Comparison of Neural Networks in Speech Recognition

Project Report

Course:	CSIT 697 Masters Project in Computer Science - SPRING 2020
Instructor:	Dr. Dawei Li
Submitted By:	Litha Thampan
Supervisory Committee:	Dr. Dawei Li Dr. John Jenq Dr. Weitian Wang

Acknowledgement

On the very outset of this report, I would like to express my sincere and heartfelt obligation towards all the personages who have helped in this endeavor. Without their active guidance, help, cooperation and encouragement, I would not have made headway in this project.

I am ineffably indebted to **Dr. Dawei Li** for his continuous guidance and encouragement to accomplish this assignment. His calm and collected attitude helped me to get the project on the right direction and allowed me to create a complete work.

I would also like to thank the experts [**Dr. John Jenq** and **Dr. Weitian Wang**] who were involved in the validation for this project

I would also like to extend my gratitude to **Montclair State University, Montclair, New Jersey** for giving me this opportunity.

I also acknowledge with my deep sense of reverence, my gratitude towards the members of my family who have always supported me morally as well as economically.

Thanking you,
Litha Thampan

Table of Contents

Speech Recognition: Comparison of Neural Networks

Litha Thampan
thampanl1@montclair.edu
Montclair State University New Jersey, USA

Abstract

Speech Recognition is one of the areas in machine learning with high amount of interest. Sequence to Sequence models of neural networks (NN) are considered to be the de facto way for machine learning in the area of speech recognition. This project is a comparison of different neural network models that are applied on speech data for end to end recognition. Different combinations of NN layers offered by Keras are used for comparing various models. A final model is created based on the behavior of various models. The project also provides a live ASR (automatic speech recognition) feature that can be applied to any model.

1. Introduction

The process of transcribing speech in an input audio stream is known as speech recognition. Since the input and output are sequences of data, input being sequence of sound waves and output being sequence of letters, a sequence to sequence learning process fits perfectly for this kind of process. Recurrent neural network (RNN) is widely used in speech recognition since the output transcription influences the letters or words that follows them. This project is exploring character-based learning approach from acoustic features created as Spectrogram or MFCC (Mel-frequency cepstral coefficients).

The base code used in the project is from training framework code of Udacity's AIND VUI Capstone Model [1]. This template code is helpful for getting started with the development of Keras models. As part of this project, the code base was also re-platformed to work on latest versions of TensorFlow (2.1), CUDA (10.1) and other python modules. A live conversion module is also added as part of this project.

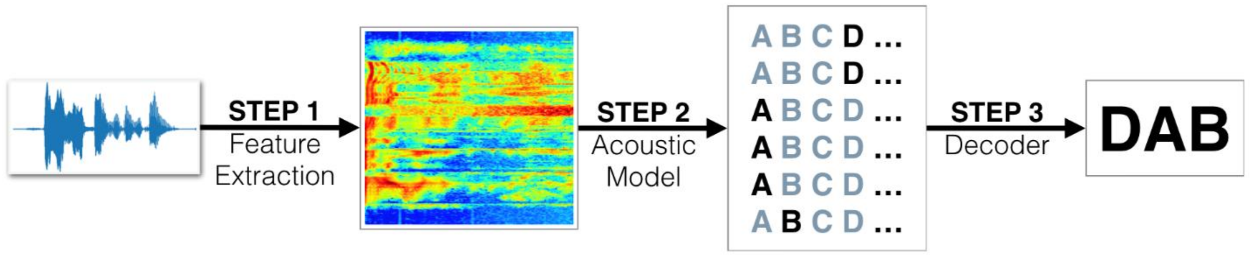


Figure 1: Overall Pipeline [1]

Figure 1 showcases the overall design of the model. As the first step the wav file is converted into a computable feature model such as Spectrogram or MFCC. This is used as features for the training and validation of models. Librispeech also provides text transcriptions of the wav files as text files. These are converted to numerical form using a character-to-number encoder which is then used as labels for the training and validation exercise. A character-based language model is used with Connectionist temporal classification (CTC) loss calculation to arrive at different probable outputs. CTC Decoder is applied to predict the output and validate against the actual text. Majority of the work is put in defining different acoustic models and finetuning the hyperparameters for optimal executions in terms of time and loss reduction.

2. Related Work

Some of major state-of-the-art (SOTA) speech recognition frameworks in recent times are Facebook's wav2letter, NVIDIA's openseq2seq and Baidu Research's deepspeech. All these frameworks are continuously improved on a day to day basis and has influenced in some of the decisions in this project. The Spectrogram feature definitions are defined by Baidu Research's deepspeech [2]. Spectrogram and MFCC features were used in research performed by Gevaert et al., which performed multilayer feedforward networks with backpropagation and radial basis functions [3]. Language models can contain lexicons (words) of the particular language to allow the deep learning to understand vocabulary. This project uses a character-based language models which has been proven to be as effective by Likhomanenko et al., [4]. Lexicon-free approach allows decoder to handle out-of-vocabulary (OOV) words: the decoder and the language model are both responsible for scoring words and restrict the vocabulary. Facebook's research on sequence to sequence speech recognition with Time Depth's Separable layer introduces a fully convoluted network after the convolutional layer [5]. HMM (Hidden Markov Model) models with bidirectional layers were analyzed using CTC loss functions in the research conducted by Maas et al. [6]

3. Data Description

Dataset used for this project is the Librispeech from openslr.org. The LibriSpeech corpus is derived from audiobooks that are part of the LibriVox project and contains 1000 hours of speech sampled at 16 kHz [7]. The dataset is further divided into distinct speech set of different sizes as shown in Table 1. This is the dataset that are used in almost all the state-of-the-art speech recognition systems. This data set also provides pre-built language models of the speech set. In this project, these pre-built language models are not utilized. Instead, a character-based language model is created.

subset	hours	per-speaker minutes	female speakers	male speakers	total speakers
dev-clean	5.4	8	20	20	40
test-clean	5.4	8	20	20	40
dev-other	5.3	10	16	17	33
test-other	5.1	10	17	16	33
train-clean-100	100.6	25	125	126	251
train-clean-360	363.6	25	439	482	921
train-other-500	496.7	30	564	602	1166

Table 1: Librispeech data subsets [7]

Among these subsets of data, dev-clean is used in this project to work with various models as training data set and test-clean is used as the validation data set. The final model is trained using train-clean-360 as the training dataset and test-clean as the validation data set.

4. Data Pre-Processing

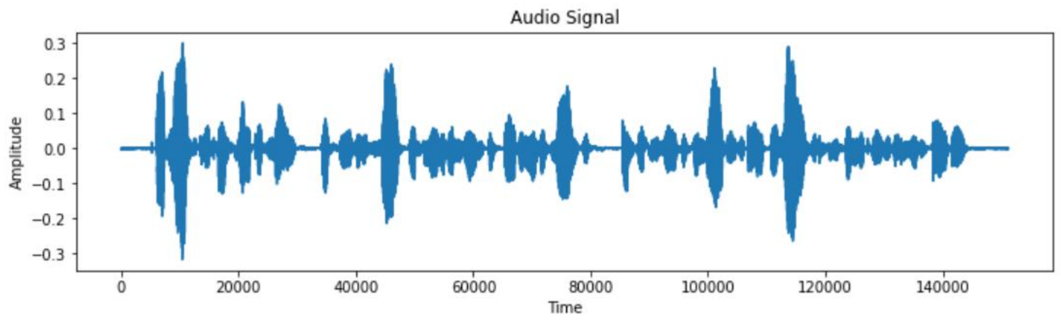
LibriSpeech is a fully processed dataset ready to be used for speech recognition machine learning models. The data set provides the audio data in `.flac` (Free Lossless Audio Codec) files which is an audio coding format for lossless compression of digital audio. For feature creations and audio visualizations, all the flac files had to be converted to a more popular wav format. The avconv utility of Linux is used to convert the flac files to wav.

Training and Validation corpus is created as json file with each audio file data in the following format

{"key": "wavefilename.wav", "duration": seconds, "text": "transcription of the wav file"}

5. Feature Extraction

There are two types of feature representations used in this analysis, namely Spectrogram and MFCC. An audio is a time-sequence of various amplitude and frequencies. It can be represented as shown in Figure 2. As it can be observed, different parts of the speech produce different amplitude and frequency. The features such as frequencies, amplitude and time are converted to digital representations such as Spectrogram or MFCC. These are used as input layers to the models.



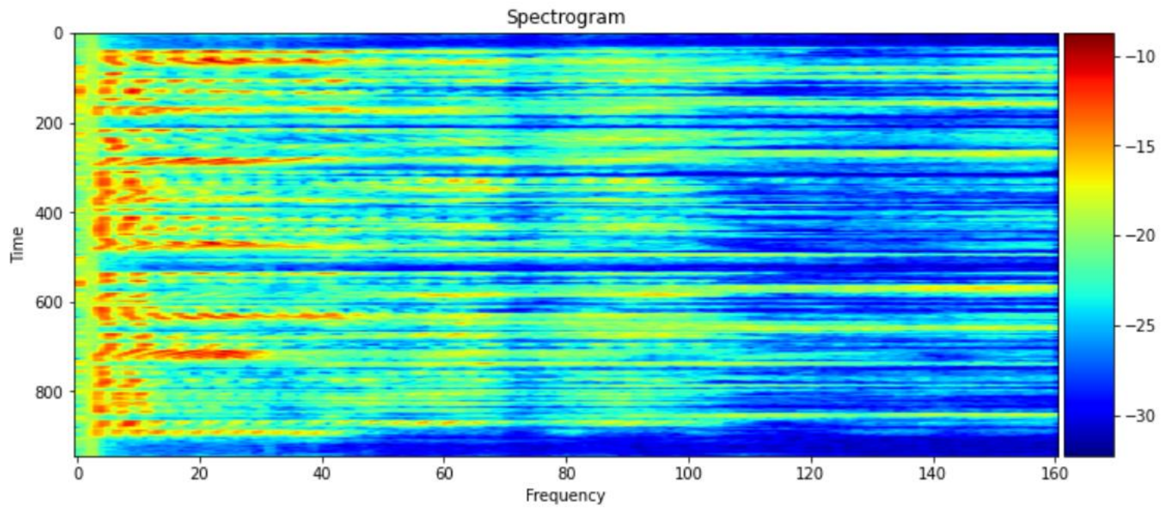
Shape of Audio Signal : (151200,)

Transcript : at last he sent word to say that he himself would be in england before the end of march and would see that the majesty of the law should be vindicated in his favour

Figure 2: Wav File Representation of one of the training entries

5.1. Spectrograms

A spectrogram is a visual representation of the spectrum of frequencies of a signal as it varies with time. When applied to an audio signal, spectrograms are sometimes called sonographs, voiceprints, or voicegrams. Spectrogram is a 3-dimensional data set where one axis represents time and other axis represent frequency. A third axis represented by different colors, signifies different amplitudes of each frequency at that point of time. Spectrogram is mainly used to identify the concept called timbre. Timbre means the quality of sound. Figure 3 shows an example of a spectrogram created by this project for the wav file shown in Figure 2. It can be observed that the spectrogram is of shape 944 (time) * 161 (frequency) * 27 (amplitude)



Shape of Spectrogram : (944, 161)

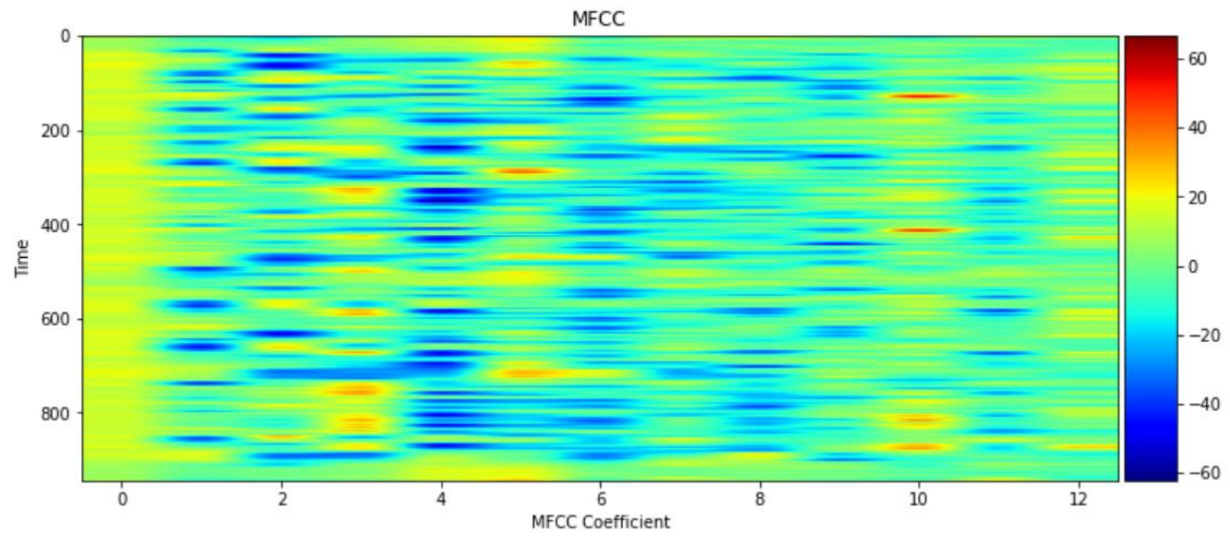
Figure 3: Spectrogram

Digital process of creating spectrogram is as follows. Digitally sampled data, in the time domain, is broken up into chunks, which usually overlap, and Fourier transformed to calculate the magnitude of the frequency spectrum for each chunk. Each chunk then corresponds to a vertical line in the image; a measurement of magnitude versus frequency for a specific moment in time (the midpoint of the chunk). These spectrums or time plots are then "laid side by side" to form the image or a three-dimensional surface or slightly overlapped in various ways, i.e. windowing [8].

