Speech Command Model

Pradeep Moturi (es16btech11016)

May 2020

Contents

1 Introduction

This is the report of my work done under Dr. G V V Sharma on building a speech command recognition model. The model built by me is based on concepts of Convolution, LSTM and Attention.

2 Create Data

1. Set 16KHz as sampling rate
2. Record 80 utterances of each command.
3. Trim each utterance to one second.
4. Save samples of each command in different folders
   Dataset/forward
   Dataset/back
   Dataset/left
   Dataset/right
   Dataset/stop

Used Audacity to do this.

3 Loading Data

I've used soundfile package to read the .wav. You may choose to use any other package like wavefile, librosa etc to do the same job.
As this is one of the slowest part, I've stored the loaded data as a numpy file for ease and speed of access. Now, I can load the data from npy file if repeating the experiment.

4 Split dataset

Do a stratified split of the dataset into train and test set with 20% as test samples.
Set a random seed for reproducing the split.

5 Augment data

Augment each audio sample by time shifting in 25000 length vectors filled with zeros.
Take steps of 500 to create 18 files per sample

6 Feature Extraction

MFCCs are most prominent features used in audio processing. Normalizing the MFCCs over the frequency axis is found to reduce effect of noise.
Kapre is a python package that provides layers for audio processing that are compatible with keras and utilize GPU for faster processing. Kapre provides us with a layer basically

Melspectrogram (padding='same', sr=16000, n_mels=39, n_dft = 1024, power_melgram=2.0, return_decibel_melgram=True, trainable_fb=False, trainable_kernel=False, name='mel_stft')

Arguments to the layer
padding: Padding when convoluting
sr: Sampling rate of audio provided
n_mels: number of coefficients to return
n_dft: width
power_melgram: exponent to raise log-mel-amplitudes before taking DCT. Using power 2 is shown to increase performance by reducing effect of noise
return_decibel_melgram: If to return log over values
trainable_fb: If filter bank trainable
trainable_kernel: If the kernel is trainable

7 Building Model

7.1 Layers

1. Using Convolutional layers ahead of LSTM is shown to improve performance in several research papers.

2. BatchNormalization layers are added to improve convergence rate.

3. Using Bidirectional LSTM is optimal when complete input is available. But this increases the runtime two-fold.

4. Final output sequence of LSTM layer is used to calculate importance of units in LSTM using a FC layer.

5. Then take the dot product of unit importance and output sequences of LSTM to get Attention scores of each time step.

6. Take the dot product of Attention scores and the output sequences of LSTM to get attention vector.

7. Add an additional FC Layer and then to output Layer with SoftMax Activation.

7.2 Hyperparameters

- **sparse_categorical_crossentropy** is used as **Loss** because only output which should be 1 is given instead of One Hot Encoding.

- **sparse_categorical_accuracy** is used as performance **Metric** for the above reason.

- **Adam** is used as **Optimizer**. Adam is adaptive learning rate optimization algorithm. This is shown to achieve a faster convergence because of having all the features of other optimization algorithms.

8 Training

- Batch size around 15 is found optimal.

- Often convergence is achieved in less than 5 epochs.

9 Testing

1. Augment the test set same as training set.

2. Extract MFCCs using same method as training set

3. Test set is passed as validation set to fit method of model.

4. The performance of model on test set is calculated after every epoch.

10 Visualize Attention

1. Now build a sub model from the trained model. Take same input layer but add 'AttentionSoftmax' layer as additional output layer.

2. Pass MFCCs of test samples to predict method.

3. Now plot log of Attention Scores and corresponding input vector before taking MFCCs on different axes.

4. We observe that Attention Scores are high on informative part.

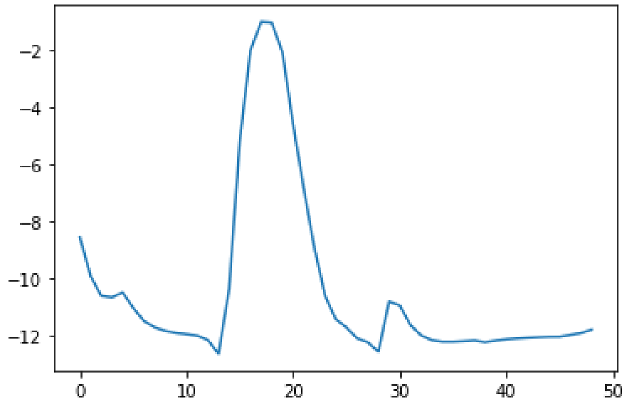

Figure 1: Attention Scores

.

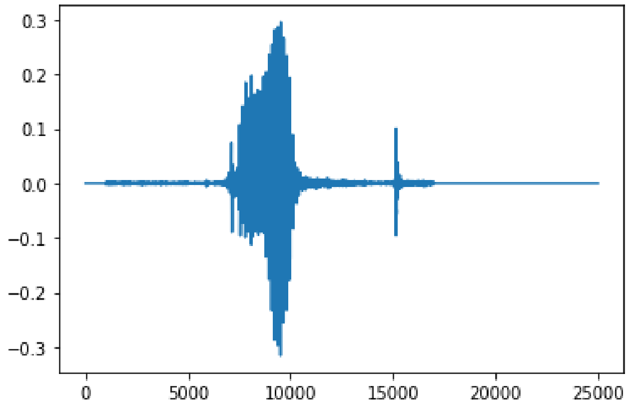

Figure 2: Raw sample

.

11 Observations

- Smaller batch size is prefferable

- Setting power_melgram=2 of Melspectogram gave faster convergence.

