Tensorflow Speech Recognition Challenge

Dongbo Wang Kexin Hui Zhaoheng Hu
University of Illinois at Urbana-Champaign
{dwang49, khui3, hu61}@illinois.edu

Abstract

Speech recognition using deep learning is a challenging task where speech signal may vary in speed, tone, accent, amplitude and etc. In this paper, we are aimed to understand English voice commands of 11 simple words from Tensorflow new released Speech Commands Dataset. We compare four methods of extracting audio features: no extraction, short time Fourier transform (STFT), filter banks (FBanks), and Mel-frequency Cepstral coefficients (MFCC). We use the preprocessed audio features and feed into a long short-term memory (LSTM) network to predict multiclass accuracy. We altered the number of hidden units in LSTM and also the loss function in order to achieve better performance. The results are promising with accuracy at 73.101%.

1. Introduction

Speech recognition is invading our lives. For example, Apple users are enjoying the benefits from Siri to get things organized, stay in touch with friends and families, and get answers for almost everything. Amazon Echo is another hands-free voice-controlled home device that allows users to play music, ask questions, make calls and so on. It is connected to Alexa Voice Service to understand any voice command from the user and execute. Therefore, the need for more accurate and faster speech recognition algorithm is urgent.

Recently, Tensorflow has released a new Speech Commands Dataset, and it inspires us to use this dataset to build an algorithm that understands such simple voice commands. To be specific, the dataset contains 65000 audio clips for one-second long utterances of 30 simple short words. There are 12 possible labels: 'yes', 'no', 'left', 'right', 'up', 'down', 'on', 'off', 'stop', 'go', 'silence', and 'unknown'. In other words, we are required to classify only 11 of them including 'silence'. And all other words that don't belong to one of the first 11 labels are labeled as 'unknown'.

As is known to all, speech recognition has long been considered a digital signal processing (DSP) problem. The time-domain sound wave can only reflect the variation of amplitude with time in one dimension. Yet in frequency domain, we can know how the sound is interacted with the medium, what the tone of the sound is, how much energy is and etc. There are more methods in frequency domain as well to help us better visualize, understand and compute the signal. As a result, we need some speech pre-processing before the recognition to extract features. Naturally, Fourier transform is used to transform the audio wave into frequency domain. It will break apart the complex sound wave into simple sinusoidal signals that make it up. We can add up how much energy is conserved in each frequency band and stack the power spectrum with respect to time as to get spectrogram. Spectrogram is important in identifying the sound waves since it gives us a straight-forward sense of how the energy of a sound signal over time is with respect to various frequencies in a single graph. Filter banks (FBanks) and Mel-Frequency Cepstral Coefficients (MFCC) [2] are two other popular methods for extracting speech features. FBanks are based on the spectrogram we get from Fourier transform and to apply multiple triangular filters, on a Mel-scale to the spectrogram to extract frequency bands. The Mel-scale aims to mimic the linearity of human voice by being more discriminative at lower frequencies and less discriminative at higher frequencies. MFCC is applied after FBanks to decorrelate the filter banks coefficients to generate a more constrained and compressed representation using Discrete Cosine Transform (DCT).

Prior to the advent of neural network, researchers often used Gaussian mixture model and hidden Markov model to do the speech recognition task. However, neural network is so powerful today that we can use deep learning to solve this task more easily and accurately. Utterance of a word can be considered as a time series of phonemes [7]. For example, the word "tomato" can be perceived as "t ah m aa t ow", a sequence of phonemes. We can use recurrent neural networks (RNN) which are capable of addressing the dynamic temporal behaviors. But in practice, RNNs are not able to learn long-term dependencies given that the gap between the relevant information and the point where it is needed can become very large sometimes [5][3]. Thus, long short-

term memory, LSTM, a special kind of RNN, is used here to learn such long-term dependencies since the time period of an utterance may be very long. In addition, we use cross entropy loss to train the network since it is a typical multi-class classification problem.

