Sentiment Analysis and Emotional Classification of Audio Recording

Ephraim Hallford
DTSC 610
Application Documentation

I. Introduction

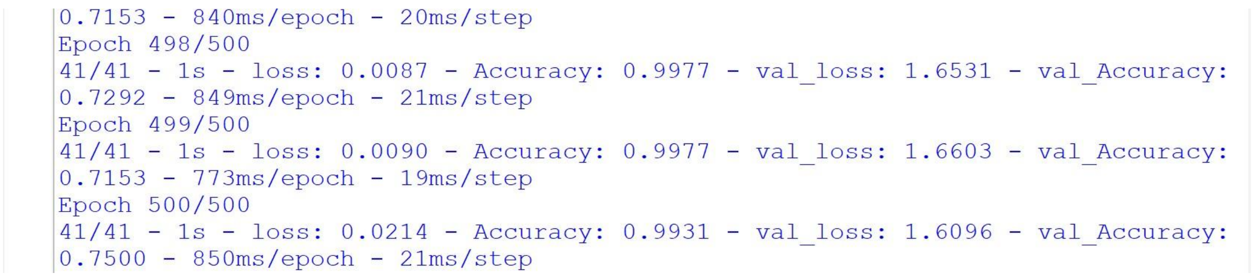
Speech Emotional Recognition (SER) is a field in machine learning that is tasked with discerning the emotions of a speaker. It has broad applications in mental healthcare, customer service and business applications [1]. State-of-the-art research has investigated deep neural networks in classification of emotions from audio [2]. Convolutional networks, a type of feed-forward neural network, which is a subset of supervised learning, have proven to be accurate classifying images [3]. Long Short Term Memory (LSTM) neural networks, a variant of recurrent neural networks, has been shown to be effective in predicting time-series data [4]. Traditional neural networks have fallen short of classifying emotions due to the complexity of emotional recognition. Issa et al. developed a model to predict the RAVDESS dataset with 71.61% accuracy [5]. The following documentation will discuss how a novel hybrid neural network consisting of Convolutional Layers, Bidirectional Long Term Short Memory Network Unit, Gaussian Noise input, and Fully Connected Dense Layers and Batch Normalization was developed and has demonstrated greater performance at 75% on the RAVDESS dataset. The following model was then implemented on a GUI application within a Tkinter framework. Textual Sentiment analysis was also added with the aid of Google's Speech Recognition API for audio transcription using the Speech Recognition package in Python. The application has the ability to play back audio and provide text of transcribed text with sentiment score, sentiment direction, subjectivity, subjectivity direction, as well as emotion predicted by text and emotion predicted by the audio that is recorded. The application further has the ability to save the transcription and predicted emotion as a text file.

Audio Processing - MFCCs

In order to train a model with the RAVDESS framework, we have to select features within a waveform in order to train a neural network. MFCCs have been used for speaker recognition [6]. Mel frequency Cepstrum Coefficient (MFCCs) are a power spectrum of fourier transform on a soundwave. They signify the spectral changes in the frequency over a time period. Speech is extremely nuanced and complex and therefore MFFcs have been shown to be excellent tools to train models to recognize speech given their ability to signify nuanced changes in frequency over time due to applying a fourier transform over a period [7]. MFFcs adequately capture important details about the sound spectral waveform and specific details that would vary across from one speaker to another [8]. We can implement and extract MFCCs from a soundwave using the open source sound processing package, Librosa, in Python [9].

CNN-LSTM Model

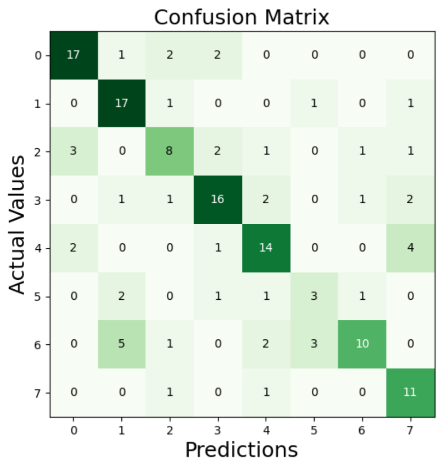
Previous research conducted by Lao et al. with CNN-LSTMmodels have shown promising results for emotional recognition with log-mel spectrograms[10]. In addition, Graves et al. demonstrated Bidirectional LSTM, which information flows back and forwards has improved upon previous models, has shown improved accuracy with speech recognition [11]. Inspired by combining these two models to achieve the best result, a novel CNN-LSTM-Bidirectional-CNN model consisting of 15 layers has stated previously achieved a 75% accuracy as demonstrated below.



```
0.7153 - 840ms/epoch - 20ms/step
Epoch 498/500
41/41 - 1s - loss: 0.0087 - Accuracy: 0.9977 - val_loss: 1.6531 - val_Accuracy:
0.7292 - 849ms/epoch - 21ms/step
Epoch 499/500
41/41 - 1s - loss: 0.0090 - Accuracy: 0.9977 - val_loss: 1.6603 - val_Accuracy:
0.7153 - 773ms/epoch - 19ms/step
Epoch 500/500
41/41 - 1s - loss: 0.0214 - Accuracy: 0.9931 - val_loss: 1.6096 - val_Accuracy:
0.7500 - 850ms/epoch - 21ms/step
```

Figure 1 Model Accuracy of CNN-BiDirectional-CNN Model

The following confusion matrix for the model generated where 0 is Neutral, 1 is calm, 2 is happy, 3 is sad, 4 is angry, 5 is fear, 6 is disgust, 7 is surprise based on the index of the array returned by the model. The dataset that was used is RAVDESS, which consists of 1440 audio files from 24 actors for the following emotions [15].



	0	1	2	3	4	5	6	7
0	17	1	2	2	0	0	0	0
1	0	17	1	0	0	1	0	1
2	3	0	8	2	1	0	1	1
3	0	1	1	16	2	0	1	2
4	2	0	0	1	14	0	0	4
5	0	2	0	1	1	3	1	0
6	0	5	1	0	2	3	10	0
7	0	0	1	0	1	0	0	11

Figure 2 Confusion Matrix of Predictions

CNN-Bidirectional-LSTM architecture

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 conv1d (Conv1D)             (None, 20, 256)           1792      

 max_pooling1d (MaxPooling1D  (None, 10, 256)          0         
 )                                                               

 conv1d_1 (Conv1D)           (None, 10, 128)           65664     

 batch_normalization (BatchN  (None, 10, 128)          512       
 ormalization)                                                   

 conv1d_2 (Conv1D)           (None, 10, 128)           32896     

 max_pooling1d_1 (MaxPooling  (None, 5, 128)           0         
 1D)                                                             

 batch_normalization_1 (Batc  (None, 5, 128)           512       
 hNormalization)                                                 

 dropout (Dropout)           (None, 5, 128)            0         

 bidirectional (Bidirectiona  (None, 5, 128)           98816     
 l)                                                              

 dense (Dense)               (None, 5, 136)            17544     

 batch_normalization_2 (Batc  (None, 5, 136)           544       
 hNormalization)                                                 

 dense_1 (Dense)             (None, 5, 136)            18632     

 batch_normalization_3 (Batc  (None, 5, 136)           544       
 hNormalization)                                                 

 dense_2 (Dense)             (None, 5, 136)            18632     

 batch_normalization_4 (Batc  (None, 5, 136)           544       
 hNormalization)                                                 

 dense_3 (Dense)             (None, 5, 136)            18632     
 dropout_1 (Dropout)        (None, 5, 136)            0

 batch_normalization_5 (Batc  (None, 5, 136)          544
 hNormalization)

 dense_4 (Dense)            (None, 5, 24)             3288

 batch_normalization_6 (Batc  (None, 5, 24)           96
 hNormalization)

 conv1d_3 (Conv1D)          (None, 5, 128)            6272

 gaussian_noise (GaussianNoi  (None, 5, 128)          0
 se)

 dropout_2 (Dropout)        (None, 5, 128)            0

 batch_normalization_7 (Batc  (None, 5, 128)          512
 hNormalization)

 dense_5 (Dense)            (None, 5, 34)             4386

 batch_normalization_8 (Batc  (None, 5, 34)           136
 hNormalization)