In this project, a 0.001 ratio is applied to the sampling rate (16kHz) resulting in 161 as the dimension of frequency. A window size of 20 is applied to mitigate the loss with step size of 10 between the windows.

5.2. MFCC

The second method used in this project to represent audio feature is MFCC. Mel-Frequency Cepstral Coefficients of a signal are small set of features which describe the overall shape of a spectral envelope [9]. MFCC feature is much lower dimensional than the spectrogram feature, which could help an acoustic model to avoid overfitting to the training dataset. The mel-frequency cepstrum (MFC) is a representation of the short-term power spectrum of a sound. Figure 4 shows an example of a MFCC created by this project for the wav file shown in Figure 2. It can be observed that the MFCC is of shape 944 (time) * 13 (MFCC Coefficient) * 120 (power spectrum). Since MFCC feature uses logarithmic scale and gives importance to the significant frequencies of the speech data, it is considered to be more efficient feature representation compared to Spectrogram.



Shape of MFCC : (944, 13)

Figure 4: MFCC

Following are the steps used in calculating MFCC [10].

1. Frame the signal into short frames.
2. For each frame calculate the periodogram estimate of the power spectrum.
3. Apply the mel filterbank to the power spectra, sum the energy in each filter.
4. Take the logarithm of all filterbank energies.
5. Take the DCT of the log filterbank energies.
6. Keep DCT coefficients 2-13, discard the rest.

In this project, the `python_speech_feature` implementation of MFCC is used which is based on the above-mentioned algorithm. It uses 13 as the dimension for the MFCC representation as shown in Figure 4.

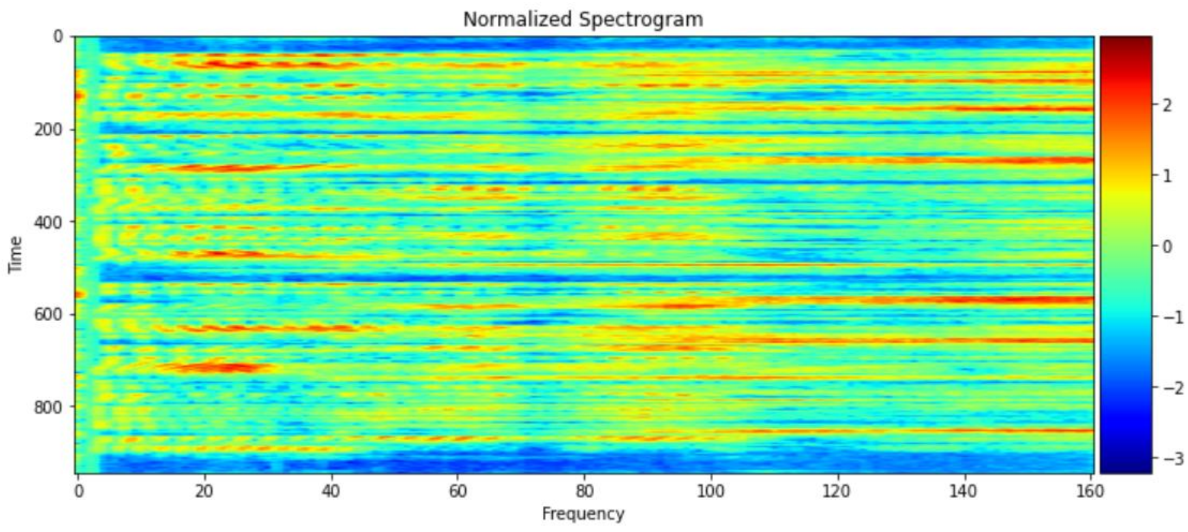
5.3. Normalization

Both Spectrogram and MFCC features are normalized to speed up the convergence. Normalization helps in reducing the impact of noise in the speech data. It is calculated with the following formula

Normalized Feature = Feature – Mean (Features) / std (Feature) + tolerance

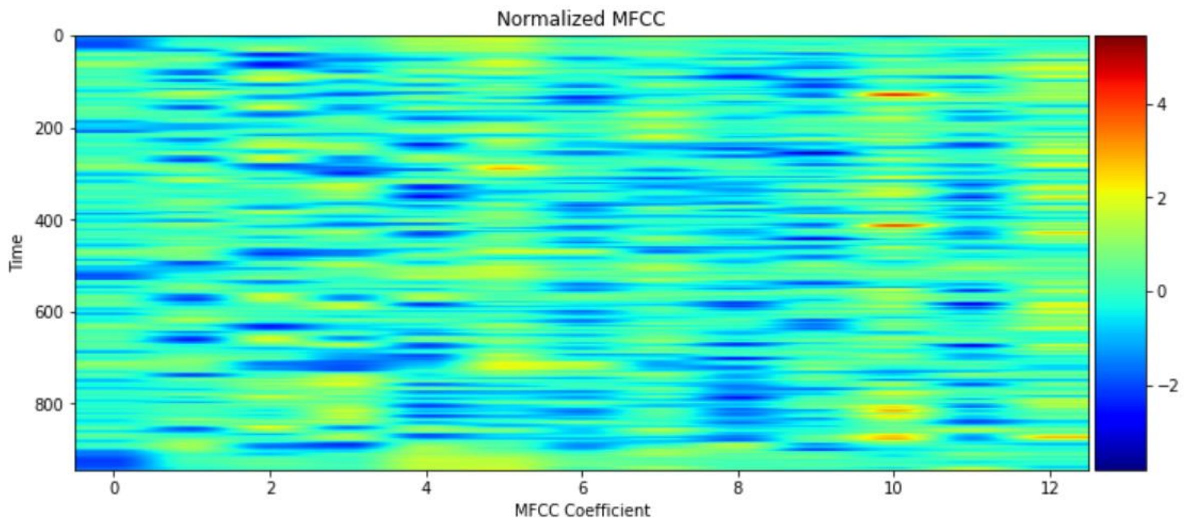
Here tolerance value is chosen as e^{-14}.

Figure 5 and Figure 6 shows the normalized spectrogram and MFCC corresponding to the ones shown in Figure 3 and Figure 4 respectively. It can be noticed that the color spectrum is normalized to have the mean of zero and more data points are showing color closer to green(zero).



Shape of Spectrogram : (944, 161)

Figure 5: Normalized Spectrogram



Shape of MFCC : (944, 13)

Figure 6: Normalized MFCC

6. Labels

Librispeech provides a transcribed text file for each audio file. This is used as the Label for learning and validation.

6.1. Label encoding

The labels are in character form and would need to be converted to mathematical representation for machine learning process to be effective. This project uses the simplest approach of character encoding by mapping an alphabet to a number. Figure 7 shows the character mapping table used in this project. The blank character is mapped to 28. The transcription is converted to a list of characters before passing as label to the neural networks. For example, a sentence such as “hello world” would be passed as [9,6,13,13,16,1,24,16,19,13,5]

'	SPACE	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27

Figure 7: Character mapping

7. Loss Calculation and Decoding

Speech recognition models use various loss calculation methods such as mean squared error, mean absolute error, cross entropy loss and connectionist temporal classification (CTC). Due to its better performing nature with speech recognition models, most of the state-of-the-art frameworks such as deepspeech and wav2letter are using CTC as the loss function.

7.1. CTC

Connectionist temporal classification solves two major problems in speech recognition.

1. Due to the different speed with which speakers produce audio, it can be difficult to pinpoint at which point of speech a character occurs.
2. It would also be difficult to determine if a character is occurring multiple times or if it is the slowness of the speech that is causing the character to appear twice. This causes difficulty in decoding.

CTC solves both these problems by introducing different permutations of the repeated character and introducing CTC-Blank character to indicate gap between separate characters or multiple occurrences of the same character. This method is proven to outperform both HMM (Hidden Markov Model) and HMM-RNN hybrid on real-world temporal classification problem [11]. CTC approach provides a much better accuracy with a trade-off for increased memory usage.

In this project, the `ctc_batch_cost` function from Keras is used for calculating the CTC loss. This takes the label, prediction and their corresponding lengths as a parameter and returns a tensor containing CTC loss for each element. For decoding, `ctc_decode` function from Keras is used which applies beam search on the input matrix to provide the most probable output. The output is then converted to text using the character mapping provided in Figure 7.

8. Framework

TensorFlow 2.1.0 is used for this project as the deep learning framework. TensorFlow is an end-to-end open source platform for machine learning. It has a comprehensive, flexible ecosystem of tools, libraries and community resources that lets researchers push the state-of-the-art in ML and developers easily build and deploy ML powered applications [12].

The base design had code templates created in TensorFlow 1.1 and Keras 2.0.3. This required the CUDA(Compute Unified Device Architecture) to be of an older version to work with GPU. GPU acceleration, even though not mandatory, but is very helpful in getting the training completed in reduced amount of time. The project upgraded the design to make the overall flow work with latest versions of technologies. Keras is being maintained within TensorFlow and the code is compatible with CUDA 10.1. This would help anyone to continue this work in contemporary versions of software. Other python modules such as `soundfile, seaborn` and `python_speech_features` are also upgraded to the latest version as of 2020 January as part of this project.

9. Activation Layers

Following are the different activation layers used in this project.

9.1. Softmax Activation

Softmax activation is considered to be the most optimal function of output layer of Neural Network. This is because the activation produces probabilities of different possible outcomes. With the combination of CTC beam search, softmax activation provides the highest probable output based on the last layer data.

$$f_i(\vec{a}) = \frac{e^{a_i}}{\sum_k e^{a_k}}$$

9.2. Tanh Activation

Tanh is the default activation function of RNNs in Keras. It is used in the initial models of the project with single RNN. Tanh function is nonlinear in nature which allows the stacking of layers. It is bound to the range (-1, 1).The gradient is stronger for tanh than sigmoid (derivatives are steeper).Like sigmoid, tanh also has a vanishing gradient problem [13].

$$f(x) = tanh(x) = \frac{2}{1+e^{-2x}} - 1$$

9.3. RELU Activation

In all the layers of most of the models in the project, rectified linear unit (RELU) is used as the activation function because of the proven advantages for speech recognition neural networks [14]. Due to the simplistic nature, RELU converges much faster than tanh or sigmoid function resulting in reduced training time.

$$RELU(x) = \begin{cases} 0 \ if \ x < 0 \\ x \ if \ x >= 0 \end{cases}$$

10. Optimizers

Optimizer used in this project for the learning process is Adam [15]. Initially SGD (Stochastic gradient descent) was tried, but Adam proved to be much efficient in loss convergence. With accelerating and decelerating momentum, Adam reaches the optimal loss in minimum number of epochs.

Keras implementation of Adam was used in this project with learning rate as 0.001. The beta_1 and beta_2 parameters were set as 0.9 and 0.999.