It seems now we have the network and we have the audio data, and we can get rid of the pre-processing and wait for a couple of hours or days to see a satisfying result. But can we? Unfortunately, the neural network today is not that powerful to directly learn something from the audio wave. The big problem is that speech varies in speed, and automatically aligning audio data of various length to a fixed-length word turns out to be extremely hard. We still need to do the preprocessing in advance. Inherited from the old fashions of DSP, we use Fourier transform, FBanks and MFCC as our feature extraction methods. Specifically, we use short-time Fourier transform (STFT) in this case since it can generate the Fourier spectrogram of multiple short segments of equal length, which can be easily fed for a network. As reference, we also feed the network with the time-domain audio wave directly, namely no extraction. We prepare the data in the same manner for all four methods, with each frame for 25ms with 10ms overlapping.

In this paper, we discuss the implementation of integration of feature extraction and a LSTM network to report multi-class accuracy on Tensorflow Speech Commands Dataset. We compare the performance of four feature extraction methods. We also alter several hyperparameters including number of hidden units in LSTM and weights of loss function to achieve higher accuracy. Results demonstrate that filter banks gives the highest recognition accuracy at 73.101%.

2. Related Work

2.1. Speech Recognition

In the area of speech recognition, researchers normally apply various methods to extract features from the plain audio data in time domain. The key technique of these methods is STFT (short time fourier transform) which will convert audio data into frequency domain spectrogram. STFT is common method used in digital signal processing. It will apply fourier transform to the audio, which will break down the signal to a series of cosine functions representing the different frequency components of the signal.

Other processes will be applied to the spectrogram, to extract features like FBanks (Filter Banks) [1], MFCC (Mel-frequency Cepstral Coefficients) [2][4]. These methods are based on the prior knowledge about the characteristic of the human voice and are mostly inherited from speech recognition tasks when deep learning is not that popular.

2.2. LSTM

In our work, we are first extracting audio features by applying some signal processing techniques like STFT to generate spectrograms so as to feed into LSTM to train our network. Thus, we will review the technically related learning based work in the following.

LSTM, namely long short term memory network, is a kind of recurrent neural network (RNN) which performs well in classifying, processing and predicting time series given time tags of unknown size and duration between important events. Compared to a standard RNN, LSTM is well-suited to solving those problems that requires learning long-term temporal dependencies due to its architecture.[3]

An LSTM network consists of memory cells, which can "remember" a value for either long or short time periods. This is because there are a set of gates in a cell to control when information enters the memory, when it's output and when it's forgotten.

3. Implementation

We break down the task into two parts: audio feature extraction, and neural networks.

3.1. Audio Feature Extraction

In the area of speech recognition, researchers normally apply various methods to extract features from the plain audio data in time domain. The key technique among these methods is STFT (short time Fourier transform), which will convert audio data into frequency domain spectrogram. Other processes will be applied to the spectrogram, for example frame overlap, removing high frequency component, applying filter banks, etc. These methods are based on the prior knowledge about the characteristic of the human voice, and are mostly inherited from speech recognition tasks when deep learning is not that popular.

However, these feature extraction methods will certainly seem unnatural to the experts of deep learning, since most of these extraction methods are linear and deep neural networks are meant to handle this well. We should be able to just feed the network with plain audio data and with carefully designed model architecture, so that the model should be able to learn the best feature extraction method on its own. Some people even go out of their way to argue that, given that the STFT is an linear operator itself, a deep model should be able to learn it as well. This is actually a pretty interesting and valid point, but there are also people who argue that STFT is such an sophisticated operation that it is extremely difficult to design a model to learn that.

With the discussion above, we decided to implement four methods of feature extraction: no extraction, STFT, FBanks (Filter Banks) and MFCC (Mel-frequency Cepstral Coefficients), where FBanks and MFCC are the methods that have

been widely used in speech recognition way before deep learning becomes popular.

1. No Extraction
   This was intended to examine the fascinating point that deep models are capable of learning the linear STFT operator. To match the operations before STFT, we cut the plain audio data in to 25ms frames with 10ms overlapping, and pack the frames together into a 2D array. Therefore, this transformation will produce the same shape data as the other methods. No other operations are applied to the audio signal.

