")

if not isinstance(microphone, sr.Microphone):

raise TypeError("`microphone` must be `Microphone` instance")

# adjust the recognizer sensitivity to ambient noise and record audio

# from the microphone

with microphone as source:

recognizer.adjust\_for\_ambient\_noise(source) # # analyze the audio source for 1 second

audio = recognizer.listen(source)

# set up the response object

response = {

"success": True,

"error": None,

"transcription": None

}

# try recognizing the speech in the recording

# if a RequestError or UnknownValueError exception is caught,

# update the response object accordingly

try:

response["transcription"] = recognizer.recognize\_google(audio)

except sr.RequestError:

# API was unreachable or unresponsive

response["success"] = False

response["error"] = "API unavailable/unresponsive"

except sr.UnknownValueError:

# speech was unintelligible

response["error"] = "Unable to recognize speech"

return response

recognizer = sr.Recognizer()

mic = sr.Microphone(device\_index=1)

response = recognize\_speech\_from\_mic(recognizer, mic)

print('\nSuccess : {}\nError : {}\n\nText from Speech\n{}\n\n{}' \

.format(response['success'],

response['error'],

'-'\*17,

response['transcription']))

temp\_str = ''

records\_all = []

while (temp\_str != 'bye'):

print('Speak it out')

response = recognize\_speech\_from\_mic(recognizer, mic)

if response['success']:

temp\_str = response['transcription']

print('You said :' ,temp\_str )

if (temp\_str != 'bye'):

records\_all.append(temp\_str)

print('Thanks for your suggetions')

print('Your suggetions on given product is:')

print(records\_all)

from textblob import TextBlob

def get\_sentiment(sentx):

'''

Utility function to classify sentiment of passed tweet

using textblob's sentiment method

'''

# create TextBlob object of passed text

analysis = TextBlob(sentx)

# set sentiment

if analysis.sentiment.polarity > 0:

return ('positive')

elif analysis.sentiment.polarity == 0:

return ('neutral')

else:

return ('negative')

# Select from collection

sentiments\_total = {'neutral': 0 , 'positive' : 0 , 'negative':0}

for recd\_sent in records\_all:

sentiment = get\_sentiment(recd\_sent)

print (sentiment,'===>',recd\_sent)

sentiments\_total[sentiment] = sentiments\_total[sentiment] + 1

print('###########################################')

print(sentiments\_total)

from matplotlib import pyplot as plt

slices = [sentiments\_total['neutral'],sentiments\_total['positive'],sentiments\_total['negative']]

activities = ['Neutral','Positive','Negative']

cols = ['c','m','r',]

plt.pie(slices,

labels=activities,

colors=cols,

shadow= True,

autopct='%1.1f%%')

plt.title('Sentiment Analysis of Voice')

plt.legend()

plt.show()

**Screenshots:**