 flatten (Flatten)          (None, 170)               0

 dense_6 (Dense)            (None, 8)                 1368

=================================================================
Total params: 291,866
Trainable params: 289,894
Non-trainable params: 1,972
_________________________________________________________________
```

II. Program Interface

The program will begin running by executing the python script EmotionalClassification.py. It can be run directly from the command line with python EmotionalClassification.py or by clicking the icon for the file in the directory. Once it is executed, it will download and import the appropriate packages. The program will then launch a window on the desktop screen. The CNN-LSTM model for the emotional classification has been built and is already in the file directory, saved under Mode3. Nonetheless, there is also a EmotionalModel.py file which can be run to make a new model. The program terminates when the user exits the window that is generated by the script. This is a window that the user will interact with throughout the program. Throughout the program, the user will be prompted to either enter information about recording or click buttons to engage with the program.

III. Program Execution

The app is a recording tool and an analytical tool for emotional classification and sentiment analysis of audio files. Given those parameters, the program can record an audio file with a specified duration given by the user in seconds and a file name, or a user can select a file from their own local computer that is in the .wav format.

A. Recording

There are two buttons as we mentioned earlier that the user can interact with. For the purposes of this section, we will discuss how the recording executes its functions. The user is prompted to enter seconds and a filename. When the user wishes to record, they must press the record button. In Figure 1, the application is in its default state prior to any user input.

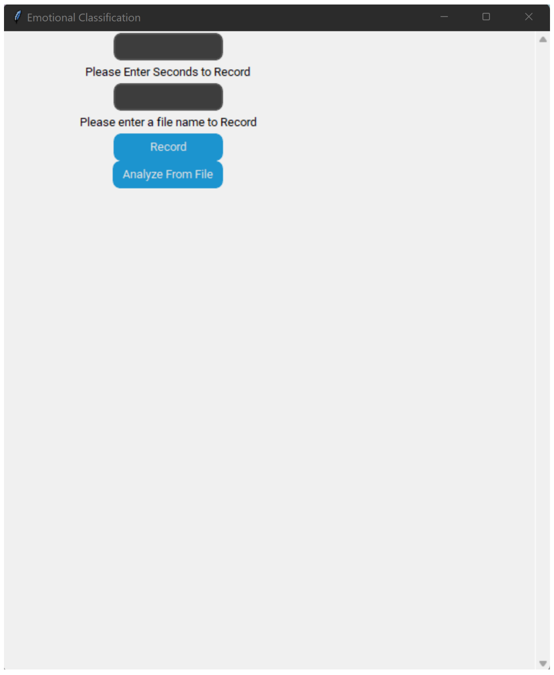

Emotional Classification
Please Enter Seconds to Record
Please enter a file name to Record
Record
Analyze From File

Figure 3 Default GUI from initial program execution

Once the user begins to record with a specified time frame, it will continue to record until a recording complete message is displayed on the GUI window.

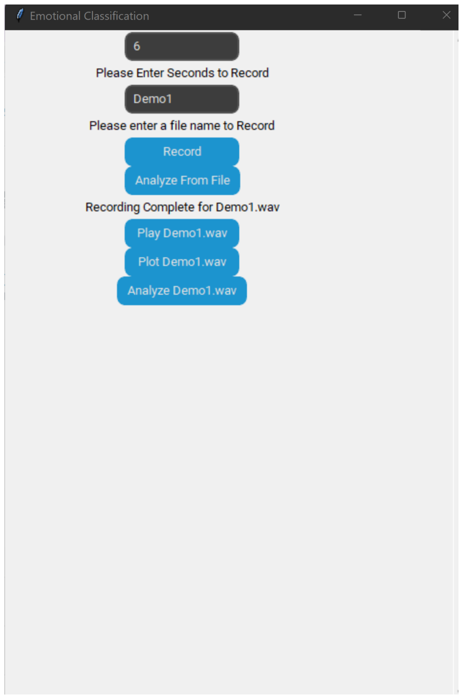


