TensorFlow Speech Recognition Challenge

ECSE523 Winter 2018 Final Project

Alex Yin [260502051], zixuan.yin@mail.mcigll.ca
Bonnie Hu [260556970], guanqing.hu@mail.mcgill.ca

***Abstract*— Simple speech command is a widely requested feature in many trending artificial intelligent applications. This report presents the implementation and testing of a Convolutional Neural Network (CNN) solution to classify simple spoken commands with tuning results of various speech preprocessing techniques. The implementation source code was released on github: https://github.com/alex-yin/speech.**

I. Introduction

The objective of this project is to devise a machine learning algorithm to classify simple spoken commands (e.g. yes, no, left, right, etc.). The training set, including labels, as well as a test set were downloaded from Kaggle's TensorFlow Speech Recognition Challenge page : https://www.kaggle.com/c/tensorflow-speech-recognition-challenge.

The rest of the current paper is structured as follows. In Section II, data preprocessing and feature design are presented including dataset separation, spectrogram generation, normalization, and silence removal. In Section III, descriptions of CNN structure are presented first, followed by a summary of optimized parameters, hardware requirements, and time and computing efficiency. Section IV presents validation and testing results, as well as some critical parameter tuning results. Section V completes the paper with discussion and conclusion.

II. Problem Representation

A. Data Set

TensorFlow challenge provides 65,000 one-second utterances of 30 words. For fast prototyping, this project uses 10 out of the 30 words including 'yes', 'no', 'up', 'down', 'left', 'right', 'on', 'off', 'go', and 'stop', with totally 23,677 utterances. 80% of the utterances were used for training and validation and the rest 20% were used for testing. Detailed separation and number of utterances are presented in Table II.

TABLE I: Dataset Summary

	Percentage (%)	Number of utterance
Training	76	17,994
Validation	4	948
Testing	20	4,735
Total		23,677

B. Feature Design and Selection

1) Spectrogram and window size: The original dataset contains audio files in WAV format which is incompatible with CNN. Spectrogram is a way to convert audio wave into image. Since the audios used in this project are all 1-second long, their spectrograms are images with fixed width and height, which is suitable to be used as CNN inputs.

The choice of window size to generate the spectrogram is not obvious and requires tuning experiments to optimize. Detailed tuning results are presented in Section IV. The model achieves best performance when window size equals to 20ms. Knowing the sampling rate is 16,000, each segment in the spectrogram represents the average frequency features of 320 consecutive samples. Spectrograms are also processed with log scale in order to enhance the time and frequency dynamics. A sound wave and its spectrogram are shown in Figure 1.

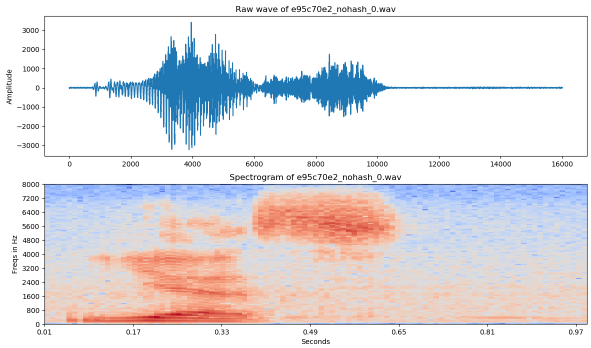

Fig. 1: Audio wave and spectrogram of "Yes".

2) Normalization: In practice, CNN's performance tends to be improved with normalization on the input images. Experiments were used to investigate the performance difference between using the normalized spectrograms and the original ones, which shows a 1.5% increase of the test accuracy when using normalization.

3) Voice activity detector (VAD): One of the more difficult problem in speech recognition is speech segmentation: detecting where the word and the silence are actually at. Although the speech commands in this project are only 1-second long and contain only 1 word, the silence before and after each word is random and may cause performance difference. VAD is a simple algorithm using speech energy

and total energy ratio to detect if a sequence of samples are speech or silence. Based on experimental results, this project did not use VAD due to performance degradation.

