Configuration Manual

MSc Research Project
Data Analytics

Aggarwal Aditi
Student ID: x18137156

School of Computing
National College of Ireland

Supervisor: Prof. Christian Horn

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Aggarwal Aditi
Student ID:	x18137156
Programme:	Data Analytics
Year:	2019
Module:	MSc Research Project
Supervisor:	Prof. Christian Horn
Submission Due Date:	12/12/2019
Project Title:	Configuration Manual
Word Count:	1781
Page Count:	37

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	
Date:	11th December 2019

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	☐
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	☐
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	☐

Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Contents

Configuration Manual

Aggarwal Aditi
x18137156

1 Introduction

This configuration manual provides the instructions and information required to set up and implement the Convolutional Recurrent Neural Network (CRNN) for speech emotion classification (SER). The internal details such as process inputs and outputs, file storage, file manipulation, application development environment, and configurable parameters have been discussed in detail. This is a technical manual associated with the thesis report which describes the concepts and functionalities relevant to CRNN.

2 Application Environment

2.1 Hardware

- Processor: 2.3 GHz Dual-Core Intel Core i5

- Memory: 8 GB 2133 MHz LPDDR3

- Graphics: Intel Iris Plus Graphics 640 1536 MB

2.2 Software

- MacOS Catalina 10.15.1

- Anaconda: The open-source and free Anaconda distribution of R and python programming languages lets the user perform scientific computing such as machine learning applications, data science, predictive analysis and many more. This software includes data science packages compatible with macOS, Linux and Windows. The desktop graphical user interface (GUI) of Anaconda distribution is well-known as Anaconda Navigator. This GUI enables them to launch the application and manage the conda packages which save the users from using command-line commands. The navigator provides access to eight different applications by default. Jupyter Notebook is one of the applications installed in the navigator which has been used to design and implement the SER model. Python 3 is the latest version of Anaconda which is supported by Jupyter has been used for this project.

3 Application

3.1 Data Extraction

Purpose: To extract dataset of speech audio clips for eight classes of emotions as available online.

Data Source: A zip file of 1440 audio clips titled as 'Audio_Speech_Actors_01-24' is available at below-mentioned link:
https://zenodo.org/record/1188976#.XersLpP7Su4

Process Steps:
Download the zip file and unzip it. Save the unzipped folder in the desired location (folder is stored at location mentioned in 'filepath' variable).

3.2 Data Preparation

Source Code: Data_Preparation

Purpose: To transform the raw data into the suitable format for further processing.

Parameters:

- filepath: The folder path where the extracted raw data is stored.
- dirPath= Path of where a new folder titled 'paddedAudio' to be created.
- outPath= Path where all the padded audios get stored (same as dirPath).
- dirPath2= Path where the second directory gets created.
- toFolder = Path to move balanced data to a new folder.
- fromFolder = Path of padded audio files.
- emoPath= Path where eight different emotions of same speaker are stored. This folder was manually created to understand the audio signals visually. Each file of eight emotions is added to this folder.
- imgPath= This path is same as emoPath folder where all plots get stored.

Process Steps:

1. Plot a bar chart to check the issue of class imbalance across eight classes of emotion. (see section 4.1, cell [4] on page 6)
2. Librosa which is a python audio library has been used to read the audio signal. (see section 4.1, cells [5],[8]-[11] on page 7)
3. All audio clips are padded and transformed to the same length of 5.3s. (see section 4.1, cell [5] on page 7)

4. A new folder is created with the name of 'modelData' and all the files from 'paddedAudio' folder are copied to the new folder in order to secure the original dataset. (see section 4.1, cell [6] on page 8)

5. Then the contents of 'modelData' folder are used to resolve the class imbalance issue observed in step 1. The 'neutral' class of emotion comprised of 96 records is removed in order to have balanced classes.(see section 4.1, cell [7] on page 8)

6. To understand the difference between eight classes of emotions, raw audio is plotted for three different speech representatives: MFFCC, mel spectrogram and energy. (see section 4.1, cells [8]-[11] on page 9 to 13)

3.3 Feature Extraction

Source Code: Feature_Extract

Purpose: To extract five speech representatives from the preprocessed audio clips and save them in numpy (array) format.

Parameters:

- frame_size= The size of each frame in an audio signal. The frame size can be changed and tested for 25ms and 50ms frame as done in this research.
- source_path= The path where the second directory ('modelData' folder) consists prepared for model gets created (Number of audio files used for model: 1344).
- features_path= The path where numpy array of features and labels gets stored.

Process Steps:

1. Each audio clip obtained after pre-processing passes through all four functions: extract_features, feature_normalize, frames, one_hot_encode.

2. The foremost step loads the speech signal and normalizes it using 'feature_normalize' named function. (see section 4.2, cell [3] on page 14)

3. The two-digit emotion identifier is extracted from the audio file name as a string and stored in the variable named 'emotion'. The identifier is appended and the list of emotions for each file gets stored in 'labels' named variable as an array. (see section 4.2, cell [3] on page 14 and 15)

4. Then the audio signal is divided into frame size of 100ms using a function named 'frames' which ensures each frame has an overlap of 50% samples from the previous frame. The value of frame size is defined with 'frame_size' variable which is set to 2208 samples, 100ms in time. Total frames: 104(see section 4.2, cell [3] on page 14 and 15)

5. All the five features are extracted from each frame of a signal using a function named 'extract_features'. Then the extracted features are appended and the list of features is converted to an array. This array gets stored in 'features' named variable. (see section 4.2, cells [4]-[5] on page 16)
   Shape of features (1344, 104, 182) and Shape of labels: (1344,7)

6. Categorical labels are converted to binary vector using the function named 'one_hot_encode'. (see section 4.2, cell [6] on page 16)

7. Then obtained features and labels get stored in the desired location (features_path). (see section 4.2, cell [7] on page 17)

3.4 Model and Evaluation

Source Code: Model

Purpose: To classify seven classes of emotions based on extracted five speech representatives using the proposed model.

Parameters:

- features_path= The path where numpy array of features and labels gets stored.
- bestModel: The path for saving the best model obtained during training.

Process Steps:

1. The array of features and labels are loaded from the location (features_path). (see section 4.3, cell [3] on page 18)

2. Dataset is split into two: training dataset and testing dataset. 1008 files (75%) are used to train the model and 336 files (25%) for testing the model. (see section 4.3, cell [4] on page 18)

3. The values of six parameters that were optimized are set based on the outcome received in section 3.5. (see section 4.3, cell [5] on page 18)

4. The model was designed with three functions: CRNN_model_build, train_model, frames, show_summary_stats. (see section 4.3, cells [6]-[8] on page 19 and 20)

5. Once the values are set for the model, the function name 'train_model' model starts with setting up the input. The model is created with the function named 'CRNN_model_build' and gets the 3D input. Then the model gets trained on the training dataset and outputs the model and its history. (see section 4.3, cell [9] on page 21)

6. The accuracy of the model is evaluated and the variable named 'history' is passed to the function named 'show_summary_stats' which demonstrates the summary statistics. (see section 4.3, cells [10]-[11] on page 23)

7. Labels were transformed in desired form to make the confusion matrix. (see section 4.3, cells [12]-[15] on page 25)

8. The summary of predictions is displayed through a confusion matrix. (see section 4.3, cells [16]-[17] on page 25)

9. Dictionary is used to convert the labels into strings to make a the classification report which is generated to evaluate the performance of model based on the other metric. (see section 4.3, cells [18]-[20] on page 25 and 26)

3.5 Hyperparameter Optimization and Evaluation

Source Code: Optimization

Purpose: To identify optimal hyperparameters which control the learning algorithm.

Parameters:

- features_path= The path where numpy array of features and labels gets stored.
- outPath= The path where the CSV file with all the tried values and outcome gets stored.