Figure 4 GUI application once recording is complete

There are three buttons on the GUI screen after the recording is completed. The first button will playback the recording that was made. The Second button as illustrated above in Figure 4 will plot the waveform of the recording and the third button will transcribe and analyze the recording that was made and provide its predicted emotion and sentiment. If the user selects plot, it will display the waveform of the recording as illustrated in Figure 5.

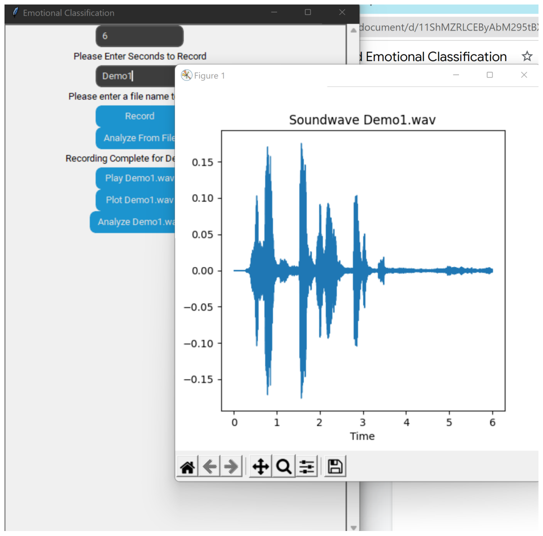


Figure 5 Soundwave displayed on after clicking plot Demo1.wav

If the user wishes to record again, they can as well and it will create a secondary recording with a different name. The user can playback, plot its waveform and analyze it as well in the same fashion.

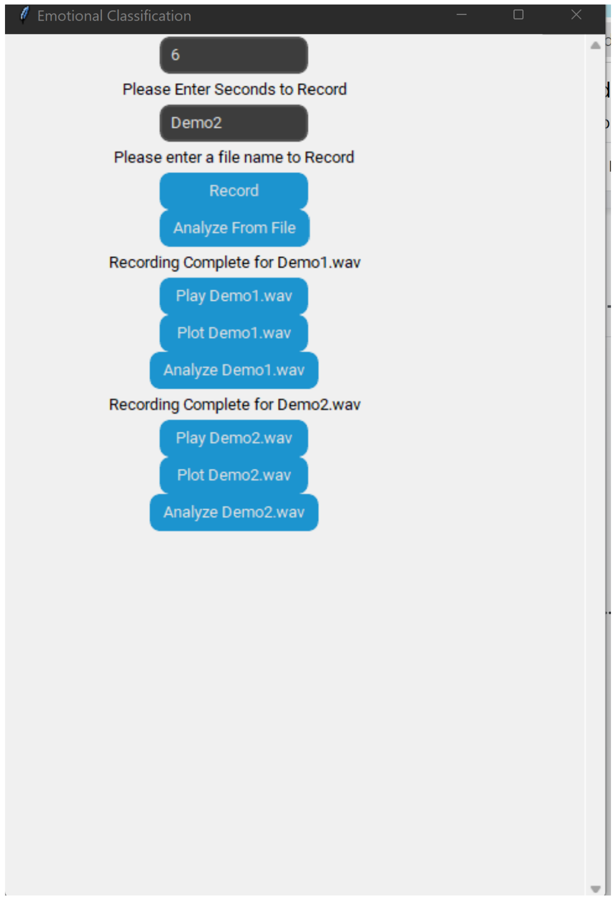
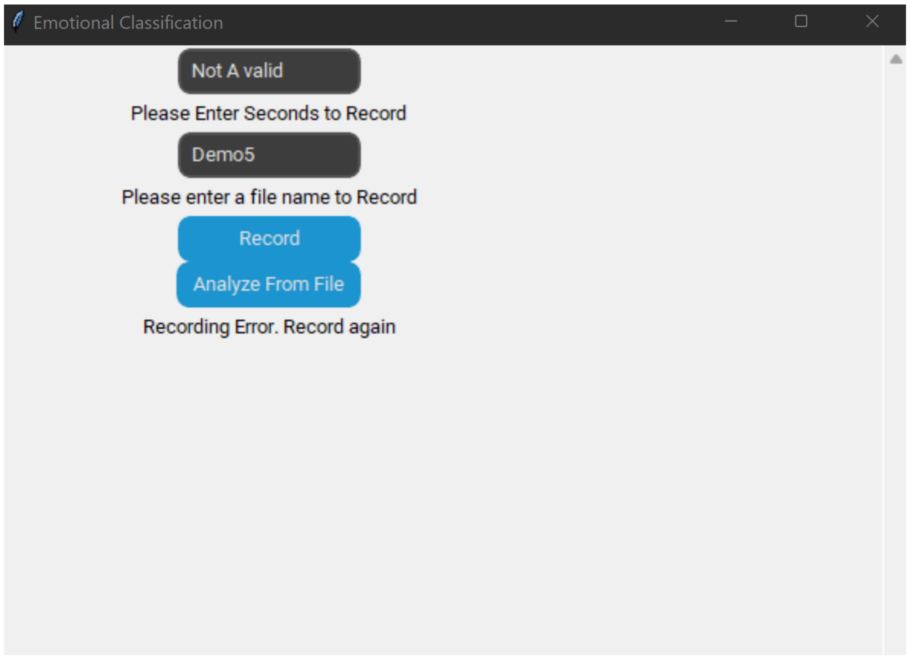


Figure 6 Second instance of a recording

Should the user wish to analyze a third recording and so forth, the instances of each recording will continue to show and the user can scroll through each record for the specific recording that they wish to analyze. There is a degree of error handling as well in the recording. Suppose for whatever reason a recording name is left blank or the user provides something other than an integer in the output. As illustrated in Figure 7, an error message will prompt the user to record with a filename.


Emotional Classification

Not A valid

Please Enter Seconds to Record

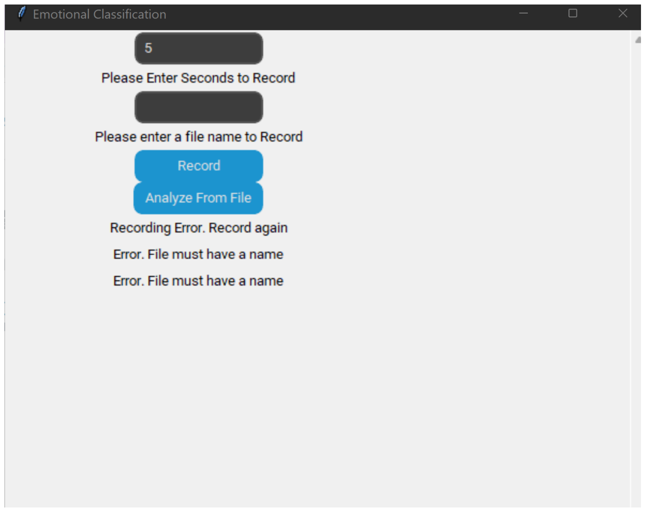
Demo5

Please enter a file name to Record

Recording Error. Record again

Figure 7 Error Message if the user enters invalid entry for duration

In addition as illustrated in Figure 8, there is an error if the user fails to enter a filename.


Emotional Classification

5

Please Enter Seconds to Record

Please enter a file name to Record

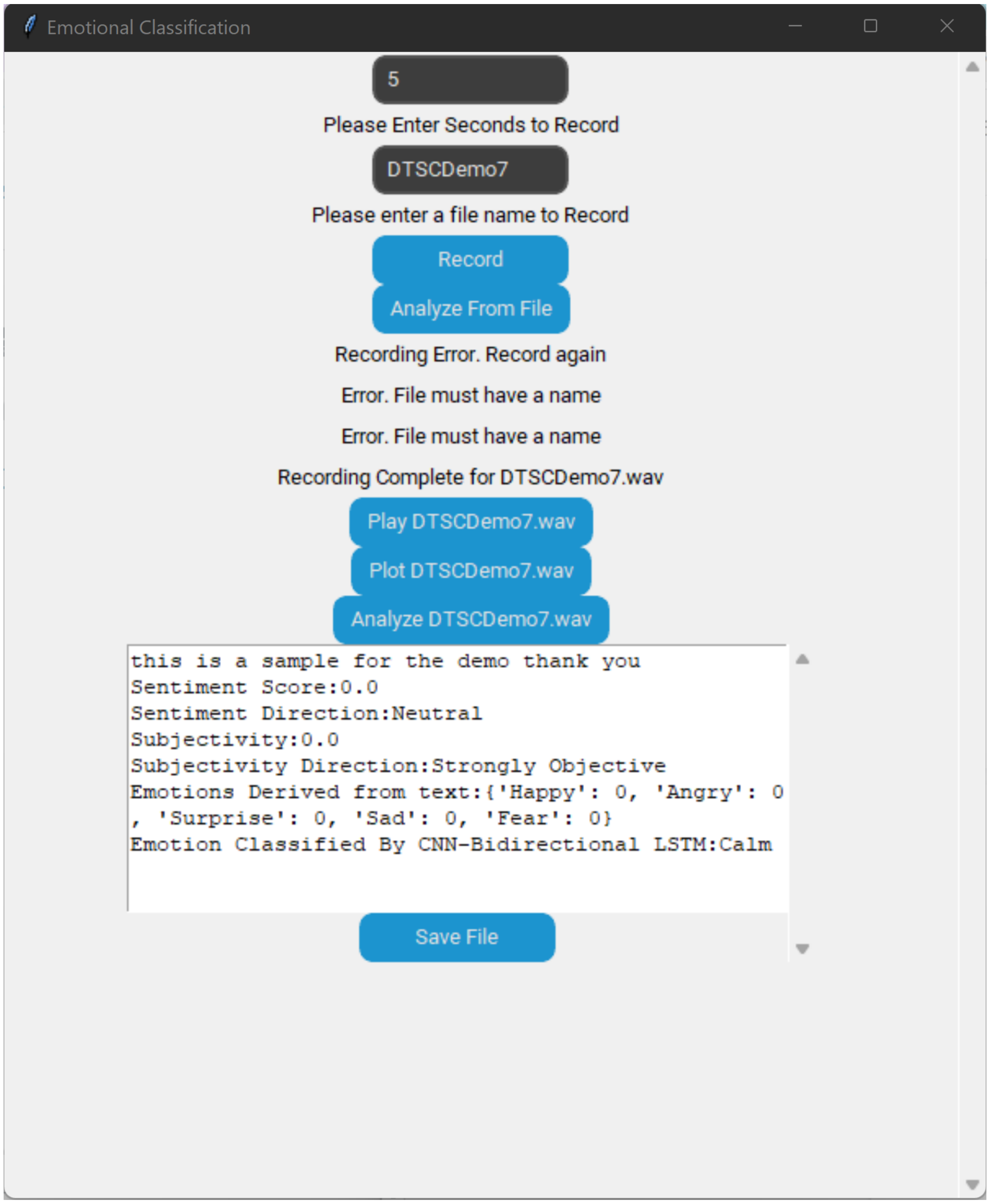
Recording Error. Record again

Error. File must have a name

Error. File must have a name

Figure 8 Error Message given if user fails to enter a file name

Now that we have seen basic validation for the inputs we can look at how the analysis was provided. When the user clicks analyze the file name that they selected, a Textbox will appear with the transcribed text, sentiment and predicted emotion as illustrated in Figure 7.

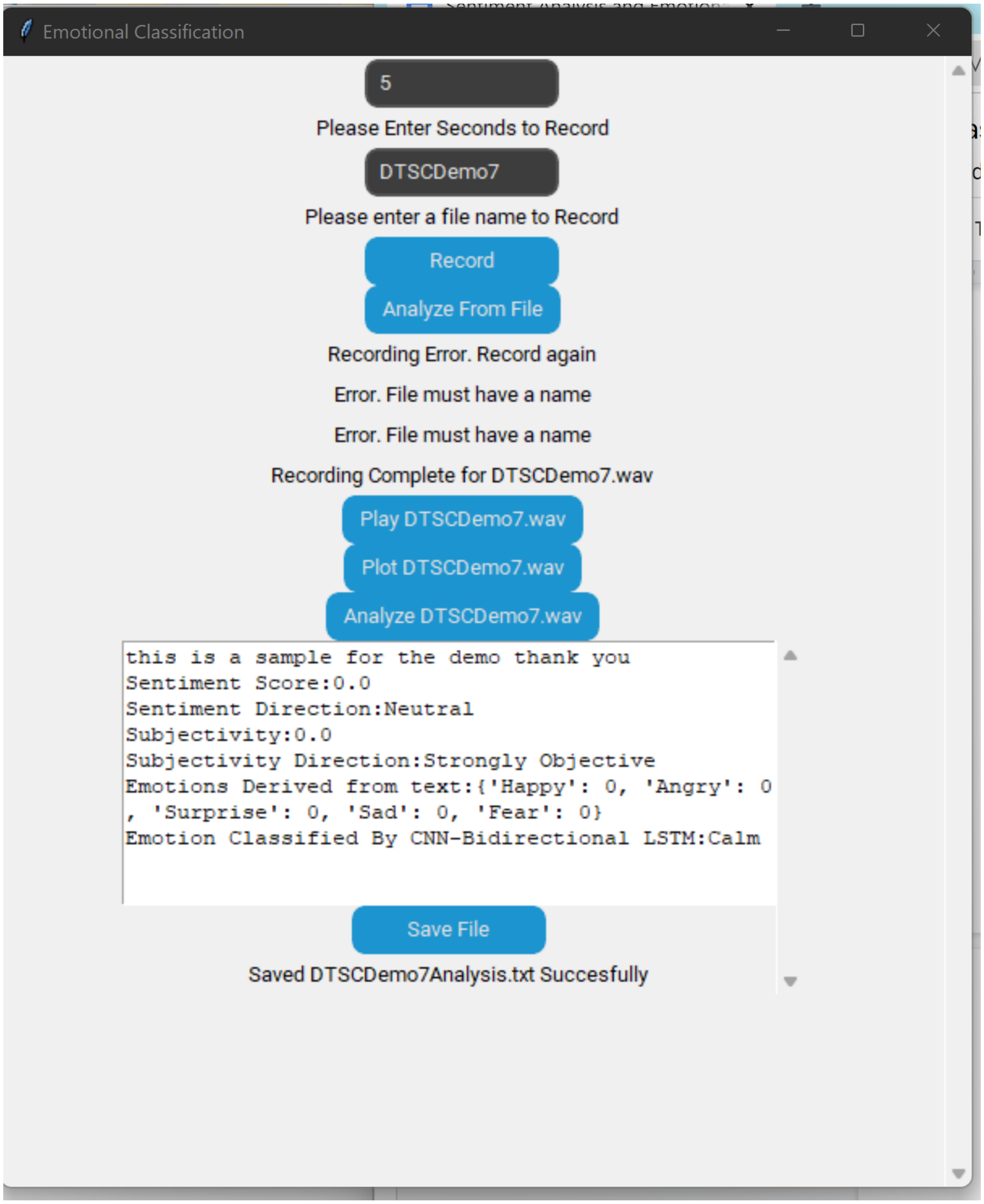


