FINAL PROJECT REPORT

EMOTION CLASSIFICATION USING SPEECH INPUT

Deep Learning 6303

Akshat Saini, Xiao Qi, Adhithya Kiran

INTRODUCTION

Abstract

The Ryerson Audio-Visual Database of Emotional Speech and Song (RAVDESS) contains 7,356 files (total size: 24.8 GB). The database contains 24 professional actors (12 female, 12 male), vocalizing two statements in English language. Speech includes calm, happy, sad, angry, fearful, surprise, and disgust expressions, and the song contains calm, happy, sad, angry, and fearful emotions. Each expression is produced at two levels of emotional intensity (normal, strong), with an additional neutral expression. The data set used for the final project specifically contains the speech data files and not the songs. There are 8 target classes in total and 1440 different audio files producing the target emotions, equally distributed between male and female actors.

Selection of emotions

Eight target emotions were selected for speech: neutral, calm, happy, sad, angry, fearful, surprise, and disgust. Calm and neutral are marked as baseline conditions, while the remaining emotions constitute the set of six basic or fundamental emotions that are thought to be culturally universal[1]. The concept of primary emotions has a long history in science. While using the model of emotion has been criticized in the past, it is fundamentally a practical choice in the creation and labeling of emotion sets.

EDA and Data Pre-Processing

Since the Data is in the format of Audio in .wav files, we converted the data into something that can be understood by the model to efficiently recognize different patterns and changes, while capturing shifts in tone and pitch frequency. This is usually done by converting the .wav files into mel values. For this particular purpose we used the python package Librosa which helped convert the different 1440 voice notes into their respective mel-spectrogram values, capturing shifts in tone and frequency which can help classify the different audio files into one of the 8 target emotion classes. The github repository for the project can be found using the link. The repository contains the different branches and a main readme file providing instructions on how to execute the code.

Each of the audio files in the data set are marked or named in a specific format which allows us to automate the data processing task using a python script.
For example, an audio file in the data set is named such as
'03-01-01-01-01-01-01.wav'

Understanding file formatting:

Modality (01 = full-AV, 02 = video-only, 03 = audio-only). - All files are 03.

Vocal channel (01 = speech, 02 = song). - All files are 01.

Emotion (01 = neutral, 02 = calm, 03 = happy, 04 = sad, 05 = angry, 06 = fearful, 07 = disgust, 08 = surprised).

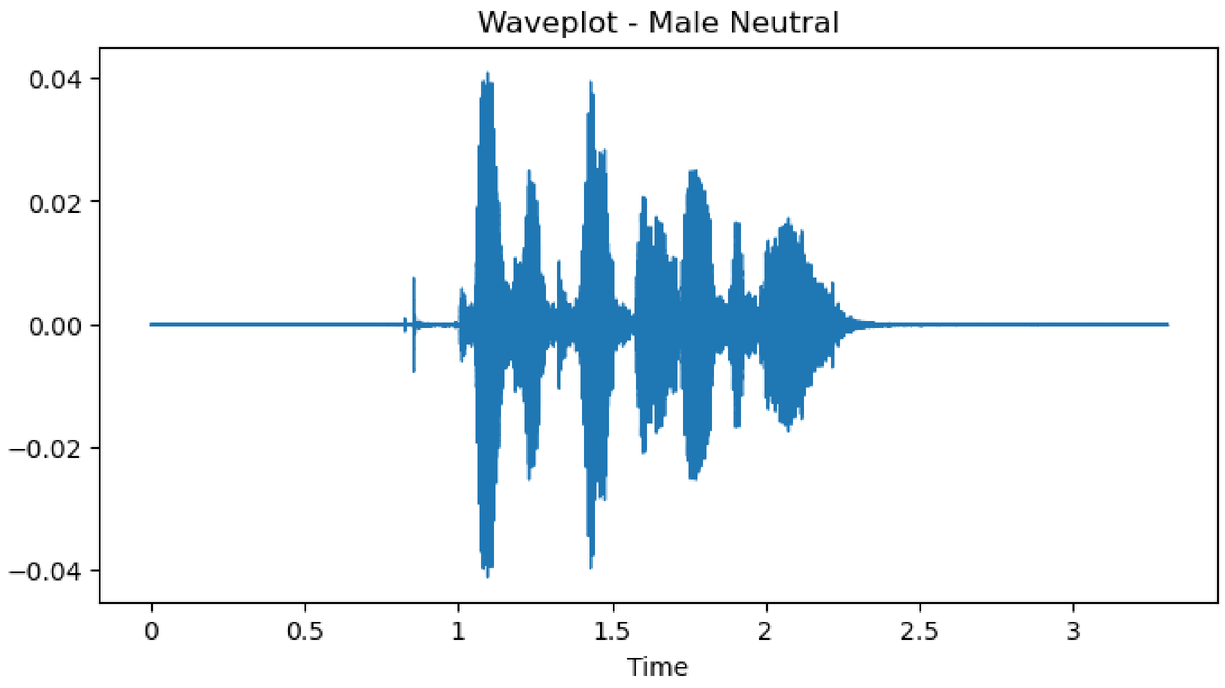
Emotional intensity (01 = normal, 02 = strong). NOTE: There is no strong intensity for the 'neutral' emotion.

