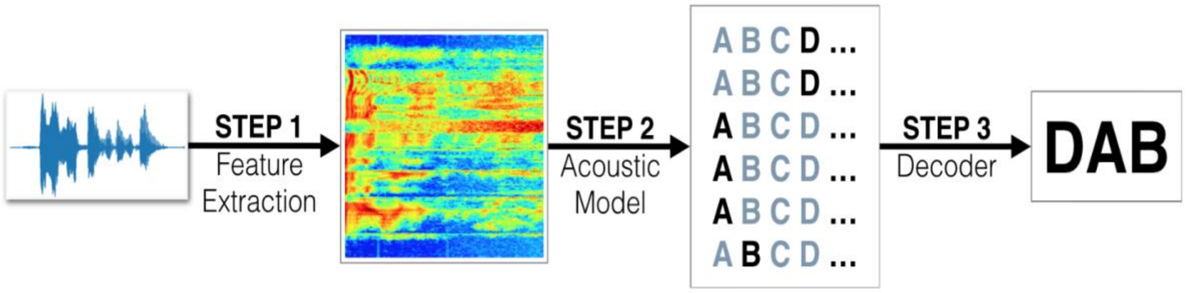
Speech Recognition with Neural Networks

Pruthvij Thakar | Gurtej Khanooja
{thakar.p, khanooja.g} @husky.neu.edu
CSYE 7245, Fall 2017, Northeastern University

Introduction: -

Speech recognition is an interesting topic in deep learning because of its difficulty and its great applications in life: Amazon Echo, Apple Siri, speech typer, etc. The most common architecture used in speech recognition is recurrent neural network (RNN) and its variants. We have implemented various variations in network architecture based on RNN and reported and compared the results and reasoned for choosing the best architecture.

The general outline for creating end to end speech recognition can be seen below:



- **STEP 1** is a pre-processing step that converts raw audio to one of two feature representations that are commonly used for ASR.

- **STEP 2** is an acoustic model which accepts audio features as input and returns a probability distribution over all potential transcriptions. After learning about the basic types of neural networks that are often used for acoustic modeling, you will engage in your own investigations, to design your own acoustic model!

- **STEP 3** in the pipeline takes the output from the acoustic model and returns a predicted transcription.

The workflow being followed is as follows:

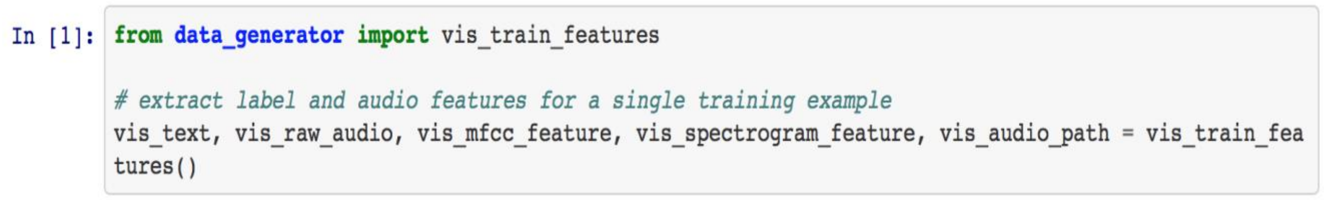
- Data Collection
- Data Preprocessing
- Acoustic Features for Speech Recognition
- Deep Neural Networks for Acoustic Modeling
    - RNN
    - RNN + TimeDistributed Dense
    - Deeper RNN + TimeDistributed Dense
    - Bidirectional RNN + TimeDistributed Dense
    - Our own model arcitecture
- Comapre Models
- Final Model

<u>The Data:</u>

We begin by investigating the dataset that will be used to train and evaluate your pipeline. Voxforge provides dataset of English-read speech, designed for training and evaluating models for ASR (Automatic Speech Recognization). The dataset contains 130 hours of speech. We will work with a small subset in this project, since larger-scale data would take a long while to train.

In the code cells below, you will use the vis_train_features module to visualize a training example.

- transcribed text (label) for the training example.
- raw audio waveform for the training example.
- frequency cepstral coefficients (MFCCs) for the training example.
- spectrogram for the training example
- the file path to the training example

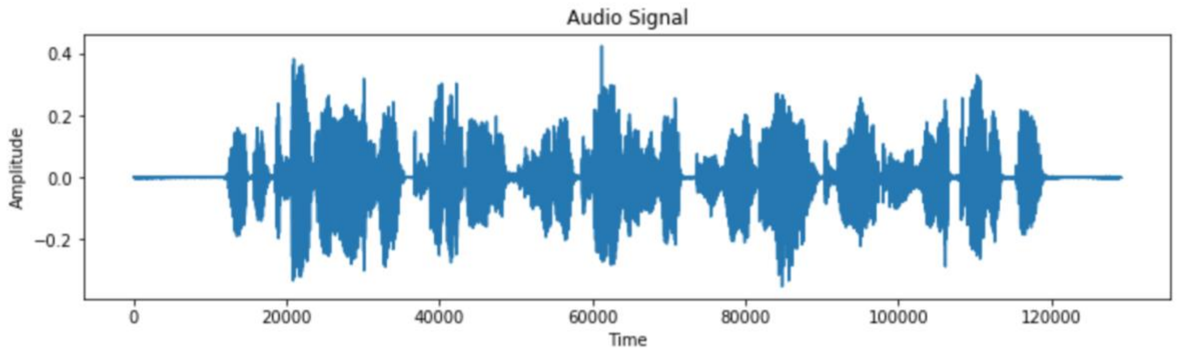


```
In [1]: from data_generator import vis_train_features

        # extract label and audio features for a single training example
        vis_text, vis_raw_audio, vis_mfcc_feature, vis_spectrogram_feature, vis_audio_path = vis_train_fea
        tures()
```

There are 3126 total training examples

The following code cell visualizes the audio waveform for your chosen example, along with the corresponding transcript.

In [2]:
```python
from IPython.display import Markdown, display
from data_generator import vis_train_features, plot_raw_audio
from IPython.display import Audio
%matplotlib inline

# plot audio signal
plot_raw_audio(vis_raw_audio)
# print length of audio signal
display(Markdown('**Shape of Audio Signal** : ' + str(vis_raw_audio.shape)))
# print transcript corresponding to audio clip
display(Markdown('**Transcript** : ' + str(vis_text)))
# play the audio file
Audio(vis_audio_path)
```

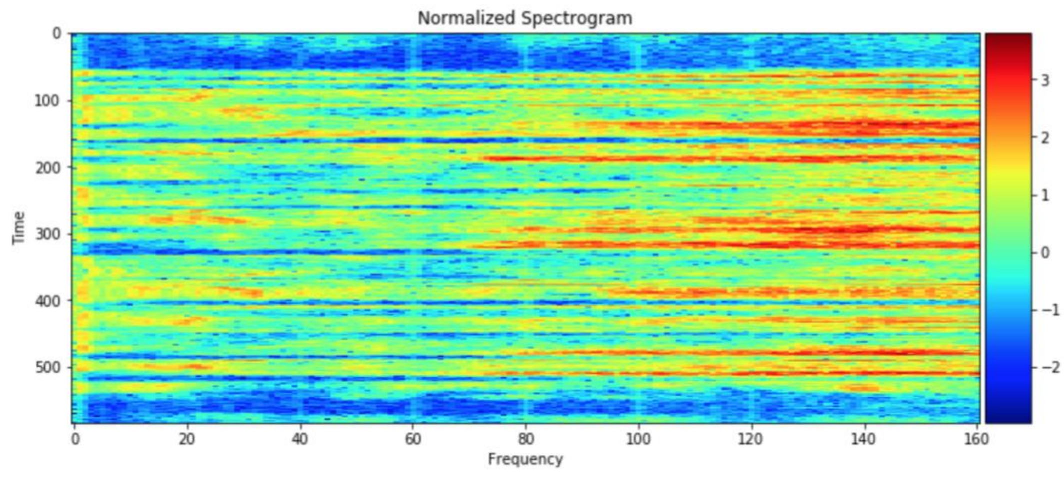


Acoustic Features for Speech Recognition:

Raw audio is generally not a good option for using input to neural networks. There are acoustic features which could be used to get better results. We explored two acoustic features for the purpose of the project: spectrograms and MFCCs.