11. Models and Observations

Following are the models that were used for evaluation in this project. All of them were created using combination of different network layers offered by `tensorflow.keras`

11.1. Single layer RNN Models

Given their effectiveness in modeling sequential data, first set of models contain a single layer of RNNs. As shown in Figure 8, the RNN will take the time sequence of audio features as input. Input layer of audio features (Spectrogram or MFCC) was directly fed to the RNN layer and the output was directly used in a SoftMax activation.

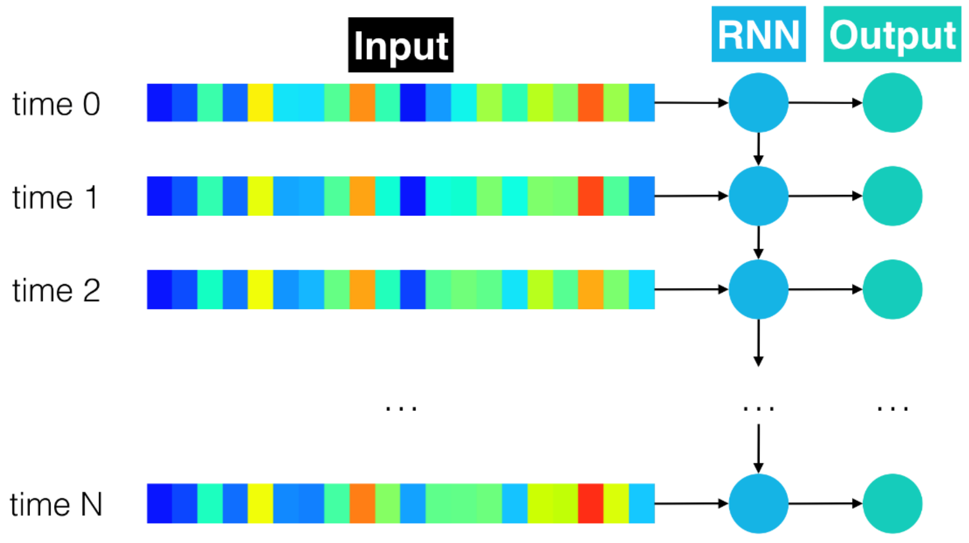


Figure 8: Single layer RNN [1]

At each time step, the speaker pronounces one of 28 possible characters, including each of the 26 letters in the English alphabet, along with a space character (" "), and an apostrophe (').

The output of the RNN at each time step is a vector of probabilities with 29 entries, where the ith entry encodes the probability that the ith character is spoken in the time sequence. (The extra 29th character is an empty "character" used to pad training examples within batches containing uneven lengths.).Figure 9 shows an equivalent, rolled depiction of the RNN that shows the output layer in greater detail.

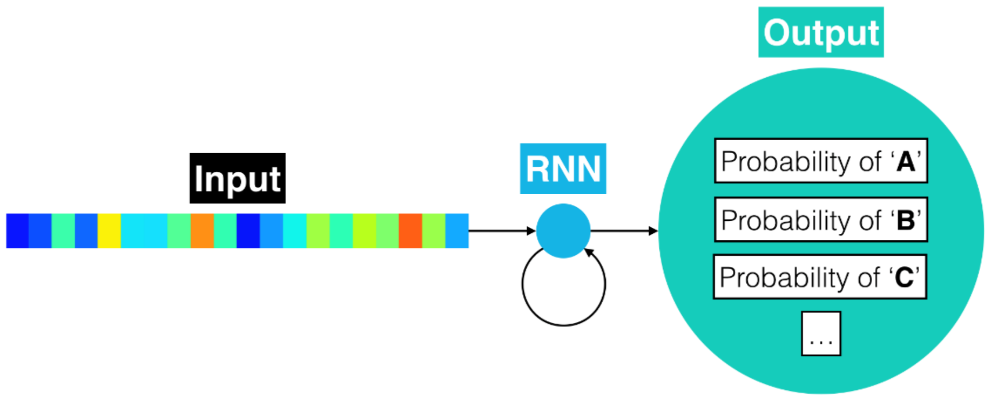


Figure 9: Single layer RNN rolled [1]

11.1.1. Simple RNN

The first single layer model tried is Simple RNN which is a fully connected RNN where the output is to be fed back to input.

Keras `SimpleRNN` layer is used for this model. Model summaries of MFCC and Spectrogram are as shown below.

```
Model: "simple_mfcc"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
the_input (InputLayer)        [(None, None, 13)]        0         
_________________________________________________________________
rnn (SimpleRNN)               (None, None, 29)          1247      
_________________________________________________________________
softmax (Activation)          (None, None, 29)          0         
=================================================================
Total params: 1,247
Trainable params: 1,247
Non-trainable params: 0
_________________________________________________________________
```

```
Model: "simple_spectrogram"
_________________________________________________________________
Layer (type)                    Output Shape              Param #   
=================================================================
the_input (InputLayer)          [(None, None, 161)]       0         
_________________________________________________________________
rnn (SimpleRNN)                 (None, None, 29)          5539      
_________________________________________________________________
softmax (Activation)            (None, None, 29)          0         
=================================================================
Total params: 5,539
Trainable params: 5,539
Non-trainable params: 0
_________________________________________________________________
```

This layer has the drawback of vanishing gradients when used in networks with more layers added. Vanishing gradients occurs as more layers using certain activation functions are added to neural networks, the gradients of the loss function approaches zero, making the network hard to train. Vanishing gradient problem is for higher impact in sequence to sequence models. Long short-term memory (LSTM) and Gated Recurrent Unit (GRU) networks solve this problem by adding a forget logic to remove lingering influences on weights.

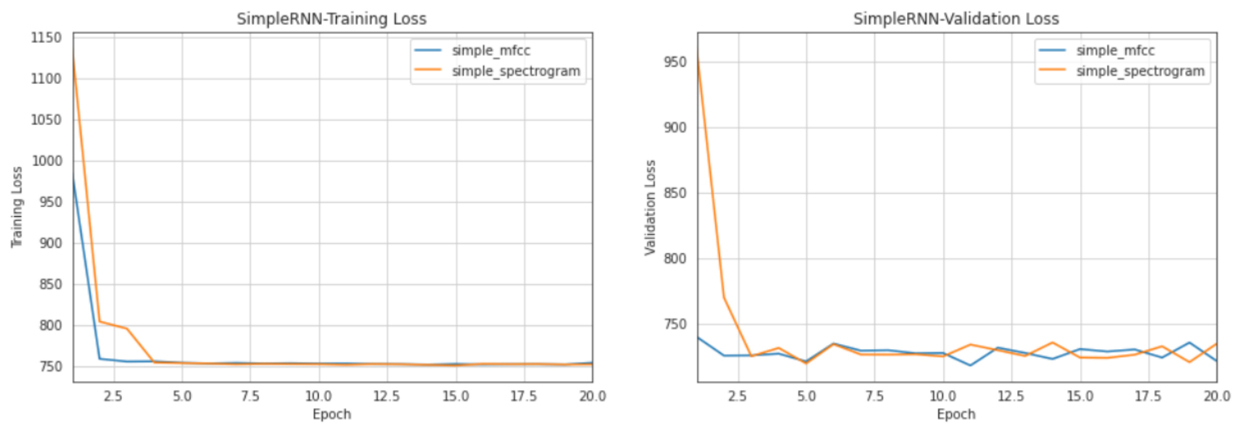


Figure 10: Training and Validation Loss for Simple RNN model

Training and validation loss for the model during 20 epochs is as shown in Figure 10 . Training loss for both MFCC and Spectrogram models plateaus around 750 and validation loss around 730.

11.1.2. LSTM

Second single layer model tried in this project is LSTM. As shown in Figure 11 , LSTM introduces a cell state value which helps in keeps track of weights that needs to be forgotten at later layers. That solves the vanishing gradient problem.

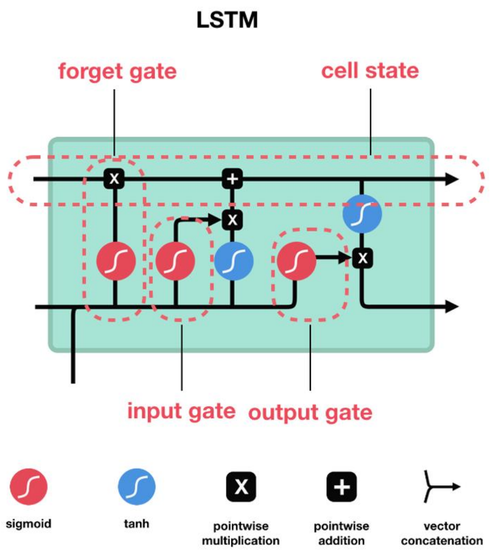


Figure 11: LSTM [16]

Keras `LSTM` layer is used for this model. Model summaries of MFCC and Spectrogram are as shown below.

```
Model: "lstm_mfcc"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
the_input (InputLayer)       [(None, None, 13)]        0
_________________________________________________________________
rnn (LSTM)                   (None, None, 29)          4988
_________________________________________________________________
softmax (Activation)         (None, None, 29)          0
=================================================================
Total params: 4,988
Trainable params: 4,988
Non-trainable params: 0
_________________________________________________________________

Model: "lstm_spectrogram"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
the_input (InputLayer)       [(None, None, 161)]       0
_________________________________________________________________
rnn (LSTM)                   (None, None, 29)          22156
_________________________________________________________________
softmax (Activation)         (None, None, 29)          0
=================================================================
Total params: 22,156
Trainable params: 22,156
```

```
Non-trainable params: 0
_________________________________________________________________
```

Training and validation loss for the model during 20 epochs is as shown in Figure 12. Training loss levels around 750 and validation loss around 730 for spectrogram model. For MFCC model, training loss is around 780 and validation loss is around 750.

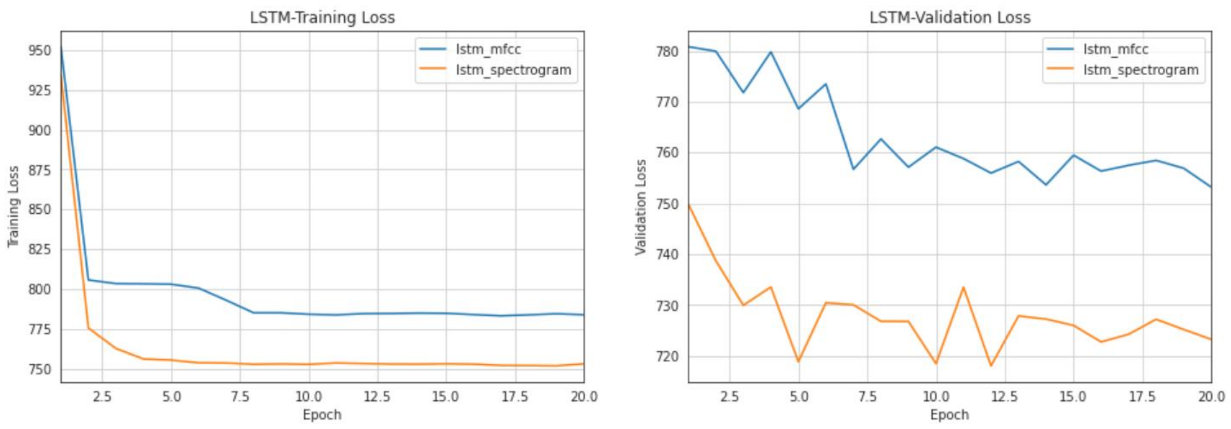


Figure 12: Training and Validation loss for LSTM model

11.1.3. GRU

Third single layer model tried in this project is GRU. As shown in Figure 13 , GRU is the newer generation of recurrent neural networks and is pretty similar to an LSTM. GRU is much more simplified and uses a hidden state to transfer information. It only has two gates, a reset gate and update gate. This makes it perform much faster compared to LSTM at the same time solves vanishing gradient problem. In various studies and the references therein, it has been noted that GRU RNN is comparable to, or even outperforms, the LSTM in most cases [17]. Due to this case, all further models are using GRU as the RNN layers.