Statement (01 = "Kids are talking by the door", 02 = "Dogs are sitting by the door").

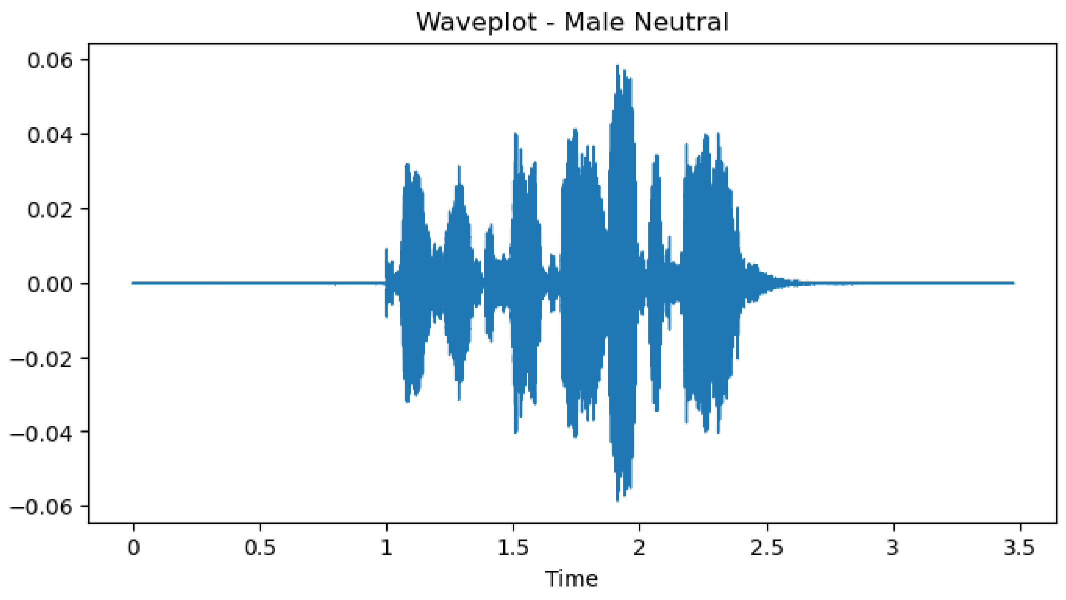
Repetition (01 = 1st repetition, 02 = 2nd repetition).

Actor (01 to 24. Odd numbered actors are male, even numbered actors are female).

Since we only have two statements in the entire data set, 01 = "Kids are talking by the door", 02 = "Dogs are sitting by the door", it is extremely important to capture shifts in tone and pitch of a file, since emotion can vary for the very same sentence. For example, the sentence: “I am here”, can be said in both a neutral and happy manner.