Spectrograms:

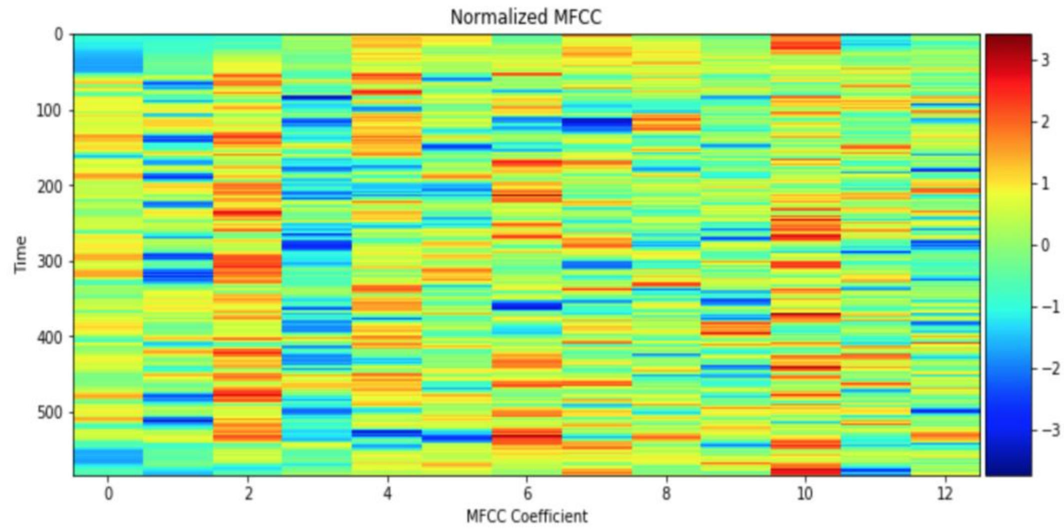
The first option for an audio feature representation is the spectrogram. The code given below returns the spectrogram as a 2D tensor, where the first (*vertical*) dimension indexes time, and the second (*horizontal*) dimension indexes frequency. To speed the convergence of your algorithm, we have also normalized the spectrogram. (can be seen quickly in the visualization below by noting that the mean value hovers around zero, and most entries in the tensor assume values close to zero.



Shape of Spectrogram : (584, 161)

Mel-Frequency Cepstral Coefficients (MFCCs):

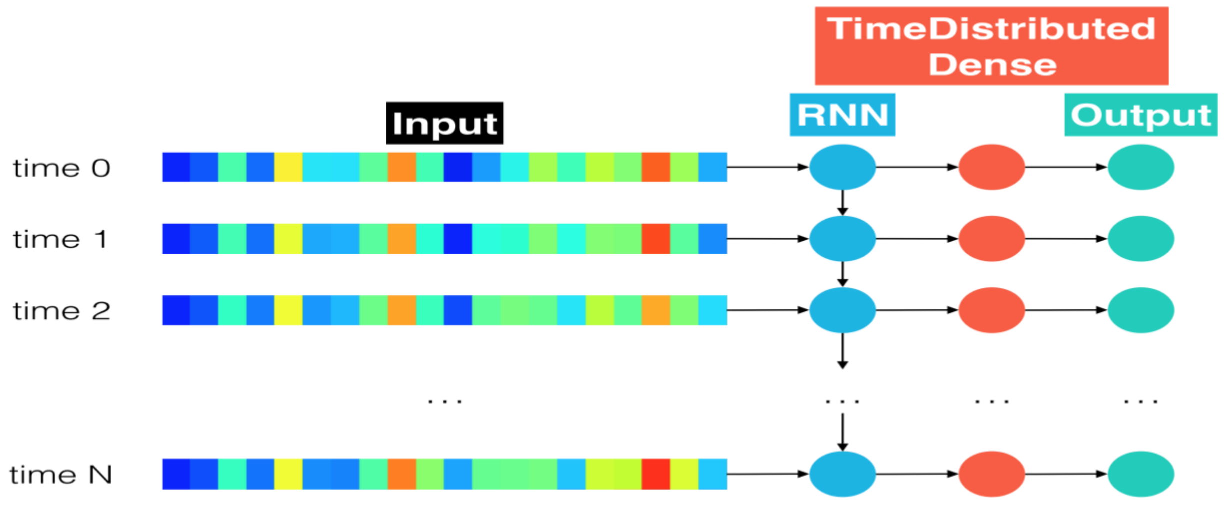
The second option for an audio feature representation is MFCCs. The main idea behind MFCC features is the same as spectrogram features: at each time window, the MFCC feature yields a feature vector that characterizes the sound within the window. Note that the MFCC feature is much lower-dimensional than the spectrogram feature, which could help an acoustic model to avoid overfitting to the training dataset.



Shape of MFCC : (584, 13)

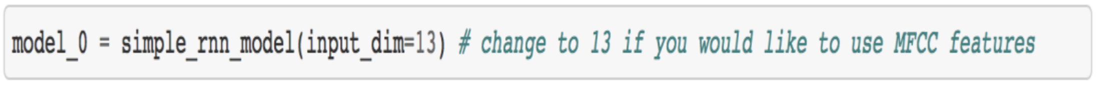
Deep Neural Networks for Acoustic Modeling

Model 0 (RNN + Time distributed dense)

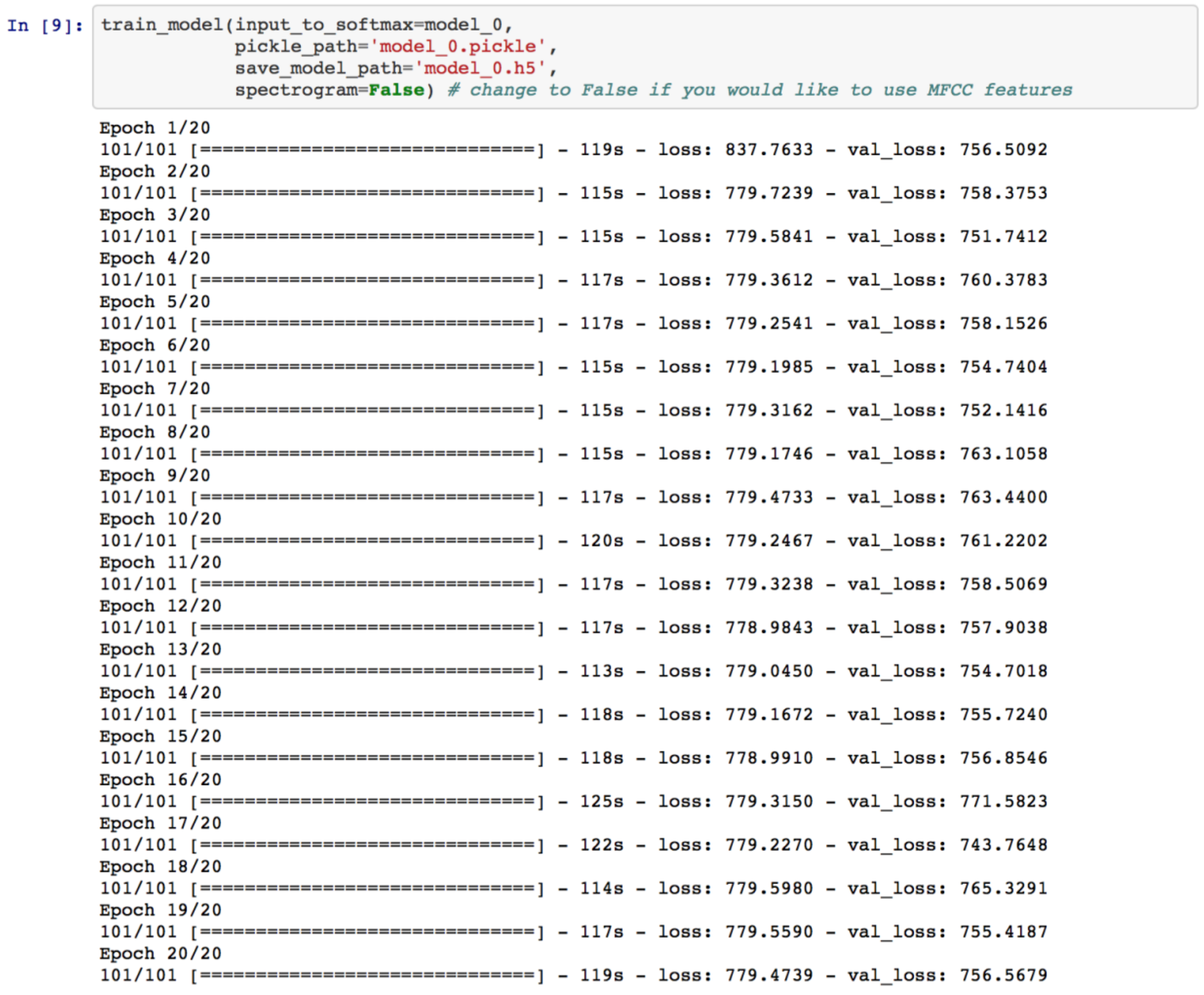


Model 0: Fully connected layer after rnn: This model is a small upgrade on the model 0, in the model 0 we use outputs from RNN and from those we are calculating loss and probs for the spoken words. This is not good in practices as we have seen. The idea of the Model 1 is to add a fully connected layer between RNN part and output probs. In this way we are getting more information which we use to get better predictions. This add-on really helped in the matter of decreasing loss. As we can see on the left graph, model immediately started with loss of about 300. But this model didn't improve over time at both training loss and validation loss.