2. STFT (Short Time Fourier Transform)
   This method will generate the full frequency spectrogram of the audio signal. The STFT operation has several parameters that could be altered: frame size, overlapping size, fft number and window options. We followed the conventional settings that are widely used by other research: frame size is 25ms, overlapping size is 10ms, 512 point fft and hamming window.

3. FBanks (Filter Banks)
   This method will generate partial frequency spectrogram that has been chopped and separated into different frequency bands. FBanks follows directly after STFT and applies several triangular-shaped filters distributed over the major frequency regions that the human voice resides upon. We used 40 filters thus the extracted features are reduced to 40 values per frame.

4. MFCC (Mel-frequency Cepstral Coefficients)
   This method will generate highly constrained spectrogram that has been chopped and modified even more than the FBanks method. MFCC is used directly after FBanks, and applies DCT (Discrete Cosine Transform) to generate decorrelated filter banks coefficients. As a result, each frame has 12 values only. [2]

3.2. LSTM

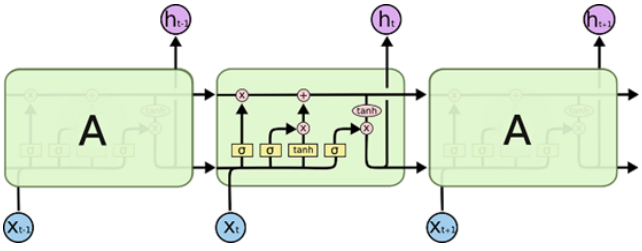


Figure 1: Example of a Standard LSTM Cell

A standard LSTM cell has the structure as the Figure 1. Each time, both last cell state and output are delivered back to the cell again and processed by different units in the cell to update cell state and generate new output.

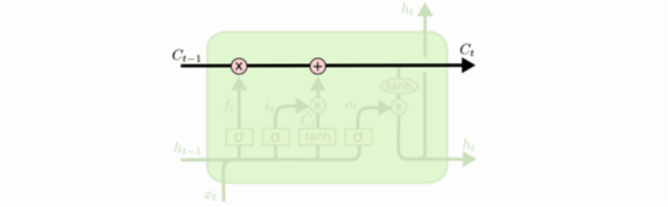


Figure 2: Horizontal Line in an LSTM Cell

The highlighted horizontal line in the Figure 2 represents the cell state. It takes last cell state and update it to get new state. This process is controlled by different gates, which can be divided into several units.

1. The first unit is a sigmoid operation, it takes the last output and new input, combines them and applies a sigmoid operation to them. This will generate values between 0 to 1 to decide which part in the previous cell state should be forgotten: 0 means "totally forget" and 1 means "reserve all". A mathematical representation for this unit is shown on Figure 3a and we get f_t as the output.

2. The next unit will select the new information that should be added to cell state. Sigmoid operation here gives an output i_t, which performs like a filter. $Tanh$ function, on the other hand, can map inputs to $\tilde{C}_t$, which is in the range between -1 and 1 since sometimes we want to add negative information to the state.

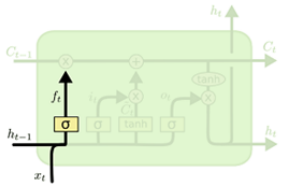
3. After above two steps, we get filters f_t and i_t that can be used to select what to forget and what to add. Also we have a mapped information $\tilde{C}_t$. Therefore, we apply filters separately by doing multiplication and add two results together, which will give us a new cell state C_t and pass back to cell next time.

4. The last unit will decide the output of current cell. O_t acts as a selector by applying sigmoid to the combination of x_t and h_{t-1}. $Tanh$ gate will take the new state as input and give its outputs as candidates to O_t. After multiplication, h_t will be the output of current cell.

There are two inputs and two outputs for each LSTM cell. From C_{t-1} to C_t, there are only linear operations including elementwise addition and multiplication so it is taking charge of long-term memory. In comparison, from h_{t-1} to h_t, there are some non-linear operation, so it reflecting the short-term memory.