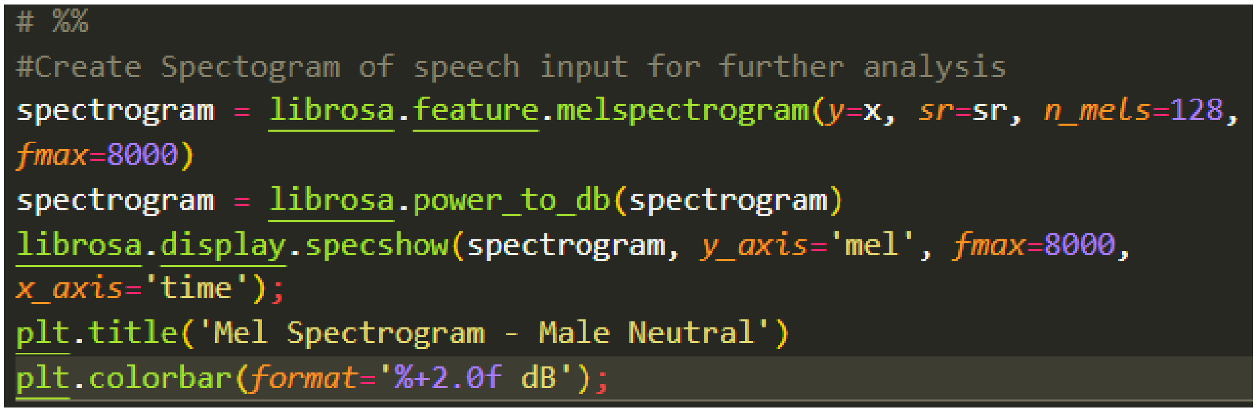


Waveplot of the sentence: “Kids are talking by the door” in a **neutral** manner



Waveplot of the sentence: "Kids are talking by the door" in a **happy** manner

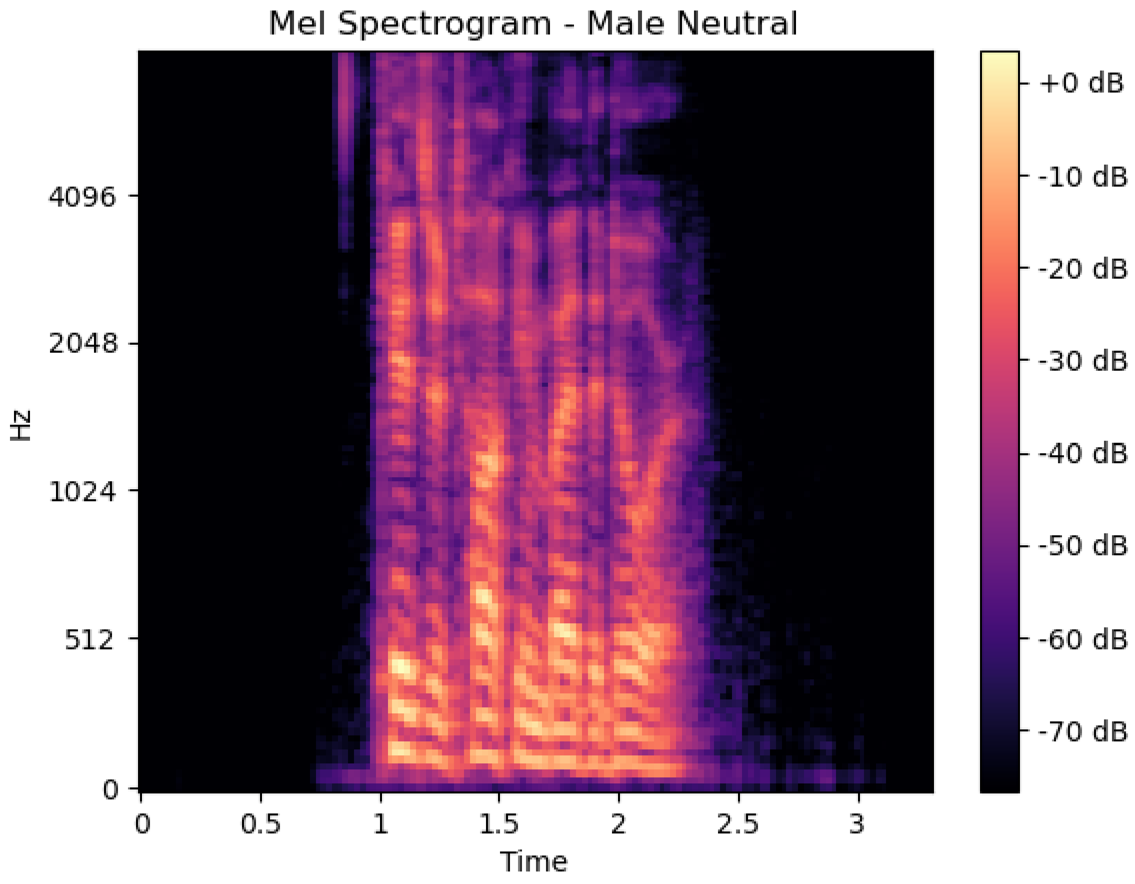
From the above two wave plots we can clearly notice the subtle changes for the same sentence, by the same person, just said in a different emotion. Once we have the audio time series values, we can use the librosa package to generate the mel-spectrogram values like so:



```
# %%
#Create Spectogram of speech input for further analysis
spectrogram = librosa.feature.melspectrogram(y=x, sr=sr, n_mels=128,
fmax=8000)
spectrogram = librosa.power_to_db(spectrogram)
librosa.display.specshow(spectrogram, y_axis='mel', fmax=8000,
x_axis='time');
plt.title('Mel Spectrogram - Male Neutral')
plt.colorbar(format='%+2.0f dB');
```

Here, the input x are the audio time series values and sr is the sampling rate of the input audio file.