Implementation:

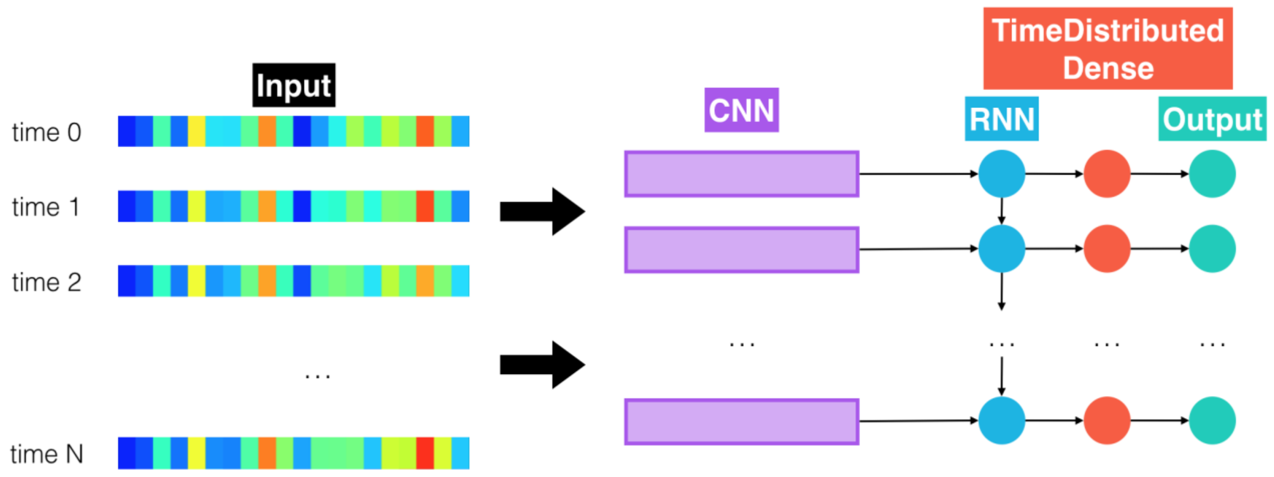


```
model_0 = simple_rnn_model(input_dim=13) # change to 13 if you would like to use MFCC features
```



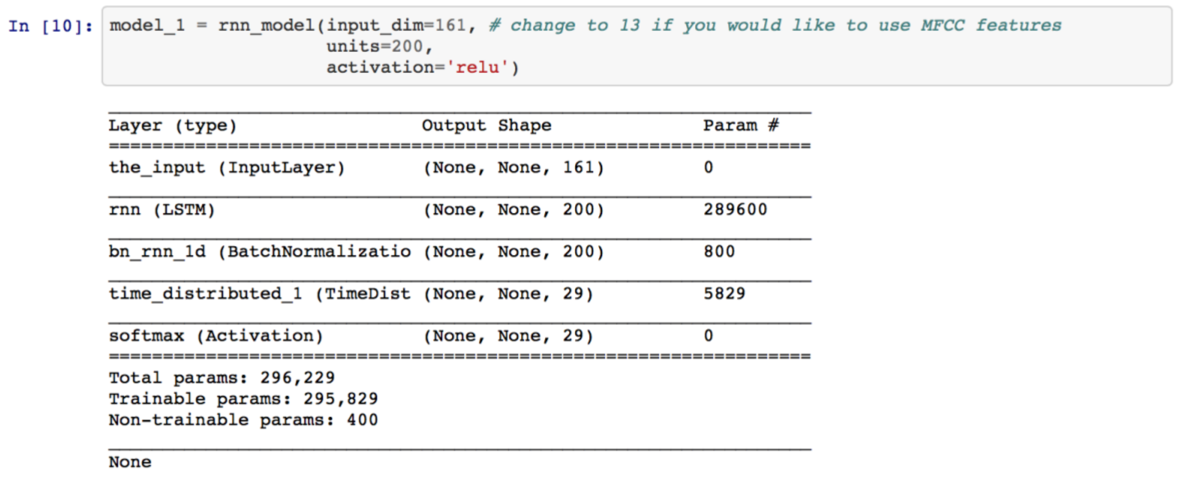
```
In [9]: train_model(input_to_softmax=model_0,
                    pickle_path='model_0.pickle',
                    save_model_path='model_0.h5',
                    spectrogram=False) # change to False if you would like to use MFCC features

        Epoch 1/20
        101/101 [==============================] - 119s - loss: 837.7633 - val_loss: 756.5092
        Epoch 2/20
        101/101 [==============================] - 115s - loss: 779.7239 - val_loss: 758.3753
        Epoch 3/20
        101/101 [==============================] - 115s - loss: 779.5841 - val_loss: 751.7412
        Epoch 4/20
        101/101 [==============================] - 117s - loss: 779.3612 - val_loss: 760.3783
        Epoch 5/20
        101/101 [==============================] - 117s - loss: 779.2541 - val_loss: 758.1526
        Epoch 6/20
        101/101 [==============================] - 115s - loss: 779.1985 - val_loss: 754.7404
        Epoch 7/20
        101/101 [==============================] - 115s - loss: 779.3162 - val_loss: 752.1416
        Epoch 8/20
        101/101 [==============================] - 115s - loss: 779.1746 - val_loss: 763.1058
        Epoch 9/20
        101/101 [==============================] - 117s - loss: 779.4733 - val_loss: 763.4400
        Epoch 10/20
        101/101 [==============================] - 120s - loss: 779.2467 - val_loss: 761.2202
        Epoch 11/20
        101/101 [==============================] - 117s - loss: 779.3238 - val_loss: 758.5069
        Epoch 12/20
        101/101 [==============================] - 117s - loss: 778.9843 - val_loss: 757.9038
        Epoch 13/20
        101/101 [==============================] - 113s - loss: 779.0450 - val_loss: 754.7018
        Epoch 14/20
        101/101 [==============================] - 118s - loss: 779.1672 - val_loss: 755.7240
        Epoch 15/20
        101/101 [==============================] - 118s - loss: 778.9910 - val_loss: 756.8546
        Epoch 16/20
        101/101 [==============================] - 125s - loss: 779.3150 - val_loss: 771.5823
        Epoch 17/20
        101/101 [==============================] - 122s - loss: 779.2270 - val_loss: 743.7648
        Epoch 18/20
        101/101 [==============================] - 114s - loss: 779.5980 - val_loss: 765.3291
        Epoch 19/20
        101/101 [==============================] - 117s - loss: 779.5590 - val_loss: 755.4187
        Epoch 20/20
        101/101 [==============================] - 119s - loss: 779.4739 - val_loss: 756.5679
```

Model 1(CNN + RNN)