III. Implementation

A. Network structure

The CNN model used in this project consists of 4 Convolution (Conv) layers and 3 fully-connected (FC) layers. The structure is similar to LeNet [3] CNN model used for handwritten digit recognition. We apply batch normalization [4] and rectified linear units (ReLU) [5] activation at each Conv layer. Max pooling was also applied after each Conv layer to reduce output dimension. Summary of the architecture is shown in the table below.

TABLE II: Dataset Summary

Layer type	Filter Shape	Input Size
Conv1	3x3x1x16	100x161x1
MaxPooling	2x2	100x161x16
Conv2	3x3x16x16	50x80x16
MaxPooling	2x2	50x80x16
Conv3	3x3x16x32	25x40x16
MaxPooling	2x2	25x40x32
Conv4	3x3x16x32	12x20x32
MaxPooling	2x2	12x20x32
FC1	128	1920
FC2	64	128
Output FC	10 or 15	64

B. Optimal parameter summary

It is common practice to decay learning rate while training to increase the probability to converge to local minimum. The proposed network uses Adam optimizer [1] which decays the learning rate based on adaptive estimates of lower-order moments. Adam optimizer helps the training process achieve good results in shorter time.

In general, using smaller batch size leads to higher test accuracy and less test variance with the cost of training time. Experiments have been done on batch size. Some of the results are presented in Section IV. The proposed network uses a relatively small batch size at 5 with 50 epochs, which achieves 94.8% accuracy.

Smaller batch size is good for the early stage convergence, but might be too aggressive for late stages. [2] suggested to increase the batch size during training to achieve same learning curve as learning rate decay, but in a faster pace. Therefore, after the small-batch-size training, the proposed network increased the batch size to 25 for another 50 epochs, which improved the accuracy further to 96%.

C. Hardware requirements and running time

The proposed algorithm were tested with GTX1070 GPU using Keras library. The best practice model of this project took 1 hour to train.

The computation power required to run this project is fairly low. When the batch size is 100, it only takes 9s for each epoch, while smaller batch size at 5 leads to longer running time at 50s/epoch. Training on CPU is feasible with large batch size (e.g. >100), which requires approximately 60 times longer.

IV. Results

The current section pertains to detailed analysis of the achieved results.

A. Main results

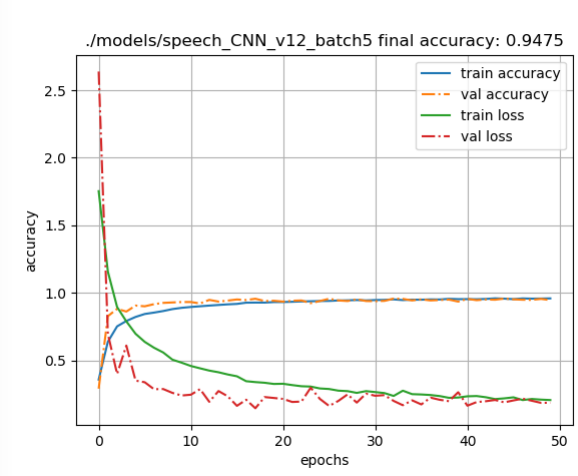

Fig. 2: Training result with batch size=5, epoch=50

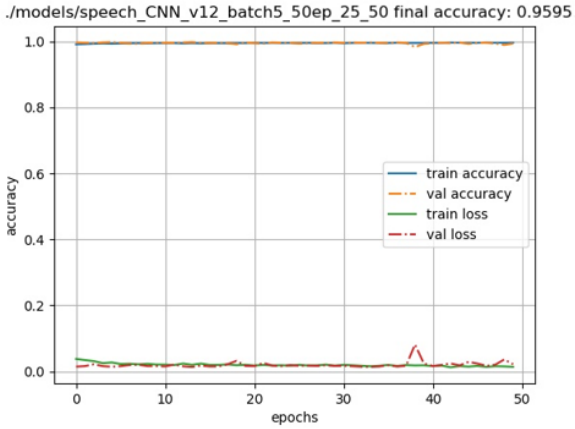

Fig. 3: Training result after increasing batch size

Using the implementation proposed in Section III, the training results are shown in Figure 2 and 3. It achieves 94.8% accuracy at the first training stage with small batch size, then the accuracy further improved to 96% with higher batch size. The loss of validation is constantly below the training loss, demonstrating balanced convergence.