12 Files

- Src/DataGenerator.py: Augments the data
- Src/FeatureExtractor.py: Extracts MFCC coefficients
- Src/TrainModel.py: Trains model and saves it in h5 file
- ColabNotebook.ipynb: Use this for experimental purpose

13 Further

- Different augmentation techniques like adding noise, changing pitch, speed etc. [https://medium.com/@makcedward/data-augmentation-for-audio-76912b01fdf6]
- We can change the arguments to Melspectrogram
- Changing the model architecture like layers and units in layers.
- Further the scope of project to check performance on Google's Speech Command Datasets (v1 and v2) and participate in Kaggle challenge by google [https://www.kaggle.com/c/tensorflow-speech-recognition-challenge/]

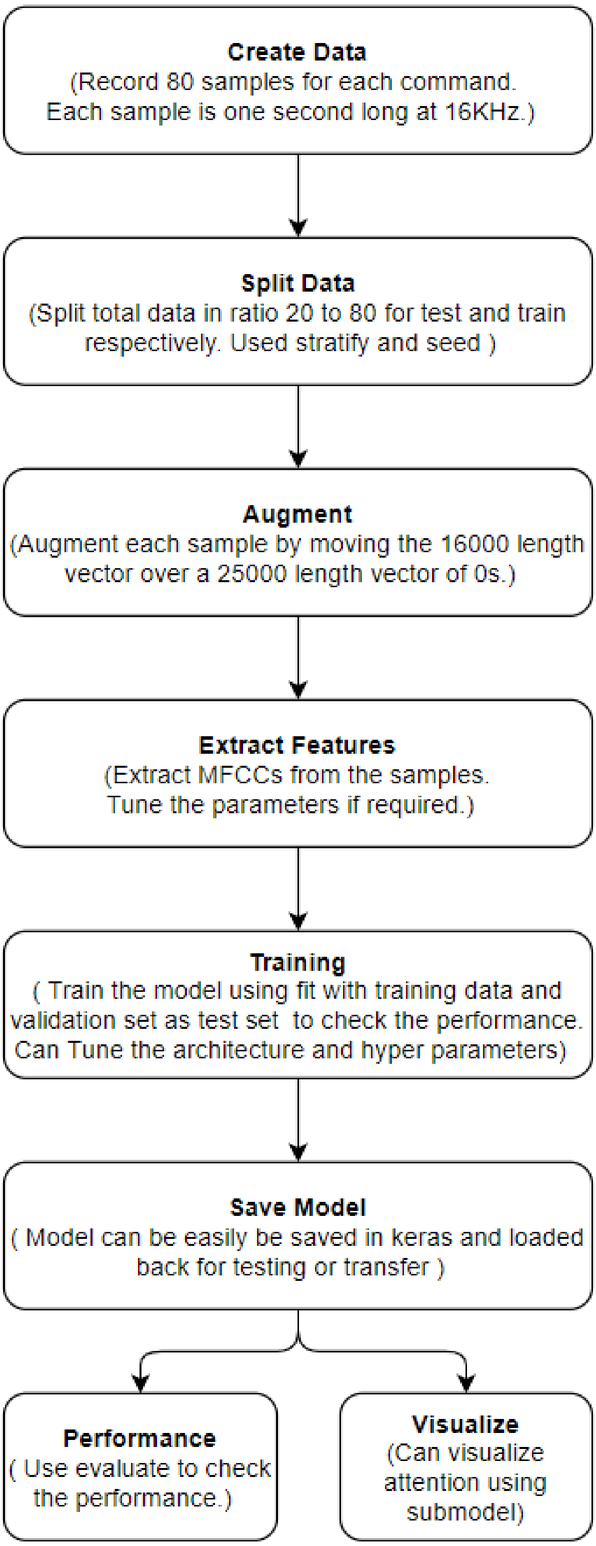

Figure 3: Data Flow Diagram

.

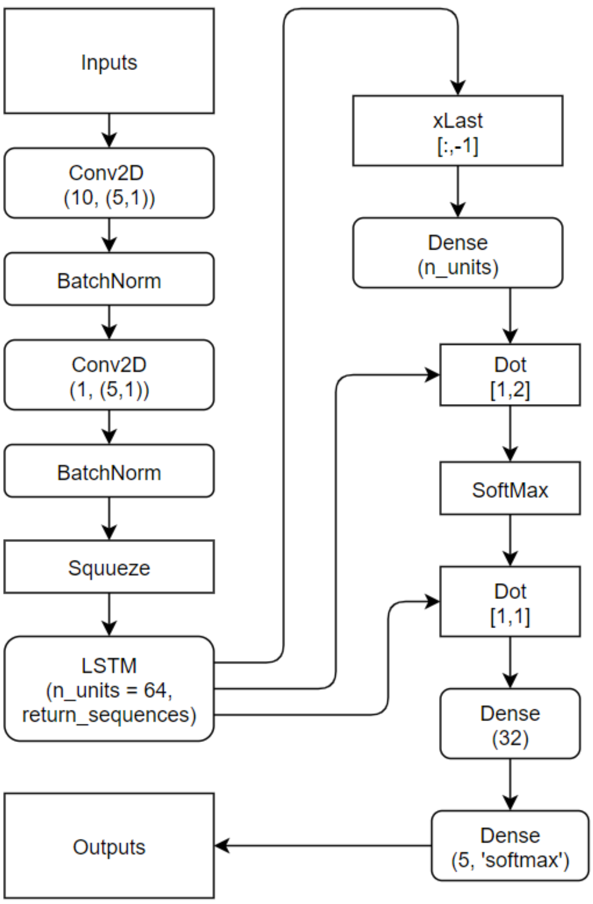

Figure 4: Model Diagram

.