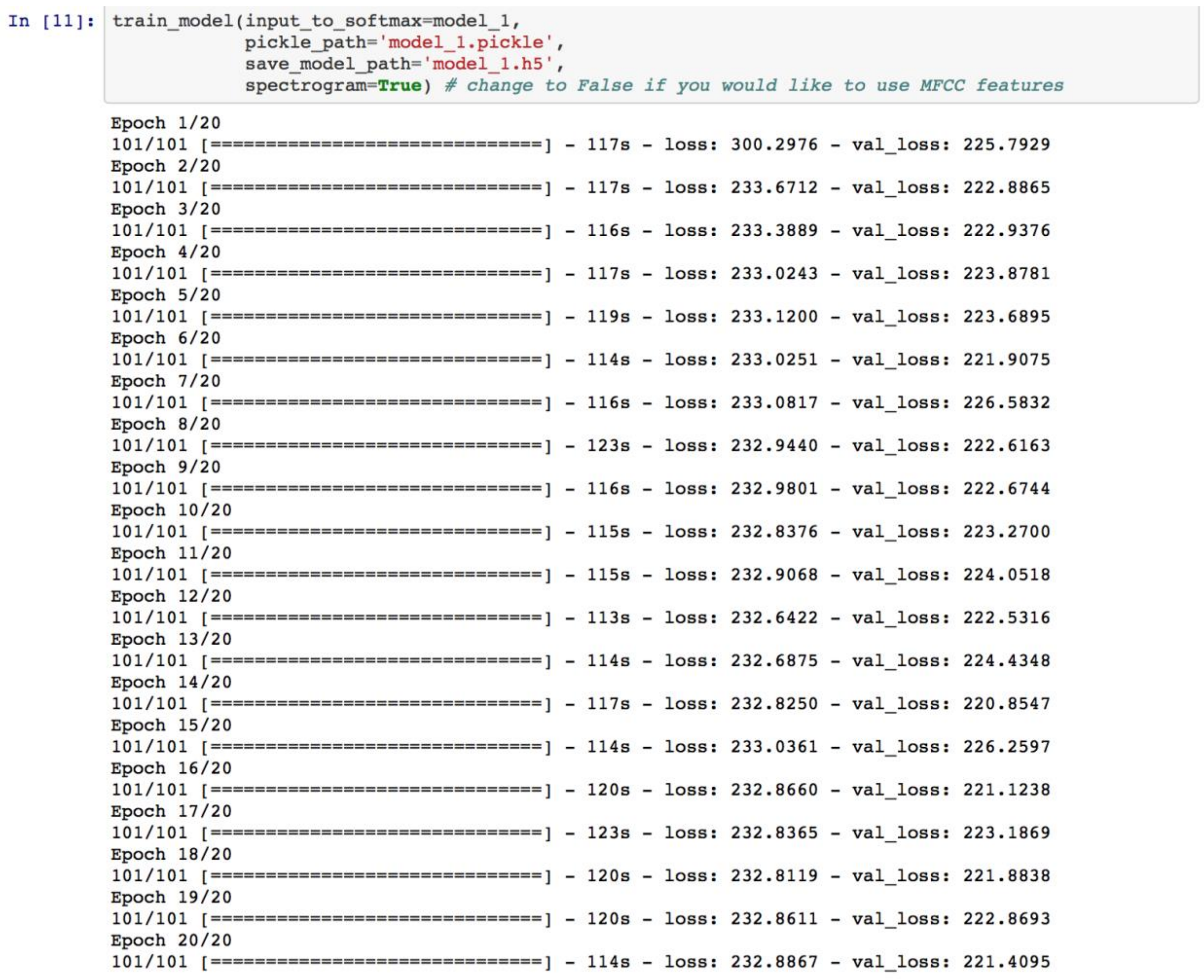
Model 1: CNN model: In this model we have added CNN layer at the beginning (before RNN layer). This way we are analyzing Spectrogram or MFCC features with convolutional kernels. With this strategy we are able to get only important features and give them to the RNN part of the network. This strategy showed the best results of all models tested. Training loss kept decreasing over time and also validation loss decreased a lot. This results helped me in deciding what layers/architecture to use in the Final model.

Implementation



```
In [10]: model_1 = rnn_model(input_dim=161, # change to 13 if you would like to use MFCC features
                             units=200,
                             activation='relu')
```

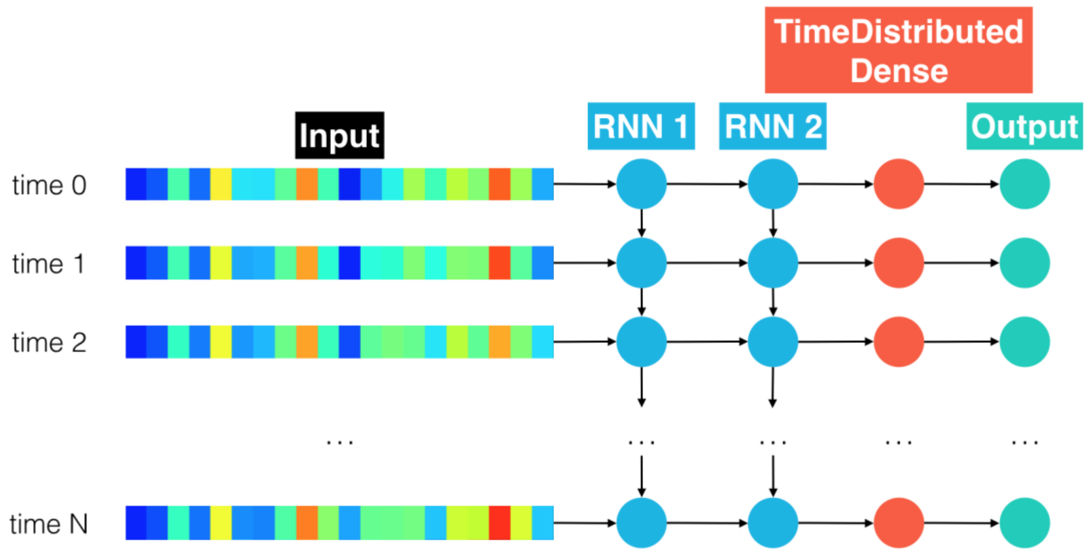
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
the_input (InputLayer)       (None, None, 161)         0
_________________________________________________________________
rnn (LSTM)                   (None, None, 200)         289600
_________________________________________________________________
bn_rnn_1d (BatchNormalizatio (None, None, 200)         800
_________________________________________________________________
time_distributed_1 (TimeDist (None, None, 29)          5829
_________________________________________________________________
softmax (Activation)         (None, None, 29)          0
=================================================================
Total params: 296,229
Trainable params: 295,829
Non-trainable params: 400
_________________________________________________________________
None
```



```
In [11]: train_model(input_to_softmax=model_1,
                     pickle_path='model_1.pickle',
                     save_model_path='model_1.h5',
                     spectrogram=True) # change to False if you would like to use MFCC features
```

```
Epoch 1/20
101/101 [==============================] - 117s - loss: 300.2976 - val_loss: 225.7929
Epoch 2/20
101/101 [==============================] - 117s - loss: 233.6712 - val_loss: 222.8865
Epoch 3/20
101/101 [==============================] - 116s - loss: 233.3889 - val_loss: 222.9376
Epoch 4/20
101/101 [==============================] - 117s - loss: 233.0243 - val_loss: 223.8781
Epoch 5/20
101/101 [==============================] - 119s - loss: 233.1200 - val_loss: 223.6895
Epoch 6/20
101/101 [==============================] - 114s - loss: 233.0251 - val_loss: 221.9075
Epoch 7/20
101/101 [==============================] - 116s - loss: 233.0817 - val_loss: 226.5832
Epoch 8/20
101/101 [==============================] - 123s - loss: 232.9440 - val_loss: 222.6163
Epoch 9/20
101/101 [==============================] - 116s - loss: 232.9801 - val_loss: 222.6744
Epoch 10/20
101/101 [==============================] - 115s - loss: 232.8376 - val_loss: 223.2700
Epoch 11/20
101/101 [==============================] - 115s - loss: 232.9068 - val_loss: 224.0518
Epoch 12/20
101/101 [==============================] - 113s - loss: 232.6422 - val_loss: 222.5316
Epoch 13/20
101/101 [==============================] - 114s - loss: 232.6875 - val_loss: 224.4348
Epoch 14/20
101/101 [==============================] - 117s - loss: 232.8250 - val_loss: 220.8547
Epoch 15/20
101/101 [==============================] - 114s - loss: 233.0361 - val_loss: 226.2597
Epoch 16/20
101/101 [==============================] - 120s - loss: 232.8660 - val_loss: 221.1238
Epoch 17/20
101/101 [==============================] - 123s - loss: 232.8365 - val_loss: 223.1869
Epoch 18/20
101/101 [==============================] - 120s - loss: 232.8119 - val_loss: 221.8838
Epoch 19/20
101/101 [==============================] - 120s - loss: 232.8611 - val_loss: 222.8693
Epoch 20/20
101/101 [==============================] - 114s - loss: 232.8867 - val_loss: 221.4095
```

Model 2(Deep RNN + Time distributed dense)



Model 2: Deep RNN Network: This model is made to have many RNN layers. In the testing (above) we have used 2 RNN layers + fully connected layer after the last RNN layer. This model showed the same training results as our Model 1. At the beginning of the training process it started to decrease loss and after 2.5 epochs it just stopped and kept that amount of loss until the end of the training. Validation loss stayed constant throughout whole training process.

Implementation:

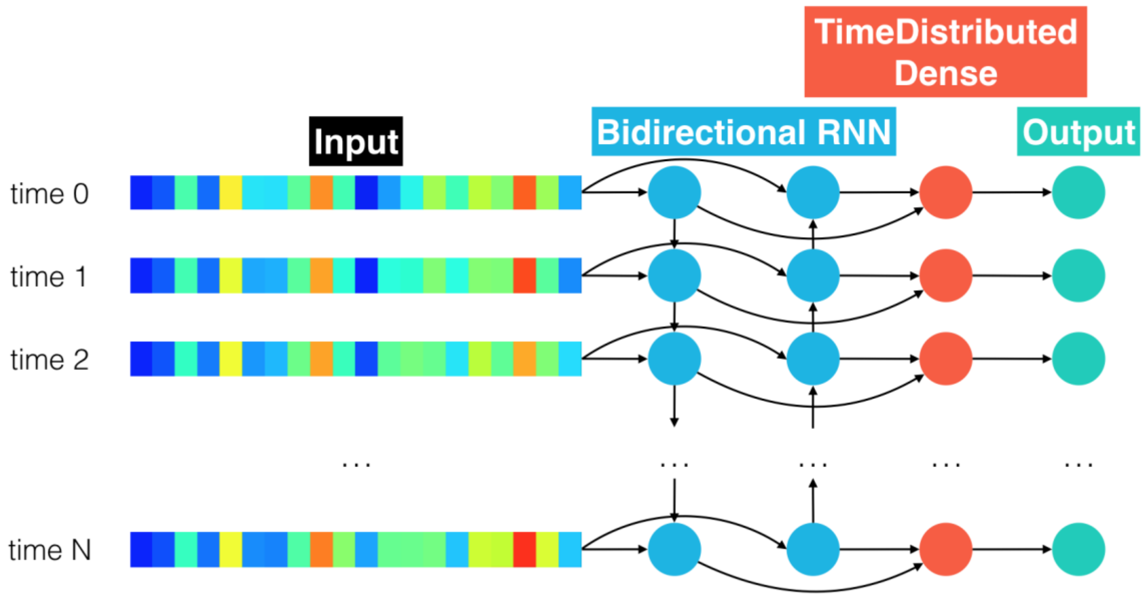
In [12]:
```
model_2 = cnn_rnn_model(input_dim=161, # change to 13 if you would like to use MFCC features
                        filters=200,
                        kernel_size=11,
                        conv_stride=2,
                        conv_border_mode='valid',
                        units=200)
```