```
Recording Error. Record again
Error. File must have a name
Error. File must have a name
Recording Complete for DTSCDemo7.wav
```

```
this is a sample for the demo thank you
Sentiment Score:0.0
Sentiment Direction:Neutral
Subjectivity:0.0
Subjectivity Direction:Strongly Objective
Emotions Derived from text:{'Happy': 0, 'Angry': 0
, 'Surprise': 0, 'Sad': 0, 'Fear': 0}
Emotion Classified By CNN-Bidirectional LSTM:Calm
```

Figure 9 Textbox with transcribed text, sentiment score, sentiment direction, Classified Emotions

The user can then save analysis as a text file by clicking on Save File. It will display the output after with the file anime that the user selected and it will save the file.



Emotional Classification
5
Please Enter Seconds to Record
DTSCDemo7
Please enter a file name to Record
Record
Analyze From File

Recording Error. Record again

Error. File must have a name

Error. File must have a name

Recording Complete for DTSCDemo7.wav

```
this is a sample for the demo thank you
Sentiment Score:0.0
Sentiment Direction:Neutral
Subjectivity:0.0
Subjectivity Direction:Strongly Objective
Emotions Derived from text:{'Happy': 0, 'Angry': 0
, 'Surprise': 0, 'Sad': 0, 'Fear': 0}
Emotion Classified By CNN-Bidirectional LSTM:Calm
```

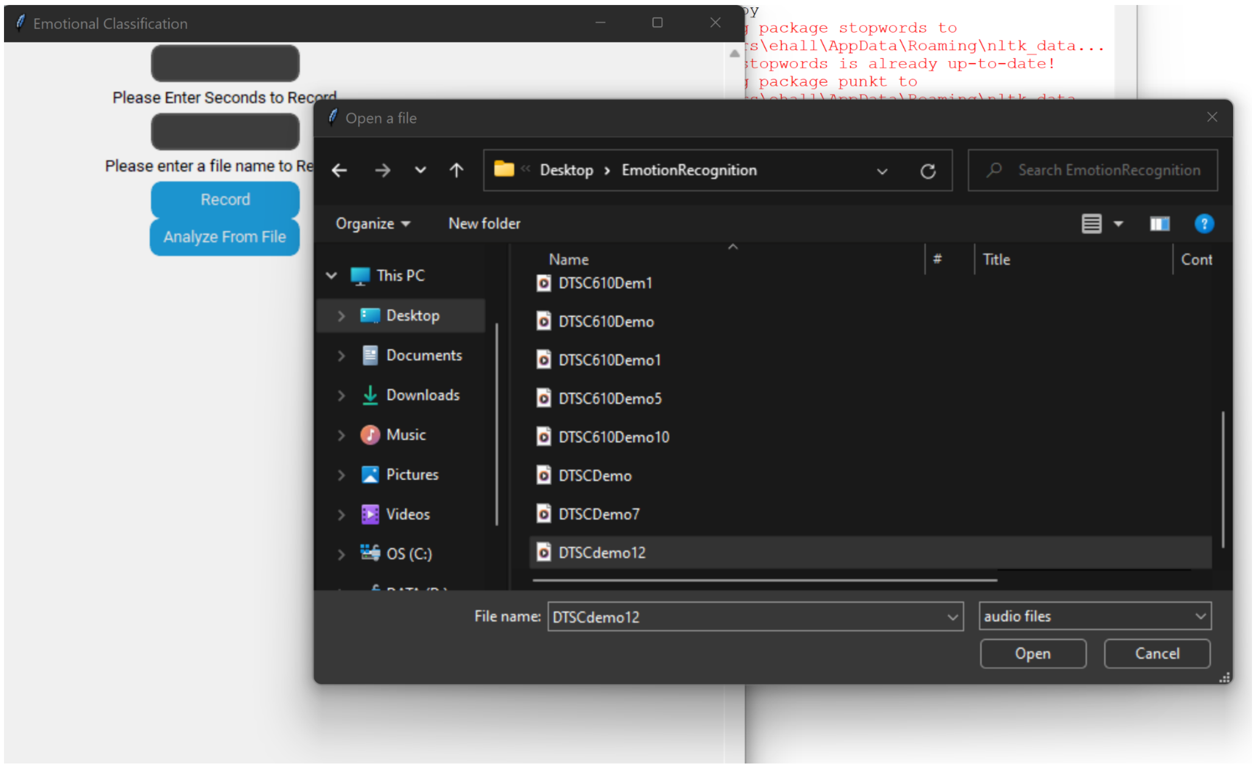
Saved DTSCDemo7Analysis.txt Succesfully

Figure 10 Saved File with filename of textbox

There is error handling as well for each step. For whatever reason that the transcription fails, it will display that the transcription failed and it will prompt the user to either record again or they can select another file.

B. Analyze From File

The user can select to analyze the audio file. The format must be a wav format. As illustrated in Figure 9, a browse window pops up when the user selects analyze from file.



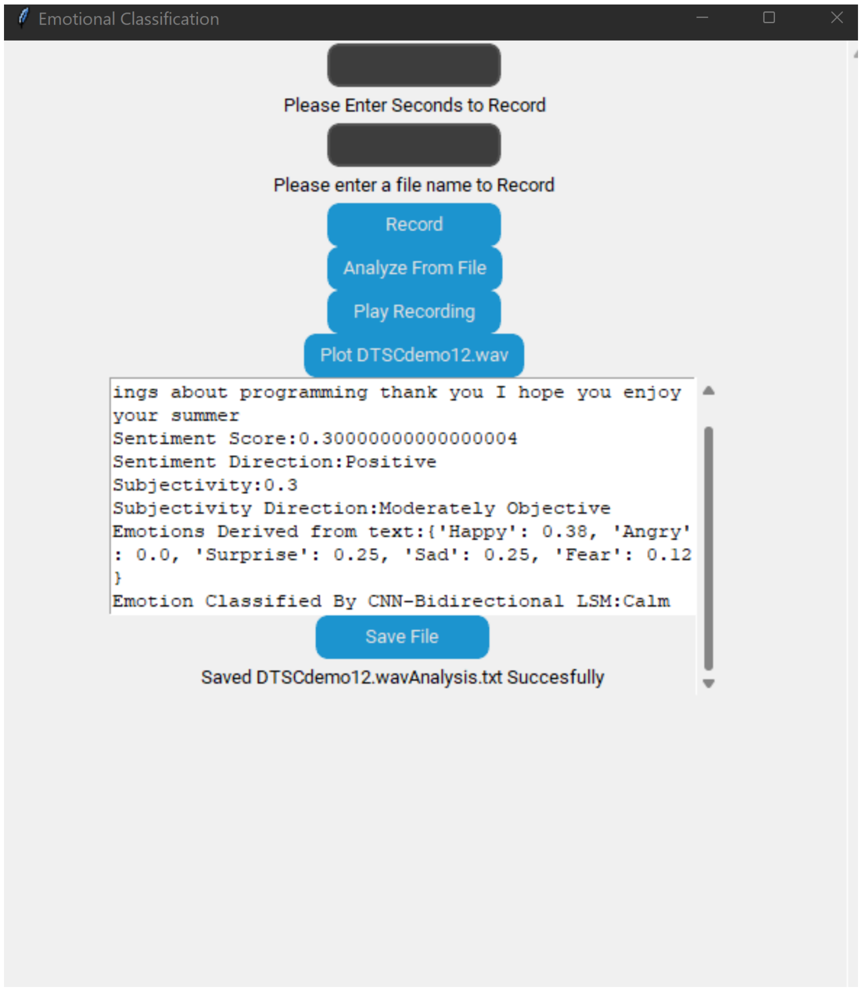
```
py
g package stopwords to
rs\ehall\AppData\Roaming\nltk_data...
stopwords is already up-to-date!
g package punkt to
```

DTSC610Dem1
DTSC610Demo
DTSC610Demo1
DTSC610Demo5
DTSC610Demo10
DTSCDemo
DTSCDemo7
DTSCdemo12

File name: DTSCdemo12
audio files

Figure 11 Selecting a file

As an example, a user can load a sample file and it will print out the same choices as before. The file's name will be output on each of the buttons and on the plot as well as the saved text file.