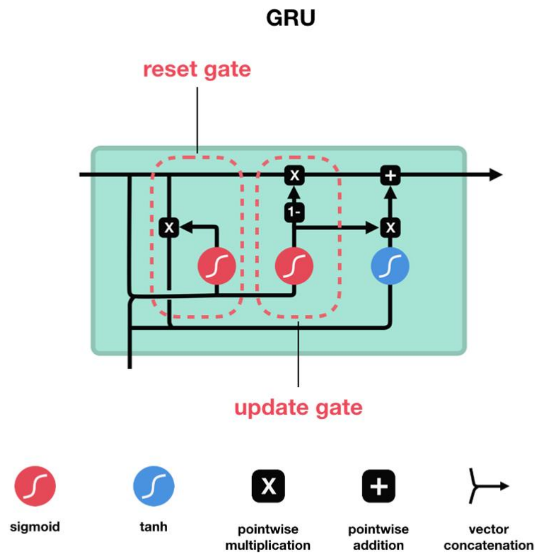


Figure 13: GRU [16]

Keras `GRU` layer is used for this model. Model summaries of MFCC and Spectrogram are as shown below.

```
Model: "gru_mfcc"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
the_input (InputLayer)       [(None, None, 13)]        0
_________________________________________________________________
rnn (GRU)                    (None, None, 29)          3828
_________________________________________________________________
softmax (Activation)         (None, None, 29)          0
=================================================================
Total params: 3,828
Trainable params: 3,828
Non-trainable params: 0
_________________________________________________________________
```

```
Model: "gru_spectrogram"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
the_input (InputLayer)       [(None, None, 161)]       0
_________________________________________________________________
rnn (GRU)                    (None, None, 29)          16704
_________________________________________________________________
softmax (Activation)         (None, None, 29)          0
=================================================================
Total params: 16,704
Trainable params: 16,704
Non-trainable params: 0
_________________________________________________________________
```

Training and validation loss for the models during 20 epochs is as shown in Figure 14. Training loss plateaus for both MFCC and Spectrogram models around 750 and validation loss plateaus around 730.

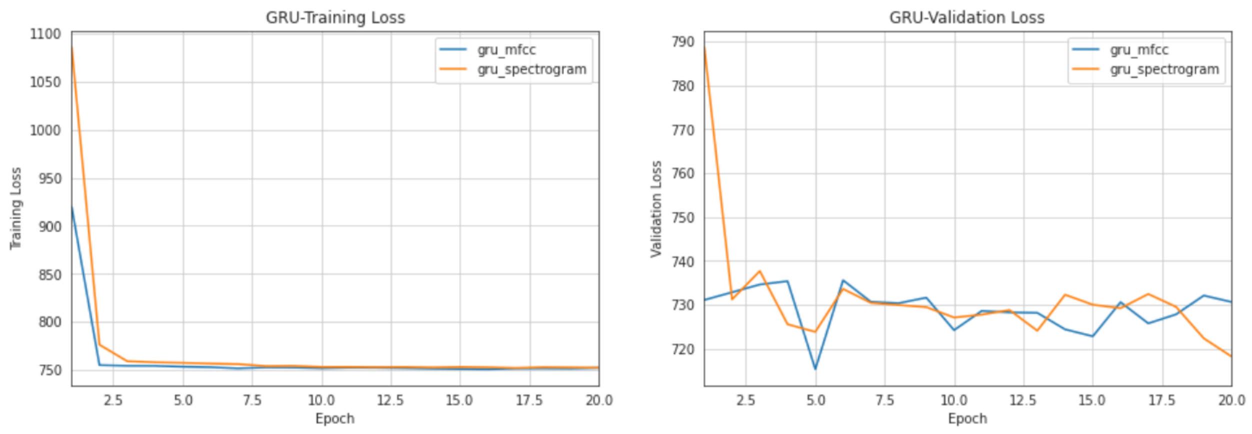


Figure 14: Training and Validation Loss for GRU Model

11.2. RNN + TimeDistributed Dense Model

TimeDistributed Dense wrapper and the BatchNormalization layer are applied to the GRU RNN layer in the next model. Batch normalization is added to the recurrent layer to reduce training times. The TimeDistributed layer is used to find more complex patterns in the dataset. It applies a layer to every temporal slice of the output of RNN. The unrolled snapshot of the architecture is shown in Figure 15.

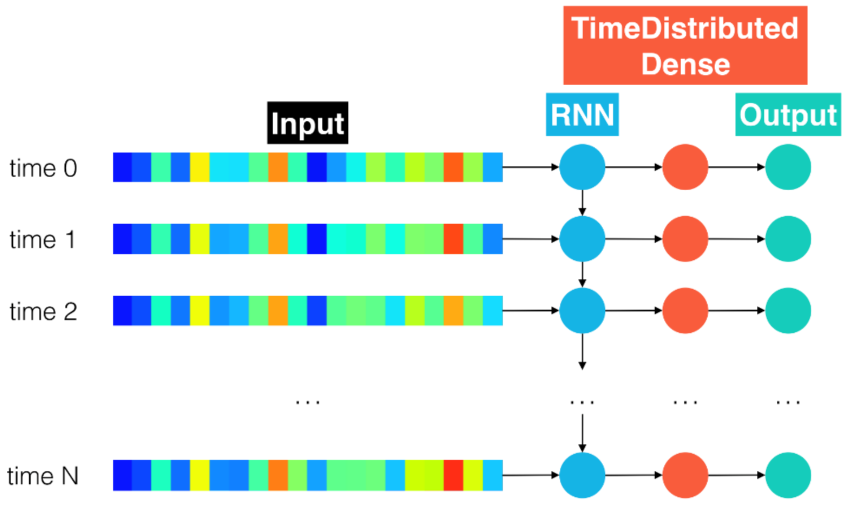


Figure 15: RNN + TimeDistributed Dense Model [1]

Figure 16 shows an equivalent, rolled depiction of the RNN that shows the (TimeDistributed) dense and output layers in greater detail.

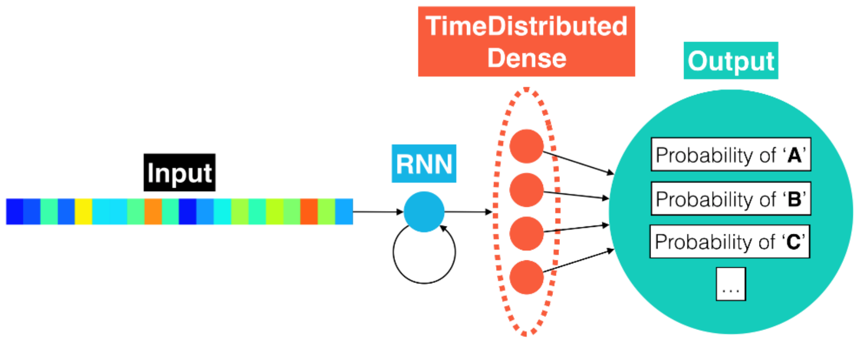


Figure 16: RNN +TimeDistributed Dense Model Rolled [1]

As part of the project, 200 output nodes are created from a GRU RNN with RELU activation applied on top of the input layer. After the GRU RNN layer a BatchNormalization layer is applied. After that TimeDistributed (Dense) layer is added before the output layer. Model summaries of MFCC and Spectrogram are as shown below.

```
Model: "rnn_mfcc"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
the_input (InputLayer)       [(None, None, 13)]        0         
_________________________________________________________________
rnn (GRU)                    (None, None, 200)         129000    
_________________________________________________________________
bn_rnn_1d (BatchNormalizatio (None, None, 200)         800       
_________________________________________________________________
time_distributed_2 (TimeDist (None, None, 29)          5829      
_________________________________________________________________
softmax (Activation)         (None, None, 29)          0         
=================================================================
Total params: 135,629
Trainable params: 135,229
Non-trainable params: 400
_________________________________________________________________
```

```
Model: "rnn_spectrogram"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
the_input (InputLayer)       [(None, None, 161)]       0
_________________________________________________________________
rnn (GRU)                    (None, None, 200)         217800
_________________________________________________________________
bn_rnn_1d (BatchNormalizatio (None, None, 200)         800
_________________________________________________________________
time_distributed_3 (TimeDist (None, None, 29)          5829
_________________________________________________________________
softmax (Activation)         (None, None, 29)          0
=================================================================
Total params: 224,429
Trainable params: 224,029
Non-trainable params: 400
_________________________________________________________________
```

Training and validation loss for the model during 20 epochs is as shown in Figure 17. Training loss and validation loss levels for both MFCC and Spectrogram model around 130 and 140 respectively. MFCC models showed slightly better performance than Spectrogram models.

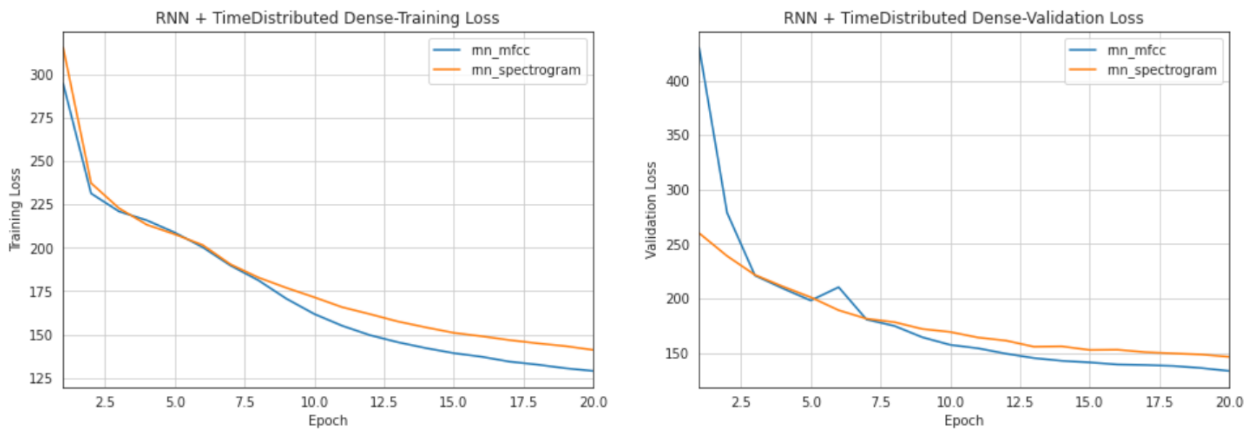


Figure 17: Training and Validation loss for RNN + TimeDistributed Dense model

11.3. CNN + RNN + TimeDistributed Dense Model

The next architecture adds an additional level of complexity, by introducing a 1D convolution layer to the input layer. CNN layer provides an additional depth to earlier model which allows more information to be processed. The architecture is shown in Figure 18

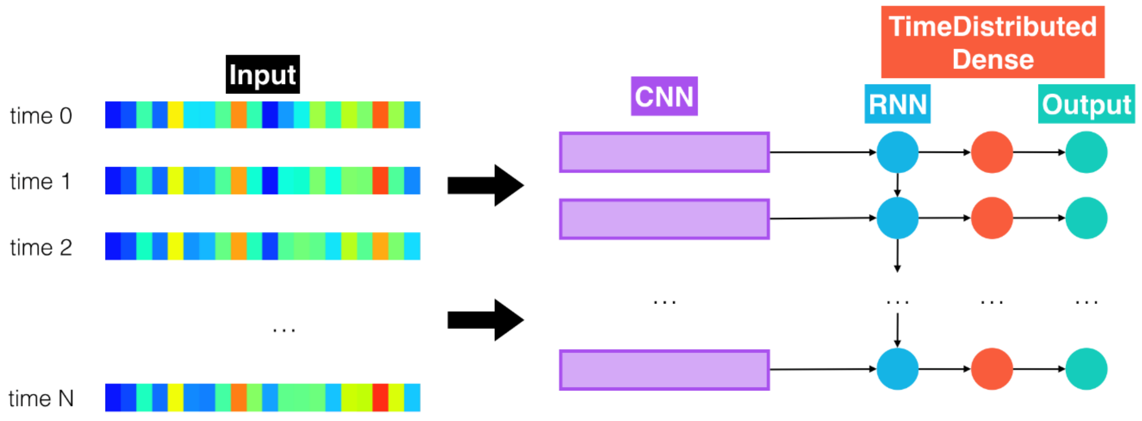


Figure 18: CNN + RNN + TimeDistributed Dense Model [1]

In this project, CNN Layer was applied with kernel size as 11, number of filters as 200, padding as "valid" and activation as "RELU". This is followed by the layers used in RNN + TimeDistributed Dense model. Model summaries of MFCC and Spectrogram are as shown below.

```
Model: "cnn_rnn_mfcc"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
the_input (InputLayer)       [(None, None, 13)]        0
_________________________________________________________________
conv1d (Conv1D)              (None, None, 200)         28800
_________________________________________________________________
bn_conv_1d (BatchNormalizati (None, None, 200)         800
_________________________________________________________________
rnn (GRU)                    (None, None, 200)         241200
_________________________________________________________________
gru_rnn (BatchNormalization) (None, None, 200)         800
_________________________________________________________________
time_distributed_4 (TimeDist (None, None, 29)          5829
_________________________________________________________________
softmax (Activation)         (None, None, 29)          0
=================================================================
Total params: 277,429
Trainable params: 276,629
Non-trainable params: 800
_________________________________________________________________

Model: "cnn_rnn_spectrogram"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
the_input (InputLayer)       [(None, None, 161)]       0
_________________________________________________________________
conv1d (Conv1D)              (None, None, 200)         354400
_________________________________________________________________
bn_conv_1d (BatchNormalizati (None, None, 200)         800
_________________________________________________________________
rnn (GRU)                    (None, None, 200)         241200
_________________________________________________________________
gru_rnn (BatchNormalization) (None, None, 200)         800
_________________________________________________________________
time_distributed_5 (TimeDist (None, None, 29)          5829
_________________________________________________________________
softmax (Activation)         (None, None, 29)          0
```

```
=================================================================
Total params: 603,029
Trainable params: 602,229
Non-trainable params: 800
_________________________________________________________________
```

Training and validation loss for the model during 20 epochs is as shown in Figure 19 . Training loss levels for both MFCC and Spectrogram model around 100 and validation loss around 130.

