Individual Report

Adhithya Kiran

Introduction and Project Goal:

The final project aims to analyze and compare various dynamic models, such as RNN or LSTM, and a convolutional model like CNN, in their ability to classify emotions following the listening of a speech or a short voice note.

Individual contribution and Code Implementation:

I have used the REVDESS dataset, where I have used an emotion dataset which other two models used as well.

The data set uses the naming convention from where we can identify the class of each file.

The RAVDESS (Ryerson Audio-Visual Database of Emotional Speech and Song) dataset is a collection of speech and song recordings by professional actors expressing different emotions. The dataset contains 7356 files in total, with each file named according to a specific naming convention.

The naming convention of the RAVDESS dataset is as follows:

The first two digits (01-24) represent the speaker ID.
The third digit (1 or 2) represents whether the recording is speech (1) or song (2).

The fourth digit (1-8) represents the emotion expressed in the recording, where 1 = neutral, 2 = calm, 3 = happy, 4 = sad, 5 = angry, 6 = fearful, 7 = disgust, and 8 = surprised.
The fifth digit (01-48) is simply an index number to differentiate between multiple recordings from the same speaker and with the same emotion.

Data visualization and Preprocessing

I have created a dataframe to store all emotions of data with their paths. I used this dataframe to extract features for our model training.

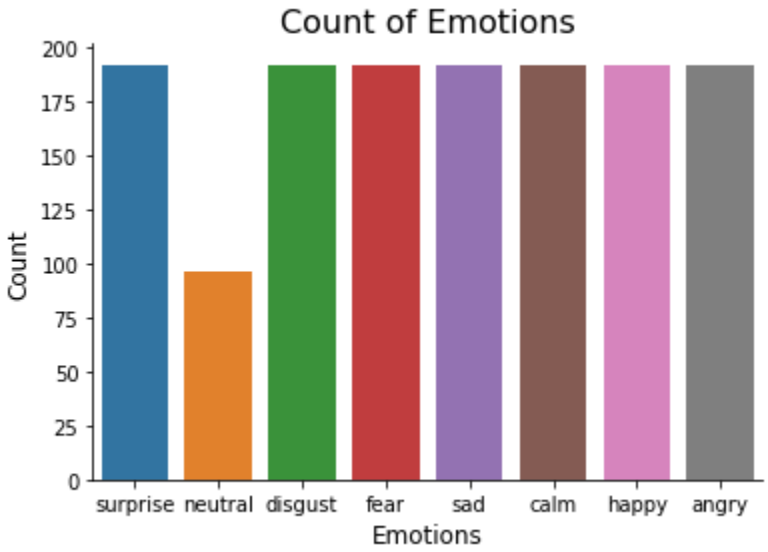
I have changed the integers to represent the emotions to actual emotions.

```
ravdess_directory_list = os.listdir(Ravdess)
file_emotion = []
file_statement = []
file_path = []
for dir in ravdess_directory_list:
   # as their are 20 different actors in our previous directory we need to
extract files for each actor.
   actor = os.listdir(Ravdess + dir)
   for file in actor:
       part = file.split('.')[0]
       part = part.split('-')
       # third part in each file represents the emotion associated to that
auido.
       file_emotion.append(int(part[2]))
       file_statement.append(int(part[4]))
       file_path.append(Ravdess + dir + '/' + file)

# dataframe for emotion of files
# emotion_df = pd.DataFrame(file_emotion, columns=['Emotions'])

# dataframe for path of files.
# path_df = pd.DataFrame(file_path, columns=['Path'])
# Ravdess_df = pd.concat([emotion_df, path_df], axis=1)
Ravdess_df = pd.DataFrame({"Emotions": file_emotion, "Statement":
file_statement, "Path": file_path})

# changing integers to actual emotions.
Ravdess_df.Emotions.replace(
```

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

Data Visualization

Data visualization for the count of emotions in each set of emotions to see any data imbalance.