```
ings about programming thank you I hope you enjoy
your summer
Sentiment Score:0.30000000000000004
Sentiment Direction:Positive
Subjectivity:0.3
Subjectivity Direction:Moderately Objective
Emotions Derived from text:{'Happy': 0.38, 'Angry'
: 0.0, 'Surprise': 0.25, 'Sad': 0.25, 'Fear': 0.12
}
Emotion Classified By CNN-Bidirectional LSM:Calm
```

Saved DTSCdemo12.wavAnalysis.txt Succesfully

Figure 12 From a File

IV. Input and Output

The following input/output structure is as follows: we have two functionalities in the application. The first functionality as we discussed is recording an audio file in wav format for a specified duration of seconds.

Inputs:

1. A user is prompted to enter a string of numbers, which is then converted into an integer and a string filename. The file is routinely called throughout the program by its name and loaded into the application when the application requires it.
2. The user is prompted to record via a microphone. The microphone must be turned on for the program to work. It will default to the default microphone in settings.

Output:

1. The recording output is a .wav file.
2. Second output is transcribed text and its analysis, as well as the analysis of the audio file. This can be saved as a .txt file. All of these are displayed in the text box and can be saved to the local directory.
    a. Text transcription - the voice recording is transcribed into text
    b. Textual Sentiment Analysis - the text that was transcribed is analyzed with Textblob and returns sentiment score
    c. Speech Emotional Recognition - the audio file is analyzed with CNN-LSTM model and output is a classified emotion.

3.An output is a plot, which was created using Matplotlib library, that can be saved as a .gif

V. Program Structure

Application

The application consists of two main files. The model that is generated with EmotionModel.py saved as Mode3 in the directory, and the application which implements the model. For the first section discussing the program structure, we will discuss the overall application architecture from a Tkinter object. The program starts by importing the necessary libraries for the program to operate. The following packages for the application are:

- PyAudio
- Wave
- time
- Os
- Speech Recognition
- Nltk
- Tkinter
- CustomKinter
- Librosa
- Matplotlib
- Winsound
- Datetime
- Sklearn
- Keras and tensorflow
- Numpy
- Pandas
- text2Emotion
- (‘omw-1.4’) nltk module
- Pip.internal, Sys, and subprocess

The following program operates from the Tkinter object. A Tkinter object is a main.loop() which iterates and updates the window [6].The Tkinter object has buttons that have commands. Each of these commands can call a function such as plot, analyze, play and so on. All of the functionality is elicited by clicking Tkinter buttons. CustomTkinter is a library that is used to have a more modern appearance [7]. Each time a button is clicked it generates a new recording instance. In order to allow the window to be scrollable, there must be a scrollable object frame that can have a SrollBar added to its vertically to scroll throughout the frame.

The following Schema follows as the program executes

1. Function to install
2. Import Libraries with Try, Except calling install() function defined earlier (This is script will install packages with pip if they aren't installed)
3. Create the Tkinter window object
4. Create Tkinter MainFrame and SecondFrame within a Tkinter window so the scrollbar can allow the window to show repeated recording instances in the Tkinter root object
5. Create and Pack two Tkinter Entry Widgets for inputting the seconds to record and the filename. Using StringVar() and .get() methods to return filename throughout file as an object instance
6. Create and Pack Two Customer Tkinter Buttons on the main root that will execute two defined functions browse() and record(). When buttons are clicked, the functions will execute.
7. For the purposes of this schema, we will then discuss first how function record() operates.
8. record() has no parameters but uses the global variable duration1 and duration with the tkinter .get() method in order to return the string values that can be fed into later sections of the code.
9. record() logic works by nesting the logic in a try-except in which it will create a PyAudio object and then read chunks of data from the microphone and append that into a frames list using stream.read() and append that to a list. It will then save that data in a wave format using the python wave module based on a condition if the filename was provided.
10. record() will then pack a label onto the GUI Tkinter window stating that the recording was successful.
11. Following a recording instance being successful, Playsound from the Winsound library using a lamda function, will be created and then used in conjunction with a custom Tkinter button object that can be called to play the recorded audio.
12. A customTkinter button object for display plot will then be packed onto the GUI Tkinter window object that is called when it is clicked.
13. A custom Tkinter button, transcribe, object for the function transcribe2() is created and packed on the widget.

14. If the recording instance fails on the try except, it will print a recording error and pack the label to the GUI window.
15. The following functions will now be discussed in detail beginning with DisplayPlot object
16. DisplayPlot object is a button that executes the function plotWave when it is is clicked. plotWave returns the name of the filanem and passes it in with the .wav format. It then loads the data from the audio file using Librosa.load(). A matplotlib figure is then generated with lb.display and the plot is shown
17. Transcribe object is a button that calls the function transcribe2. Transcribe2 is nested in a try-except that returns an error label if the transcription fails. Speech_recognition Recognizer object is created and passes filename as a parameter. It returns the transcribed text from the audio file using recognize_google() method. Transcribe2 then creates a Textbox using Tkinter widget and then inserts the following text using Textblob and sentiment from the transcribed text. Transcribe2 then uses Librosa to load the file that was recorded previously into y and sampling rate that will later be loaded into a matrix and reshape with numpy to be fed into the model generated in the EmotionModel.py file. In addition, transcribe2 calls the following functions isSentimentDir, which provides the sentiment direction and isSubjectiveDir, which provides the subjectivity of the transcribed text.
18. Transcribe2 then generates the result from the Mode3 model and makes a prediction and calls the function returnEmotion, which translates the int type return prediction into a string, and then inserts that prediction into the textbox widget created in the Tkinter instance.
19. Transcribe2 then packs the Textbox into the Tkinter frame that was created initially.
20. Finally, transcribe packs a saveButton object, which is a customTkinter button that calls save() function when it is clicked.
21. save() function is a function that opens a text file with the returned name of file using the .get() method for the textbox widget earlier. It opens a text file and writes the contents of the textbox, saves and closes the file.
22. The other functionality that is within the app is opening from a file in the directory of a local computer.
23. The customTkinter object openButton, which calls the function browse() when it is clicked, allows the user to specify what file that they want to analyze.
24. Browse() follows a similar structure to the record with a try-except to return if it fails and defining a global variable filename1 that will be called throughout instead of the .get() method that was used with Entry widget for record(). The only variant is the savebrowse() instead of save() that calls the filename1 created globally by the selected filename.

EmotionalModel.py

EmotionalModel.py is the file that creates the Mode3 model that is called by the application. The following pip dependencies must be installed for the model to run.

- Pandas
- Os
- Numpy
- Sklearn
- Tensorflow
- Keras
- Mlxtend
- Librosa
- Librosa.display
- Matplotlib

The following Schema is followed in the EmotionalModel.py

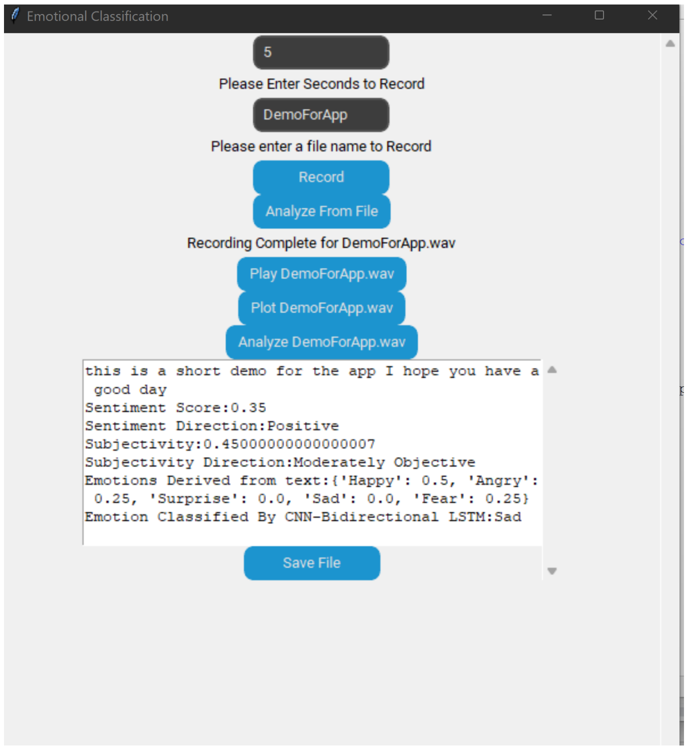
1. The model first must retrieve all the data files. The model file was already loaded in mode3. This is only run if you want to return the model with the data in the file. The data RAVDESS dataset is included in the zip file with the source code.
2. The first section of the code will retrieve the paths and create a dataframe in pandas with the emotions and the paths of each file. We then create a pandas data frame, and a combined pandas data frame with emotions and path
3. A for loop iterates through the paths and loads with Librosa into a data and list sr.
4. This is passed in into an array called extracted.
5. Extracted is then appended to a list X.
6. X is them converted into a matrix
7. X is converted into a pandas DataFrame
8. X Matrix is the concatenated with a combined path and is fed into scitkitlearns train_test_split to return X_train, X_test, y_train and y_Train datasets
9. Each of these is converted into a numpy array
10. X_Train and X_Test np arrays are reshaped to add a column with np.expand_dims to feed into model.
11. LabelEncoder() converts the string values into hot encoded 0-7.
12. A sequential model is built with 15 layers as illustrated with the architecture we discussed earlier. Batch normalization layers are used to normalize inputs
13. The following architecture has nested Conv1d layers with Batch Normalization between each Convolutional layer. This is done to learn the features of the MFFcs as we discussed earlier. Each batch normalization layer normalizes the inputs so only true features are learned. We then drop out 10% of the neurons to help it generalize. We then feed that input into a bidirectional LSTM. This is done so it will learn the sequence-dependent attributes that occur on the waveform. The information flows forward and then flows backward so it captures more of the density of features. After this, it is fed into 4 fully connected dense layers. The general idea behind this is that the dense layers will be able to learn and weigh the features of the BiDirectional LSTM layer. Finally, the output is fed into 1 convolutional layer in order to learn any outlying features that might have

been missed. It's normalized and flatten and fed into a dense layer with 8 outputs and a softmax function to return probabilities of belonging to one of the 8 classes.

14. The model is compiled and run with Adamax. Adamax is a variant of Adam for optimization algorithm that has been shown to be effective in deep learning architectures [13]. Its learning rate has been tuned to .0011. Categorical cross entropy has been demonstrated to be an effective loss function for multiple classes in deep learning architectures [14]. The model was run for 500 epochs.
15. Model is fitted with model.fit() in keras and then saved. Scikit Learn is used to plot the confusion matrix as demonstrated in Figure 2.
16. The file rerun with the data provided to generate another model. Further tweaks to the file can make further improvements to the model.

VI. Examples

Classification Application Demo1