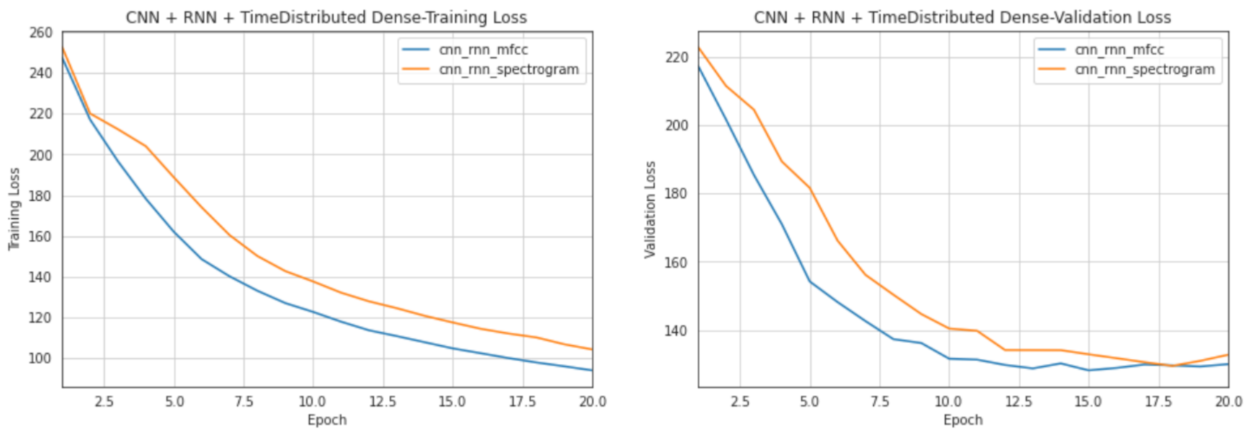


Figure 19: Training and Validation loss for CNN + RNN + TimeDistributed Dense model

11.4. Deep RNN + TimeDistributed Dense Model

The next architecture is a multi-level GRU model. This allows a number of GRU to be added back to back which can process long sequences and interdependencies. Figure 20 shows the architecture of the model with number of layers as 2.

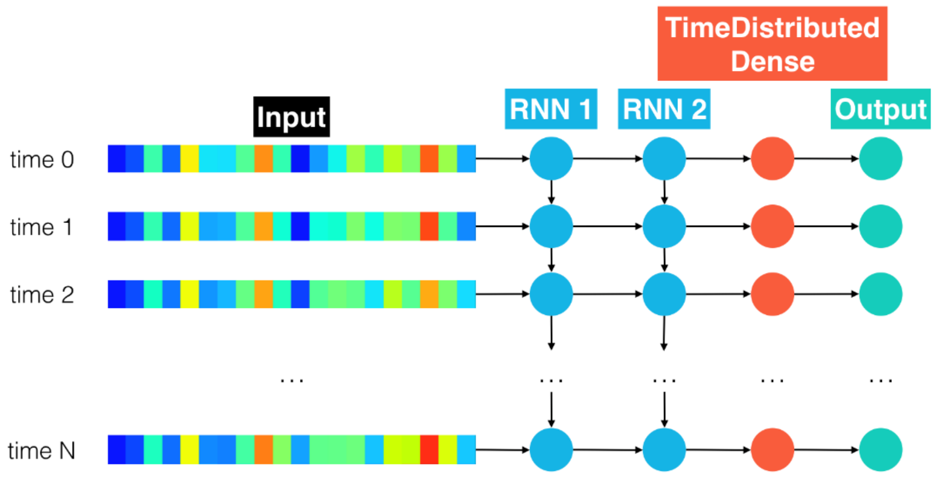


Figure 20: Deep RNN + TimeDistributed Dense Model [1]

In this project, deep RNN model is implemented with depth as 2. Same GRU layer is applied in both levels. Model summaries of MFCC and Spectrogram are as shown below.

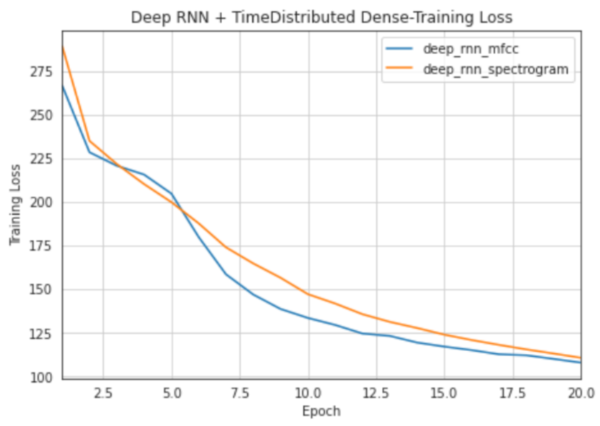
Model: "deep_rnn_mfcc"

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
the_input (InputLayer)       [(None, None, 13)]        0         
_________________________________________________________________
gru (GRU)                    (None, None, 200)         129000    
_________________________________________________________________
bt_rnn_1 (BatchNormalization (None, None, 200)         800       
_________________________________________________________________
gru_1 (GRU)                  (None, None, 200)         241200    
_________________________________________________________________
bt_rnn_last_rnn (BatchNormal (None, None, 200)         800       
_________________________________________________________________
time_distributed_6 (TimeDist (None, None, 29)          5829      
_________________________________________________________________
softmax (Activation)         (None, None, 29)          0         
=================================================================
Total params: 377,629
Trainable params: 376,829
Non-trainable params: 800
_________________________________________________________________
```

Model: "deep_rnn_spectrogram"

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
the_input (InputLayer)       [(None, None, 161)]       0         
_________________________________________________________________
gru_2 (GRU)                  (None, None, 200)         217800    
_________________________________________________________________
bt_rnn_1 (BatchNormalization (None, None, 200)         800       
_________________________________________________________________
gru_3 (GRU)                  (None, None, 200)         241200    
_________________________________________________________________
bt_rnn_last_rnn (BatchNormal (None, None, 200)         800       
_________________________________________________________________
time_distributed_7 (TimeDist (None, None, 29)          5829      
_________________________________________________________________
softmax (Activation)         (None, None, 29)          0         
=================================================================
Total params: 466,429
Trainable params: 465,629
Non-trainable params: 800
_________________________________________________________________
```

Training and validation loss for the model during 20 epochs is as shown in Figure 21. Training loss for both MFCC and Spectrogram model plateaus around 110. Validation loss plateaus around 125 for MFCC and 135 for Spectrogram.



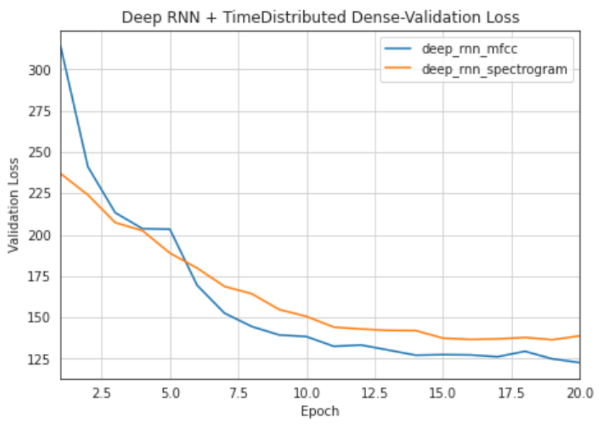


Figure 21: Training and Validation loss for Deep RNN + TimeDistributed Dense model

11.5. Bidirectional RNN + TimeDistributed Dense Model

One shortcoming of conventional RNNs is that they are only able to make use of previous context. In speech recognition, where whole utterances are transcribed at once, there is no reason not to exploit future context as well. Bidirectional RNNs (BRNNs) do this by processing the data in both directions with two separate hidden layers which are then fed forward to the same output layer. Figure 22 shows the architecture of the model with bidirectional RNNs.

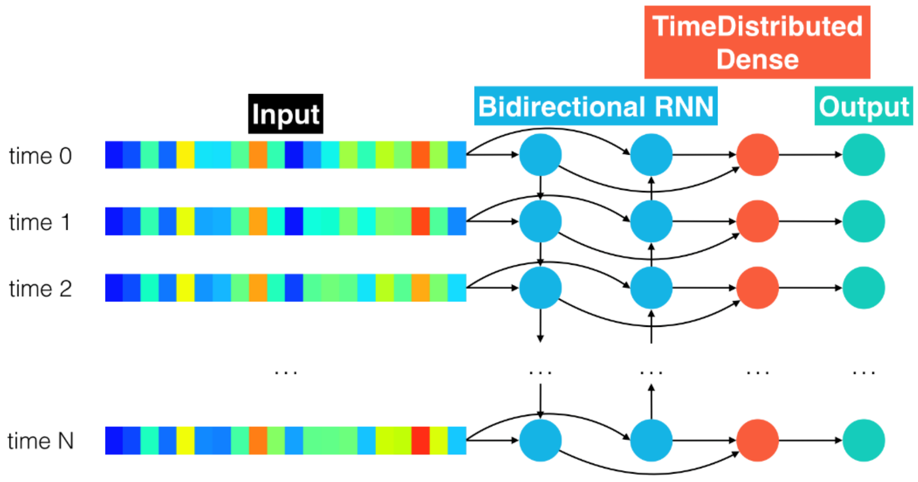


Figure 22: Bidirectional RNN + TimeDistributed Dense Model [1]

In this project, Bidirectional layer from Keras is applied on GRU with "`concat`" merge mode. Model summaries of MFCC and Spectrogram are as shown below.

```
Model: "bidirectional_rnn_mfcc"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
the_input (InputLayer)       [(None, None, 13)]        0
_________________________________________________________________
bidirectional (Bidirectional (None, None, 400)         258000
_________________________________________________________________
time_distributed_8 (TimeDist (None, None, 29)          11629
_________________________________________________________________
```

```
softmax (Activation)         (None, None, 29)          0
=================================================================
Total params: 269,629
Trainable params: 269,629
Non-trainable params: 0
_________________________________________________________________

Model: "bidirectional_rnn_spectrogram"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
the_input (InputLayer)       [(None, None, 161)]       0
_________________________________________________________________
bidirectional_1 (Bidirection (None, None, 400)         435600
_________________________________________________________________
time_distributed_9 (TimeDist (None, None, 29)          11629
_________________________________________________________________
softmax (Activation)         (None, None, 29)          0
=================================================================
Total params: 447,229
Trainable params: 447,229
Non-trainable params: 0
_________________________________________________________________
```

Using the Adam optimizer with the values used in the other models was causing a problem of exploding gradients in the Bidirectional RNN model. To resolve this, the optimizer was used with learning rate as 0.0005, beta_1 as 0.9 and beta_2 as 0.9999. This reduced the rate of convergence but prevented the model from overshooting loss value.

