Report

- Introduction

The objective of this assignment was for us to have hands on experience with audio signal processing and sequence modelling using RNN and LSTM.

In this assignment I learned how to manipulate audio data. How to load, preprocess and visualize it and tailor it for the task of Deep Learning.

Extracting features from the audio samples, building RNN and LSTM Deep Learning models for audio classification. And finally comparing the different models with each other using metrics like accuracy, loss and confusion matrics.

- Data Preprocessing

The feature I've used are MFCC features. Essentially, it's a way to represent the short-term power spectrum of a sound which helps machines understand and process human speech more effectively. MFCC coefficients contain information about the rate changes in the different spectrum bands. If a cepstral coefficient has a positive value, the majority of the spectral energy is concentrated in the low-frequency regions. On the other hand, if a cepstral coefficient has a negative value, it represents that most of the spectral energy is concentrated at high frequencies.

Regarding the parameters of MFCC, I've kept n_mfcc(number of mfcc features to output)as 40 in order to capture more frequency detail.

n_fft(size of the FFT window) as 400.

hop_length(number of samples to move between each frame)as 160.

n_mels(number of mel filterbanks used) as 40.

For padding I've used torch.nn.utils.rnn.pad_sequence while creating batches in Dataloader.

And for feature scaling I used the method provided in the tasks of the assignment.

I did face a few challenges in padding and normalization due to different shape and structures of the data I was working with.

- Model Architectures

I learned the concepts of both RNN and LSTM from Statquest plus a few different videos.

RNNs allow the network to "remember" past information by feeding the output from one step into next step. This helps the network understand the context of what has already happened and make better predictions based on that.

The input to a RNN model is: (batch_size, seq_length, features)

Where the batch size is the number of audio clips that are being fed in the model. Sequence length is the number of time steps the clips have been divided into.

And features is the number of mfcc features which describes each timestep.

In my RNN model, there is a simple vanilla RNN layer with hidden units=128. And then a fully connected layer to predict the subclasses. The activation function used is tanh, which maps the inputs to a range of -1 to 1. And dropout is optionally introduced to reduce overfitting.

Similarly, in the LSTM model, the vanilla RNN layer is replaced with an LSTM layer which helps capture long-term dependencies in sequential data more effectively. The LSTM uses both sigmoid and tanh functions. A dropout layer has been added to prevent overfitting

And the final linear layer which is used to make multiclass classification.

For the CNN-LSTM architecture,I have used 2 Convolution layers along with relu and mapooling. The mel spectrograms were passed to the 2D Convolution layer(Conv2d) which is used to extract important features from the spectrogram. ReLU activation function is used to introduce non linearity to the model in order to learn complex patterns. Then maxpooling was applied in order to reduce the dimension of the output but preserving the features.

The output of the CNN layers is then reshaped/permuted so that it has the correct shape to be fed into LSTM layers.

Lazy initialization of lstm layer is used and classification is made.

- Training & Evaluation

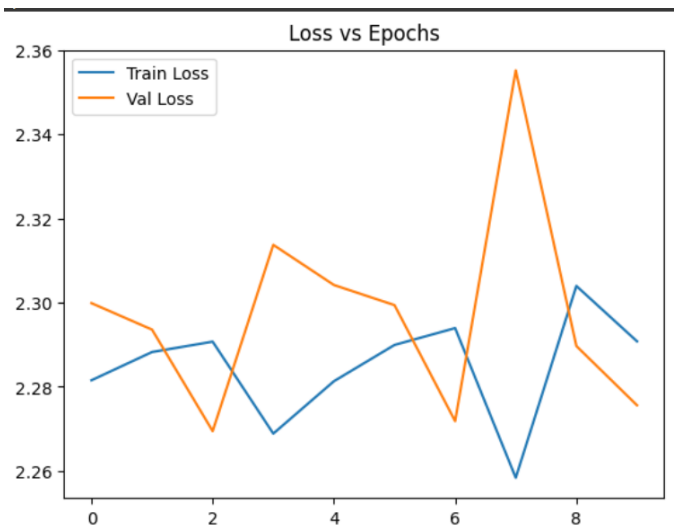
I have used CrossEntropyLoss as my loss function and the optimizer is Adam. For my vanilla RNN model I tried with 10, 20 and even 30 epochs but my loss and accuracy were almost constant for all pf the three.