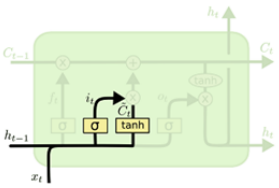
3.3. Loss Function (Unbalanced Weights)

Basically, the loss function we are using is cross entropy loss, because this is a typical multiple class classification



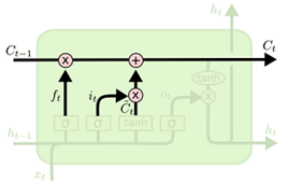
$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

(a) First Unit



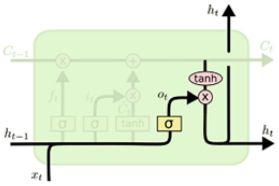
$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

(b) Second Unit



$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

(c) Third Unit



$$o_t = \sigma\left(W_o \; [h_{t-1}, x_t] \; + \; b_o\right)$$
$$h_t = o_t * \tanh\left(C_t\right)$$

(d) Fourth Unit

Figure 3: LSTM Walk Through

problem where each sample only belongs to one class.

$$L = -\frac{1}{N}(\sum_{i=1}^{N} \mathbf{y_i} \cdot \log \hat{\mathbf{y}}_\mathbf{i}) \tag{1}$$

However, the dataset is highly unbalanced. There are totally 31 files in the training set that are roughly the same size, with each file representing for one word (including "silence"). We are required to distinguish only 11 of the 31 words and classify all other words into a single class called "unknown". That means around 66.7% of the data belongs to the same class, making this dataset seriously unbalanced. In order to deal with this, we introduce class weights to the loss function: 0.15 for the "unknown" class and 1 for all other classes. 0.15 is chosen as 1 over its frequency count.

3.4. Implementation Details

1. Extract features by converting audio to spectrogram.

   (a) Pre Emphasis (Low Pass Filtering to Remove High Frequencies)
   We run test on both with and without pre emphasis, since people mentioned that high frequency actually conveys lots of information that potentially benefits the learning problem.

   (b) STFT
   We follow the convention for STFT that frame size is 25ms, overlapping is 10ms, hamming window, and 512 FFT (zero-padding). In addition, due to the symmetry of FFT, we only keep half of the FFT results. As a result, 1 second audio is converted to 101 frames with each frame containing 257 samples, which is packed as a 2D array.

   (c) FBanks
   We applied filter banks to the STFT results. Each frame now only has 40 samples.

   (d) MFCC
   DCT is applied to the FBanks results. Each frame now only has 12 samples.

   (e) Mean Removal and Normalization
   We did the normalization by removing the mean of the spectrogram and the variance. Thus, each spectrogram contains roughly the same energy.

2. Given the spectrogram images as input, we split a spectrogram into rows, and feed into the LSTM cell row by row. In this way, each spectrogram is given to LSTM as a sequence, and LSTM will learn the patterns of different spectrogram images, which corresponds to different words.