Process Steps:

1. The array of features and labels are loaded from the location (features_path). (see section 4.4, cell [3] on page 28)
2. Dataset is split into two: training dataset and testing dataset. 1008 files (75%) are used to train the model and 336 files (25%) for testing the model. (see section 4.4, cell [4] on page 28)
3. This code focused on optimizing six parameters: epochs, activation, learning rate, Adam decay, dropout rate, and batch size. A set of default parameters is defined under the variable name 'default_parameters'. Each parameter is assigned with a range of values to find the optimal parameter values for the proposed model. (see section 4.4, cell [5] on page 28)
4. The model was designed CRNNmodelbuild. (see Feature_Extract, cell [6] on page 28)
5. Gaussian process is applied using 'gp_minimize' function which belongs to 'skopt' python library. This intends to find the minimum of a noisy function using a function named as 'fitness' over the range provided in the function's argument named as 'dimensions'. (see section 4.4, cells [7]-[9] on page 29 and 30)
6. 'fitness' function takes a single list of parameters as input and 'use_named_args' has been used as a decorator to call it directly with named arguments. This function is called 20 number of times to find the minimum, the value of 20 is set to the 'n_calls' arguments of 'gp_minimize' function. (see section 4.4, cells [7]-[9] on page 29 and 30)
7. The best accuracy out of those 20 evaluations is displayed with the set of best parameters. (see section 4.4, cells [10]-[11] on page 32)
8. The hyper tuned model outputs the best combination of parameters out of 50 evaluations. Those 20 evaluations get stored in the location provided in 'outPath' variable. The best set of hyper tuned parameters are used to train the model. (see section 4.4, cells [12]-[15] on page 32 to 34)
9. Labels were transformed in desired form to make the confusion matrix. (see section 4.4, cells [16]-[22] on page 36)
10. The summary of predictions is displayed through a confusion matrix. (see section 4.4, cells [23]-[24] on page 36)

11. Dictionary is used to convert the labels into strings to make a the classification report which is generated to evaluate the performance of model based on the other metric. (see section 4.4, cells [25]-[28] on page 36 and 37)

4 Code Artefacts

4.1 Data_Preparation

Data_Preparation

December 10, 2019

[1]:
```
# Data preparation
```

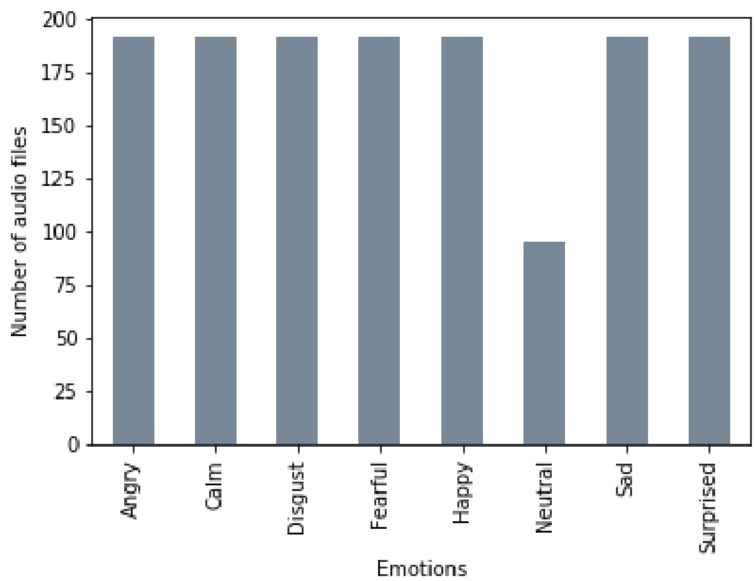
[2]:
```
# Pad audio files to same length and deleting 'neutral' class of emotion.
```

[3]:
```
# Import Libraries

import glob, os, numpy
from pydub import AudioSegment
import librosa
import matplotlib as mpl
import pandas as pd
%matplotlib inline
import collections
import matplotlib.pyplot as plt
import shutil
import librosa.display
import numpy as np
from scipy.io import wavfile as wav
import sklearn
```

[4]:
```
# Count of audio files in eight classes of emotion
# Data dictionary

emo={
  '01':'Neutral',
  '02':'Calm',
  '03':'Happy',
  '04':'Sad',
  '05':'Angry',
  '06':'Fearful',
  '07':'Disgust',
  '08':'Surprised'
}

labels=[]
filepath="./Desktop/Python/Audio_Speech_Actors_01-24/Actor_*/*.wav"
for file in glob.glob(filepath):
```

1

```python
    file_name=os.path.basename(file)
    emotion=file_name.split("-")[2]
    label=emo[emotion] # Number lables to name labels using data dictionary
    labels.append(label)

labelsDf = pd.DataFrame({'Emotions': labels}) # list to dataframe
barChart=labelsDf.groupby('Emotions', as_index=False).size()
barChart.plot(kind='bar',color='lightslategray')

plt.ylabel("Number of audio files")

plt.show()
```



[5]:
```python
# Create a new folder: "paddedAudio"
# Padded audio files are stored in a new folder.
# Each audio file is tranformed to the duration of 5.3seconds

dirPath1='./Desktop/Python/Code/PaddedAudio'
filepath="./Desktop/Python/Code/Audio_Speech_Actors_01-24/Actor_*/*.wav"
outPath='./Desktop/Python/Code/PaddedAudio/'
```

```
os.mkdir(dirPath1)

count=0
for file in glob.glob(filepath):
    y, sr = librosa.load(file)
    duration = librosa.get_duration(y=y, sr=sr) # Extracts the duration of audio
    duration  = duration * 1000                 # Converts duration in␣
 ↪milliseconds
    padding_ms = 5300 - duration  # Calculated the milliseconds of silence␣
 ↪needs to be added

    silence = AudioSegment.silent(duration=padding_ms)
    audio = AudioSegment.from_wav(file)
    padded = audio + silence  # Adds calculated silence after the audio
    file_name=os.path.basename(file)
    padded.export(outPath+ file_name, format='wav')
    count=count+1

print("Total", count, "files are padded and moved a new folder.")
```

```
Total 1440 files are padded and moved a new folder.
```

[6]:
```
# Make a copy of padded audio files

dirPath2='./Desktop/Python/Code/modelData'
path = os.getcwd()
toFolder = os.path.join(path, 'Desktop/Python/Code/modelData/')
fromFolder = os.path.join(path, 'Desktop/Python/Code/paddedAudio/')

os.mkdir(dirPath2)
print("New folder named 'modelData' is created.")

for f in os.listdir(fromFolder):
    shutil.copy2(os.path.join(fromFolder, f), toFolder)
print("All the audio files from 'paddedAudio' folder copied to 'modelData'␣
 ↪folder")
```

```
New folder named 'modelData' is created.
All the audio files from 'paddedAudio' folder copied to 'modelData' folder
```

[7]:
```
# Deletes neutral files code as 01 at the third position of file name.
# Remove imbalanced class.
modeldataPath="./Desktop/Python/Code/modelData/*.wav"
count=0
for file in glob.glob(modeldataPath):
    file_name=os.path.basename(file)
```

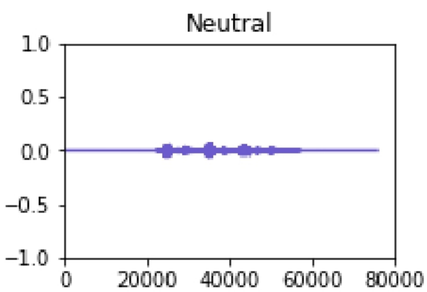
3

```
    emotion=file_name.split("-")[2] #splits the filename by "-" and picks up␣
↪the third value
    if emotion == '01':
        os.remove(file)
        count=count+1
print(count, "audio files belonged to neutral class are removed!")
```