B. Results on batch size

Figure 4 is the training history with batch size equal to 25, in comparison with the training results in Figure 2. Smaller batch size leads to less accuracy difference between the training and validation set, conveying better generalization which leads to better test accuracy.

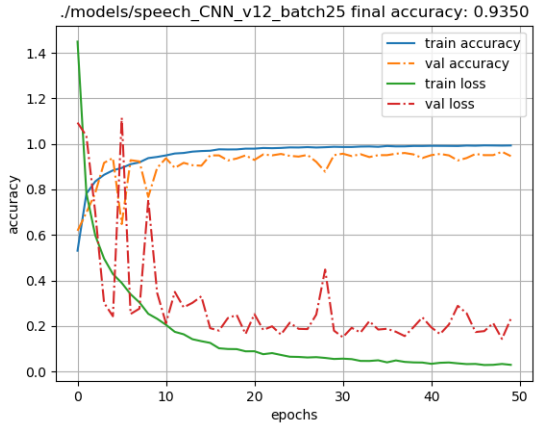

Fig. 4: Training result with batch size=25, epoch=50

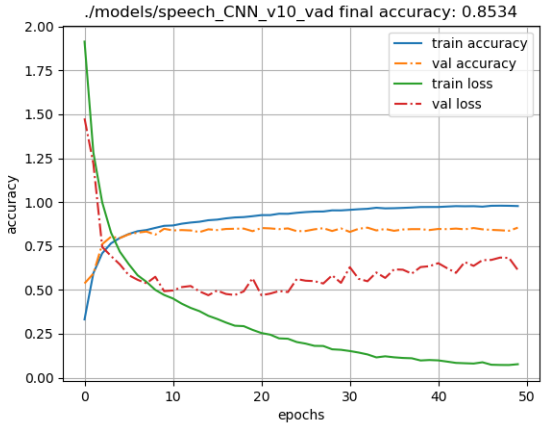

Fig. 6: Training result with VAD

C. Results on window size

When generating spectrogram from audio file, the window-size parameter introduces a trade-off between time resolution and frequency resolution. Larger window size preserves better frequency resolution of the speech, but decreases the resolution in time dimension, and vise versa. Experiments have been done using a sequence of different window sizes ranging from 15ms to 40ms in order to optimize the spectrogram for the network. Experiment results can be seen in Figure 5 which suggests using window size at 20ms for performance peak.

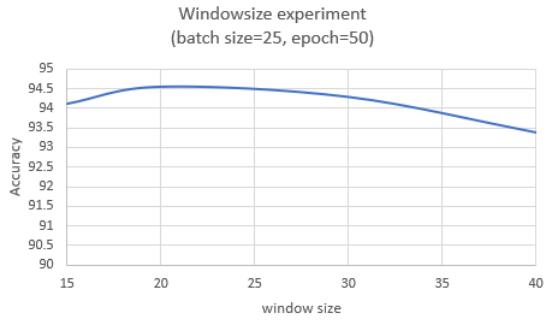

Fig. 5: Window size results

D. Results on VAD

Silence removal could increase the performance of a speech model in the way that same vocabularies become more consistent on the time line, therefore, easier for the model to catch their common features. This idea did not work as expected in this project. The training results were obtained with same settings as the results in Figure 4 except with silence removed audio input. The speech model trained with silence removed audios started to have overfitting issue after 20 epochs with validation accuracy oscillates around 90%. The model did not manage to generalize to new data, either, with test accuracy at 85%.

V. Discussion

There are a few methods and techniques we've tried to improve the accuracy of the model, such as hyper parameter tuning, spectrogram normalization and silence removal. While there are many other aspects we can adjust to improve the accuracy, but we focused on areas that doesn't require exhaustive searching or high computation power. For instance, CNN architecture search could be very useful to find accurate and efficient models, but due to the enormous design space and long training time for each network such process could take weeks. Overall, these optimization are usually conducted to improve the accuracy by a small fraction from the baseline. It might not seem much in practice, but such is known as the bottleneck of CNN models and a lot of researches were conducted to get those percentage in various ways.