Plotting the mel-spectrogram values on the X-Y plane results in a frequency time plot of the audio file. For the above example (neutral tone and first sentence), the Mel-Spectrogram plot looks like:



Further, we created a pandas dataframe containing the mel-spectrogram values of all the audio files. This helped in streamlining the overall process as we could then use the same dataframe for different models.

Data Augmentation

The original data set had 1440 different audio files. This seemed insufficient for creating a successful model. Hence, to tackle this particular problem we adopted the approach of data augmentation while maintaining balance within the data set.

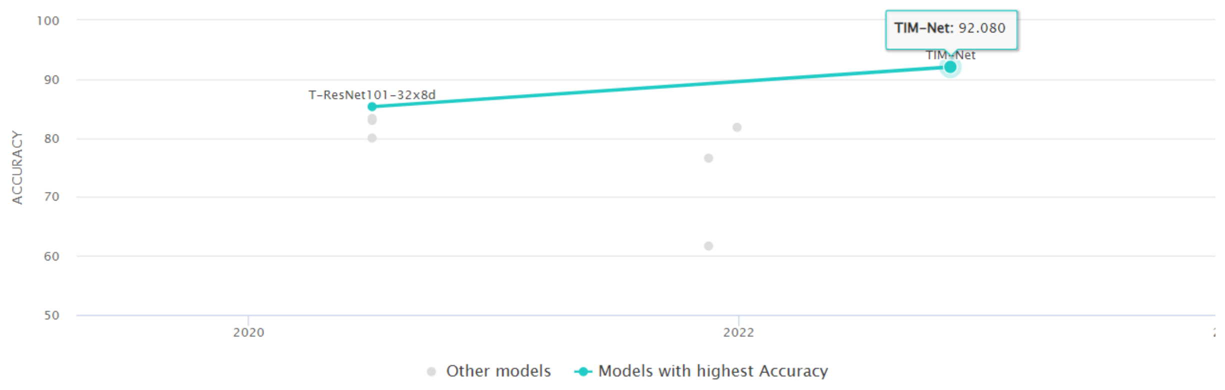
Audio augmentation or transformation can be done by using many techniques such as adding noise, change in pitch, audio shift etc. among many others.
We used the above methods to create multiple augmentations of the original audio file and added to our dataframe, increasing the size and variability in the final data set.

Padding
Because the original data is audio files which have different length, and the data augmentation changes its length as well. In order to get the vectors for the later usage, we need to do some padding work to make each vector the same length. So we found out the longest vector and added zeros for the other vectors. After doing that, we got the same length vector for training the models.

Data Normalization
Normalization is a preprocessing step that can help improve the performance of a model when working with audio data. Normalization usually refers to scaling the input data to a specific range or standardizing it to have zero mean and unit variance. Here, we used standardization as the normalization method.

Project Goal and Models

Historically, CNN models have performed optimally or best on audio data sets. This can primarily be attributed to the fact that CNN models are very nice at pattern recognition or mapping values to their target.



Our main goal for this project was to compare the performance of dynamic neural networks such as RNN and LSTM against a CNN model on the same dataset and using similar augmentation techniques.

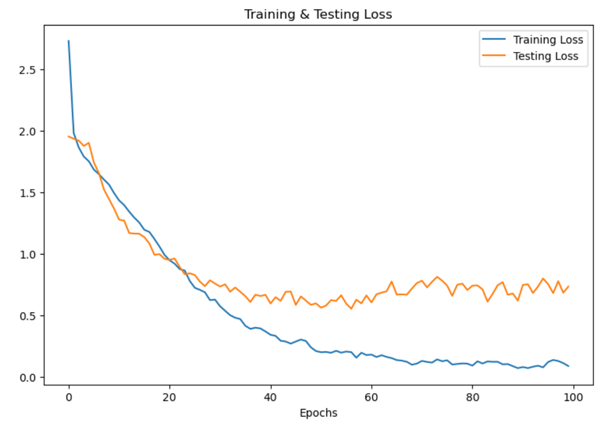
CNN Model
We started off by building a vanilla CNN model for which we just used the original dataset of 1440 audio files. Without any data augmentation or transformation, the CNN model performed an accuracy of 45% on the test set.
We conducted hyperparameter tuning and the model performance was more or less in the same region.