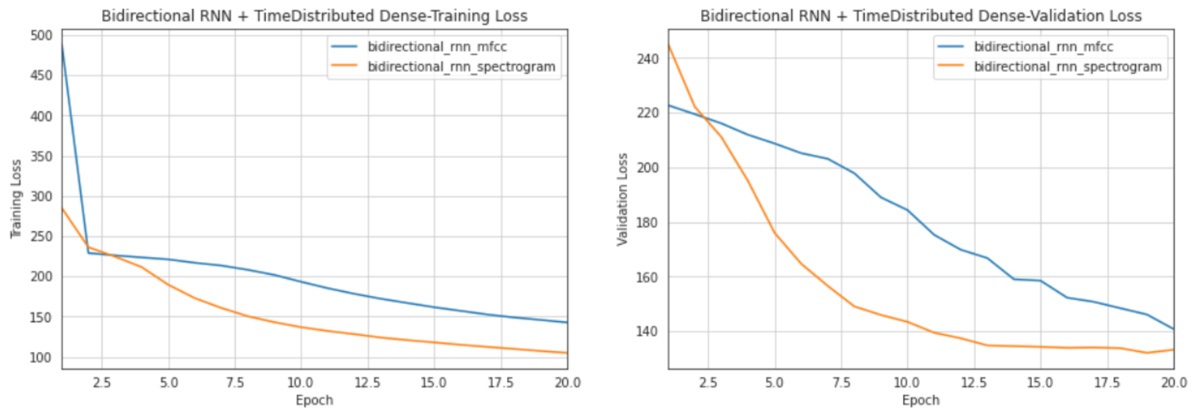


Figure 23: Training and Validation loss for Bidirectional RNN + TimeDistributed Dense model

Training and validation loss for the model during 20 epochs is as shown in Figure 23Figure 19. The training loss plateaus for MFCC around 145 and Spectrogram around 105. The validation loss reaches 140 for MFCC and plateaus around 130 for Spectrogram. This is the only model where Spectrogram outperformed MFCC in terms of loss.

11.6.Final Model

Figure 24 and Figure 25 shows the comparison of models trained with MFCC and Spectrogram respectively. It can be observed that, for MFCC models, both `cnn_rnn` and `deep_rnn` showed lowest loss values. The models `cnn_rnn, deep_rnn` and `bidirectional_rnn` showed the lowest loss values among the Spectrogram models. A final model is created with the combination of `cnn_rnn_model` and `deep_rnn_model` since these layers showed consistently better accuracy.

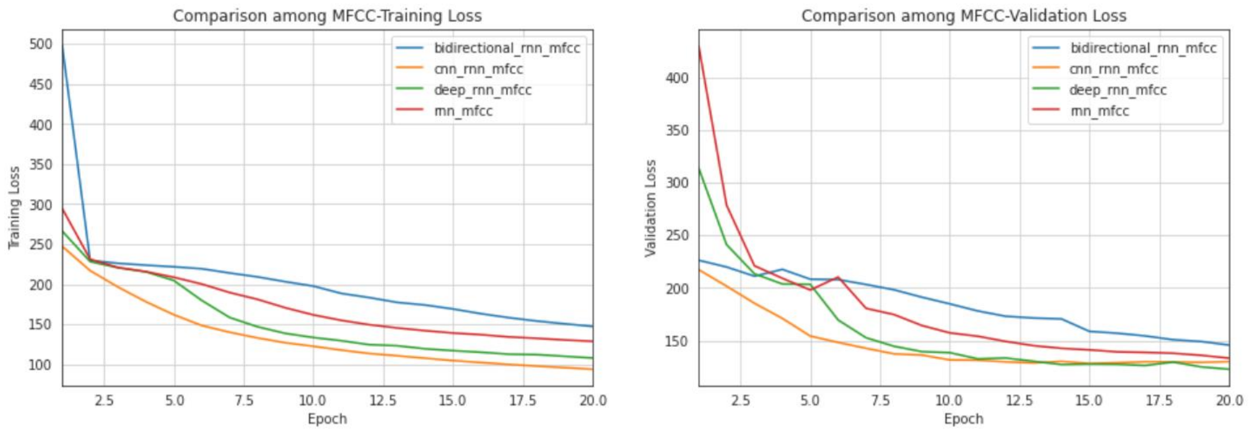


Figure 24: Comparison of MFCC models

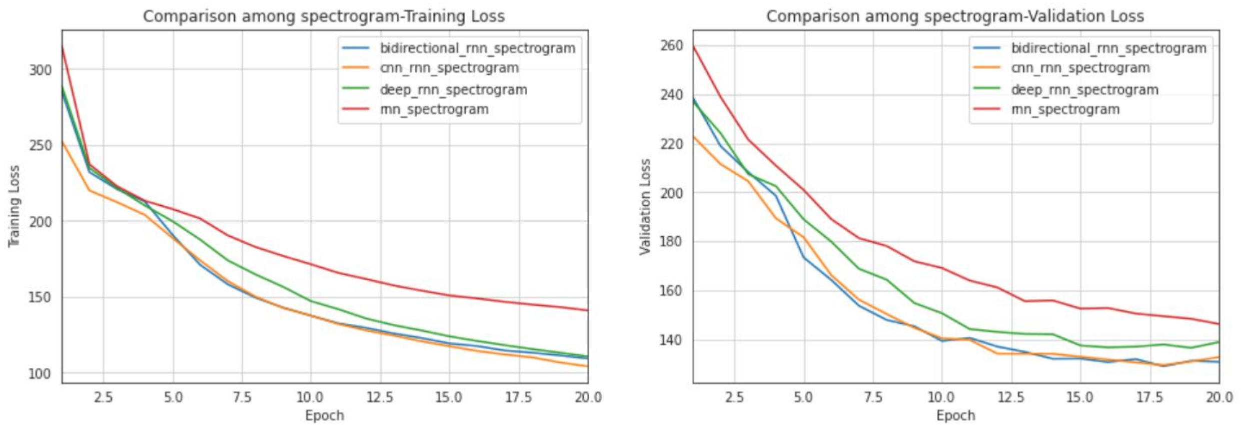


Figure 25: Comparison of Spectrogram models

11.6.1. Model Architecture

Several hyperparameter tuning is performed to find the best combination to reduce the CTC loss. Figure 26 shows the architecture of the final model.

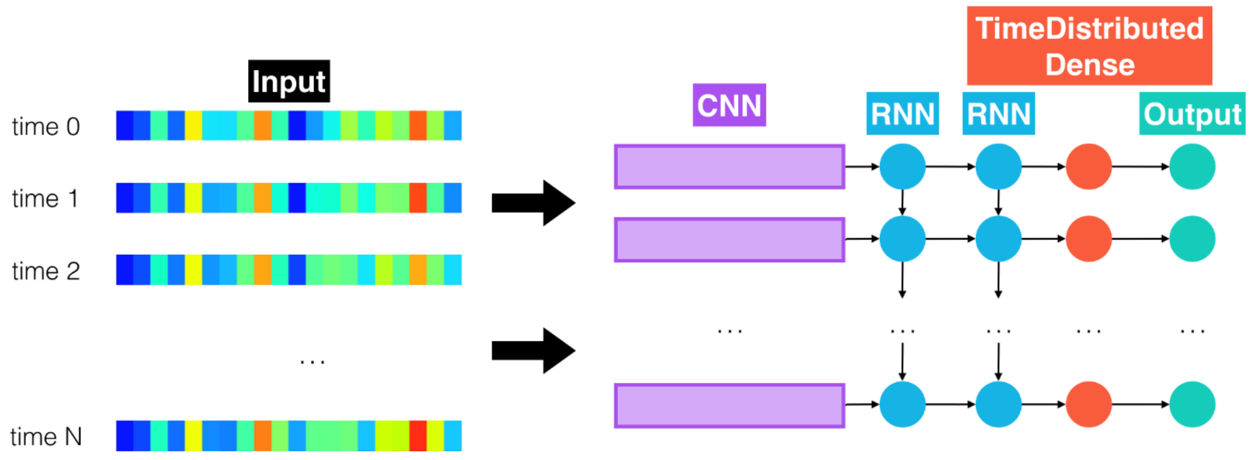


Figure 26: Final Model

Model summaries of MFCC and Spectrogram are as shown below. CNN layer is created with 200 nodes with kernel size as 11.

```
Model: "final_model_mfcc"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
the_input (InputLayer)       [(None, None, 13)]        0
_________________________________________________________________
layer_1_conv (Conv1D)        (None, None, 200)         28800
_________________________________________________________________
conv_batch_norm (BatchNormal (None, None, 200)         800
_________________________________________________________________
rnn_1 (GRU)                  (None, None, 250)         339000
_________________________________________________________________
bt_rnn_1 (BatchNormalization (None, None, 250)         1000
_________________________________________________________________
final_layer_of_rnn (GRU)     (None, None, 250)         376500
_________________________________________________________________
bt_rnn_final (BatchNormaliza (None, None, 250)         1000
_________________________________________________________________
time_distributed_10 (TimeDis (None, None, 29)          7279
_________________________________________________________________
softmax (Activation)         (None, None, 29)          0
=================================================================
Total params: 754,379
Trainable params: 752,979
Non-trainable params: 1,400
_________________________________________________________________

Model: "final_model_spectrogram"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
the_input (InputLayer)       [(None, None, 161)]       0
_________________________________________________________________
layer_1_conv (Conv1D)        (None, None, 200)         354400
_________________________________________________________________
conv_batch_norm (BatchNormal (None, None, 200)         800
_________________________________________________________________
rnn_1 (GRU)                  (None, None, 250)         339000    
_________________________________________________________________
bt_rnn_1 (BatchNormalization (None, None, 250)         1000      
_________________________________________________________________
final_layer_of_rnn (GRU)     (None, None, 250)         376500    
_________________________________________________________________
bt_rnn_final (BatchNormaliza (None, None, 250)         1000      
_________________________________________________________________
time_distributed_11 (TimeDis (None, None, 29)          7279      
_________________________________________________________________
softmax (Activation)         (None, None, 29)          0         
=================================================================
Total params: 1,079,979
Trainable params: 1,078,579
Non-trainable params: 1,400
_________________________________________________________________
```

The model is named as `final_light_model`. Training and validation loss for the model during 20 epochs is as shown in Figure 27. For Spectrogram, training loss reaches 130 and validation loss around 125. For MFCC, training loss reaches 90 and validation loss around 105.

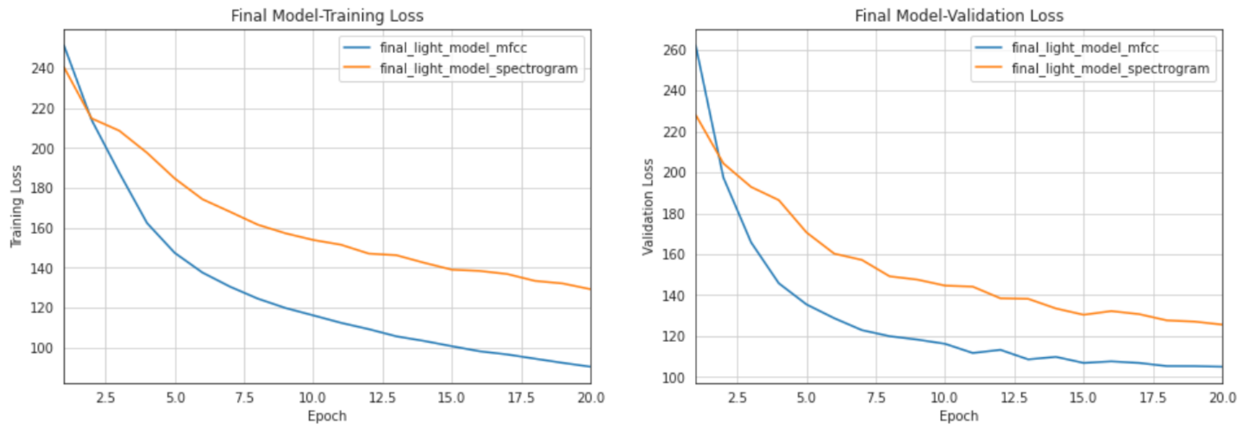
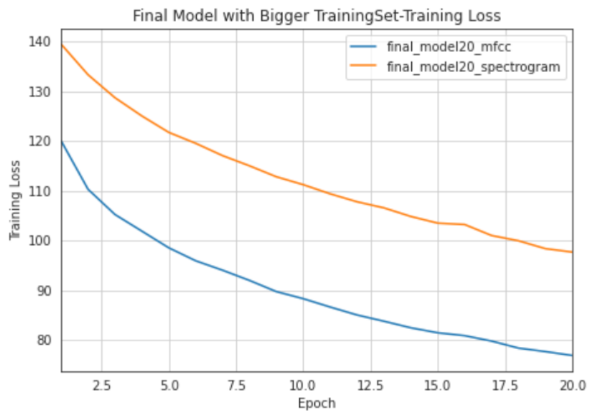


Figure 27: Training and Validation loss for final model

11.6.2. Impact of training size

Final model showed much better validation loss than all previous models. The model was trained again on a bigger dataset (train-360-clean) with 20 epochs. Training and validation loss for the model is shown in Figure 28. MFCC model performed better both on training and validation. Spectrogram model reached training loss of 98 and validation loss of 77 in 20 epochs, where MFCC model reached training loss of 77 and validation loss of 60.