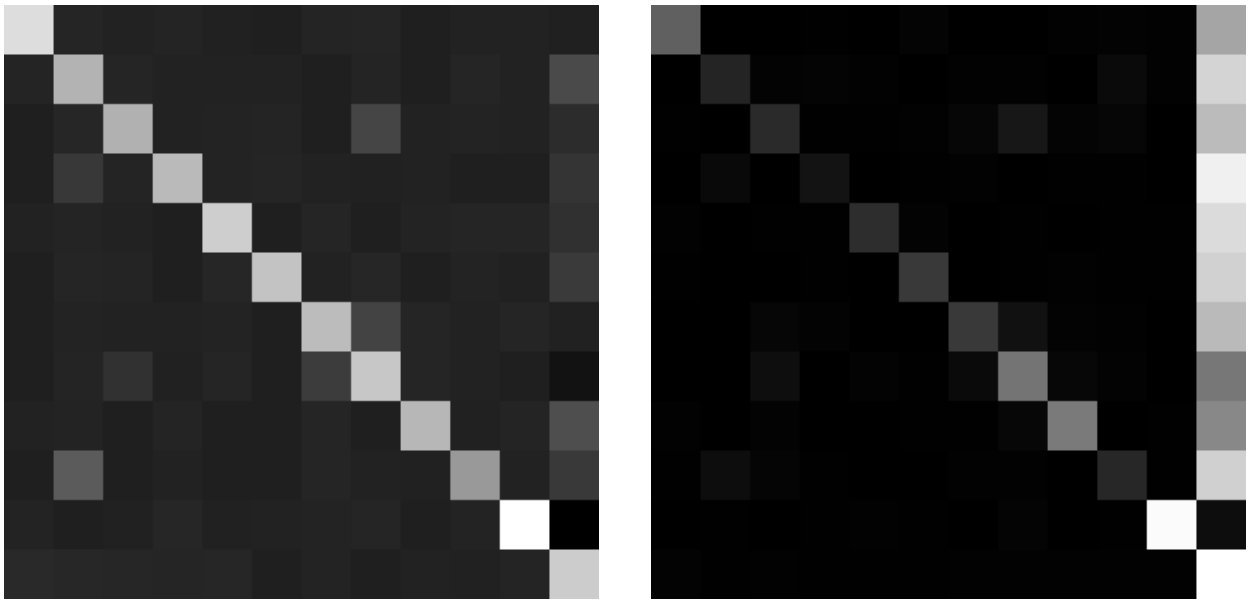
3. The LSTM cell we are using is BasicLSTMCell. The impacts of using different types of cells, for example ConvLSTMCell, will be included in our future work.

4. We also explored the impacts of super-parameters, for example the number of hidden units in LSTM network and different methods of feature extraction.

4. Experiments and Results

The original dataset contains multiple folders named after the word, and each folder contains multiple audio clips for roughly one second long respectively.

We pre-read all the audio files to extract the raw data, zero-pad them to the same length (16000 samples corresponds to 1 second of audio with sampling rate of 16000Hz) and pack the data from the same folder into a 2D array of [Batch x samples] and save it into .mat files.



(a) Weighted Loss Function

(b) Unweighted Loss Function

Figure 4: Confusion Matrix for Comparison Between Loss Functions

The dataset is separated into train, val and test, which has been pre-defined (chosen randomly by the contest organizer).

In terms of labels, we have in total 12 class labels: "yes", "no", "up", "down", "left", "right", "on", "off", "stop", "go", "slience", and "unknown". Each is assigned to an integer number from 0 to 11.

Since the entire data is roughly 2GB, relatively small, we load all the data into memory before training, and keep shuffling and sending batches to the model in the similar manner of tensorflow MNIST dataset.

For the results, we output both the accuracy for each class, that is the percentage of examples that are correctly classified as their corresponding words, and also a nicely-visualized confusion matrix. Confusion matrix makes it easy to see if the network is doing well for predicting the label since the brighter the cell it appears, the higher the accuracy it is, and we only care about the diagonal. We calculate multiclass accuracy, the average accuracy for all classes. The horizontal axis of the confusion matrix is the predicted label from class 0 to class 11, and the vertical axis is the truth label from class 0 to class 11.

4.1. Comparison Between Weighted and Unweighted Loss Functions

As we can see from the confusion matrix in Figure 4, unbalanced dataset benefits hugely from having a class weighted loss function. We chose FBanks without pre-emphasis as our feature extraction method, and set the number of hidden units as 128. Obviously, unweighted loss function makes much more mistakes than weighted loss function when predicting the class.

4.2. Comparison Among Different Feature Extraction Methods

As discussed before, we have brought forward four feature extraction methods: no extraction, STFT, FBanks, and MFCC. We also manipulated with pre-emphasis here. In specific, for each feature extraction method, we do two experiments, with and without pre-emphasis, to see if this strategy of balancing the energy of low versus high frequencies works well.