```
this is a short demo for the app I hope you have a
 good day
Sentiment Score:0.35
Sentiment Direction:Positive
Subjectivity:0.45000000000000007
Subjectivity Direction:Moderately Objective
Emotions Derived from text:{'Happy': 0.5, 'Angry':
 0.25, 'Surprise': 0.0, 'Sad': 0.0, 'Fear': 0.25}
Emotion Classified By CNN-Bidirectional LSTM:Sad
```

Figure 13

VII. Improvements and Extensions

The following improvements can be made to the GUI. A menu for selecting prior recordings. Importing from the web could be a potential new extension. Accuracy can be improved. Currently, the model is only 75% accurate. Further fine tuning of the emotional Model can be done with new architecture. A functionality to have emotional output could be implemented.

VIII. Difficulties Encountered

Developing a model that was accurate was the greatest challenge. The initial accuracy was in the low 20%. After multiple iterations and fine tuning with additional layers, the accuracy improved to the mid seventies. Additional feature extraction techniques can be implemented. The size of the model was very limited. A larger dataset could be employed to the train the model or data augmentation techniques could be employed. There were only 1440 vocal samples in the RAVDESS dataset [15]. The dataset was labeled with the following format as instructed by the authors of the RAVDESS. Additional samples that could be included to improve the model.

IX. Conclusion

Speech Emotional Recognition is a state-of-the-art field in artificial intelligence and machine learning. Various approaches have been implemented to classify a speaker's emotions. After review of the literature, such as Issa (2020)'s CNN model and Zhou's (2019) 1D CNN-LSTM, a novel hybrid CNN-LSTM-CNN model was implemented in Keras and an application was developed with GUI Tkinter library. The model had an accuracy of 75%, which outperformed Issa's model on the same dataset. The Tkinter application that was built employed Speech Recognition, text emotion and textblob to add textual insight on the transcribed texts. This is a primarily utility tool and research tool that can record multiple files and quickly apply a model to make predictions on a recording dataset. Other models can be easily applied with simple changes to the application. Further refinement of this tool can allow for additional extensions such as a connection to the web with an API. The primary tool functionalities allow for recording a short clip and returning a predicted emotion or analyzing a text and returning an emotion and transcription. The ability to save the files generated by this schema allows for quick and easy analysis of files promptly.

X. References

1. Ingale, A.B. and Chaudhari, D.S., 2012. Speech emotion recognition. *International Journal of Soft Computing and Engineering (IJSCE)*, *2*(1), pp.235-238.
2. Fayek, H.M., Lech, M. and Cavedon, L., 2017. Evaluating deep learning architectures for Speech Emotion Recognition. *Neural Networks*, *92*, pp.60-68.
3. Guo, T., Dong, J., Li, H. and Gao, Y., 2017, March. Simple convolutional neural network on image classification. In *2017 IEEE 2nd International Conference on Big Data Analysis (ICBDA)* (pp. 721-724). IEEE.
4. Yu, Y., Si, X., Hu, C. and Zhang, J., 2019. A review of recurrent neural networks: LSTM cells and network architectures. *Neural computation*, *31*(7), pp.1235-1270.
5. Issa, D., Demirci, M.F. and Yazici, A., 2020. Speech emotion recognition with deep convolutional neural networks. Biomedical Signal Processing and Control, 59, p.101894.

6. Tiwari, V., 2010. MFCC and its applications in speaker recognition. *International journal on emerging technologies*, *1*(1), pp.19-22.
7. Gaikwad, S.K., Gawali, B.W. and Yannawar, P., 2010. A review on speech recognition technique. *International Journal of Computer Applications*, *10*(3), pp.16-24.
8. Zhao, J., Mao, X. and Chen, L., 2019. Speech emotion recognition using deep 1D & 2D CNN LSTM networks. *Biomedical signal processing and control*, *47*, pp.312-323.
9. Graves, A., Jaitly, N. and Mohamed, A.R., 2013, December. Hybrid speech recognition with deep bidirectional LSTM. In *2013 IEEE workshop on automatic speech recognition and understanding* (pp. 273-278). IEEE.
10. *Graphical User Interfaces with Tk — Python 3.10.4 documentation*. (2022). Tkinter. https://docs.python.org/3/library/tk.html
11. T. (2022). *GitHub - TomSchimansky/CustomTkinter: A modern and customizable python UI-library based on Tkinter*. GitHub. https://github.com/TomSchimansky/CustomTkinter
12. McFee, Brian, Colin Raffel, Dawen Liang, Daniel PW Ellis, Matt McVicar, Eric Battenberg, and Oriol Nieto. "librosa: Audio and music signal analysis in python." In Proceedings of the 14th python in science conference, pp. 18-25. 2015.
13. Kingma, D.P. and Ba, J., 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.
14. Rusiecki, A., 2019. Trimmed categorical cross-entropy for deep learning with label noise. *Electronics Letters*, *55*(6), pp.319-320.
15. Livingstone SR, Russo FA (2018) The Ryerson Audio-Visual Database of Emotional Speech and Song (RAVDESS): A dynamic, multimodal set of facial and vocal expressions in North American English. PLoS ONE 13(5): e0196391. https://doi.org/10.1371/journal.pone.0196391.
16. Team, K. (2022). *Keras documentation: Developer guides*. Keras. https://keras.io/guides/

XI. Appendices (source code)

EmotionalClassification.py

```
#Imports libraries that will may be necesasry to run the program
#We Must First check to see if the package are installed
import sys
import subprocess

def check_install(package):
    subprocess.check_call([sys.executable, '-m', 'pip', 'install',
package])


try:
    import pyaudio
except ImportError:
    check_install('PyAudio')
    import pyaudio
```

```python
import wave
import time
import os
try:
    import speech_recognition as sr
except ImportError:
    check_install('SpeechRecognition')
    import speech_recognition as sr

from textblob import *
try:
    from tkinter import *
except ImportError:
    check_install('tk')

from tkinter import *
from tkinter import ttk
import tkinter.messagebox as MessageBox
try:
    import librosa as lb
except ImportError:
    install('librosa')
    import librosa as lb
import librosa
try:
    import matplotlib.pyplot as plt
except ImportError:
    install('matplotlib')
    import matplotlib.pyplot as plt
import librosa.display as lbd
from tkinter import*
from datetime import datetime
import time
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
try:
    import sklearn
except ImportError:
    install('scikit-learn')
    import sklearn
try:
    import nltk
except ImportError:
    install('nltk')
    import nltk

from sklearn import *
try:
    import tensorflow as tf
except ImportError:
    install('tensorflow')
    import tensorflow as tf
```

```
from tensorflow import keras
try:
    import numpy as np
except ImportError:
    install('numpy')
    import numpy as np
try:
    import pandas as pd
except ImportError:
    install('pandas')
    import pandas as pd
try:
    import customtkinter
except ImportError:
    install('customtkinter')
    import customtkinter
try:
    import text2emotion as te
except ImportError:
    install('text2emotion')
    import text2emotion as te
from tkinter import filedialog as fd
from tkinter.filedialog import asksaveasfile
nltk.download('omw-1.4')
from winsound import *


def features_record(data,sr):
    # Mel Coeffienct cepstrum is the power spectrumm ajfter a fourier
transform has been applied to the signal
    mfcc = librosa.feature.mfcc(y=data, sr=sr)
    mfccsmean = np.mean(mfcc.T,axis=0)
    return mfccsmean


def savebrowse():
    filename2 = os.path.basename(filename1)
    txtfile = filename2 + "Analysis.txt"
    saveText = open(txtfile,"w")
    saveText.write(tbox.get(1.0,END))
    saveText.close()
    textLabel1 = customtkinter.CTkLabel(secondFrame, text=("Saved" +" " +
txtfile + " " + "Succesfully"),text_color="#000000")
    textLabel1.pack()



def save():
    txtfile = duration1.get()+"Analysis.txt"
    saveText = open(txtfile,"w")
    saveText.write(tbox.get(1.0,END))
    saveText.close()
    textLabel1 = customtkinter.CTkLabel(secondFrame, text=("Saved" +" " +
txtfile + " " + "Succesfully"),text_color="#000000")
```