Data Augmentation

Performed data augmentation. The dataset is balanced but i found that the dataset has only few train data. So inorder to diversiy and exapnd the dataset.

```
def noise(data):
   noise_amp = 0.035*np.random.uniform()*np.amax(data)
   data = data + noise_amp*np.random.normal(size=data.shape[0])
   return data

def stretch(data, rate=0.85):
   return librosa.effects.time_stretch(data, rate=rate)

def shift(data):
```

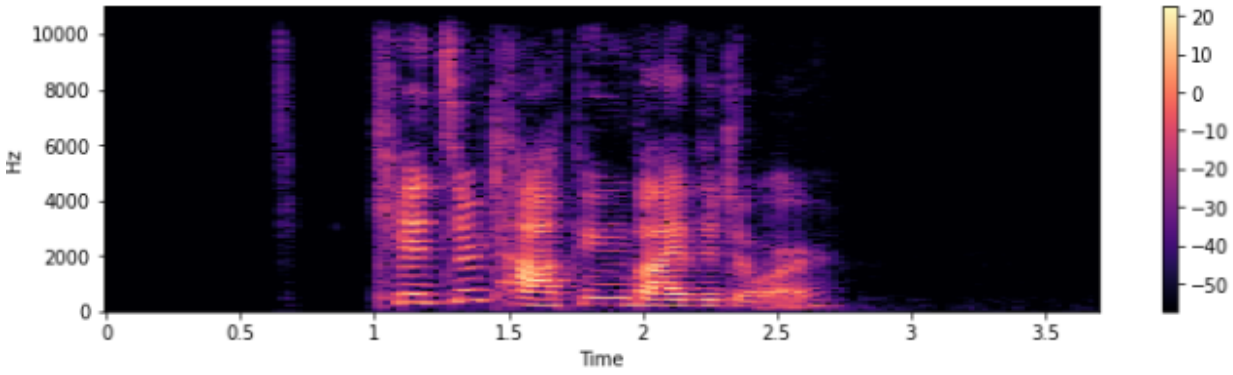
```
    shift_range = int(np.random.uniform(low=-5, high = 5)*1000)
    return np.roll(data, shift_range)

def pitch(data, sampling_rate, pitch_factor=0.7):
    return librosa.effects.pitch_shift(data, sr=sampling_rate,
n_steps=pitch_factor)
```

Feature Extraction:

I used MFCC mel Frequency Cepstral Coefficients. duration of the audio is set to 2.5 and offset satrting and enfding is set to 0.6 that no audio in start and ending.

```
def extract_features(data):
    # MFCC
    mfcc = librosa.feature.mfcc(y=data, sr=sample_rate)
    result = mfcc
    return result

# funtion to transform audio
def transform_audio(data, fns):
    fn = random.choice(fns)
    if fn == pitch:
        fn_data = fn(data, sampling_rate)
    elif fn == "None":
        fn_data = data
    elif fn in [noise, stretch]:
        fn_data = fn(data)
    else:
        fn_data = data
    return fn_data


def get_features(path):
    # duration and offset are used to take care of the no audio in start and the
ending of each audio files as seen above.
    data, sample_rate = librosa.load(path, duration=2.5, offset=0.6)


    #randomly transforming audio data
    fns = [noise, pitch, "None"]

    fn1_data = transform_audio(data, fns)
    fn2_data = transform_audio(fn1_data, fns)
```

```
    res1 = extract_features(fn2_data)
    result = [np.array(res1[:,:108])]

    fn1_data = transform_audio(data, fns)
    fn2_data = transform_audio(fn1_data, fns)
    res2 = extract_features(fn2_data)[:,:108]
    result.append(res2)#np.vstack((result, res2)) # stacking vertically

    fn1_data = transform_audio(data, fns)
    fn2_data = transform_audio(fn1_data, fns)
    res3 = extract_features(fn2_data)[:,:108]
    result.append(res3)#np.vstack((result, res3)) # stacking vertically

    return result
```



OneHotEncoder

I have used one hot encoding for the output.