```
_________________________________________________________________
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)         240600
_________________________________________________________________
bn_rnn_1d (BatchNormalizatio (None, None, 200)         800
_________________________________________________________________
time_distributed_2 (TimeDist (None, None, 29)          5829
_________________________________________________________________
softmax (Activation)         (None, None, 29)          0
=================================================================
Total params: 602,429
Trainable params: 601,629
Non-trainable params: 800
_________________________________________________________________
None
```

In [13]:
```
train_model(input_to_softmax=model_2,
            pickle_path='model_2.pickle',
            save_model_path='model_2.h5',
            spectrogram=True) # change to False if you would like to use MFCC features
```

```
Epoch 1/20
101/101 [==============================] - 63s - loss: 249.1161 - val_loss: 279.6265
Epoch 2/20
101/101 [==============================] - 62s - loss: 192.2664 - val_loss: 181.4677
Epoch 3/20
101/101 [==============================] - 60s - loss: 156.6647 - val_loss: 154.3840
Epoch 4/20
101/101 [==============================] - 60s - loss: 140.1063 - val_loss: 144.7106
Epoch 5/20
101/101 [==============================] - 60s - loss: 129.1700 - val_loss: 135.3142
Epoch 6/20
101/101 [==============================] - 60s - loss: 121.0549 - val_loss: 132.8940
Epoch 7/20
101/101 [==============================] - 61s - loss: 114.1555 - val_loss: 131.7514
Epoch 8/20
101/101 [==============================] - 66s - loss: 108.3093 - val_loss: 132.3798
Epoch 9/20
101/101 [==============================] - 65s - loss: 102.7633 - val_loss: 130.5701
Epoch 10/20
101/101 [==============================] - 65s - loss: 98.1989 - val_loss: 130.0429
Epoch 11/20
101/101 [==============================] - 64s - loss: 93.3816 - val_loss: 128.8333
Epoch 12/20
101/101 [==============================] - 63s - loss: 89.0302 - val_loss: 129.7670
Epoch 13/20
101/101 [==============================] - 61s - loss: 85.0271 - val_loss: 130.5819
Epoch 14/20
101/101 [==============================] - 64s - loss: 81.2161 - val_loss: 131.1650
Epoch 15/20
101/101 [==============================] - 64s - loss: 77.4708 - val_loss: 133.3010
Epoch 16/20
101/101 [==============================] - 63s - loss: 73.7549 - val_loss: 138.3304
Epoch 17/20
101/101 [==============================] - 62s - loss: 70.4167 - val_loss: 136.0437
Epoch 18/20
101/101 [==============================] - 62s - loss: 67.1312 - val_loss: 144.0295
Epoch 19/20
101/101 [==============================] - 62s - loss: 64.0253 - val_loss: 144.0692
Epoch 20/20
101/101 [==============================] - 62s - loss: 60.9438 - val_loss: 147.9649
```

Model 3(Deeper RNN + Time distributed dense)



Model 3: Bidirectional RNN layers: This model uses Bidirectional Rnn layers, which is logical use in this use case. With using Bidirectional Rnn we are able to analyze input signal from the beginning to the end and from the end to the beginning. This model showed similar results to models 1 and 3.

Implementation

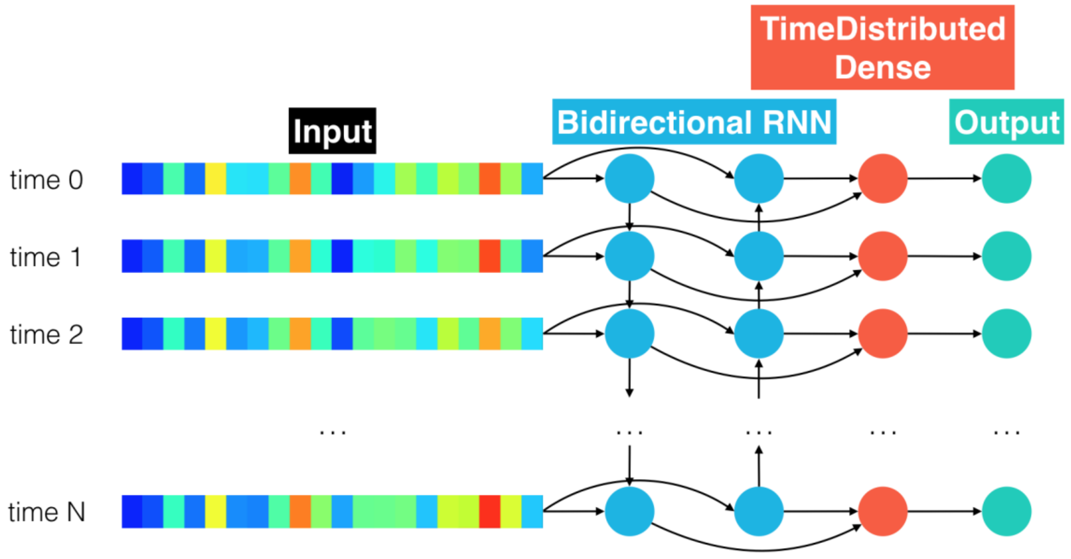
In [14]:
```
model_3 = deep_rnn_model(input_dim=161, # change to 13 if you would like to use MFCC features
                         units=200,
                         recur_layers=2)
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
the_input (InputLayer)       (None, None, 161)         0         