```
    textLabel1.pack()


def transcribe2():
    try:
        r = sr.Recognizer()
        filename = duration1.get()
        filename = filename + ".wav"
        with sr.AudioFile(filename) as source:
            data_from_audio = r.record(source)
            text = r.recognize_google(data_from_audio)
            print(text)


        vertical = Scrollbar(secondFrame,orient='vertical')
        vertical.pack(side=RIGHT,fill='y')

        #Declares tbox as a global variable
        global tbox
        tbox =
Text(secondFrame,height=10,width=50,yscrollcommand=vertical.set)
        vertical.config(command=tbox.yview)
        tbox.insert(INSERT,text)
        blob = TextBlob(text)
        sentiment = blob.sentiment.polarity
        subjective = blob.sentiment.subjectivity
        #inserts text into textbox for each analyzed portion
        tbox.insert(INSERT,'\nSentiment Score:')
        tbox.insert(INSERT,sentiment)
        tbox.insert(INSERT,'\nSentiment Direction:')
        tbox.insert(INSERT,isSentimentDir(sentiment))
        tbox.insert(INSERT,'\nSubjectivity:')
        tbox.insert(INSERT,subjective)
        tbox.insert(INSERT,'\nSubjectivity Direction:')
        tbox.insert(INSERT,isSubjectiveDir(subjective))
        TextEmotion = te.get_emotion(text)
        tbox.insert(INSERT,'\nEmotions Derived from text:')
        tbox.insert(INSERT,TextEmotion)
        y, sampling_rate = lb.load(filename, sr=44000)
        voice = features_record(y,sampling_rate)
        X_Matrix = np.matrix(voice)
        df = pd.DataFrame(X_Matrix)
        X_test = np.array(df)
        X_test = np.expand_dims(X_test,axis=2)
        model = keras.models.load_model('Mode3')
        result = model.predict(X_test)
        y_pred=np.argmax(result, axis=1)
        emotion = returnEmotion(y_pred)
        tbox.insert(INSERT,'\nEmotion Classified By CNN-Bidirectional
LSTM:')
        tbox.insert(INSERT,emotion)
        tbox.config(state='disabled')
        tbox.pack()
        saveButton = customtkinter.CTkButton(secondFrame,text="Save 
File",command=save)
        saveButton.pack()

    except Exception as e:
        errorLabel= customtkinter.CTkLabel(secondFrame, text="Recording 
cannot Be Transcribed. Record again",text_color="#000000")
        errorLabel.pack()



def plotWave():
    #This returns the name and allows us to plot the waveform and the 
features of the audiofile
    filename = duration1.get()
    filename = filename + ".wav"
    y, sr = lb.load(filename, sr=44000)
    fig = plt.Figure(figsize=(5,5))

    # We'll show each in its own subplot
    plt.figure(figsize=(5,5))
    lb.display.waveshow(y, sr=sr)
    plt.title('Soundwave' + " "+ filename)

    plt.show()


def plotBrowseWave():
    #This returns the name and allows us to plot the waveform and the 
features of the audiofile
    y, sr = lb.load(filename1, sr=44000)
    fig = plt.Figure(figsize=(5,5))

    # We'll show each in its own subplot
    plt.figure(figsize=(5,5))
    lb.display.waveshow(y, sr=sr)
    file2 = os.path.basename(filename1)
    plt.title('Soundwave' + " "+ file2)

    plt.show()

#returns an emotion based on a hot encoding for CNN-BiDirectional LSTM 
model created and loaded. This definitions are provided
#in the ReturnModel file that was included in the zip file. The current 
accuracy of the model is 75% for 8 emotions. The model returns
#a value 0 to 7 based on the hot encoding. Please refer to EmotionModel 
included in zip file for further information.
def returnEmotion(y_pred):
    if int(y_pred) == 0:
        return "Neutral"
    elif int(y_pred) ==1:
        return "Calm"
    elif int(y_pred) ==2:
        return "Happy"
    elif int(y_pred) ==3:
        return "Sad"
    elif int(y_pred) ==4:
        return "angry"
    elif int(y_pred) ==5:
        return "fearful"
    elif int(y_pred) ==6:
        return "disgust"
    else:
       return "surprised"

#Creates a recording instance that can be called by Tkinter button
def record():
    #Try Catch in order to try recording and returning an error messgage
if it fails to record
    try:
        chunk = 1024
        sample_format = pyaudio.paInt16
        channels = 2
        fs = 44100
        p = pyaudio.PyAudio()
        print("Begin Recording")
        frames = []
        filename = duration1.get()
        seconds = int(duration.get())
        start = time.time()
        stream = p.open(format=sample_format,
        channels=channels,
        rate=fs,
        frames_per_buffer=chunk,
                input=True)
        for i in range(0, int(fs / chunk * seconds)):
            data = stream.read(chunk)
            frames.append(data)



        stream.stop_stream()
        stream.close()
        p.terminate()
        print('Finished Recording')
        end = time.time()
        elapsedTime = round((end - start),2)
        print("Total Time Elapsed:",elapsedTime)
        if filename !="":
            filename =filename + ".wav"
            wf = wave.open(filename, 'wb')
            wf.setnchannels(channels)
            wf.setsampwidth(p.get_sample_size(sample_format))
            sample_format = pyaudio.paInt16
            wf.setframerate(fs)
            wf.writeframes(b''.join(frames))
            wf.close()
            completeLabel = customtkinter.CTkLabel(secondFrame,
text=("Recording Complete for"+" "+filename),text_color="#000000")
            completeLabel.pack()
            play = lambda: PlaySound(filename, SND_FILENAME)
            play_button =
customtkinter.CTkButton(secondFrame,text=("Play"+"
"+filename),command=play)
            play_button.pack()
            displayPlot =
customtkinter.CTkButton(secondFrame,command=plotWave,text=('Plot'+"
"+filename))
            displayPlot.pack()
            transcribe =
customtkinter.CTkButton(secondFrame,text=("Analyze"+"
"+filename),command=transcribe2)
            transcribe.pack()
        else:
            noNameLabel = customtkinter.CTkLabel(secondFrame, text="Error.
File must have a name",text_color="#000000")
            noNameLabel.pack()

    except ValueError:
        errorLabel= customtkinter.CTkLabel(secondFrame, text="Recording
Error. Record again",text_color="#000000")
        errorLabel.pack()



#returns the sentiment of a text that is transcribed
def sentiment_score(text):
    blob = TextBlob(text)
    sentiment = blob.sentiment.polarity
    textLabel = customtkinter.CTkLabel(secondFrame, text=("This is the
sentiment:",sentiment))
    return sentiment



#allows a user to browse for a fiile and then transcribe text and return
emotion from an audio file
def browse():
    try:
        r = sr.Recognizer()
        filetypes = (
            ('audio files', '*.wav'),
            ('All files', '*.*')
        )

        global filename1
        filename = fd.askopenfilename(
            title='Open a file',
            initialdir='/',
            filetypes=filetypes)
        with sr.AudioFile(filename) as source:
            data_from_audio = r.record(source)
            text = r.recognize_google(data_from_audio)
            print(text)
        filename1 = filename

        play = lambda: PlaySound(filename, SND_FILENAME)
        play_button = customtkinter.CTkButton(secondFrame,text="Play 
Recording",command=play)
        play_button.pack()
        filename2 = os.path.basename(filename1)
        displayPlot = 
customtkinter.CTkButton(secondFrame,command=plotBrowseWave,text=('Plot'+" 
"+filename2))
        displayPlot.pack()
        vertical = Scrollbar(secondFrame,orient='vertical')
        vertical.pack(side=RIGHT,fill='y')

        global tbox
        tbox = 
Text(secondFrame,height=10,width=50,yscrollcommand=vertical.set)
        vertical.config(command=tbox.yview)
        tbox.insert(INSERT,text)
        #tbox.config(state='disabled')
        blob = TextBlob(text)
        sentiment = blob.sentiment.polarity
        print(sentiment)
        subjective = blob.sentiment.subjectivity
        tbox.insert(INSERT,'\nSentiment Score:')
        tbox.insert(INSERT,sentiment)
        tbox.insert(INSERT,'\nSentiment Direction:')
        tbox.insert(INSERT,isSentimentDir(sentiment))
        tbox.insert(INSERT,'\nSubjectivity:')
        tbox.insert(INSERT,subjective)
        tbox.insert(INSERT,'\nSubjectivity Direction:')
        tbox.insert(INSERT,isSubjectiveDir(subjective))
        TextEmotion = te.get_emotion(text)
        tbox.insert(INSERT,'\nEmotions Derived from text:')
        tbox.insert(INSERT,TextEmotion)
        y, sampling_rate = lb.load(filename, sr=44000)
        voice = features_record(y,sampling_rate)
        X_Matrix = np.matrix(voice)
        df = pd.DataFrame(X_Matrix)
        X_test = np.array(df)
        X_test = np.expand_dims(X_test,axis=2)
        model = keras.models.load_model('Mode3')
        result = model.predict(X_test)
        y_pred=np.argmax(result, axis=1)
        emotion = returnEmotion(y_pred)
        tbox.insert(INSERT,'\nEmotion Classified By CNN-Bidirectional 
LSM:')
        tbox.insert(INSERT,emotion)
        tbox.config(state='disabled')
        tbox.pack()
        saveButton = customtkinter.CTkButton(secondFrame,text="Save 
File",command=savebrowse)
        saveButton.pack()