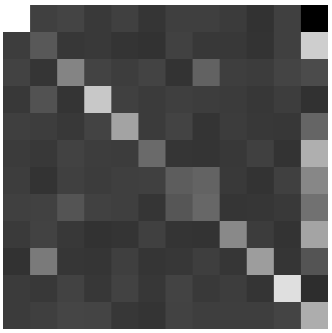
Based on the confusion matrix in Figure 5, pre-emphasis doesn't make a big difference for deep models and it may even reduce the accuracy by a little bit.

STFT and FBanks work almost equally well, or FBanks is slightly better. MFCC performs okay, probably because too much information is thrown away. No extraction gives the worst performance as expected since audio features are not that straight-forward for a network to learn.

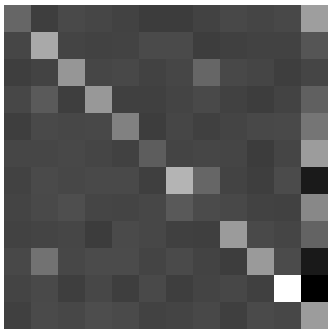
4.3. Comparison Among Different RNN Settings

According to the previous experiment, we set the feature extraction method to be filter banks without pre-emphasis. We only modify the number of hidden units to be 64, 128, and 256 in this case.

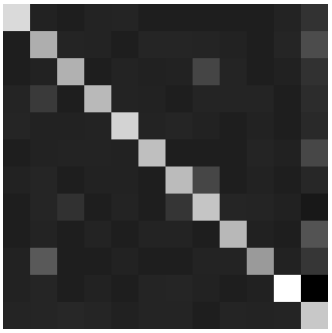
In Figure 6, the accuracy for 64 hidden units is a little bit lower than that for 128 hidden units, which makes senses since 64 may not be sufficient to learn the features. Yet increasing the number of hidden units won't help with the ac-