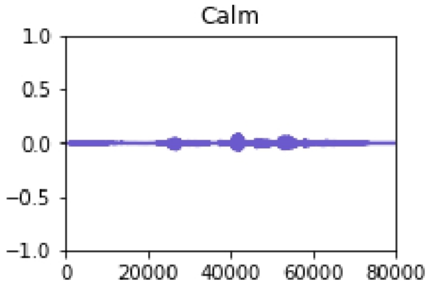
```
96 audio files belonged to neutral class are removed!
```

[8]:
```
# Plotting Waveforms
# dpi is set to 300 in order to use these plots for thesis report!

emoPath='./Desktop/Python/Code/emoPath/*.wav'
imgPath='./Desktop/Python/Code/emoPath/'
for file in glob.glob(emoPath):
    x, sr = librosa.load(file)
    plt.figure(figsize=(3,2))
    plt.plot(x, color='slateblue')
    ax = plt.axes()
    ax.set_facecolor("white")
    filename = os.path.basename(file)
    name = filename.split(".")[0]
    plt.title(name, color='black')
    plt.ylim(-1, 1)
    plt.xlim(0,80000)
    plt.savefig(imgPath+ 'wav' + name + '.png', dpi=300)
    plt.show()
```

```
/Users/aditiaggarwal/anaconda3/lib/python3.7/site-
packages/matplotlib/figure.py:98: MatplotlibDeprecationWarning:
Adding an axes using the same arguments as a previous axes currently reuses the
earlier instance.  In a future version, a new instance will always be created
and returned.  Meanwhile, this warning can be suppressed, and the future
behavior ensured, by passing a unique label to each axes instance.
  "Adding an axes using the same arguments as a previous axes "
```



4

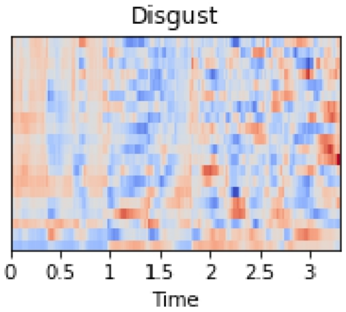


[9]:
```
# Plotting MFCCS

for file in glob.glob(emoPath):
    x , sr = librosa.load(file, sr=22050)
    print(type(x), type(sr))

    mfccs = librosa.feature.mfcc(x, sr=sr)
    #Displaying  the MFCCs:
    mfccs = sklearn.preprocessing.scale(mfccs, axis=1)
    plt.figure(figsize=(3,2))
    librosa.display.specshow(mfccs, sr=sr, x_axis='time')
    filename = os.path.basename(file)
    name = filename.split(".")[0]
    plt.title(name, color='black')
    plt.xlim(0,3.3)
    plt.savefig(imgPath+ 'MFCC' + name + '.png', dpi=300)
    plt.show()
```

```
<class 'numpy.ndarray'> <class 'int'>
```

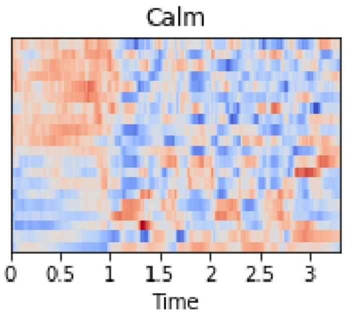
```
/Users/aditiaggarwal/anaconda3/lib/python3.7/site-
packages/sklearn/preprocessing/data.py:172: UserWarning: Numerical issues were
encountered when centering the data and might not be solved. Dataset may contain
too large values. You may need to prescale your features.
  warnings.warn("Numerical issues were encountered "
/Users/aditiaggarwal/anaconda3/lib/python3.7/site-
packages/sklearn/preprocessing/data.py:189: UserWarning: Numerical issues were
encountered when scaling the data and might not be solved. The standard
deviation of the data is probably very close to 0.
  warnings.warn("Numerical issues were encountered "
```

7




```
<class 'numpy.ndarray'> <class 'int'>
```

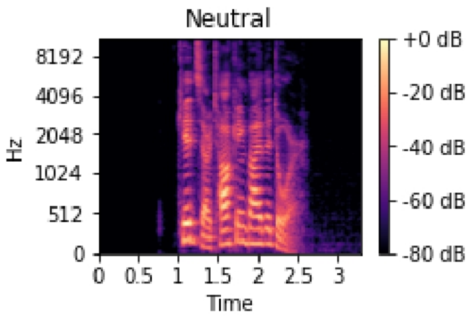
```
/Users/aditiaggarwal/anaconda3/lib/python3.7/site-
packages/sklearn/preprocessing/data.py:172: UserWarning: Numerical issues were
encountered when centering the data and might not be solved. Dataset may contain
too large values. You may need to prescale your features.
  warnings.warn("Numerical issues were encountered "
/Users/aditiaggarwal/anaconda3/lib/python3.7/site-
packages/sklearn/preprocessing/data.py:189: UserWarning: Numerical issues were
encountered when scaling the data and might not be solved. The standard
deviation of the data is probably very close to 0.
  warnings.warn("Numerical issues were encountered "
```

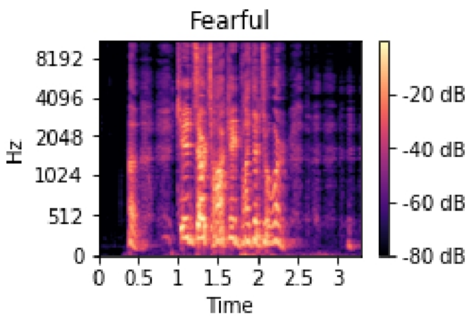


[10]:
```
# Mel Spectrogram
plt.figure()
for file in glob.glob(emoPath):
```

```python
n_mels = 128
x, sr = librosa.load(file)
filename = os.path.basename(file)
name = filename.split(".")[0]
plt.figure(figsize=(3,2))
plt.title(name, color='black')
S = librosa.feature.melspectrogram(y=x, sr=sr,
                                   n_mels=n_mels)
S_DB = librosa.power_to_db(S, ref=np.max)
librosa.display.specshow(S_DB, sr=sr,
                         x_axis='time', y_axis='mel')
plt.colorbar(format='%+2.0f dB')
plt.xlim(0,3.3)
plt.savefig(imgPath+ 'Mel' + name + '.png', dpi=300)
plt.show()
```

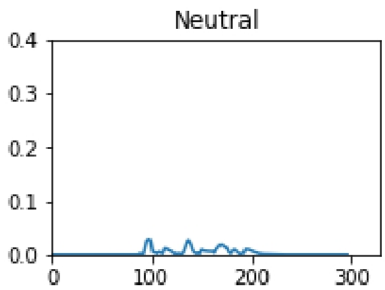
```
<Figure size 432x288 with 0 Axes>
```

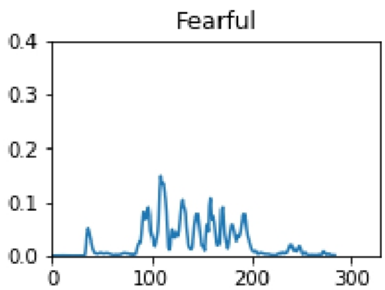




13

```
[11]: # Plotting Root Square Mean Energy:

for file in glob.glob(emoPath):
    x, sr = librosa.load(file)
    filename = os.path.basename(file)
    name = filename.split(".")[0]
    hop_length = 256
    frame_length = 512

    rmse = librosa.feature.rms(x, frame_length=frame_length,␣
↪hop_length=hop_length, center=True)
    rmse = rmse[0]

    plt.figure(figsize=(3,2))
    plt.plot(rmse)
    plt.title(name, color='black')
    plt.xlim(0,330)
    plt.ylim(0,.4)
    plt.savefig(imgPath+ 'Energy' + name + '.png', dpi=300)
```