On the design of the CNN model, we applied techniques that are known to improve the accuracy from past researches. For instance, batch normalization [4] and dropout [6] were used to speed up the training process and prevent network from overfitting respectively. On the other hand, we've also hand tuned some design parameters whose effect are unknown and problem-dependent. Reducing the batchsize in early stage and increasing the batchsize in late stage helped improve the testing accuracy from 94.8% to 96.0%. Additionally, a callback function was used to discard the overfitting epochs when validation accuracy starts to decrease. It helped avoid overfitting model, which can decrease the model's performance by 1-2%. We've also tuned the decay rate for learning rate, however the results showed no clear trend or strong indication that the parameter has significant impact on the accuracy.

In terms of audio processing, we explored spectrogram normalization and various VAD algorithms. The spectrogram normalization is particularly helpful for improving the test score. Since each audio recorded has different volume intensity, the range of spectral intensity for two audio could be on 2 different level, even from the same class. By normalizing all values proportionally to a range of [0, 1], we can not

only remove the effect of volume intensity, but also retain the formants transition and other dynamic changes in frequency domain. Moreover, many machine learning researchers have indicated that small input values are favorable for neural networks. Normalizing image intensity range to [0, 1] or [-1, 1] is a common practice in image recognition. From experiment results, we've seen a improvement of 1.5% from spectrogram normalization.

For silence removal, we've tried several different VAD algorithms and experimented the effect. From the results, we conclude that simple silence removal algorithm using VAD doesn't help the accuracy, but on the contrary decrease the accuracy by 8%. The problem with silence removal on this dataset is that the length of silence in each audio has a big range and high variance. Therefore, simple silence removal algorithm simply can not adapt to the various length of silence, resulting in voiced part being cut short or a lot of silence remain from sample to sample. Moreover, each selected voice spectrogram is scaled to the same size (for CNN), which introduce distortion in frequency domain especially for shorter voiced audios. Given these conditions, simple silence removal techniques are unlikely to improve the accuracy. It's worth to note that if the CNN model is trained on audios with silence removal, then input test audios must also be processed prior to CNN prediction. By not using silence removal, the framework is simpler and has higher throughput.

Other than the simple 10 command word dataset, we've also tried to extend number of classes to 15 by adding audio of numbers from one to five. The accuracy on the 15 words dataset is 93.56% (50 epochs).

VI. Conclusion

In this project, we've implemented a CNN-based short voice command recognition framework based on the TensorFlow speech recognition dataset. We've tested and optimized our design on the CNN and preprocessing procedures to improve the accuracy of the system. We've found various methods and techniques that could improve on baseline accuracy. Finally, we achieved 96.0% accuracy on 10 command words dataset and 93.5% on 15 command words dataset. Given the resource and time, we believe that some other optimization can be applied to further improve the accuracy and meet the commercial standard accuracy.

References

[1] Diederik P. Kingma and Jimmy Ba, Adam: A Method for Stochastic Optimization, CoRR, vol. abs/1412.6980, 2014.

[2] Samuel L. Smith and Pieter-Jan Kindermans and Quoc V. Le, Don't Decay the Learning Rate, Increase the Batch Size, CoRR, vol. abs/1711.00489, 2017

[3] Y. LeCun, L. Bottou, Y. Bengio and P. Haffner. Gradient-based learning applied to document recognition. Proceedings of the IEEE, November 1998.

[4] Sergey Ioffe, Christian Szegedy. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. CoRR, abs/1502.03167, 2015.

[5] Nair, Vinod and Hinton, Geoffrey E. Rectified Linear Units Improve Restricted Boltzmann Machines. ICML 2010.

[6] Nitish S., Geoffrey H., Alex K., Ilya S. and Ruslan S. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. Journal of Machine Learning Research 2014.