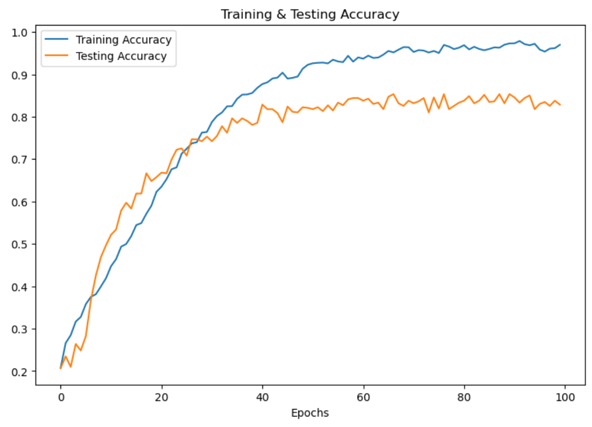
Following this, we used the existing CNN model with the same network architecture after performing data transformation and augmentation techniques such as noise, shift and pitch change. This both increased the overall size of the dataset and added variability in the final data. After fitting the model on train data, the model generated an accuracy of 85% on test data.

This boost in accuracy can be attributed to factors such as normalization, data augmentation and transformation.

We further compared the performance of this CNN model with dynamic neural networks like RNN and LSTM.

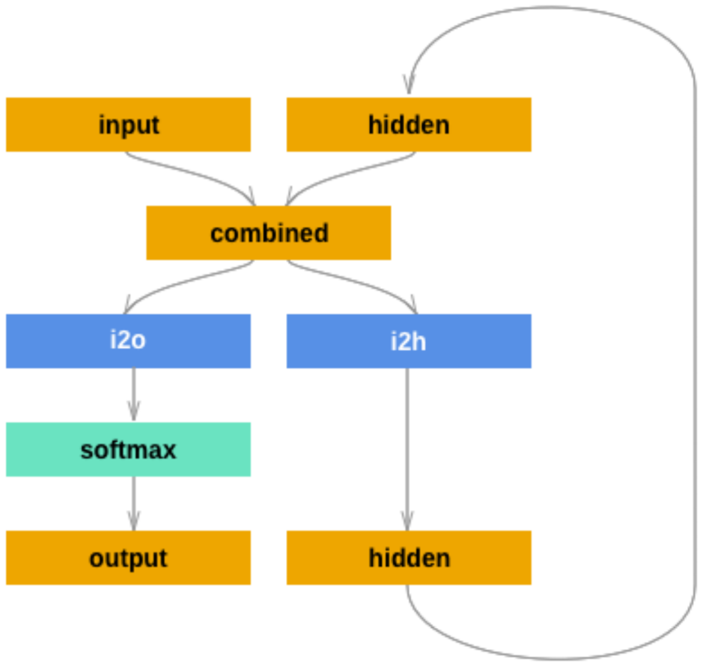
The accuracy and loss plots for the CNN model look like the below plots:
These were generated on both the training and the test sets:



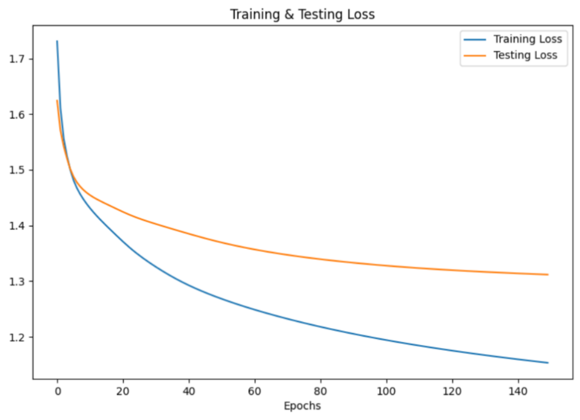


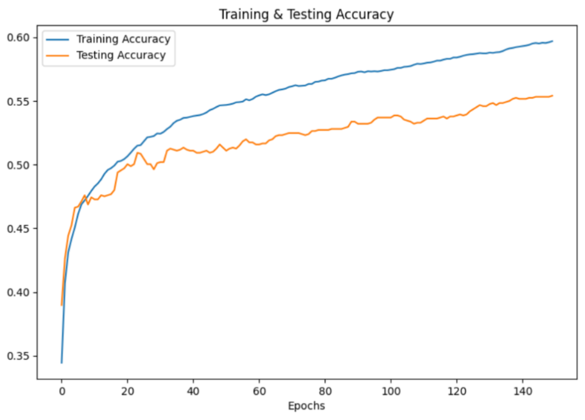
Elman RNN Model

The Elman RNN model was introduced in 1990. It has three layers: an input layer, a hidden layer, and an output layer. The hidden layer is connected to the input and output layers. This is the architecture of the Elman model, which is built on PyTorch. The input size is 50; the hidden size is 96 and the output size is 8.



In our project, We set the epoch as 150 to train the model. For the training process, we can find that the Elman RNN's accuracy is very low. Before the data augmentation, the test set accuracy is only 39% on vanilla dataset. After doing data augmentation, the test accuracy increases to 55%. Compared to the Lstm and Cnn model, the accuracy of Elman RNN is still very slow.





LSTM Model

We have used LSTM Model to build the model to classify emotions in audio files.We 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.

We 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.

We have build an LSTM model with 9 layers.

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.

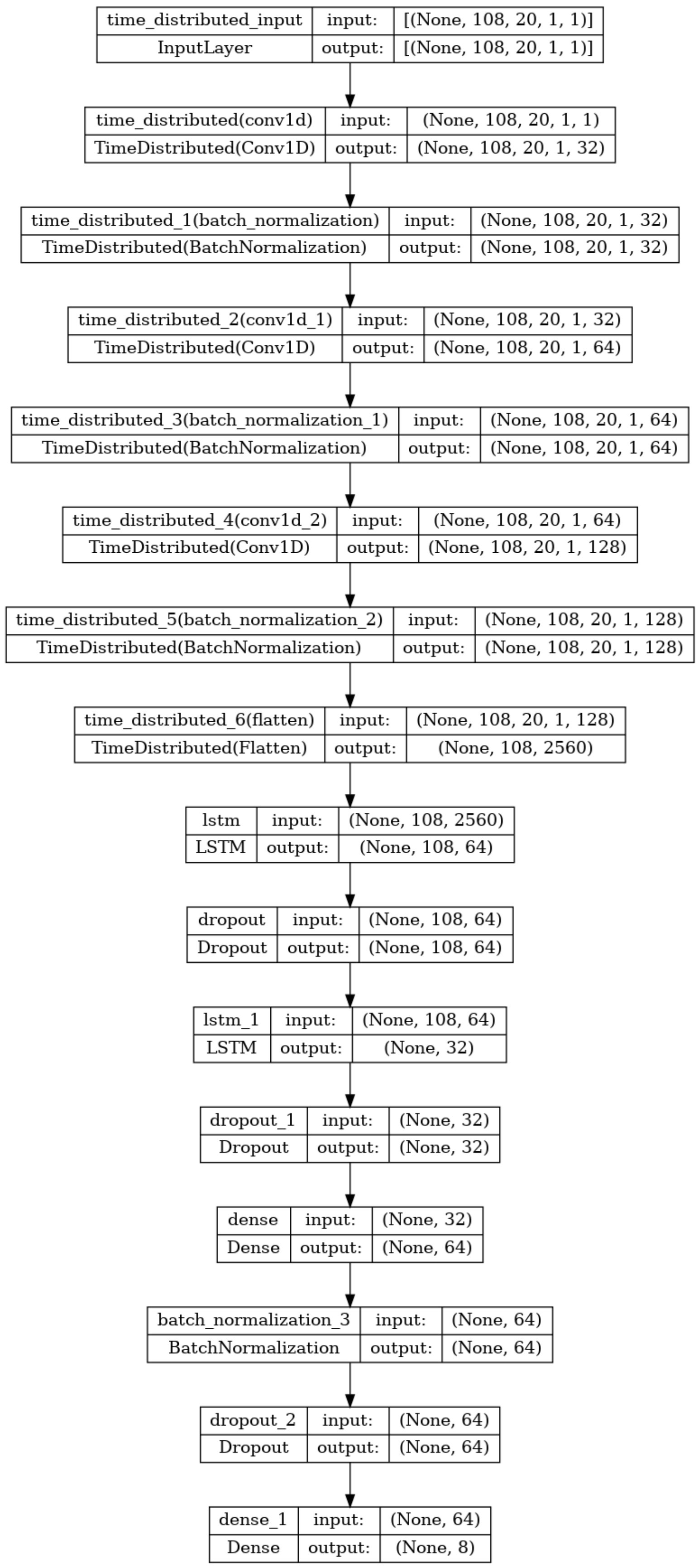
LSTM layer: The model includes an LSTM layer, which is a type of recurrent neural network that can capture temporal dependencies in the input data. LSTM layers are particularly effective when the data has long-term dependencies or when there is a time lag between input and output.