16

4.2 Feature_Extract

Feature_Extract

December 10, 2019

[1]:
```python
# Feature Extraction
```

[2]:
```python
#Importing librarires

import glob
import pandas as pd
import os
import librosa
import numpy as np
from sklearn.model_selection import train_test_split
```

[3]:
```python
def feature_normalize(dataset):
    mu = np.mean(dataset, axis=0)
    sigma = np.std(dataset, axis=0)
    return (dataset - mu) / sigma

def frames(data, frame_size):
    start = 0
    while start < len(data):
        yield int(start), int(start + frame_size)
        start += (frame_size / 2) # stepping at half window size

def extract_features(file ,bands = 40):
    frame_size = 2208 # For 100ms frame
                      #frame size: 552 for 25ms frame
                      #frame size: 552*2 = 1104 for 50ms frame



    features=[]
    labels=[]

    feature1= []
    feature2= []
    feature3= []
    feature4= []
    feature5= []
```

1

```
    fileNew=glob.glob(file)

    for i, sound_file_path in enumerate(fileNew):
        sound_clip,s = librosa.load(sound_file_path)
        sound_clipN = feature_normalize(sound_clip)
        file_name=os.path.basename(sound_file_path)
        emotion=file_name.split("-")[2]
        mfccs = []
        mels=[]
        chromas=[]
        ton=[]
        rms=[]
        zcrs=[]

        # framing of audio clips
        for (start,end) in frames(sound_clip,frame_size):


            if(len(sound_clip[start:end]) == frame_size):

                signal = sound_clip[start:end]
                signalN = sound_clipN[start:end] # normalize the values

                mfcc = np.mean(librosa.feature.mfcc(y=signalN, sr=s, n_mfcc =␣
↪bands).T, axis=0)
                mel=np.mean(librosa.feature.melspectrogram(signal, sr=s).
↪T,axis=0)
                stft=np.abs(librosa.stft(signal)) #Short time fourier transform
                chroma=np.mean(librosa.feature.chroma_stft(S=stft, sr=s).
↪T,axis=0)
                rmse=np.mean(librosa.feature.rms(y=signal).T,axis=0) #␣
↪Root-Mean-Square Energy
                zcr=np.mean(librosa.feature.zero_crossing_rate(signal).
↪T,axis=0) # Zero-Crossing Rate

                # Append frames from each file
                mfccs.append(mfcc)
                mels.append(mel)
                chromas.append(chroma)
                rms.append(rmse)
                zcrs.append(zcr)

        # Emotion per file
        labels.append(emotion)

        # Append files
```

```
        feature1.append(mfccs)
        feature2.append(mels)
        feature3.append(chromas)
        feature4.append(rms)
        feature5.append(zcrs)


        # Convert list to array
        f1=np.array(feature1)
        f2=np.array(feature2)
        f3=np.array(feature3)
        f4=np.array(feature4)
        f5=np.array(feature5)



    print("MFCC:",f1.shape, "Mel:",f2.shape,"Chroma:",f3.shape,"RMSE:",f4.
 ↪shape,"ZCR:",f5.shape)

    features=np.concatenate([f1,f2,f3,f4,f5],axis=2)

    print("Shape of features array:", features.shape)
    return np.array(features), np.array(labels,dtype = np.str)

def one_hot_encode(labels):
    return np.asarray(pd.get_dummies(labels), dtype = np.float32)
```

[4]:
```
source_path="./Desktop/Python/Code/modelData/*.wav"
features,labels = extract_features(source_path, bands = 40)
print("All features and labels are extracted for 100ms frame size")
```

```
/Users/aditiaggarwal/anaconda3/lib/python3.7/site-
packages/librosa/core/pitch.py:146: UserWarning: Trying to estimate tuning from
empty frequency set.
  warnings.warn('Trying to estimate tuning from empty frequency set.')
```

```
MFCC: (1344, 104, 40) Mel: (1344, 104, 128) Chroma: (1344, 104, 12) RMSE: (1344,
104, 1) ZCR: (1344, 104, 1)
Shape of features array: (1344, 104, 182)
All features and labels are extracted for 100ms frame size
```

[5]:
```
print("Shape of labels:",labels.shape)
```

```
Shape of labels: (1344,)
```

[6]:
```
# Categorical varibales encoded as binary vectors.
labels = one_hot_encode(labels)
```

[7]:
```
# Save features and labels to 'Features' folder.

features_path='./Desktop/Python/Features/'

np.save(features_path + 'X_100ms',features)
np.save(features_path + 'Y_100ms',labels)
```

4.3 Model

Model

December 10, 2019

[1]:
```
# Convolutional Recurrent Neural Network
```

[2]:
```
#Importing libraries : Check libraries
from sklearn.model_selection import train_test_split
import numpy as np
import os
import keras
from keras.models import Sequential, Model
from keras.layers import Input, Dense, LSTM, Dropout, Activation␣
 ↪#TimeDistributed
from keras.layers import Conv1D, MaxPooling1D, Flatten, BatchNormalization
from keras.callbacks import ModelCheckpoint, TensorBoard, ReduceLROnPlateau
from keras.optimizers import Adam
import pandas as pd
from keras import regularizers
import librosa
import matplotlib.pyplot as plt
```

```
Using TensorFlow backend.
```

[3]:
```
# Fetch features and labels data
features_path='./Desktop/Python/Features/'

X=np.load(features_path + 'X_100ms.npy')
y=np.load(features_path +'Y_100ms.npy')
```

[4]:
```
# Holdout approach : train-test split
X_train, X_test, y_train, y_test = train_test_split(
        X, y , test_size=0.25, random_state=10)
```

[5]:
```
# Based on optimized outcome
learning_rate=0.00075
adam_decay=0.00052
batch_size = 24
epochs = 70
dropout_rate=0.00267
activation='relu'
```

1

[6]:
```
# Model design

num_layers = 3
kernel_size = 5
conv_filters = 56
lstm = 96
num_hidden = 64
l2penalty = 0.001
num_classes = 7


def CRNN_model_build(model_input):
    print('Building model...')
    layer = model_input

    ### 3 1D Convolution Layers
    for i in range(num_layers):
        # give name to the layers
        layer = Conv1D(
                filters=conv_filters,
                kernel_size=kernel_size,
                kernel_regularizer=regularizers.l2(l2penalty),
                name='convolution_' + str(i + 1)
            )(layer)
        layer = BatchNormalization(momentum=0.9)(layer)
        layer = Activation(activation)(layer)
        layer = MaxPooling1D(2)(layer)
        layer = Dropout(dropout_rate)(layer)
    ## LSTM Layer
    layer = LSTM(lstm,return_sequences=False)(layer)
    layer = Dropout(0.4)(layer)

    ## Dense Layer
    layer = Dense(num_hidden,kernel_regularizer=regularizers.l2(l2penalty), ␣
↪name='dense1')(layer)
    layer = Dropout(0.4)(layer)

    ## Softmax Output
    layer = Dense(num_classes)(layer)
    layer = Activation('softmax', name='output_realtime')(layer)
    model_output = layer
    model = Model(model_input, model_output)


    opt = Adam(lr=learning_rate, decay= adam_decay)
    model.compile(
            loss='categorical_crossentropy',
```

2

```
            optimizer=opt,
            metrics=['accuracy']
        )

    print(model.summary())
    return model
```

[7]:
```
def train_model(x_train, y_train, x_val, y_val):

    n_features = x_train.shape[2]
    n_frames=x_train.shape[1]
    input_shape = (n_frames, n_features)
    model_input = Input(input_shape, name='input')

    model = CRNN_model_build(model_input)
    bestModel='./Desktop/Python/model.h5'

    checkpoint_callback = ModelCheckpoint(bestModel, monitor='val_accuracy',␣
↪verbose=1,
                                          save_best_only=True, mode='max')

    reducelr_callback = ReduceLROnPlateau(
                monitor='val_acc',  factor=0.75, verbose=0, mode='auto',␣
↪cooldown=30, min_lr=0.0001
            )
    callbacks_list = [checkpoint_callback, reducelr_callback]

    # Fit the model and get training history.
    print('Training...')
    history = model.fit(x_train, y_train, batch_size=batch_size, epochs=epochs,
                        validation_data=(x_val, y_val), verbose=1,␣
↪callbacks=callbacks_list)

    return model, history
```