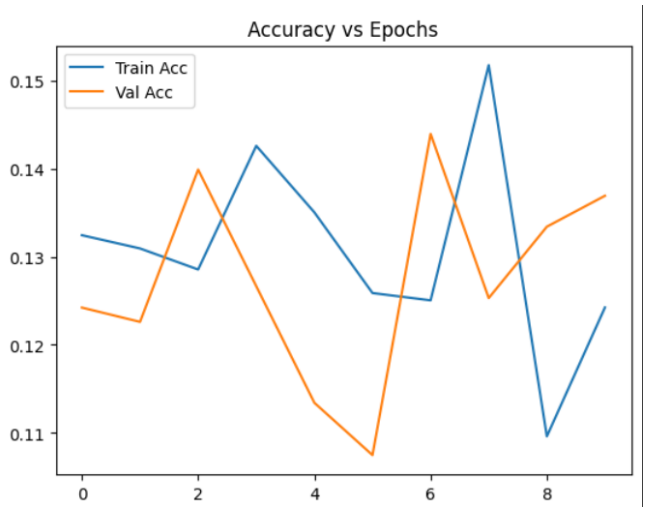
For the LSTM model, first I tried with 10 epochs, but I flet it can improve, then I tried 15 epochs and gained the best accuracy.

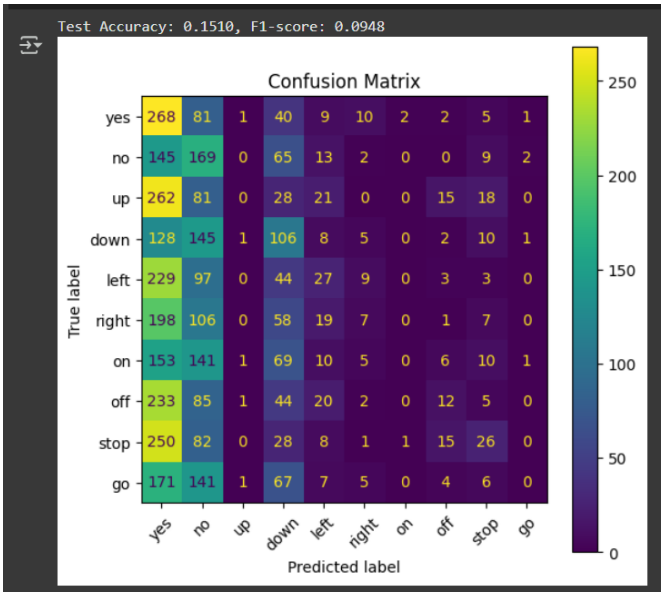
For the CNN Lstm model I tried 10, 15 as well as 20 epochs.

Training Validation Curves:

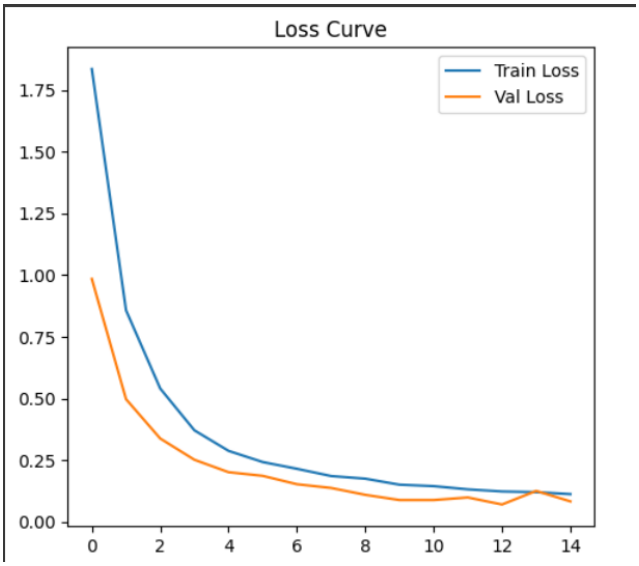
1)RNN Model:

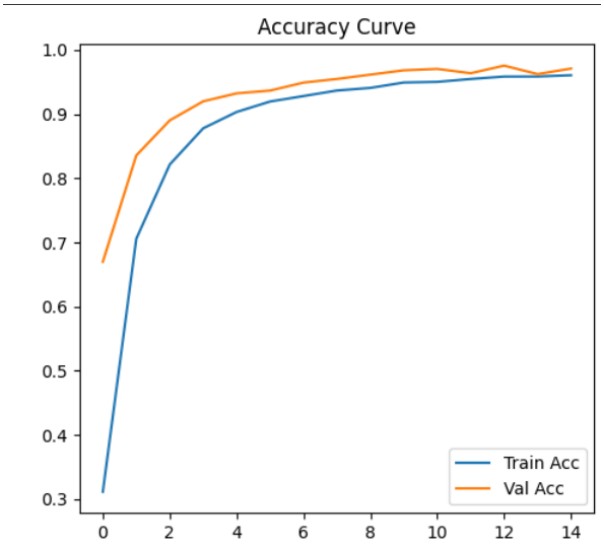


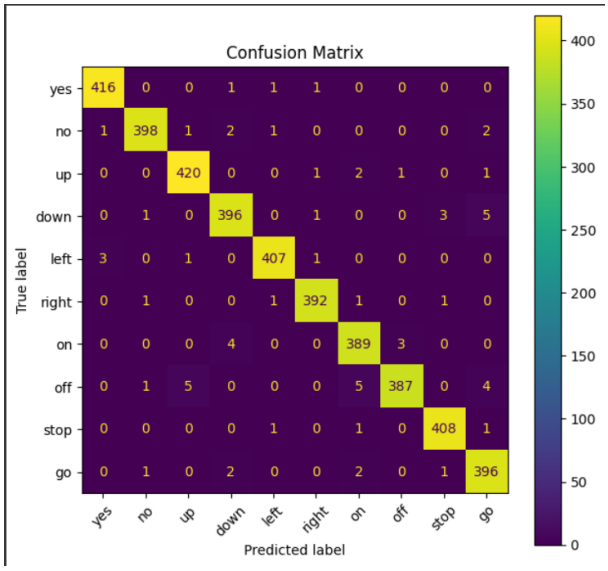



Test Accuracy: 0.1510, F1-score: 0.0948
Confusion Matrix
yes 268 81 1 40 9 10 2 2 5 1
no 145 169 0 65 13 2 0 0 9 2
up 262 81 0 28 21 0 0 15 18 0
down 128 145 1 106 8 5 0 2 10 1
left 229 97 0 44 27 9 0 3 3 0
right 198 106 0 58 19 7 0 1 7 0
on 153 141 1 69 10 5 0 6 10 1
off 233 85 1 44 20 2 0 12 5 0
stop 250 82 0 28 8 1 1 15 26 0
go 171 141 1 67 7 5 0 4 6 0
yes no up down left right on off stop go
True label
Predicted label
250
200
150
100
50
0

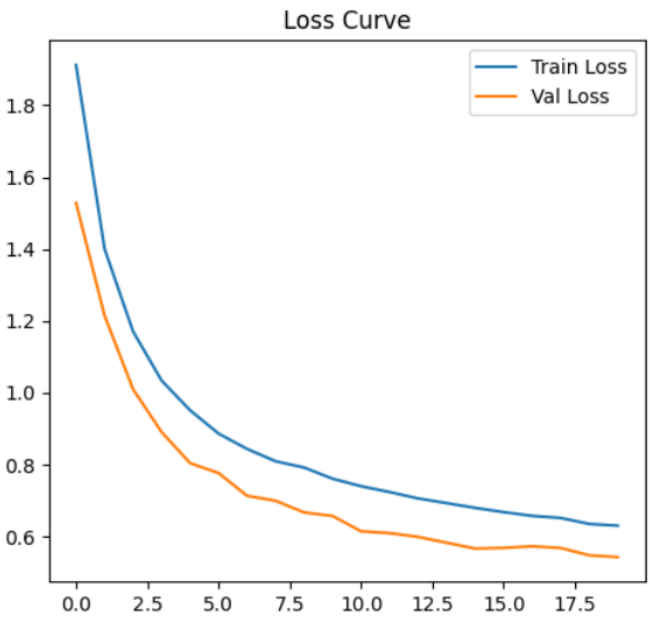
2) LSTM Model:

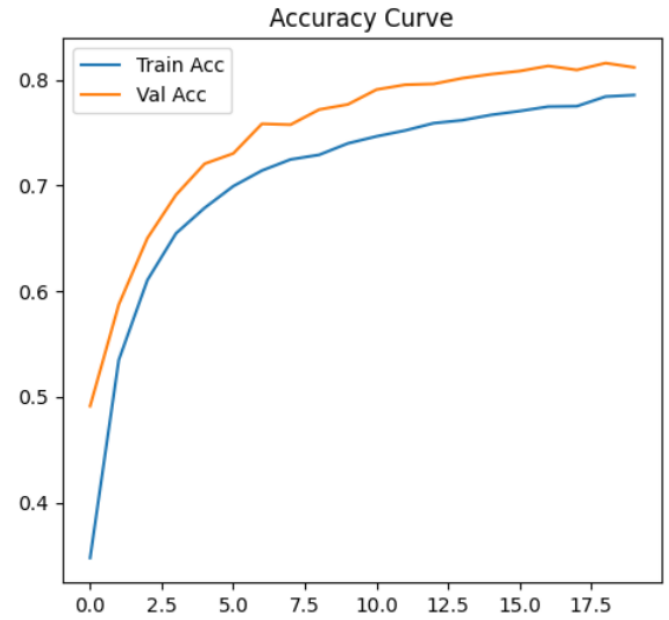

Loss Curve
Train Loss
Val Loss
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
0
2
4
6
8
10
12
14

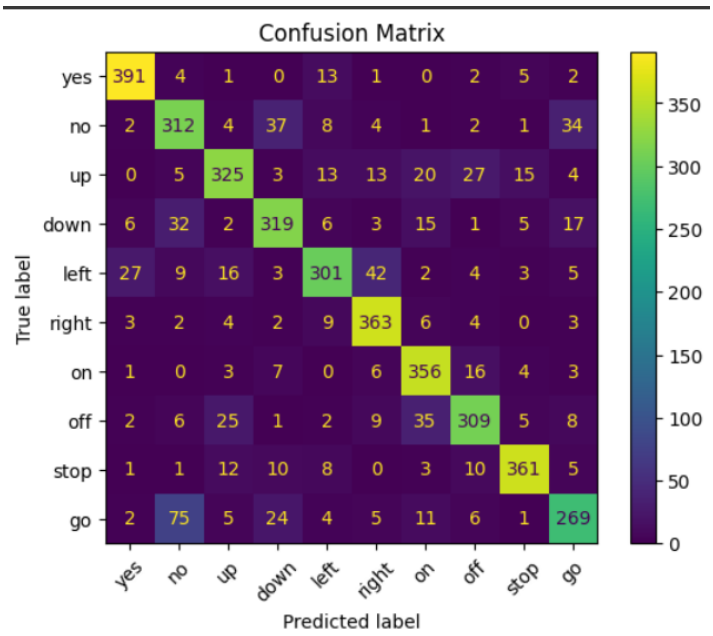

Accuracy Curve
Train Acc
Val Acc
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0
2
4
6
8
10
12
14


Confusion Matrix
yes 416 0 0 1 1 1 0 0 0 0
no 1 398 1 2 1 0 0 0 0 2
up 0 0 420 0 0 1 2 1 0 1
down 0 1 0 396 0 1 0 0 3 5
left 3 0 1 0 407 1 0 0 0 0
right 0 1 0 0 1 392 1 0 1 0
on 0 0 0 4 0 0 389 3 0 0
off 0 1 5 0 0 0 5 387 0 4
stop 0 0 0 0 1 0 1 0 408 1
go 0 1 0 2 0 0 2 0 1 396
yes no up down left right on off stop go
True label
Predicted label
400
350
300
250
200
150
100
50
0

3) CNN-LSTM Model:







- Comparative Analysis

The best performing model of all the three was the LSTM model with an accuracy of 98.4%. Then comes the CNN-LSTM model with an accuracy of ----. The worst performing model was the vanilla RNN model which failed to capture long term dependencies

Th LSTM model converged the fastest reaching 98% accuracy in just 15 epochs. While the CNN-LSTM Model took - epochs to reach an accuracy of __.

From the confusion matrices it can be seen that the simple RNN model wrongly classifies most of its samples as yes, which may be a result of underfitting. As seen from the matrices of LSTM and CNN-LSTM it is evident that they are better at audio classification than a simple RNN model.

- ## Error Analysis & Improvements

The vanilla RNN often misclassified samples as "yes," mainly due to class imbalance and its limited capacity to capture complex temporal features in audio. Confusion matrices showed it struggled with distinguishing between similar-sounding words, indicating poor generalization.

When we switched to LSTM and CNN-LSTM models, major improvement was seen. With LSTM model performing nearly perfect.

Hyper tuning n_mffc to 40 from the common 13 also increased my models accuracy.

I tried different learning rates and 0.001 proved to be the best.

- ## Conclusion

Overall, the LSTM model performed the best while the vanilla RNN model failed to capture the features and make proper predictions.

Incorporating attention mechanisms or using transformer based models along with advanced data augmentation would further improve the accuracy of these models.