Model

LSTM Model

I have used LSTM Model to build the model to classify emotions in audio files.I believed that LSTM can perform better on training audio data.This is because LSTM can effectively capture long-term dependencies in the input data, which is crucial for processing audio data, where the signal may contain complex patterns and dependencies that span over a long period of time.

In the context of training an audio file, the LSTM layer in the model is used to process the sequence of audio frames in a temporal order and capture the long-term dependencies in the audio signal. By doing so, the model can learn to identify the underlying patterns and features in the audio data that are relevant for the given classification task, such as speech recognition or music genre classification.

I have used 80%,20% train test split and the data processing is all identical as other two models including python package librosa for preprocessing, augmentation,feature extraction,normalization ect, one hot encoding ect.

Initially I have used a model with just one TimeDistributed Conv1D, which gives accuracy of 75% after 100 epochs. Later I added more TimeDistributed Conv1D layers to make the model complex, with 2 layers of LSTM. This model gives me a train accuracy of 90% and Test accuracy of 80% in 35 epochs. Inorder not to overfit , I have only used 40 epochs in the latest code.

I have build an LSTM model with 9 layers:

6 layers of TimeDistributed Conv1D with BatchNormalization and ReLU activation.
1 layer of TimeDistributed Flatten.
2 layers of LSTM with Dropout regularization.
1 layer of Dense with BatchNormalization, ReLU activation, and Dropout regularization.
1 layer of Dense with softmax activation.

The model is a deep learning model for sequence classification. It has a total of 9 layers. The first 6 layers are TimeDistributed layers that apply 1D convolutional filters to each time step of the input sequence independently. The number of filters in each convolutional layer increases from 32 to 128. Batch normalization is applied after each convolutional layer to speed up the training process.

The seventh layer is a TimeDistributed Flatten layer that flattens the output of the convolutional layers into a 2D matrix.

The eighth and ninth layers are LSTM layers, which are a type of recurrent neural network (RNN) that are commonly used for sequence classification tasks. The first LSTM layer has 64 units and returns sequences, while the second LSTM layer has 32 units and does not return sequences. Dropout regularization with a rate of 0.2 is applied after each LSTM layer to prevent overfitting.

The last three layers are fully connected layers. The first dense layer has 64 units with a ReLU activation function, followed by a batch normalization layer and dropout regularization with a rate of 0.2. The second dense layer has 8 units with a softmax activation function, which outputs the probability distribution over the 8 possible emotion classes.

Overall, the model is designed to take in an input sequence of speech or music features, apply convolutional and LSTM layers to extract temporal and spectral features, and use fully connected layers to classify the input sequence into one of 8 emotional categories.

Why does LSTM have better performance compared to RNN?