[8]:
```
def show_summary_stats(history):
    # List all data in history
    print(history.history.keys())

    # Summarize history for accuracy
    plt.plot(history.history['accuracy'])
    plt.plot(history.history['val_accuracy'])
    plt.title('model accuracy')
    plt.ylabel('accuracy')
    plt.xlabel('epoch')
    plt.legend(['train', 'test'], loc='upper left')
    plt.show()
```

```
# Summarize history for loss
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```

[9]:
```
model, history  = train_model(X_train, y_train, X_test, y_test)
```

```
WARNING: Logging before flag parsing goes to stderr.
W1210 04:41:12.908634 4787482048 deprecation.py:506] From
/Users/aditiaggarwal/anaconda3/lib/python3.7/site-
packages/tensorflow_core/python/ops/resource_variable_ops.py:1630: calling
BaseResourceVariable.__init__ (from tensorflow.python.ops.resource_variable_ops)
with constraint is deprecated and will be removed in a future version.
Instructions for updating:
If using Keras pass *_constraint arguments to layers.
W1210 04:41:13.089951 4787482048 module_wrapper.py:139] From
/Users/aditiaggarwal/anaconda3/lib/python3.7/site-
packages/keras/backend/tensorflow_backend.py:4070: The name tf.nn.max_pool is
deprecated. Please use tf.nn.max_pool2d instead.


Building model…
Model: "model_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input (InputLayer)           (None, 104, 182)          0
_________________________________________________________________
convolution_1 (Conv1D)       (None, 100, 56)           51016
_________________________________________________________________
batch_normalization_1 (Batch (None, 100, 56)           224
_________________________________________________________________
activation_1 (Activation)    (None, 100, 56)           0
_________________________________________________________________
max_pooling1d_1 (MaxPooling1 (None, 50, 56)            0
_________________________________________________________________
dropout_1 (Dropout)          (None, 50, 56)            0
_________________________________________________________________
convolution_2 (Conv1D)       (None, 46, 56)            15736
_________________________________________________________________
batch_normalization_2 (Batch (None, 46, 56)            224
_________________________________________________________________
activation_2 (Activation)    (None, 46, 56)            0
```

```
_________________________________________________________________
max_pooling1d_2 (MaxPooling1 (None, 23, 56)            0
_________________________________________________________________
dropout_2 (Dropout)          (None, 23, 56)            0
_________________________________________________________________
convolution_3 (Conv1D)       (None, 19, 56)            15736
_________________________________________________________________
batch_normalization_3 (Batch (None, 19, 56)            224
_________________________________________________________________
activation_3 (Activation)    (None, 19, 56)            0
_________________________________________________________________
max_pooling1d_3 (MaxPooling1 (None, 9, 56)             0
_________________________________________________________________
dropout_3 (Dropout)          (None, 9, 56)             0
_________________________________________________________________
lstm_1 (LSTM)                (None, 96)                58752
_________________________________________________________________
dropout_4 (Dropout)          (None, 96)                0
_________________________________________________________________
dense1 (Dense)               (None, 64)                6208
_________________________________________________________________
dropout_5 (Dropout)          (None, 64)                0
_________________________________________________________________
dense_1 (Dense)              (None, 7)                 455
_________________________________________________________________
output_realtime (Activation) (None, 7)                 0
=================================================================
Total params: 148,575
Trainable params: 148,239
Non-trainable params: 336
_________________________________________________________________
None
Training…

W1210 04:41:17.092154 4787482048 module_wrapper.py:139] From
/Users/aditiaggarwal/anaconda3/lib/python3.7/site-
packages/keras/backend/tensorflow_backend.py:422: The name tf.global_variables
is deprecated. Please use tf.compat.v1.global_variables instead.


Train on 1008 samples, validate on 336 samples
Epoch 1/70
1008/1008 [==============================] - 5s 5ms/step - loss: 2.1326 -
accuracy: 0.2361 - val_loss: 1.8650 - val_accuracy: 0.4048

Epoch 00001: val_accuracy improved from -inf to 0.40476, saving model to
./Desktop/Python/model.h5
Epoch 2/70
  72/1008 [=>…] - ETA: 2s - loss: 1.8439 - accuracy:
```

5

```
accuracy: 0.9960 - val_loss: 1.1390 - val_accuracy: 0.7827

Epoch 00065: val_accuracy did not improve from 0.81845
Epoch 66/70
1008/1008 [==============================] - 2s 2ms/step - loss: 0.2048 -
accuracy: 0.9851 - val_loss: 1.3132 - val_accuracy: 0.7649

Epoch 00066: val_accuracy did not improve from 0.81845
Epoch 67/70
1008/1008 [==============================] - 2s 2ms/step - loss: 0.1785 -
accuracy: 0.9950 - val_loss: 1.2227 - val_accuracy: 0.7917

Epoch 00067: val_accuracy did not improve from 0.81845
Epoch 68/70
1008/1008 [==============================] - 2s 2ms/step - loss: 0.1762 -
accuracy: 0.9950 - val_loss: 1.1968 - val_accuracy: 0.7857

Epoch 00068: val_accuracy did not improve from 0.81845
Epoch 69/70
1008/1008 [==============================] - 2s 2ms/step - loss: 0.1856 -
accuracy: 0.9950 - val_loss: 1.0764 - val_accuracy: 0.8065

Epoch 00069: val_accuracy did not improve from 0.81845
Epoch 70/70
1008/1008 [==============================] - 2s 2ms/step - loss: 0.1807 -
accuracy: 0.9960 - val_loss: 1.0935 - val_accuracy: 0.8065

Epoch 00070: val_accuracy did not improve from 0.81845
```

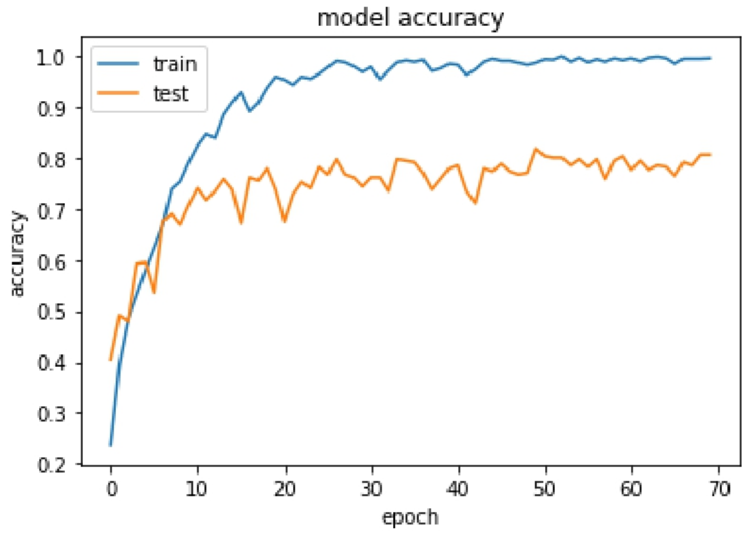
[10]:
```
accuracy = model.evaluate(X_test,y_test)
print('Test set\n  Loss: {:0.3f}\n  Accuracy: {:0.3f}'.
 ↪format(accuracy[0],accuracy[1]))
```

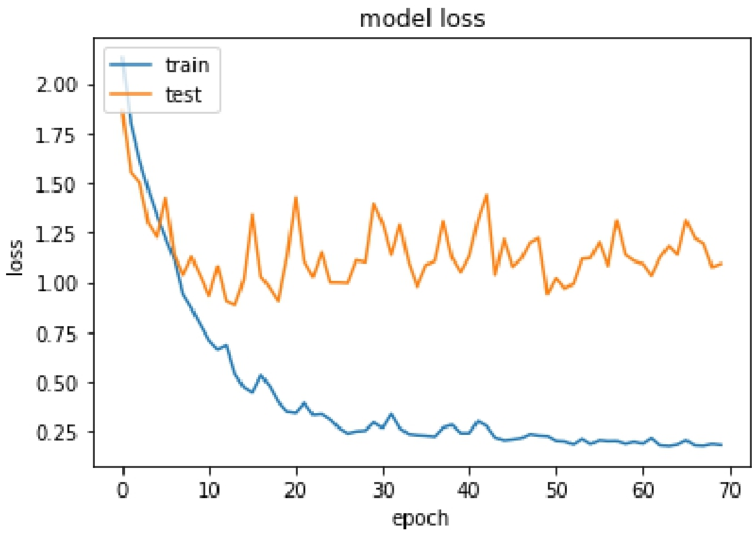
```
336/336 [==============================] - 0s 594us/step
Test set
  Loss: 1.094
  Accuracy: 0.807
```

[11]:
```
show_summary_stats(history)
```