_________________________________________________________________
lstm_1 (LSTM)                (None, None, 200)         289600    
_________________________________________________________________
bt_rnn_1 (BatchNormalization (None, None, 200)         800       
_________________________________________________________________
lstm_2 (LSTM)                (None, None, 200)         320800    
_________________________________________________________________
bt_rnn_last_rnn (BatchNormal (None, None, 200)         800       
_________________________________________________________________
time_distributed_3 (TimeDist (None, None, 29)          5829      
_________________________________________________________________
softmax (Activation)         (None, None, 29)          0         
=================================================================
Total params: 617,829
Trainable params: 617,029
Non-trainable params: 800
_________________________________________________________________
None
```

In [15]:
```
train_model(input_to_softmax=model_3,
            pickle_path='model_3.pickle',
            save_model_path='model_3.h5',
            spectrogram=True) # change to False if you would like to use MFCC features
```

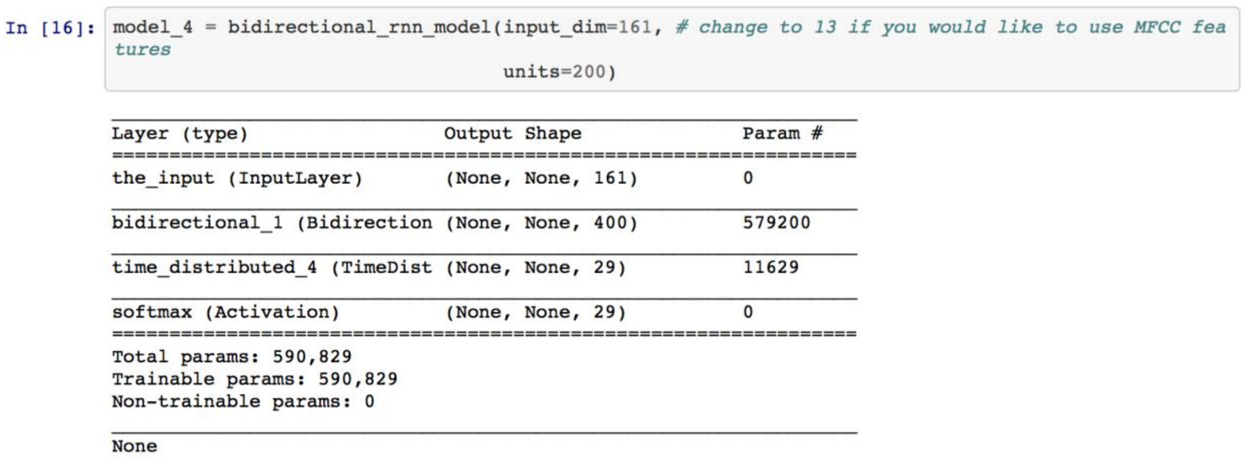
```
Epoch 1/20
101/101 [==============================] - 283s - loss: 299.6555 - val_loss: 223.3386
Epoch 2/20
101/101 [==============================] - 263s - loss: 234.0082 - val_loss: 223.4137
Epoch 3/20
101/101 [==============================] - 260s - loss: 233.3815 - val_loss: 222.8910
Epoch 4/20
101/101 [==============================] - 264s - loss: 233.2315 - val_loss: 226.2374
Epoch 5/20
101/101 [==============================] - 261s - loss: 233.0499 - val_loss: 223.1657
Epoch 6/20
101/101 [==============================] - 262s - loss: 232.9850 - val_loss: 222.8622
Epoch 7/20
101/101 [==============================] - 264s - loss: 233.0063 - val_loss: 224.3826
Epoch 8/20
101/101 [==============================] - 261s - loss: 233.0306 - val_loss: 222.3476
Epoch 9/20
101/101 [==============================] - 262s - loss: 232.7922 - val_loss: 222.7110
Epoch 10/20
101/101 [==============================] - 261s - loss: 232.7517 - val_loss: 222.6574
Epoch 11/20
101/101 [==============================] - 262s - loss: 233.0781 - val_loss: 223.4762
Epoch 12/20
101/101 [==============================] - 263s - loss: 232.8227 - val_loss: 222.2776
Epoch 13/20
101/101 [==============================] - 261s - loss: 233.0050 - val_loss: 225.1248
Epoch 14/20
101/101 [==============================] - 260s - loss: 233.0632 - val_loss: 223.2428
Epoch 15/20
101/101 [==============================] - 261s - loss: 232.7974 - val_loss: 221.2334
Epoch 16/20
101/101 [==============================] - 261s - loss: 232.8635 - val_loss: 223.2759
Epoch 17/20
101/101 [==============================] - 262s - loss: 232.9474 - val_loss: 220.7706
Epoch 18/20
101/101 [==============================] - 263s - loss: 232.8470 - val_loss: 223.5131
Epoch 19/20
101/101 [==============================] - 261s - loss: 232.8623 - val_loss: 223.4732
Epoch 20/20
101/101 [==============================] - 262s - loss: 232.8314 - val_loss: 222.8387
```

Model 4: Bidirectional RNN + Time Distributed Dense



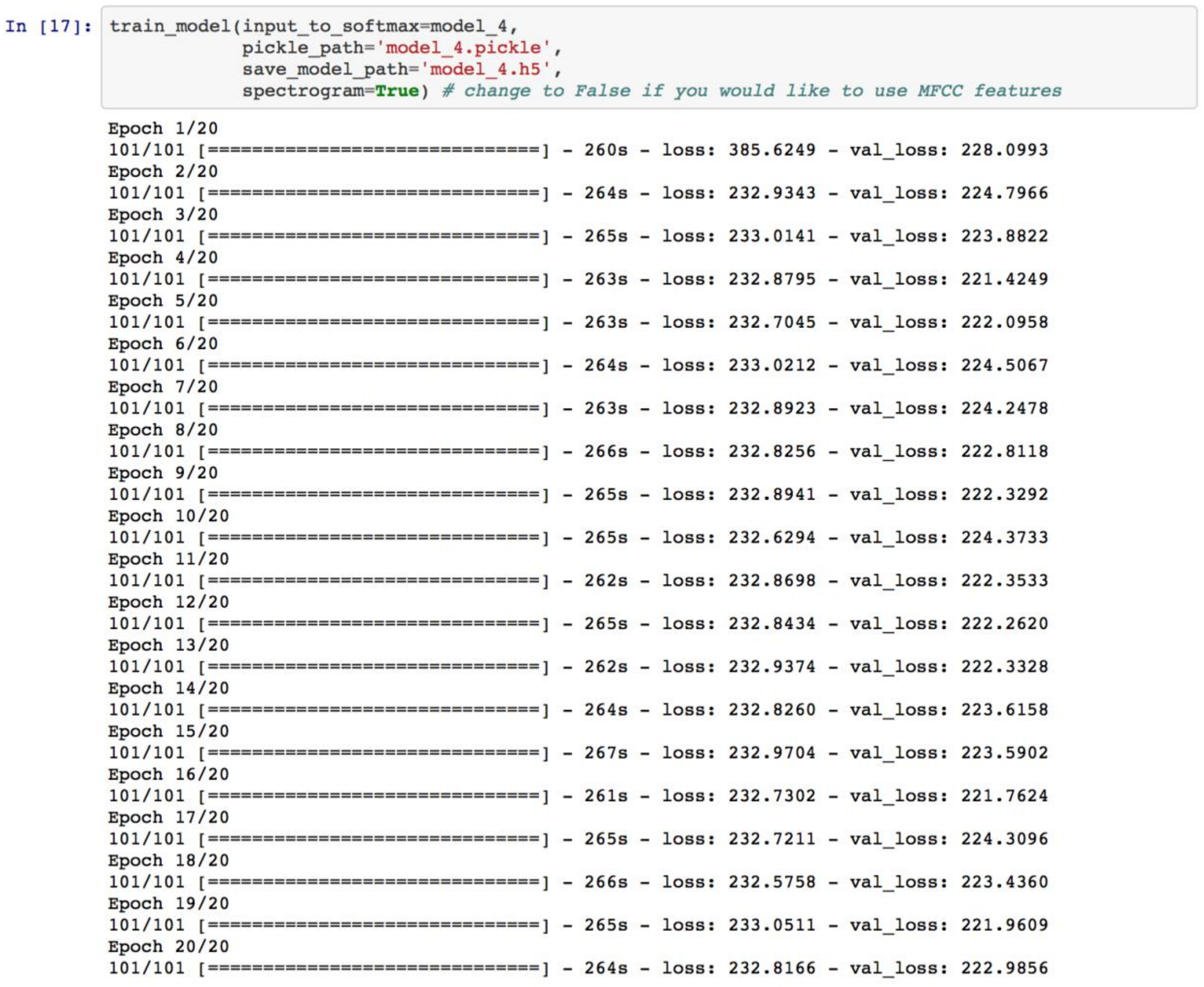
One shortcoming of conventional RNNs is that they are only able to make use of previous context. In speech recognition, where whole utterances are transcribed at once, there is no reason not to exploit future context as well. Bidirectional RNNs (BRNNs) do this by processing the data in both directions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

Implementation:



```
In [16]: model_4 = bidirectional_rnn_model(input_dim=161, # change to 13 if you would like to use MFCC fea
         tures
                                           units=200)
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
the_input (InputLayer)       (None, None, 161)         0         
_________________________________________________________________
bidirectional_1 (Bidirection (None, None, 400)         579200    
_________________________________________________________________
time_distributed_4 (TimeDist (None, None, 29)          11629     
_________________________________________________________________
softmax (Activation)         (None, None, 29)          0         
=================================================================
Total params: 590,829
Trainable params: 590,829
Non-trainable params: 0
_________________________________________________________________
None
```



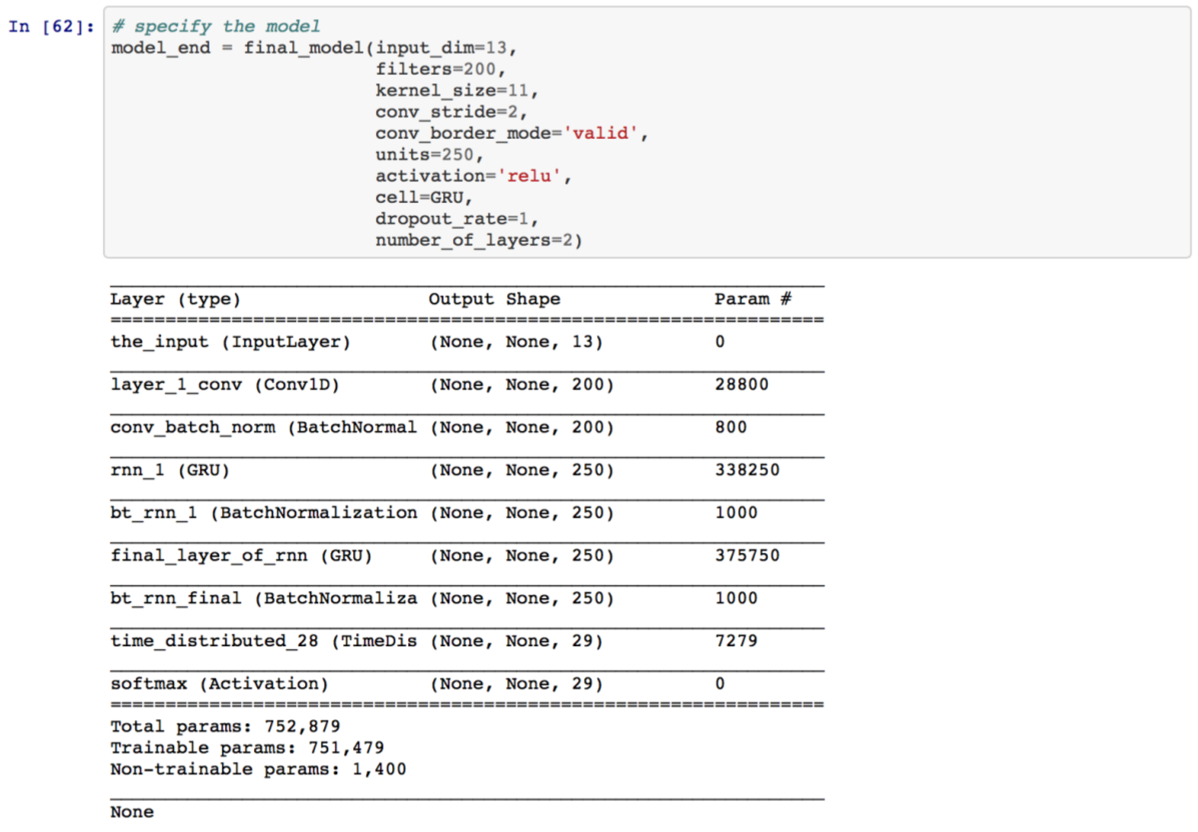
```
In [17]: train_model(input_to_softmax=model_4,
                     pickle_path='model_4.pickle',
                     save_model_path='model_4.h5',
                     spectrogram=True) # change to False if you would like to use MFCC features
```

```
Epoch 1/20
101/101 [==============================] - 260s - loss: 385.6249 - val_loss: 228.0993
Epoch 2/20
101/101 [==============================] - 264s - loss: 232.9343 - val_loss: 224.7966
Epoch 3/20
101/101 [==============================] - 265s - loss: 233.0141 - val_loss: 223.8822
Epoch 4/20
101/101 [==============================] - 263s - loss: 232.8795 - val_loss: 221.4249
Epoch 5/20
101/101 [==============================] - 263s - loss: 232.7045 - val_loss: 222.0958
Epoch 6/20
101/101 [==============================] - 264s - loss: 233.0212 - val_loss: 224.5067
Epoch 7/20
101/101 [==============================] - 263s - loss: 232.8923 - val_loss: 224.2478
Epoch 8/20
101/101 [==============================] - 266s - loss: 232.8256 - val_loss: 222.8118
Epoch 9/20
101/101 [==============================] - 265s - loss: 232.8941 - val_loss: 222.3292
Epoch 10/20
101/101 [==============================] - 265s - loss: 232.6294 - val_loss: 224.3733
Epoch 11/20
101/101 [==============================] - 262s - loss: 232.8698 - val_loss: 222.3533
Epoch 12/20
101/101 [==============================] - 265s - loss: 232.8434 - val_loss: 222.2620
Epoch 13/20
101/101 [==============================] - 262s - loss: 232.9374 - val_loss: 222.3328
Epoch 14/20
101/101 [==============================] - 264s - loss: 232.8260 - val_loss: 223.6158
Epoch 15/20
101/101 [==============================] - 267s - loss: 232.9704 - val_loss: 223.5902
Epoch 16/20
101/101 [==============================] - 261s - loss: 232.7302 - val_loss: 221.7624
Epoch 17/20
101/101 [==============================] - 265s - loss: 232.7211 - val_loss: 224.3096
Epoch 18/20
101/101 [==============================] - 266s - loss: 232.5758 - val_loss: 223.4360
Epoch 19/20
101/101 [==============================] - 265s - loss: 233.0511 - val_loss: 221.9609
Epoch 20/20
101/101 [==============================] - 264s - loss: 232.8166 - val_loss: 222.9856
```

Note: The initial scope was to implement the LAS (Listen, Attend and Spell) model from one of the papers published by Google but the main aim for the project was speech recognition. We reduced the scope to implementation of 5 models as described above whose model architecture was inspired by one of the project which was part of udacity’s artificial nano-degree program. We were quite not satisfied with the performance of all the models implemented above so to improve further we did some hyper parameter tuning and using the learnings derived from the above list of model we tried to create our own version of the model. The implementation of the final accepted model and reasoning for it is described below. Implementation of all models have been done on keras.

Model 5: Final Model (Own Architecture):

Implementation:



```
In [62]: # specify the model
         model_end = final_model(input_dim=13,
                                 filters=200,
                                 kernel_size=11,
                                 conv_stride=2,
                                 conv_border_mode='valid',
                                 units=250,
                                 activation='relu',
                                 cell=GRU,
                                 dropout_rate=1,
                                 number_of_layers=2)
```

```
_________________________________________________________________
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)         338250    
_________________________________________________________________
bt_rnn_1 (BatchNormalization (None, None, 250)         1000      
_________________________________________________________________
final_layer_of_rnn (GRU)     (None, None, 250)         375750    
_________________________________________________________________
bt_rnn_final (BatchNormaliza (None, None, 250)         1000      
_________________________________________________________________
time_distributed_28 (TimeDis (None, None, 29)          7279      
_________________________________________________________________
softmax (Activation)         (None, None, 29)          0         
=================================================================
Total params: 752,879
Trainable params: 751,479
Non-trainable params: 1,400
_________________________________________________________________
None
```

In [63]:
```
from keras.optimizers import RMSprop
train_model(input_to_softmax=model_end,
            pickle_path='model_end.pickle',
            save_model_path='model_end.h5',
            spectrogram=False)
```