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.

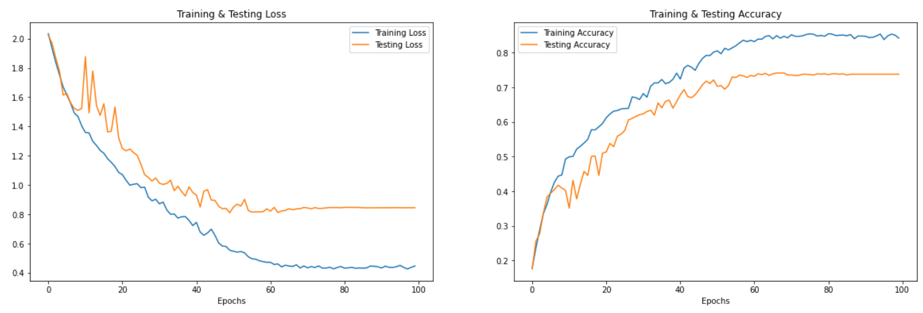
BatchNormalization layer: The BatchNormalization layer have helped to stabilize the training process and improve the generalization performance of the model.

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

Performance Comparison and Final Analysis

Compared with the LSTM model, Elman RNN has a lower accuracy. This is because LSTM has a more complex architecture that incorporates memory cells and gating mechanisms. However, the Elman model has its advantages, it has faster training rates and lower computational complexity. For our model, the training loop for Elman is 2-3 seconds, but for the Lstm model, it's 20 seconds.

Compared between CNN model and RNN model, it seems that CNN model is more suitable for this task. It's because this is a classification problem, and the audio files have only 3 seconds length. So the sequences of the data are not as important as the patterns in classification.

	CNN	Elman RNN	LSTM
Train accuracy	0.91	0.60	0.80
Test accuracy	0.85	0.55	0.75

Summary

The project focused on the classification of Ryerson Audio-Visual Database of Emotional Speech. We built three models separately to do this task and compare those three models. It shows that the CNN model is most suitable for this task, which has a test accuracy of 0.85. And the Elman RNN model has the least accurate result between the three models.

This could be because of its simple architecture and also because this particular dataset did not require the benefits or the usage of a feedback loop or delay, which models like RNN and LSTM can provide.

Finally, for future improvements and comparisons, we can try creating a third dataset just for conducting a comparative analysis between the different models. This approach can help us to avoid cherry picking the best scores, as there can be cases where models that have a high test accuracy can perform low on unseen data compared to models that had a low accuracy on test set, which can end up performing better on unseen data points.

References

Ekman P, Sorenson ER, Friesen WV. Pan-cultural elements in facial displays of emotion. Science. 1969;164(3875):86–8. Pmid:5773719

https://zenodo.org/record/1188976#.YFZuJ0j7SL8

Project Github Repository:
https://github.com/akisaini/DL-Project.git

Speech Emotion Recognition in Neurological Disorders Using Convolutional Neural Network
https://www.researchgate.net/publication/344308187_Speech_Emotion_Recognition_in_Neurological_Disorders_Using_Convolutional_Neural_Network

https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0196391

https://paperswithcode.com/dataset/ravdess

https://pytorch.org/tutorials/intermediate/char_rnn_classification_tutorial.html

https://github.com/gabrielloye/RNN-walkthrough/blob/master/main.ipynb

https://www.researchgate.net/publication/344308187_Speech_Emotion_Recognition_in_Neurological_Disorders_Using_Convolutional_Neural_Network

https://www.kaggle.com/code/blitzapurv/speech-emotion-recognition-using-lstm