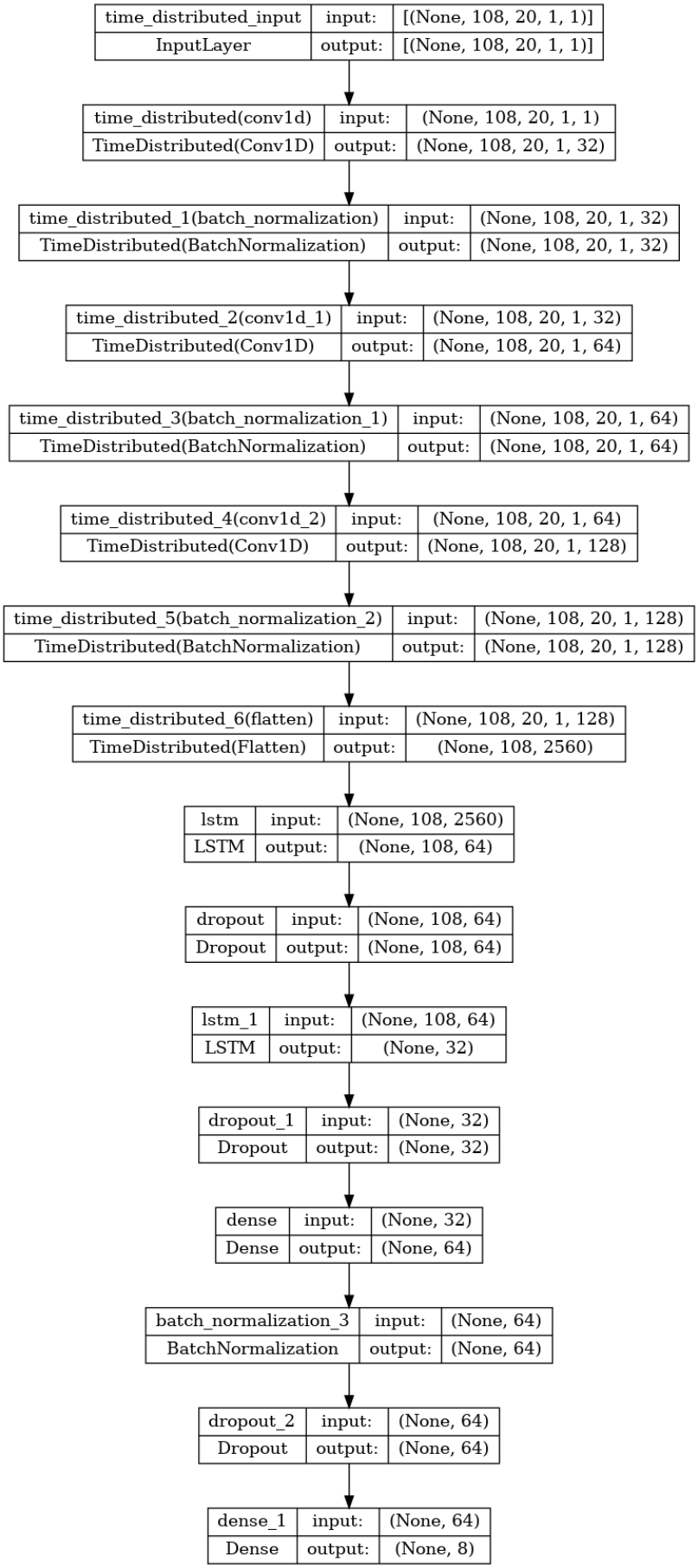
Convolutional layers: The use of convolutional layers in the model have helped to extract meaningful features from the input data. Convolutional layers are particularly effective at capturing local patterns in sequential data, such as changes in amplitude or frequency.

LSTM layer: The model includes an LSTM layer, which is a type of recurrent neural network that can capture temporal dependencies in the input data. LSTM layers are particularly effective when the data has long-term dependencies or when there is a time lag between input and output.

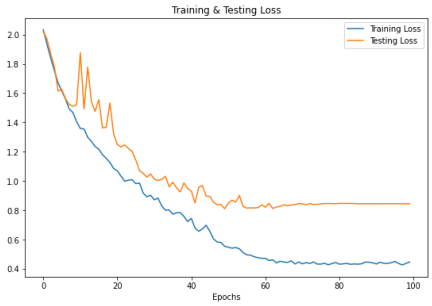
Dropout layers: Dropout layers were used in the model to reduce overfitting, which can occur when the model learns noise or idiosyncrasies in the training data that do not generalize to new data.

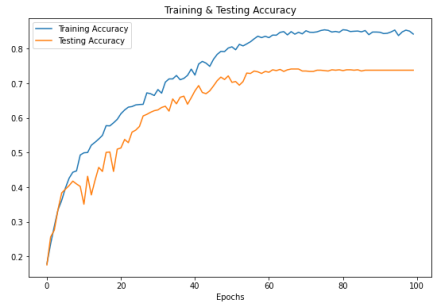
BatchNormalization layer: The BatchNormalization layer may have helped to stabilize the training process and improve the generalization performance of the model. This layer normalizes the activations of the previous layer, making the training process more stable and efficient.