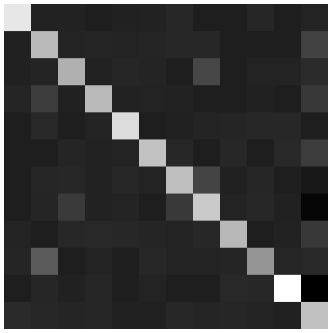
(a) No Extraction without Pre-emphasis



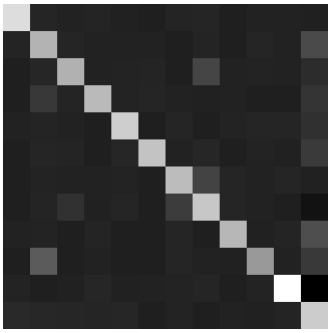
(b) No Extraction with Pre-emphasis



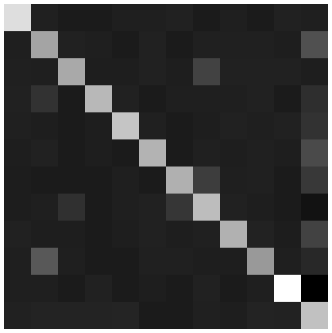
(c) STFT without Pre-emphasis



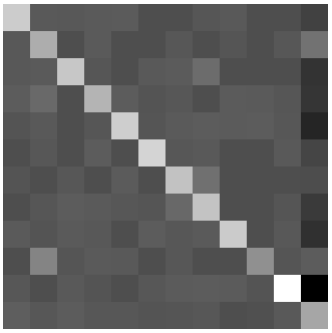
(d) STFT with Pre-emphasis



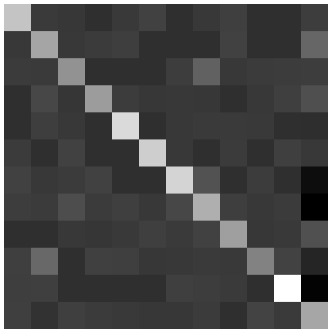
(e) FBanks without Pre-emphasis



(f) FBanks with Pre-emphasis

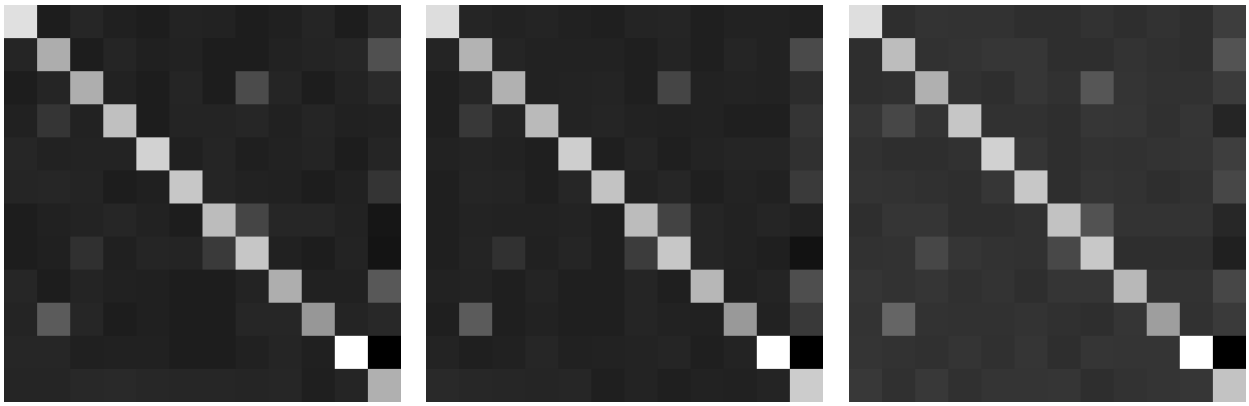


(g) MFCC without Pre-emphasis



(h) MFCC with Pre-emphasis

Figure 5: Confusion Matrix for Comparison Among Feature Extraction Methods



(a) 64 Hidden Units (b) 128 Hidden Units (c) 256 Hidden Units

Figure 6: Confusion Matrix for Comparison Among Hidden Unit Number

	Loss Function		Feature Extraction								Hidden Unit Number		
Accuracy (%)	Unweighted	Weighted	No Extraction		STFT		FBanks		MFCC		64 Hidden Units	128 Hidden Units	256 Hidden Units
			w/o pre	w/ pre	w/o pre	w/ pre	w/o pre	w/ pre	w/o pre	w/ pre			
class “yes”	35.156	84.551	78.172	20.242	83.038	85.251	84.551	84.862	71.389	62.503	81.214	84.551	82.73
class “no”	13.095	65.602	14.58	52.303	63.239	66.266	65.602	59.649	53.754	49.693	60.216	65.602	67.04
class “up”	15.441	64.777	31.355	43.112	64.534	62.088	64.777	61.882	68.363	41.732	60.442	64.777	61.247
class “down”	7.115	69.0	57.138	44.141	68.423	65.651	69.0	68.243	57.586	45.982	67.82	69.0	71.273
class “left”	16.479	78.0	43.238	33.075	79.567	80.449	78.0	74.224	72.315	72.02	75.316	78.0	76.783
class “right”	20.849	72.937	21.563	15.561	70.872	69.49	72.937	66.392	76.019	66.994	70.712	72.937	71.918
class “on”	20.732	69.945	16.44	57.469	69.224	67.976	69.945	64.959	65.806	69.133	66.271	69.945	69.74
class “off”	42.366	74.592	20.292	6.898	73.806	72.832	74.592	70.749	66.016	53.936	70.598	74.592	72.252
class “stop”	44.5778	67.539	33.393	45.49	67.705	65.184	67.539	65.584	70.871	47.156	60.397	67.539	64.79
class “go”	13.343	54.0	40.776	44.852	54.586	50.4	54.0	55.047	37.403	34.838	50.818	54.0	52.847
class “slience”	91.453	99.559	65.937	93.496	99.216	95.504	99.559	99.359	99.249	87.345	94.278	99.559	98.216
class “unknown”	93.088	76.713	47.07	46.001	74.838	69.118	76.713	72.202	49.494	50.14	61.341	76.713	69.961
Overall	34.558	73.101	39.163	41.887	72.42	70.851	73.101	70.263	65.772	56.789	68.285	73.101	71.567

Table 1: Per Class Accuracy Table

curacy as the performance from 256 hidden units are worse than that from 128 hidden units.