```
dict_keys(['val_loss', 'val_accuracy', 'loss', 'accuracy', 'lr'])
```

13


model accuracy
train
test
accuracy
epoch
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0
10
20
30
40
50
60
70


model loss
train
test
loss
epoch
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0
10
20
30
40
50
60
70

14

[12]:
```
y_pred=model.predict(X_test)
y_pred = np.argmax(y_pred, axis=1)
```

[13]:
```
#categorical to binary vector
def one_hot_encode(labels):
    return np.asarray(pd.get_dummies(labels), dtype = np.float32)

predictions=one_hot_encode(y_pred)
```

[14]:
```
predictions=[np.where(r==1)[0][0] for r in predictions]
```

[15]:
```
Ytest=[np.where(r==1)[0][0] for r in y_test]
```

[16]:
```
from sklearn.metrics import confusion_matrix

cm=confusion_matrix(Ytest, predictions )

index = [  'calm','happy', 'sad','angry','fearful','disgust','surprised']
columns = [  'calm','happy', 'sad','angry','fearful','disgust','surprised']
```

[17]:
```
import pandas as pd
confusionMatrix = pd.DataFrame(cm,index,columns)
confusionMatrix
```

[17]:

	calm	happy	sad	angry	fearful	disgust	surprised
calm	47	0	0	0	0	1	0
happy	0	29	6	2	4	0	8
sad	6	2	32	1	3	1	2
angry	0	0	0	46	3	3	1
fearful	0	1	4	3	43	1	3
disgust	1	0	2	0	1	38	0
surprised	0	2	1	0	1	2	36

[18]:
```
emotions={

  0:'calm',
  1:'happy',
  2:'sad',
  3:'angry',
  4:'fearful',
  5:'disgust',
  6:'surprised'
}
```

[19]:
```
# numbers to emotion names
import numpy as np
test=[]
```

15

```
for i, item in enumerate(Ytest):
    test.append(emotions[item])

pred=[]

for i, item in enumerate(predictions):
    pred.append(emotions[item])
```

[20]:
```
from sklearn.metrics import classification_report
report = classification_report(test, pred)
print(report)
```

```
              precision    recall  f1-score   support

       angry       0.88      0.87      0.88        53
        calm       0.87      0.98      0.92        48
     disgust       0.83      0.90      0.86        42
     fearful       0.78      0.78      0.78        55
       happy       0.85      0.59      0.70        49
         sad       0.71      0.68      0.70        47
   surprised       0.72      0.86      0.78        42

    accuracy                           0.81       336
   macro avg       0.81      0.81      0.80       336
weighted avg       0.81      0.81      0.80       336
```

[ ]:
```

```

4.4 Optimization

Optimization

December 10, 2019

[1]:
```
# Hyperparameter Optimization: Bayesian optimization
```

[2]:
```
#Importing libraries : check them
from sklearn.model_selection import train_test_split
import numpy as np
import os
from keras.models import Sequential, Model
from keras.layers import Input, Dense, TimeDistributed, LSTM, Dropout,␣
↪Activation
from keras.layers import Conv1D, Flatten, BatchNormalization,MaxPooling1D
from keras.layers.advanced_activations import ELU
from keras.callbacks import ModelCheckpoint, TensorBoard, ReduceLROnPlateau
from keras.optimizers import Adam
import pandas as pd
from keras import regularizers
from keras.wrappers.scikit_learn import KerasClassifier
import skopt
# !pip install scikit-optimize if  necessary
from skopt import gbrt_minimize, gp_minimize
from skopt.utils import use_named_args
from skopt.space import Real, Categorical, Integer
import librosa
import keras
import tensorflow
from tensorflow.python.keras import backend as K
from sklearn.metrics import confusion_matrix
```

```
Using TensorFlow backend.
/Users/aditiaggarwal/anaconda3/lib/python3.7/site-
packages/sklearn/externals/joblib/__init__.py:15: DeprecationWarning:
sklearn.externals.joblib is deprecated in 0.21 and will be removed in 0.23.
Please import this functionality directly from joblib, which can be installed
with: pip install joblib. If this warning is raised when loading pickled models,
you may need to re-serialize those models with scikit-learn 0.21+.
  warnings.warn(msg, category=DeprecationWarning)
```

1

[3]:
```
# Fetch features and labels data
features_path='./Desktop/Python/Features/'

X=np.load(features_path + 'X_100ms.npy')
y=np.load(features_path +'Y_100ms.npy')
```

[4]:
```
# 75:25 train:test split
X_train, X_test, y_train, y_test = train_test_split(
        X, y , test_size=0.25, random_state=10)
```

[5]:
```
#This code focuses on optimizing the below parameters:

dim_learning_rate = Real(low=1e-4, high=1e-2,␣
 ↪prior='log-uniform',name='learning_rate')
dim_activation = Categorical(categories=['relu', 'sigmoid'],name='activation')
dim_epochs= Integer(low=70, high=200, name='epochs')
dim_dropout_rate= Real(low=0,high=0.1,name='dropout_rate')
dim_batch_size = Integer(low=4, high=64, name='batch_size')
dim_adam_decay = Real(low=1e-6,high=1e-3,name="adam_decay")

dimensions = [dim_learning_rate,
              dim_activation,
              dim_epochs,
              dim_dropout_rate,
              dim_batch_size,
              dim_adam_decay
             ]
default_parameters = [1e-3,'relu',100,0.1,8, 1e-3]
```

[6]:
```
num_layers = 3
kernel_size = 5
conv_filters = 56
lstm = 96
num_hidden = 64
l2penalty = 0.001
num_classes = 7

def conv_recurrent_model_build(learning_rate, activation,dropout_rate,␣
 ↪adam_decay):
    print('Building model...')
    n_features = X_train.shape[2]
    n_frames=X_train.shape[1]
    input_shape = (n_frames, n_features)
    model_input = Input(input_shape, name='input')#n_input
    layer = model_input

    ### Three 1D Convolution Layers
```

```
    for i in range(num_layers):
        layer = Conv1D(
                filters=conv_filters,
                kernel_size=kernel_size,
                kernel_regularizer=regularizers.l2(l2penalty),
                name='convolution_' + str(i + 1)
            )(layer)
        layer = BatchNormalization(momentum=0.9)(layer)
        layer = Activation(activation)(layer)
        layer = MaxPooling1D(2)(layer)
        layer = Dropout(dropout_rate)(layer)
    ## LSTM Layer
    layer = LSTM(lstm,return_sequences=False)(layer)
    layer = Dropout(0.4)(layer)