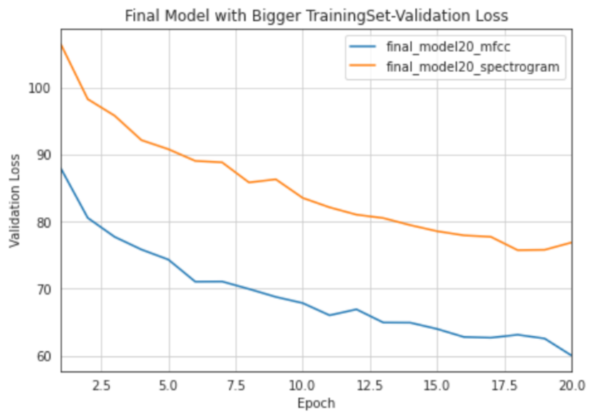
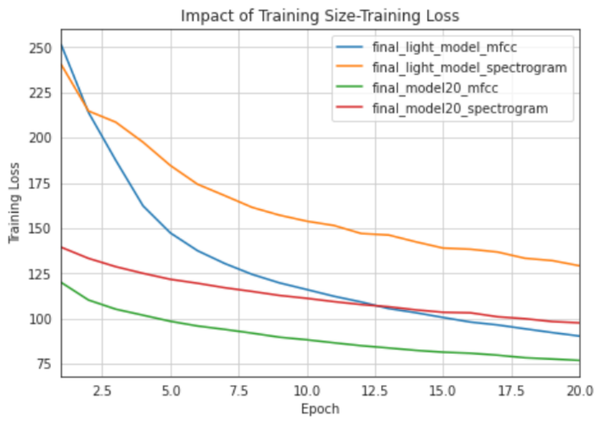


Figure 28: Training and Validation loss for final model with bigger training sets

When the training size increase from 935MB (in `final_light_model`) to 61 GB(in `final_model20`), the training loss and validation loss get lower values as shown in Figure 29. This indicates that training size has a very high impact for accuracy of the models.



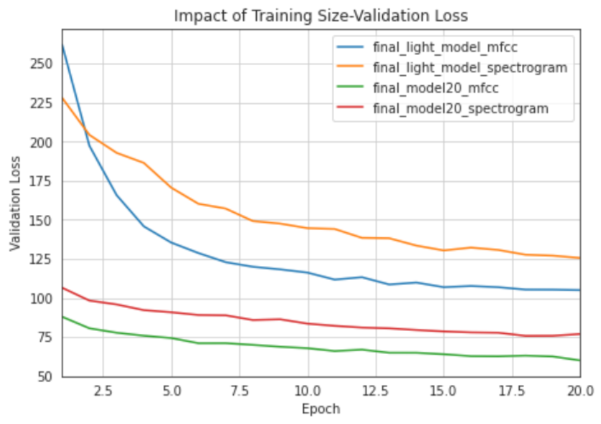


Figure 29: Impact of training size

11.6.3. Impact of increased epochs

The model was trained again on a bigger dataset (`train-360-clean`) with 30 epochs to assess if the epochs have reached optimal number. Training and validation loss for the model is shown in Figure 30. Spectrogram model reached training loss of 95 and validation loss of 70 in 30 epochs, where MFCC model reached training loss of 80 and validation loss of 63. Since this has the best accuracy, the model was named as `final_model`.

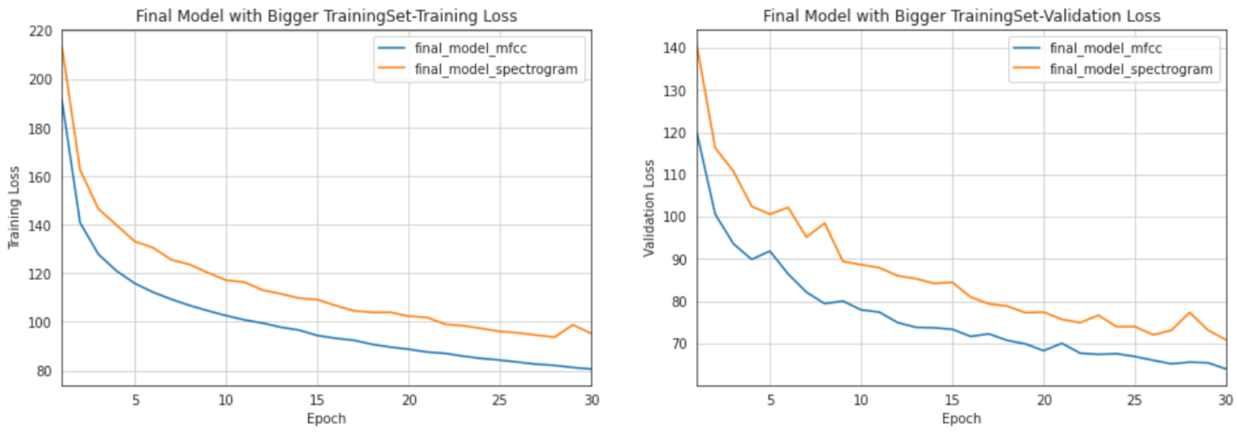


Figure 30: Training and Validation loss for final model with bigger training sets and epochs

There are slight improvements on the model accuracy with the increased epochs and indicates that the optimal number of epochs are further ahead. This can be observed in Figure 31.

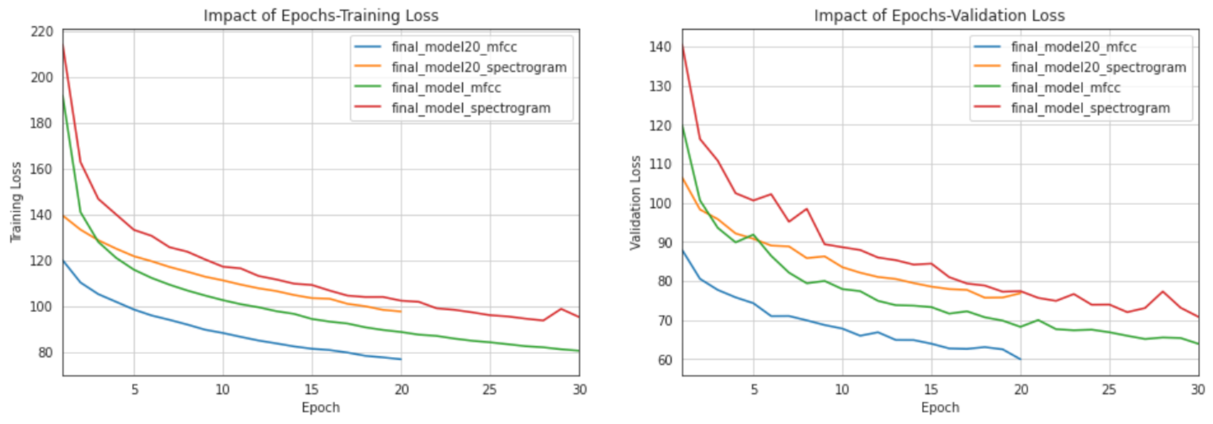


Figure 31: Impact of increased Epochs

12. Implementation Details

This section contains the details about the implementation of the project.

12.1. Overall Architecture

The overall process of the implementation is mentioned in the Figure 32 .The training and validation data sets (`dev-clean, test-clean` and `train-360-clean`) was processed using the preprocessing method mentioned in section 4.The corpus files are used as a dictionary for datasets in all modules.

In data generation module (`data_gen.py`), preprocessed files go through the feature extraction process as mentioned in section 5. These features (MFCC and Spectrogram) are saved as a data generator object. The transcription text goes through the label encoding process mentioned in section 6.1 which is also made part of the data object. A batch generator process is added to this module to support training with mini batches of data.

Data visualization (`data_viz.py`) module uses the data objects created by data generation module to visualize raw audio, Spectrogram and MFCC using matplotlib. An interactive notebook (Jupyter) is used to run the visualizations.

For further processes, the notebook also configures TensorFlow to use GPU (for parallel processing) or CPU.

A model generation module (`model_gen.py`) is used to create different models as mentioned in section 11 using TensorFlow Keras layers. Each model is returned as a separate function which is then used in the notebook to generate different Keras model objects. Activation layers mentioned in section 9 are configured separately for each function.

The generated models are trained using the model trainer (`train_util.py`). CTC loss function as mentioned in section 7.1 is created using `ctc_batch_cost` function from Keras. The `fit` function of TensorFlow is used to train the models with the minibatches generated by data generator module. Adam optimizer as mentioned in 10 are applied during the model fitting. The trained models are saved as `.h5` (HDF5 - Hierarchical Data Format) file in the result folder. Also, the training stats are stored as `.pickle` files which are then used for visualizations. Jupyter notebook is used to initiate the training with verbose output.

Model visualization (`model_viz.py`) is used for visualizing the training stats of each module. This is used for comparison of training and validation loss of different models and is displayed in the notebook interactively. Training loss and Validation loss visualizations in section 11 are generated using this module.

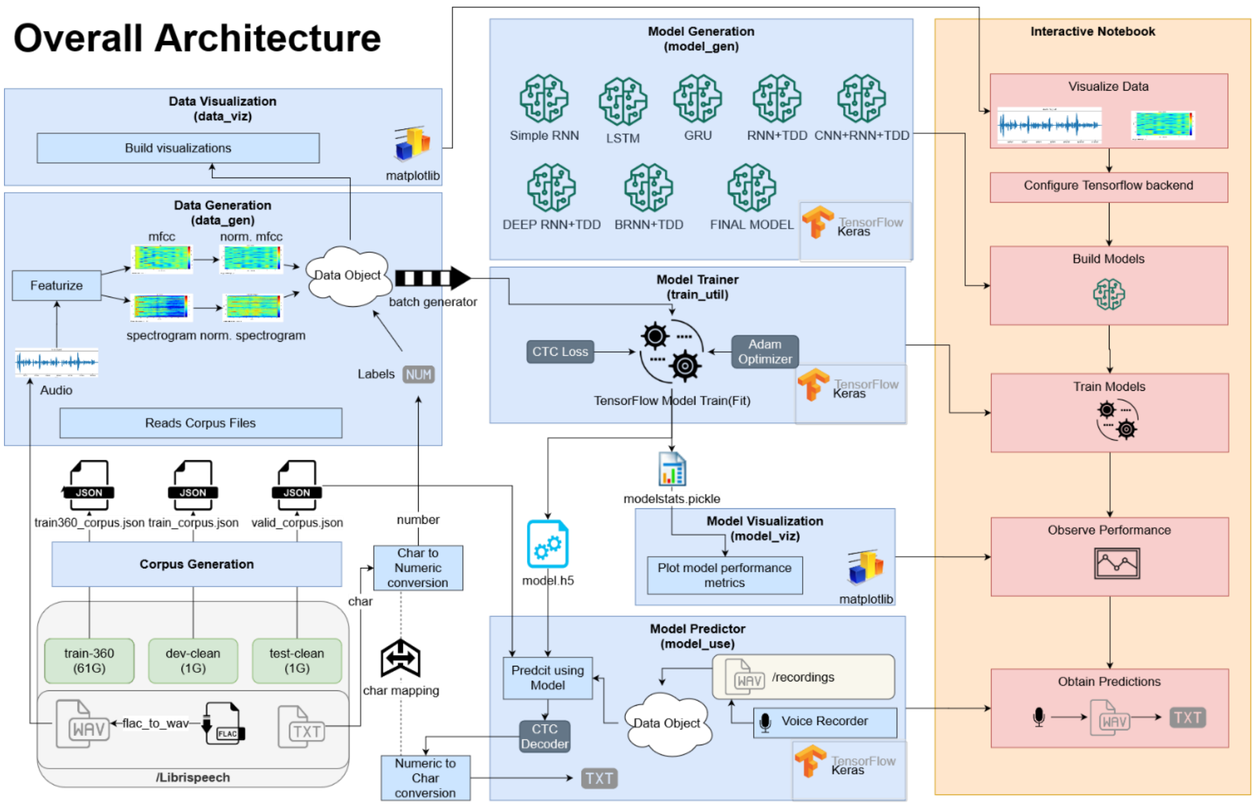


Figure 32 : Overall architecture

The saved models are used to predict the audio files from training set or testing set in the model predictor (`model_use.py`) module. This module uses CTC decoder to predict the most probable outputs followed by decoding characters using the same character encoding mentioned in 6.1 .