4.4. Discussion

Basically, the results are consistent with our discussions before as shown in Table 1. It is interesting yet conceivable that class “silence” always has nearly perfect accuracy. It makes senses since the “silence” audio is a plain audio clip without any features. The best mean accuracy we can achieve is 73.101% based on the settings below:

Feature Extraction Configurations:
>>> Method: FBank
>>> Sampling Rate: 16000 Hz
>>> Pre Emphasis Coefficient: 0
>>> Frame Length: 25 ms <-> 400 samples
>>> Overlapping Length: 15 ms <-> 240 samples
>>> FFT Length: 512
>>> Filter Banks Number: 40
>>> MFCC Coefficients Number: 12
>>> Mean Removal Normalization: True
>>> Output Data Type: <class ‘numpy.float32’>

RNN Model Configurations:
>>> Cell Type: BasicLSTMCell
>>> Hidden Unit Number: 128
>>> Learning Rate: 0.0005
>>> Step:1000000

The results are very promising yet the limitation of our settings is also obvious. The number of hidden units are tricky to decide and the research on how to choose the appropriate number of hidden units is little. On the other hand, the normal LSTM we are using might not be eligible to learn the features as the audio clip is too long for the LSTM to remember. We might want to try some more complicated architectures, for example bi-directional recurrent neural network [6]. There are also papers recently focusing

on the alleviation of feature extraction [8], which are worth trying. More significantly, the sound signal in the audio clip is also unbalanced-distributed. The word may appear very late in the clip, and may appear in the very beginning. In this case, audio temporal localization may be our next step since "slience" is easily distinguished.

5. Conclusion

In this paper, we proposed a algorithm to understand simple English voice commands from Tensorflow Speech Commands Dataset. We demonstrated that LSTM with 128 hidden units can better address this problem by using filter banks feature extraction method. We were also able to conquer the unbalance from the original dataset by adding class weights. Experimental results indicated promising performances in this task.

References

[1] A. Biem, E. Mcdermott, and S. Katagiri. A discriminative filter bank model for speech recognition. In *Proceedings of the IEEE, ICASSP-96*, pages 545–548, 1995.

[2] T. Ganchev, N. Fakotakis, and G. Kokkinakis. Comparative evaluation of various mfcc implementations on the speaker verification task. In *Proceedings of the SPECOM*, volume 1, pages 191–194, 2005.

[3] Z. C. Lipton, J. Berkowitz, and C. Elkan. A critical review of recurrent neural networks for sequence learning. *arXiv preprint arXiv:1506.00019*, 2015.

[4] L. Muda, M. Begam, and I. Elamvazuthi. Voice recognition algorithms using mel frequency cepstral coefficient (mfcc) and dynamic time warping (dtw) techniques. *arXiv preprint arXiv:1003.4083*, 2010.

[5] T. Nakatani, T. Yoshioka, K. Kinoshita, M. Miyoshi, and B.-H. Juang. Blind speech dereverberation with multi-channel linear prediction based on short time fourier transform representation. In *Acoustics, Speech and Signal Processing, 2008. ICASSP 2008. IEEE International Conference on*, pages 85–88. IEEE, 2008.

[6] J. Sotelo, S. Mehri, K. Kumar, J. F. Santos, K. Kastner, A. Courville, and Y. Bengio. Char2wav: End-to-end speech synthesis. 2017.

[7] P. Taylor. *Text-to-speech synthesis*. Cambridge university press, 2009.

[8] Y. Wang, R. Skerry-Ryan, D. Stanton, Y. Wu, R. J. Weiss, N. Jaitly, Z. Yang, Y. Xiao, Z. Chen, S. Bengio, et al. Tacotron: A fully end-to-end text-to-speech synthesis model. *arXiv preprint arXiv:1703.10135*, 2017.