    ## Dense Layer
    layer = Dense(num_hidden, kernel_regularizer=regularizers.l2(l2penalty),␣
↪name='dense1')(layer)
    layer = Dropout(0.4)(layer)

    ## Softmax Output
    layer = Dense(num_classes)(layer)
    layer = Activation('softmax', name='output_realtime')(layer)
    model_output = layer
    model = Model(model_input, model_output)


    opt = Adam(lr=learning_rate, decay= adam_decay)
    model.compile(
            loss='categorical_crossentropy',
            optimizer=opt,
            metrics=['accuracy']
        )

    return model
```

[7]:

```
@use_named_args(dimensions=dimensions)
def fitness(learning_rate, activation, epochs, dropout_rate, batch_size, ␣
↪adam_decay):

    model = conv_recurrent_model_build(learning_rate=learning_rate,

                                 activation=activation,
                                 dropout_rate=dropout_rate,
                                 adam_decay=adam_decay
                                )
```

3

```
    checkpoint_callback = ModelCheckpoint('weights_best.h5',␣
↪monitor='val_accuracy', verbose=1,
                                          save_best_only=True, mode='max')


    reducelr_callback = ReduceLROnPlateau(
                factor=0.75, verbose=0, mode='auto', cooldown=30, min_lr=0.0001
            )
    callbacks_list = [checkpoint_callback, reducelr_callback]

    blackbox = model.fit(X_train, y_train, batch_size=batch_size, epochs=epochs,
                        validation_data=(X_test, y_test), verbose=1,␣
↪callbacks=callbacks_list
                        )
    #return the validation accuracy for the last epoch.
    accuracy = blackbox.history['val_accuracy'][-1]

    # Print the classification accuracy.
    print()
    print("Accuracy: {0:.2%}".format(accuracy))
    print()


    # Delete the Keras model with these hyper-parameters from memory.
    del model


    K.clear_session()
    tensorflow.reset_default_graph()

    # the optimizer aims for the lowest score, so we return our negative␣
↪accuracy
    return -accuracy
```

[8]:
```
# Restart tensorflow
K.clear_session()
tensorflow.reset_default_graph()
```

[9]:
```
# Gaussian
gaussian_result = gp_minimize(func=fitness, #function to minimize
                            dimensions=dimensions, # bounds on each dimension
                            n_calls=20, # numbers of evaluations of func
                            noise= 0.01,
                            n_jobs=-1,
                            kappa = 5,
                            x0=default_parameters)
```

```
WARNING: Logging before flag parsing goes to stderr.
W1210 15:34:04.092263 4494056896 deprecation.py:506] From
/Users/aditiaggarwal/anaconda3/lib/python3.7/site-
packages/tensorflow_core/python/ops/resource_variable_ops.py:1630: calling
BaseResourceVariable.__init__ (from tensorflow.python.ops.resource_variable_ops)
with constraint is deprecated and will be removed in a future version.
Instructions for updating:
If using Keras pass *_constraint arguments to layers.
```

```
Building model…
```

```
W1210 15:34:04.162539 4494056896 module_wrapper.py:139] From
/Users/aditiaggarwal/anaconda3/lib/python3.7/site-
packages/keras/backend/tensorflow_backend.py:4070: The name tf.nn.max_pool is
deprecated. Please use tf.nn.max_pool2d instead.

W1210 15:34:06.129909 4494056896 module_wrapper.py:139] From
/Users/aditiaggarwal/anaconda3/lib/python3.7/site-
packages/keras/backend/tensorflow_backend.py:422: The name tf.global_variables
is deprecated. Please use tf.compat.v1.global_variables instead.


Train on 1008 samples, validate on 336 samples
Epoch 1/100
1008/1008 [==============================] - 3s 3ms/step - loss: 2.0845 -
accuracy: 0.2371 - val_loss: 1.7950 - val_accuracy: 0.3988

Epoch 00001: val_accuracy improved from -inf to 0.39881, saving model to
weights_best.h5
Epoch 2/100
1008/1008 [==============================] - 2s 2ms/step - loss: 1.8751 -
accuracy: 0.3571 - val_loss: 1.7208 - val_accuracy: 0.3988

Epoch 00002: val_accuracy did not improve from 0.39881
Epoch 3/100
1008/1008 [==============================] - 2s 2ms/step - loss: 1.7666 -
accuracy: 0.3889 - val_loss: 1.5083 - val_accuracy: 0.5238

Epoch 00003: val_accuracy improved from 0.39881 to 0.52381, saving model to
weights_best.h5
Epoch 4/100
1008/1008 [==============================] - 2s 2ms/step - loss: 1.6769 -
accuracy: 0.4345 - val_loss: 1.5532 - val_accuracy: 0.4851

Epoch 00004: val_accuracy did not improve from 0.52381
Epoch 5/100
1008/1008 [==============================] - 2s 2ms/step - loss: 1.6197 -
accuracy: 0.4573 - val_loss: 1.4122 - val_accuracy: 0.5208
```

```
weights_best.h5
Epoch 69/70
1008/1008 [==============================] - 1s 1ms/step - loss: 0.9175 -
accuracy: 0.7728 - val_loss: 1.4164 - val_accuracy: 0.6250

Epoch 00069: val_accuracy did not improve from 0.71429
Epoch 70/70
1008/1008 [==============================] - 1s 1ms/step - loss: 0.8398 -
accuracy: 0.8125 - val_loss: 1.5472 - val_accuracy: 0.6131

Epoch 00070: val_accuracy did not improve from 0.71429

Accuracy: 61.31%
```

[10]:
```
print("The best accuracy was " + str(round(gaussian_result.fun *-100,2))+"%.")
```

```
The best accuracy was 83.33%.
```

[11]:
```
#Returns parameters for the best function
gaussian_result.x
```

[11]:
```
[0.0001, 'relu', 200, 0.1, 4, 1e-06]
```

[12]:
```
#Models tested by search function

tuned_results_100ms=pd.concat([pd.DataFrame(gaussian_result.x_iters, columns =␣
 ↪["learning rate",
                                                            "activation␣
 ↪function","epochs","dropout","batch size","adam learning rate decay"]),
(pd.Series(gaussian_result.func_vals*-100, name="accuracy"))], axis=1)
```

[13]:
```
#save results to csv
outPath='./Desktop/100ms.csv'
tuned_results_100ms.to_csv(outPath,index=False)
```

[14]:
```
gaussian_model = conv_recurrent_model_build(gaussian_result.
 ↪x[0],gaussian_result.x[1],gaussian_result.x[2],gaussian_result.x[3])
gaussian_model.summary()
```

```
WARNING: Logging before flag parsing goes to stderr.
W1210 17:16:26.703282 4622884288 deprecation.py:506] From
/Users/aditiaggarwal/anaconda3/lib/python3.7/site-
packages/tensorflow_core/python/ops/resource_variable_ops.py:1630: calling
BaseResourceVariable.__init__ (from tensorflow.python.ops.resource_variable_ops)
with constraint is deprecated and will be removed in a future version.
Instructions for updating:
If using Keras pass *_constraint arguments to layers.
```

```
W1210 17:16:26.792479 4622884288 module_wrapper.py:139] From
/Users/aditiaggarwal/anaconda3/lib/python3.7/site-
packages/keras/backend/tensorflow_backend.py:4070: The name tf.nn.max_pool is
deprecated. Please use tf.nn.max_pool2d instead.