Existing training and testing sets can be used to predict the transcriptions using the saved models. Following is an example of prediction by the final model.

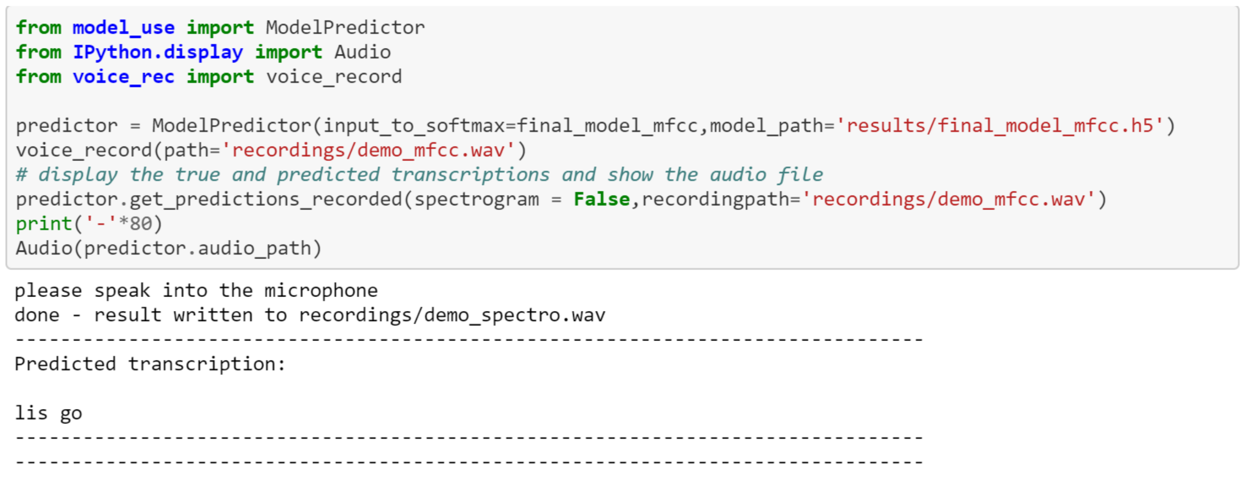
```
True transcription:

and now it had come to pass that his sole remaining ally mister samuel bozzle the ex policeman
was becoming weary of his service

--------------------------------------------------------------------------------

Predicted transcription:

ands now had com to pas atd his so remaning oli mister san no bossl the axpilisemen was becomeing
wear yef his servace

--------------------------------------------------------------------------------
```

12.2.Live Predictions

Using the saved models, a processing engine was also implemented, which processes audio for a real-time voice recording.

A voice recording module is created using `pyaudio` module from python. User is prompted for microphone input. Once the microphone frequency is detected above a threshold, the recording is detected as started. Silences are detected throughout the recording and is the continuous silence is above a specified threshold, the recording is automatically ended. This allows a normal speech of varying lengths to be recorded.

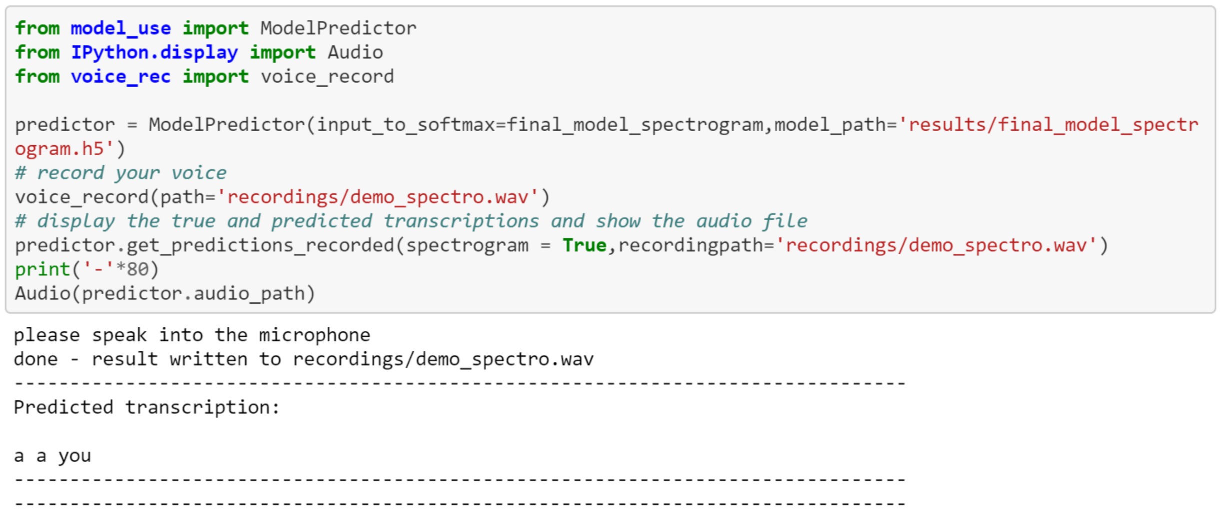
The voice recorder is used to create .wav files in the recordings folder. These recording are converted to features using the data generation module. This is then predicted using the saved model and decoded texts are printed back to the notebook interface. Figure 33 and Figure 34 showcases real-time prediction of recordings of the phrase *"Let's go"* and *"How are you"* respectively.



```python
from model_use import ModelPredictor
from IPython.display import Audio
from voice_rec import voice_record

predictor = ModelPredictor(input_to_softmax=final_model_mfcc,model_path='results/final_model_mfcc.h5')
voice_record(path='recordings/demo_mfcc.wav')
# display the true and predicted transcriptions and show the audio file
predictor.get_predictions_recorded(spectrogram = False,recordingpath='recordings/demo_mfcc.wav')
print('-'*80)
Audio(predictor.audio_path)
```

```
please speak into the microphone
done - result written to recordings/demo_spectro.wav
--------------------------------------------------------------------------------
Predicted transcription:

lis go
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
```

Figure 33: Live prediction using final_model_mfcc



```python
from model_use import ModelPredictor
from IPython.display import Audio
from voice_rec import voice_record

predictor = ModelPredictor(input_to_softmax=final_model_spectrogram,model_path='results/final_model_spectr
ogram.h5')
# record your voice
voice_record(path='recordings/demo_spectro.wav')
# display the true and predicted transcriptions and show the audio file
predictor.get_predictions_recorded(spectrogram = True,recordingpath='recordings/demo_spectro.wav')
print('-'*80)
Audio(predictor.audio_path)
```

```
please speak into the microphone
done - result written to recordings/demo_spectro.wav
--------------------------------------------------------------------------------
Predicted transcription:

a a you
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
```

Figure 34: Live prediction using final_model_spectrogram

13. Conclusion

This paper presents a deep learning model for speech recognition with Keras classifiers with TensorFlow backend. Various neural network layers are compared to find the best performing layer for the requirement. Process involved data collection, standardization of data, feature extraction, training model, testing model and comparing the results. MFCC and Spectrogram features were compared in different models. Impacts of varying training size, varying epochs was analyzed. Trade-off between loss and execution time was compared to obtain an efficient real-time solution for speech recognition with reduced use of system resources. Following are major conclusions from the project.

- MFCC as a feature outperforms Spectrogram in accuracy of the models.
- CNN layer ahead of GRU RNN improved the accuracy significantly.
- Increasing the depth of RNN layer also increased accuracy.
- Combining the best performing models (CNN+RNN+TDD and Deep RNN + TDD) further reduced the loss values.
- Model quality can be further improved by training with bigger training set and increasing the number of epochs.
- Adam optimizer provide a better stability for training deep learning network due to its self-stabilizing nature.

Following are the future directions for this project.

- Assessment of additional layers: Additional combinations of Keras layers can be tried to further expand the possibility of a better model. An example would be to use deep bidirectional RNNs.
- Beam Size Impact: Controlling the beam size for CTC batch can be analyzed to see the impact of beam size on accuracy.
- Noise reduction: Noise reduction code can be implemented in the voice recording module to generate better real-time recognitions.
- Lexicon Language Model: Lexicon language model provided in the Librispeech can be used as the Labels for training. This will get a better recognition rate for words in dictionary.

14. Repository

The code of this project can be accessed from below GitHub link.

https://github.com/lithathampan/speech_recognition_nn_comparison.git

Data set can be accessed from the below link.

http://www.openslr.org/12/

15. References

[1] "AIND-VUI-Capstone," Udacity, 2017. [Online]. Available: https://github.com/udacity/AIND-VUI-Capstone. [Accessed 15 Jan 2020].

[2] A. Hannun, C. Case, J. Casper, B. Catanzaro, G. Diamos, E. Elsen, R. Prenger, S. Satheesh, S. Sengupta, A. Coates and A. Y. Ng, "Deep Speech: Scaling up end-to-end speech recognition," 19 Dec 2014.

[3] W. Gevaert, G. Tsenov and V. Mladenov, "Neural networks used for speech recognition. Journal of Automatic Control," 2010.

[4] T. Likhomanenko, G. Synnaeve and R. Collobert, "Who Needs Words? Lexicon-Free Speech Recognition," 13 Sep 2019.

[5] A. Hannun, A. Lee, Q. Xu and R. Collobert, "Sequence-to-Sequence Speech Recognition with Time-Depth Separable Convolutions," 4 Apr 2019.

[6] A. L. Maas, X. Ziang, D. Jurafsky and A. Y. Ng, "Lexicon-Free Conversational Speech Recognition with Neural Networks," 2015.

[7] V. Panayotov, G. Chen, S. Khudanpur and D. Povey, "LIBRISPEECH: AN ASR CORPUS BASED ON PUBLIC DOMAIN AUDIO BOOKS," *ICASSP,* 2015.

[8] J. O. Smith III, "Mathematics Of The Discrete Fourier Transform (Dft) With Audio Applications," in *Mathematics Of The Discrete Fourier Transform (Dft) With Audio Applications*, 2007.

[9] musicinformationretrieval.com, "musicinformationretrieval.com," [Online]. Available: https://musicinformationretrieval.com/mfcc.html. [Accessed 20 Apr 2020].

[10] J. Lyons. [Online]. Available: http://www.practicalcryptography.com/miscellaneous/machine-learning/guide-mel-frequency-cepstral-coefficients-mfccs/. [Accessed 21 Apr 2020].

[11] A. Graves1, S. Fern´andez1, F. Gomez1 and J. Schmidhuber, "Connectionist Temporal Classification: Labelling Unsegmented Sequence Data with Recurrent Neural Networks," Jun 2006.

[12] "TensorFlow," [Online]. Available: https://www.tensorflow.org/. [Accessed 21 Apr 2020].

[13] J. Pawan, "Complete Guide of Activation Functions," 12 Jun 2019. [Online]. Available: https://towardsdatascience.com/complete-guide-of-activation-functions-34076e95d044. [Accessed 21 Apr 2020].

[14] A. L. Maas, A. Y. Hannun and A. Y. Ng, "Rectifier Nonlinearities Improve Neural Network Acoustic Models," 2014.

[15] D. P. Kingma and J. L. Ba, "Adam: A Method For Stochastic Optimization," 30 Jan 2017.

[16] M. Nguyen, "Illustrated Guide to LSTM's and GRU's: A step by step explanation," 24 Sep 2018. [Online]. Available: https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21. [Accessed 21 Apr 2020].

[17] J. Chung, C. Gulcehre, K. Cho and Y. Bengio, "Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling," 11 dec 2014.