Softmax activation: The final Dense layer in the model uses a softmax activation, which is commonly used in classification problems. The softmax function converts the output of the model into probabilities, making it easier to interpret the output and evaluate the model's performance.


time_distributed_input | input: | [(None, 108, 20, 1, 1)]
InputLayer | output: | [(None, 108, 20, 1, 1)]
time_distributed(conv1d) | input: | (None, 108, 20, 1, 1)
TimeDistributed(Conv1D) | output: | (None, 108, 20, 1, 32)
time_distributed_1(batch_normalization) | input: | (None, 108, 20, 1, 32)
TimeDistributed(BatchNormalization) | output: | (None, 108, 20, 1, 32)
time_distributed_2(conv1d_1) | input: | (None, 108, 20, 1, 32)
TimeDistributed(Conv1D) | output: | (None, 108, 20, 1, 64)
time_distributed_3(batch_normalization_1) | input: | (None, 108, 20, 1, 64)
TimeDistributed(BatchNormalization) | output: | (None, 108, 20, 1, 64)
time_distributed_4(conv1d_2) | input: | (None, 108, 20, 1, 64)
TimeDistributed(Conv1D) | output: | (None, 108, 20, 1, 128)
time_distributed_5(batch_normalization_2) | input: | (None, 108, 20, 1, 128)
TimeDistributed(BatchNormalization) | output: | (None, 108, 20, 1, 128)
time_distributed_6(flatten) | input: | (None, 108, 20, 1, 128)
TimeDistributed(Flatten) | output: | (None, 108, 2560)
lstm | input: | (None, 108, 2560)
LSTM | output: | (None, 108, 64)
dropout | input: | (None, 108, 64)
Dropout | output: | (None, 108, 64)
lstm_1 | input: | (None, 108, 64)
LSTM | output: | (None, 32)
dropout_1 | input: | (None, 32)
Dropout | output: | (None, 32)
dense | input: | (None, 32)
Dense | output: | (None, 64)
batch_normalization_3 | input: | (None, 64)
BatchNormalization | output: | (None, 64)
dropout_2 | input: | (None, 64)
Dropout | output: | (None, 64)
dense_1 | input: | (None, 64)
Dense | output: | (None, 8)

(i)LSTM Architecture





(ii) LSTM Loss Accuracy curves for Train Test

```
27/27 [==============================] - 28s 1s/step - loss: 0.1383 - accuracy: 0.9614 - val_l
27/27 [==============================] - 2s 68ms/step - loss: 0.8045 - accuracy: 0.7918
Accuracy of our model on test data :  79.18128371238708 %
27/27 [==============================] - 3s 67ms/step
              precision    recall  f1-score   support
```

Final Notes

LSTM performs better than RNN, because LSTM can capture temporal data features. Also the first 6 layers which are timedistributed layers does convolution operation in time stamps which helps in better capturing the temporal data features.

LSTM perform 80% on train set and 75 on test set, which is way better when compared to pretrained cnn model which have accuracy score of 80-85%. The highest so far is 92%.

Model can be made more complex , but in order to compare the model we should adhere to a unified complexity for all models.

More can be done on this model such as augmentation done on real time and making the model complex.

	precision	recall	f1-score	support
angry	0.79	0.85	0.82	105
calm	0.85	0.81	0.83	108
disgust	0.77	0.76	0.77	121
fear	0.84	0.77	0.81	133
happy	0.80	0.82	0.81	100
neutral	0.70	0.71	0.70	62
sad	0.67	0.76	0.71	119
surprise	0.91	0.83	0.87	107
accuracy			0.79	855
macro avg	0.79	0.79	0.79	855
weighted avg	0.80	0.79	0.79	855

References:

https://www.researchgate.net/publication/344308187_Speech_Emotion_Recognition_in_Neurological_Disorders_Using_Convolutional_Neural_Network
https://pub.towardsai.net/speech-emotion-recognition-ser-using-cnn-and-lstms-4a5dc4c314fd
https://www.kaggle.com/code/blitzapurv/speech-emotion-recognition-using-lstm
https://www.ncbi.nlm.nih.gov/pmc/articles/PMC8618559/