Building model…
Model: "model_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input (InputLayer)           (None, 104, 182)          0
_________________________________________________________________
convolution_1 (Conv1D)       (None, 100, 56)           51016
_________________________________________________________________
batch_normalization_1 (Batch (None, 100, 56)           224
_________________________________________________________________
activation_1 (Activation)    (None, 100, 56)           0
_________________________________________________________________
max_pooling1d_1 (MaxPooling1 (None, 50, 56)            0
_________________________________________________________________
dropout_1 (Dropout)          (None, 50, 56)            0
_________________________________________________________________
convolution_2 (Conv1D)       (None, 46, 56)            15736
_________________________________________________________________
batch_normalization_2 (Batch (None, 46, 56)            224
_________________________________________________________________
activation_2 (Activation)    (None, 46, 56)            0
_________________________________________________________________
max_pooling1d_2 (MaxPooling1 (None, 23, 56)            0
_________________________________________________________________
dropout_2 (Dropout)          (None, 23, 56)            0
_________________________________________________________________
convolution_3 (Conv1D)       (None, 19, 56)            15736
_________________________________________________________________
batch_normalization_3 (Batch (None, 19, 56)            224
_________________________________________________________________
activation_3 (Activation)    (None, 19, 56)            0
_________________________________________________________________
max_pooling1d_3 (MaxPooling1 (None, 9, 56)             0
_________________________________________________________________
dropout_3 (Dropout)          (None, 9, 56)             0
_________________________________________________________________
lstm_1 (LSTM)                (None, 96)                58752
_________________________________________________________________
dropout_4 (Dropout)          (None, 96)                0
_________________________________________________________________
dense1 (Dense)               (None, 64)                6208
```

303

```
_________________________________________________________________
dropout_5 (Dropout)          (None, 64)                0
_________________________________________________________________
dense_1 (Dense)              (None, 7)                 455
_________________________________________________________________
output_realtime (Activation) (None, 7)                 0
=================================================================
Total params: 148,575
Trainable params: 148,239
Non-trainable params: 336
_________________________________________________________________
```

[15]:
```
#retrain our best model architecture
gaussian_model.fit(X_train,y_train, epochs=200, batch_size=4)
gaussian_model.evaluate(X_test,y_test)
```

```
W1210 17:16:47.567802 4622884288 module_wrapper.py:139] From
/Users/aditiaggarwal/anaconda3/lib/python3.7/site-
packages/keras/backend/tensorflow_backend.py:422: The name tf.global_variables
is deprecated. Please use tf.compat.v1.global_variables instead.


Epoch 1/200
1008/1008 [==============================] - 4s 4ms/step - loss: 2.2414 -
accuracy: 0.1746
Epoch 2/200
1008/1008 [==============================] - 3s 3ms/step - loss: 2.1421 -
accuracy: 0.2044
Epoch 3/200
1008/1008 [==============================] - 3s 3ms/step - loss: 2.0487 -
accuracy: 0.2470
Epoch 4/200
1008/1008 [==============================] - 3s 3ms/step - loss: 1.9684 -
accuracy: 0.2927
Epoch 5/200
1008/1008 [==============================] - 3s 3ms/step - loss: 1.9094 -
accuracy: 0.3403
Epoch 6/200
1008/1008 [==============================] - 3s 3ms/step - loss: 1.8445 -
accuracy: 0.3591
Epoch 7/200
1008/1008 [==============================] - 3s 3ms/step - loss: 1.7845 -
accuracy: 0.3790
Epoch 8/200
1008/1008 [==============================] - 3s 3ms/step - loss: 1.7648 -
accuracy: 0.4028
Epoch 9/200
1008/1008 [==============================] - 3s 3ms/step - loss: 1.7136 -
```

```
accuracy: 0.9563
Epoch 186/200
1008/1008 [==============================] - 3s 3ms/step - loss: 0.2168 -
accuracy: 0.9812
Epoch 187/200
1008/1008 [==============================] - 3s 3ms/step - loss: 0.2502 -
accuracy: 0.9702
Epoch 188/200
1008/1008 [==============================] - 3s 3ms/step - loss: 0.2184 -
accuracy: 0.9802
Epoch 189/200
1008/1008 [==============================] - 3s 3ms/step - loss: 0.2024 -
accuracy: 0.9871
Epoch 190/200
1008/1008 [==============================] - 3s 3ms/step - loss: 0.2113 -
accuracy: 0.9821
Epoch 191/200
1008/1008 [==============================] - 3s 3ms/step - loss: 0.2208 -
accuracy: 0.9772
Epoch 192/200
1008/1008 [==============================] - 3s 3ms/step - loss: 0.2272 -
accuracy: 0.9782
Epoch 193/200
1008/1008 [==============================] - 3s 3ms/step - loss: 0.2139 -
accuracy: 0.9792
Epoch 194/200
1008/1008 [==============================] - 3s 3ms/step - loss: 0.2520 -
accuracy: 0.9643
Epoch 195/200
1008/1008 [==============================] - 3s 3ms/step - loss: 0.2229 -
accuracy: 0.9792
Epoch 196/200
1008/1008 [==============================] - 3s 3ms/step - loss: 0.2356 -
accuracy: 0.9683
Epoch 197/200
1008/1008 [==============================] - 3s 3ms/step - loss: 0.1895 -
accuracy: 0.9891
Epoch 198/200
1008/1008 [==============================] - 3s 3ms/step - loss: 0.2203 -
accuracy: 0.9732
Epoch 199/200
1008/1008 [==============================] - 3s 3ms/step - loss: 0.2373 -
accuracy: 0.9692
Epoch 200/200
1008/1008 [==============================] - 3s 3ms/step - loss: 0.1997 -
accuracy: 0.9851
336/336 [==============================] - 0s 1ms/step
```

[15]: [0.9658633058979398, 0.8125]

[16]:
```
y_pred=gaussian_model.predict(X_test)
y_pred = np.argmax(y_pred, axis=1)
```

[17]:
```
def one_hot_encode(labels):
    return np.asarray(pd.get_dummies(labels), dtype = np.float32)
```

[18]:
```
#categorical to binary vector
predictions=one_hot_encode(y_pred)
```

[19]:
```
# Binary vector to Integer
predictions=[np.where(r==1)[0][0] for r in predictions]
```

[20]:
```
Ytest=[np.where(r==1)[0][0] for r in y_test]
```

[21]:
```
# List to array
predictions=np.asarray(predictions)
```

[22]:
```
Ytest=np.asarray(Ytest)
```

[23]:
```
cm=confusion_matrix(Ytest, predictions)

index = [   'calm','happy', 'sad','angry','fearful','disgust','surprised']
columns = [  'calm','happy', 'sad','angry','fearful','disgust','surprised']
```

[24]:
```
cm_df = pd.DataFrame(cm,index,columns)
cm_df
```

[24]:

	calm	happy	sad	angry	fearful	disgust	surprised
calm	46	0	2	0	0	0	0
happy	2	35	5	2	3	0	2
sad	7	4	33	1	1	1	0
angry	0	1	2	49	0	0	1
fearful	0	3	5	2	44	0	1
disgust	1	0	2	0	1	37	1
surprised	0	4	5	1	2	1	29

[25]:
```
emotions={

  0:'calm',
  1:'happy',
  2:'sad',
  3:'angry',
  4:'fearful',
  5:'disgust',
  6:'surprised'
```

317

```
}
```

[26]:
```python
#numbers to emotion states
import numpy as np
test=[]

for i, item in enumerate(Ytest):
    test.append(emotions[item])
```

[27]:
```python
pred=[]

for i, item in enumerate(predictions):
    pred.append(emotions[item])
```

[28]:
```python
from sklearn.metrics import classification_report
report = classification_report(test, pred)
print(report)
```

```
              precision    recall  f1-score   support

       angry       0.89      0.92      0.91        53
        calm       0.82      0.96      0.88        48
     disgust       0.95      0.88      0.91        42
     fearful       0.86      0.80      0.83        55
       happy       0.74      0.71      0.73        49
         sad       0.61      0.70      0.65        47
   surprised       0.85      0.69      0.76        42

    accuracy                           0.81       336
   macro avg       0.82      0.81      0.81       336
weighted avg       0.82      0.81      0.81       336
```

[ ]:
```python
#-----end of tuned model----------
```

[ ]:
```python

```

318