```
Epoch 1/20
101/101 [==============================] - 145s - loss: 246.7676 - val_loss: 215.9699
Epoch 2/20
101/101 [==============================] - 144s - loss: 187.5169 - val_loss: 179.8064
Epoch 3/20
101/101 [==============================] - 152s - loss: 151.9236 - val_loss: 146.9361
Epoch 4/20
101/101 [==============================] - 152s - loss: 133.9369 - val_loss: 135.4676
Epoch 5/20
101/101 [==============================] - 148s - loss: 122.4069 - val_loss: 134.8893
Epoch 6/20
101/101 [==============================] - 148s - loss: 113.4147 - val_loss: 128.9967
Epoch 7/20
101/101 [==============================] - 142s - loss: 106.1782 - val_loss: 128.1952
Epoch 8/20
101/101 [==============================] - 152s - loss: 99.3979 - val_loss: 126.2738
Epoch 9/20
101/101 [==============================] - 155s - loss: 93.1851 - val_loss: 126.5522
Epoch 10/20
101/101 [==============================] - 148s - loss: 87.5127 - val_loss: 124.9496
Epoch 11/20
101/101 [==============================] - 150s - loss: 81.7004 - val_loss: 125.9966
Epoch 12/20
101/101 [==============================] - 145s - loss: 76.3927 - val_loss: 125.7233
Epoch 13/20
101/101 [==============================] - 151s - loss: 71.1534 - val_loss: 129.8055
Epoch 14/20
101/101 [==============================] - 150s - loss: 66.3418 - val_loss: 134.3553
Epoch 15/20
101/101 [==============================] - 150s - loss: 61.6126 - val_loss: 137.9712
Epoch 16/20
101/101 [==============================] - 150s - loss: 57.3928 - val_loss: 137.4264
Epoch 17/20
101/101 [==============================] - 147s - loss: 53.5605 - val_loss: 142.9806
Epoch 18/20
101/101 [==============================] - 148s - loss: 49.3938 - val_loss: 147.0497
Epoch 19/20
101/101 [==============================] - 150s - loss: 46.2359 - val_loss: 153.1326
Epoch 20/20
101/101 [==============================] - 150s - loss: 43.3314 - val_loss: 157.6372
```

Model Description and Reasoning:

While creating the Final model we have tested many different architectures. Firstly, we have started with Deep Bidirectional RNN, we used 2 and then 3 bidirectional layers. But this architecture didn't work well for us. It stayed on 500 loss through whole training. Our second attempt was to add CNN to that architecture and that improved the model a bit - it got to about 230 loss (Both training and validation loss).

Our third attempt was to add dilate parameter into convolutional layer. This didn't improve the network as well.

The next step was to create just Deep RNN with 5 GRU layers with dropout of 40%. This network was really hard to train and it did improve but for really small amount (from 760 loss to 710 in 20 epochs).

Then we changed SGD optimizer to RMSprop optimizer with learning rate to 0.02. This move didn't show any improvements overall.

We have looked at the results of the tested architectures (above, Models 1-4) and started to put a few parts together to get this architecture that we are using now.

We have used convolutional layer from Model 2, and structure from Deep RNN model. Combined them together, tried the same model with and without dropout. It seemed to work better without dropout so we trained it without dropout at all.

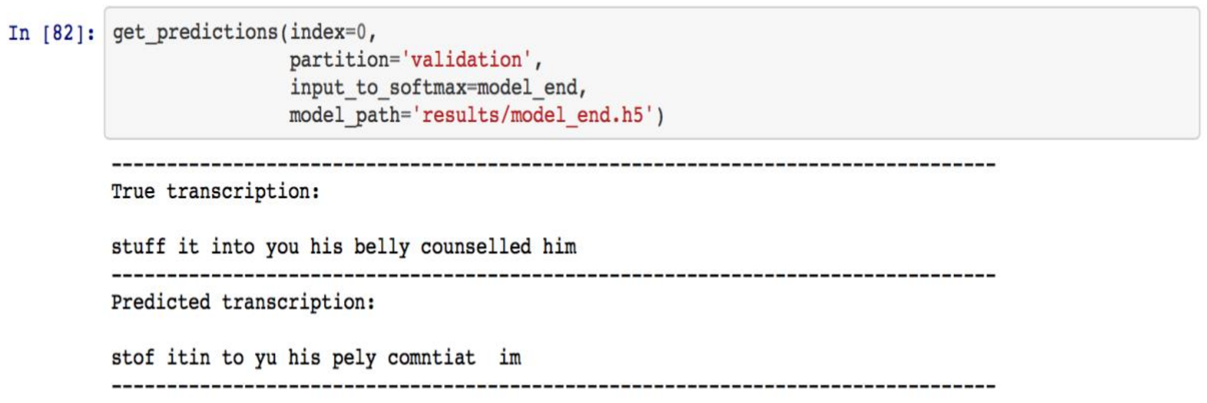
Final Model Analysis: The model was training nicely. It kept decreasing Training and validation loss over time. But there is one thing to note here, if we look at the validation loss we can see that it stopped decreasing and started to increase. This means that the network had started to overfitting to the training set.

There are a few things we can do to prevent this from happening. Early-stopping, we could have stopped training of the network after the 12 epochs.

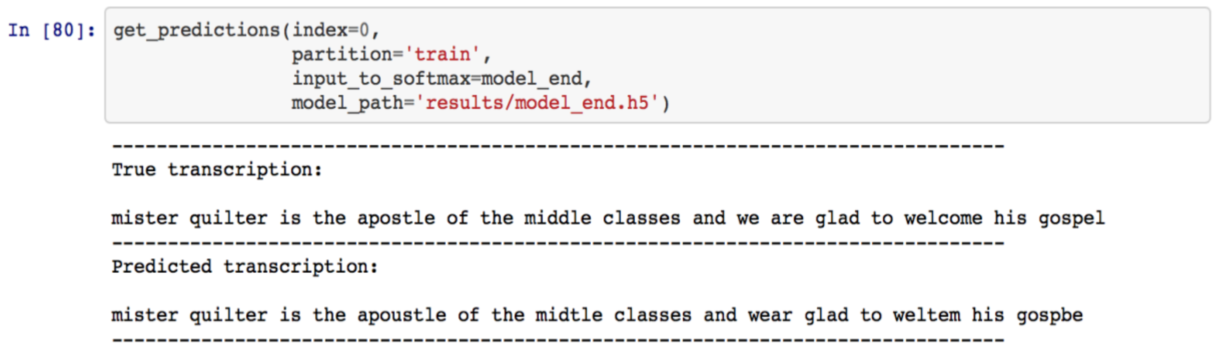
Regularization techniques such Dropout would help too. But dropout with the same setup of parameters didn't show good results, so tweaking other parameters too would help in the case of using dropout.

Predictions (Final Model)

Following are snippets of the predictions that we got from our final accepted model:



```
In [82]: get_predictions(index=0,
                         partition='validation',
                         input_to_softmax=model_end,
                         model_path='results/model_end.h5')

--------------------------------------------------------------------------------
True transcription:

stuff it into you his belly counselled him
--------------------------------------------------------------------------------
Predicted transcription:

stof itin to yu his pely comntiat  im
--------------------------------------------------------------------------------
```



```
In [80]: get_predictions(index=0,
                         partition='train',
                         input_to_softmax=model_end,
                         model_path='results/model_end.h5')

--------------------------------------------------------------------------------
True transcription:

mister quilter is the apostle of the middle classes and we are glad to welcome his gospel
--------------------------------------------------------------------------------
Predicted transcription:

mister quilter is the apoustle of the midtle classes and wear glad to weltem his gospbe
--------------------------------------------------------------------------------
```

References:

[1] Bidirectional LSTM Networks for Improved Phoneme Classification and Recognition Alex Graves1, Santiago Fern¥andez1, and J®urgen Schmidhuber
[2] K. Choi, G. Fazekas, M. Sandler, and K. Cho, ìConvolutional recurrent neural networks for music classification,î in IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), 2017.
[3] S. Hochreiter and J. Schmidhuber. Long Short-Term Memory. Neural Computation, 9(8):1735ñ1780, 1997.
[4] A. J. Robinson. An application of recurrent nets to phone probability estimation. IEEE Transactions on Neural Networks, 5(2):298ñ305, March 1994.
[5] M. Schuster and K. K. Paliwal. Bidirectional recurrent neural networks. IEEE Transactions on Signal Processing, 45:2673ñ2681, November 1997.
[6] T. N. Sainath, O. Vinyals, A. Senior, and H. Sak, ìConvolutional,long short-term memory, fully connected deep neural networks,î in IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), 2015.
[7] K. Cho, B. Van Merrienboer, C. Gulcehre, D. Bah-danau, F. Bougares, H. Schwenk, and Y. Bengio, Learning phrase representations using RNN encoderdecoder for statistical machine translation, arXiv preprint arXiv:1406.1078, 2014.
[8] D. Bahdanau, K. Cho, and Y. Bengio, ìNeural Machine Translation by Jointly Learning to Align and Translate, in ICLR, 2015.
[9] J. Chorowski, D. Bahdanau, D. Serdyuk, K. Cho, and Y. Bengio, ìAttention-Based Models for Speech Recognition, in NIPS, 2015.
[10] W. Chan, N. Jaitly, Q. Le, and O. Vinyals, ìListen, Attend and Spell: A Neural Network for Large Vocabulary Conversational Speech Recognition,î in ICASSP, 2016.
[11] D. Bahdanau, J. Chorowski, D. Serdyuk, P. Brakel, and Y. Bengio, ìEnd-to-end Attention-based Large Vocabulary Speech Recognition,î in ICASSP, 2016.