    except Exception as e:
        errorLabel= customtkinter.CTkLabel(secondFrame, text="File Cannot 
Be Analyzed. Choose another file.",text_color="#000000")
        errorLabel.pack()



def isSentimentDir(sentiment):
    if sentiment > 0:
        return "Positive"
    elif sentiment == 0:
        return "Neutral"
    else:
        return "Negative"


def isSubjectiveDir(subjective):
    if  subjective > .90:
        return "Strongly Subjective"
    elif subjective > .5:
        return "Moderately Subjective"
    elif subjective > .10:
        return "Moderately Objective"
    else:
        return "Strongly Objective"


def features_record(data,sr):
    # Mel Coeffienct cepstrum is the power spectrumm after a fourier 
transform has been applied to the signal
    mfcc = librosa.feature.mfcc(y=data, sr=sr)
    mfccsmean = np.mean(mfcc.T,axis=0)
    return mfccsmean


#Creates a scrollable frame for the app
customtkinter.set_appearance_mode("System")
customtkinter.set_default_color_theme("blue")
root= customtkinter.CTk()
mainFrame = Frame(root)
mainFrame.pack(fill=BOTH,expand=1)
mainCanvas = Canvas(mainFrame)
mainCanvas.pack(side=LEFT,fill=BOTH,expand=1)
mainScroll = Scrollbar(mainFrame,orient=VERTICAL,command=mainCanvas.yview)
mainScroll.pack(side=RIGHT,fill=Y)
mainCanvas.configure(yscrollcommand=mainScroll.set)
mainCanvas.bind('<Configure>',lambda 
e:mainCanvas.configure(scrollregion=mainCanvas.bbox("all")))
```

```
secondFrame = Frame(mainCanvas)

#This is in the secondframe, which is the srollable object frame
mainCanvas.create_window((75,0),window=secondFrame, anchor='nw')
root.title("Emotional Classification")
root.geometry('600x700')
textLabel = customtkinter.CTkLabel(secondFrame, text="Please Enter Seconds
to Record",text_color="#000000")
duration = StringVar()
duration1 = StringVar()
entry= customtkinter.CTkEntry(secondFrame,textvariable=duration).pack()
textLabel.pack()
entry1= customtkinter.CTkEntry(secondFrame,textvariable=duration1).pack()
textLabel1 = customtkinter.CTkLabel(secondFrame, text="Please enter a file
name to Record",text_color="#000000")
textLabel1.pack()
record = customtkinter.CTkButton(secondFrame,text="Record",command=record)
record.pack()
openButton = customtkinter.CTkButton(secondFrame,text='Analyze From File',
command=browse)
openButton.pack()


root.mainloop()
```

EmotionalModel.py

```
import pandas as pd
import os
import numpy as np
import matplotlib.pyplot as plt
import librosa as lb
import librosa.display
import sklearn
from sklearn import *
from sklearn.model_selection import train_test_split
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Conv1D,
MaxPooling1D, GaussianNoise, BatchNormalization, Bidirectional
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Activation, Dense
from tensorflow.keras.optimizers import *
```

```python
from tensorflow.keras.metrics import *
from keras.layers import Input, Flatten
from mlxtend.plotting import plot_confusion_matrix
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import LabelEncoder
from tensorflow.python.keras.utils import np_utils
from keras.layers import GlobalMaxPooling1D
from keras.layers import Dropout
from tensorflow.keras.layers import LSTM
from sklearn.preprocessing import StandardScaler
from tensorflow.keras.models import Sequential, save_model, load_model
#ravdess is the directory

ravdess = 'data/'
#returns a list of directory of directory ravdess
ravdess_dir = os.listdir(ravdess)

#This creates a two lists to store the files
emotions = []
paths = []

#This will retrieve all the paths in the directory. Since
#there is a subdirectory we have to aggregate and create a
#subdirectory and then concatenate each to form the total path name
#for each. We also reetrive the emotions from the file name
for dir in ravdess_dir:
    files = os.listdir(ravdess + dir)
    #for each file int he directory, it will split the file by
    for file in files:
        splitFile = file.split('.')[0]
        splitFile = splitFile.split('-')
        emotions.append(int(splitFile[2]))
        paths.append(ravdess + dir + '/' + file)

#It will create a dataframe for each emotion for each file
emotions_data = pd.DataFrame(emotions,columns=['Emotions'])
#It will create a dataframe for each Path in the directory
paths_data = pd.DataFrame(paths,columns=['Path'])
combined = pd.concat([emotions_data,paths_data],axis=1)
#This will replace integer values in the dataframe with categorical values
```

```
combined.Emotions.replace({1:'neutral', 2:'calm', 3:'happy', 4:'sad',
5:'angry', 6:'fear', 7:'disgust', 8:'surprise'}, inplace=True)



print(combined.head(10))
data = []
X=[]
labels = []

#will return the dataset for each file and sample rate
def features(data,sr):
    # Mel Coeffienct cepstrum is the power spectrumm after a fourier
transform has been applied to the signal
    mfcc = librosa.feature.mfcc(y=data, sr=sr)
    mfccsmean = np.mean(mfcc.T,axis=0)
    return mfccsmean


con = []
emotion = []

for i in paths:
    data, sr = lb.load(i)
    extracted = features(data,sr)
    extracted = np.array(extracted)
    X.append(extracted)

X_matrix = np.matrix(X)
df = pd.DataFrame(X_matrix)
combinedPath = pd.concat([combined,df],axis=1)
#print(combinedPath.head(10))
combined = combinedPath.drop(columns=['Path','Emotions'])
X_train,X_test,y_train,y_test =
train_test_split(combined,combinedPath.Emotions,test_size=.1,random_state=
2)
X_train = np.array(X_train)
y_train = np.array(y_train)
X_test = np.array(X_test)
y_test = np.array(y_test)
```

```
X_train = np.expand_dims(X_train,axis=2)
X_test = np.expand_dims(X_test,axis=2)
print(X_test)

lb = LabelEncoder()
y_train = np_utils.to_categorical(lb.fit_transform(y_train))
y_test = np_utils.to_categorical(lb.fit_transform(y_test))


model = Sequential([

                    Conv1D(256, 6,
padding='same',input_shape=(X_train.shape[1],1)),
                    MaxPooling1D(pool_size=2),
                    Conv1D(filters=128, kernel_size=2,
activation='relu',padding='same'),
                    BatchNormalization(),
                    Conv1D(filters=128,
kernel_size=2,activation='relu',padding='same'),
                    MaxPooling1D(pool_size=2),
                    BatchNormalization(),
                    Dropout(0.1),
                    Bidirectional(LSTM(64,return_sequences=True)),
                    Dense(136,activation='relu'),
                    BatchNormalization(),
                    Dense(136,activation='relu'),
                    BatchNormalization(),
                    Dense(136,activation='relu'),
                    BatchNormalization(),
                    Dense(136,activation='relu'),
                    Dropout(.2),
                    BatchNormalization(),
                    Dense(24, activation='relu'),
                    BatchNormalization(),
                    Conv1D(filters=128,
kernel_size=2,activation='relu',padding='same'),
                    GaussianNoise(0.1),
                    Dropout(0.1),
                    BatchNormalization(),
                    Dense(34,activation='relu'),
                    BatchNormalization(),
                    Flatten(),
                    Dense(units=8, activation='softmax'),

])


model.summary()

model.compile(optimizer=Adamax(learning_rate=.0011),
loss='categorical_crossentropy',metrics = ['Accuracy'])


model_history = model.fit(X_train, y_train, validation_data=(X_test,
y_test), verbose = 2, epochs=500)



y_pred = model.predict(X_test)


model.save('Mode3')

print(y_pred)

from sklearn.metrics import confusion_matrix
y_pred=model.predict(X_test)
y_pred=np.argmax(y_pred, axis=1)
y_test=np.argmax(y_test, axis=1)
cm = confusion_matrix(y_test, y_pred)
fig, ax = plot_confusion_matrix(conf_mat=cm, figsize=(6, 6),
cmap=plt.cm.Greens)
plt.xlabel('Predictions', fontsize=18)
plt.ylabel('Actual Values', fontsize=18)
plt.title('Confusion Matrix', fontsize=18)
plt.show()
print(cm)
```