Deep Learning Based Approach to Speech Recognition to Understand Simple Commands

Cheng Wang (cw17), Jiayu Li (jl145)
Final Project, Deep learning systems E533, Spring 2019

Abstract

There are many different applications and services for human speech recognition like Siri, Alexa. Many voice recognition datasets require preprocessing before a neural network model can be built on them. In the project, we combined data compression and augmentation with multiple deep learning based model to understand simple spoken commands based on one-second long utterance. The goal is to classify each audio input as one of eleven labels, including "yes, no, up, down, left, right, on, off, stop, go, unknown." We implement data compression (sample only 50% of original data) and data augmentation (add noise) technique followed by applying Convolutional Neural Network (CNN) and Long Short Term Memory (LSTM) models to train the data for better applicability of real world audio data. The classification performance is evaluated using multiclass accuracy, which is the average number of observations with the correct label. The final prediction accuracy on test data set (2823 audio files, file data size down sampled to 50%) by LSTM model is 83.8% (noise-free), 82.5% (noise-added), showing our down sampling with LSTM approach captures the main features to predict the command in the audio file using.

1. Introduction and related work

Voice assistants, such as Alexa, embedded with speech recognition technology have been emerging to perform a variety of actions after hearing a wake word or command. They can turn on lights, answer questions, play music, place online orders, etc. But, for independent makers and entrepreneurs, it's hard to build a simple speech detector using free, open data and code. Many voice recognition datasets require preprocessing before a neural network model can be built on them. Recently, TensorFlow released the Speech Commands Datasets, including 65,000 one-second long utterances of 30 short words, by thousands of different people. It is also listed on Kaggle, named "TensorFlow Speech Recognition Challenge". [1,2] In addition, many speech recognition data set have been published. Mozilla's Common Voice dataset [3] has over 500 hours from 20,000 different people. LibriSpeech [4] is a collection of 1,000 hours of read English speech. CHiME-5 [5] has 50 hours of speech recorded in people's homes,they provides large-scale corpus of real multi-speaker conversational speech recorded via commercially available multi-microphone hardware in multiple homes.

Traditional speech recognition algorithms are based on Hidden Markov models (HMS). Neural networks make fewer assumptions about feature statistical properties than HMMs and have several qualities making them attractive recognition models for speech recognition. In

2013, Alex Graves et al. proposed RNN for speech recognition [6], this method enables neural networks to perform continuous recognition.

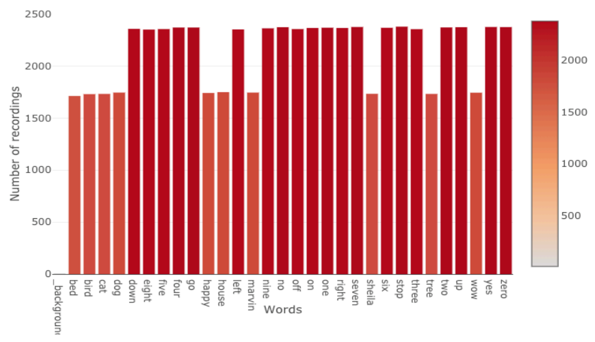
In the project, we built multiple deep learning based model to understand simple spoken commands based on one-second long utterance. The goal is to classify each audio input as one of eleven labels, including "yes, no, up, down, left, right, on, off, stop, go, silence, unknown." We implement basic neural networks algorithm to train the data followed by applying Convolutional Neural Network (CNN) and Long Short Term Memory (LSTM) models to optimize the performance.

2. Data Preprocessing

The raw data is downloaded from Kaggle challenge website, containing both training and testing files. Generally, the training data contains a few informational files and a folder of audio files. The audio folder contains subfolders with one-second clips of voice commands, with the folder name being the label of the audio clip. The test data contains an audio folder with 150,000+ files in the format clip_000044442.wav. The task is to predict the correct label.

3. Exploratory Data Analysis (EDA)

The raw data set contains 30 commands, such as "up", "yes", including 64,727 audio files. The number of audio files in each command is shown in Figure 3.1. Each command category contains 1600~2200 files, showing the data is well balanced.



Here we select all the commands files in 10 target categories, i.e., 'yes', 'no', 'up', 'down', 'left', 'right', 'on', 'off', 'stop' and 'go'. Then we randomly select 2100 audios from other categories as 'unknown' audio files. We split the curated data set into training set (20391 files, 80%), validation set (2834 files, 10%) and test set (2823 files, 10%).

Figure 3.1 Number of recordings in given label.

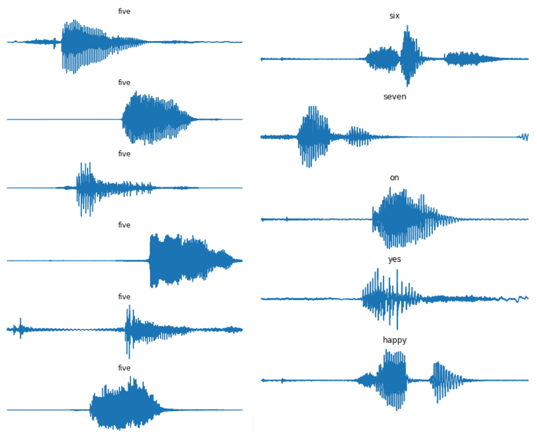


Figure 3.2 Waveforms representation of audio files: waveforms within the same word (left), and waveforms across different words (right).

Figure 3.2 shows the waveforms representation of different command audio files. It shows that the waveforms across different words are very different. In addition, the waveforms across the same word have similar waveforms, but have different amplitude profiles.

The original data is sampled with 16000 Hz. However, when down sampling the data to 8000 Hz, the data still contain the main information (major peaks in frequency domain). Therefore we reduce the dataset size matrix to 50%.

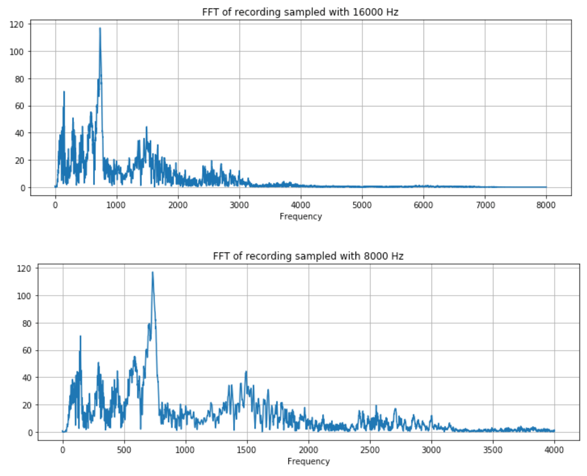


Figure 3.3 FFT of recording sampled with 16000 and 8000 Hz.

4. Data Augmentation

Data augmentation is the process by which we create new synthetic training samples by adding small perturbations on our initial training set. The objective is to make our model invariant to those perturbations and enhance its ability to generalize. The methods we use to improve voice data include: add white noise and add real environmental noise. In this project, we introduced 1200 different environmental noises used in Assignment 3. We believe this approach gives the model a certain ability to combat noise. In the experimental part, we will compare the actual effects of data augmentation.

4.1 White noise

White noise is a random signal having equal intensity at different frequencies, giving it a constant power spectral density. In discrete time, white noise is a discrete signal whose samples are regarded as a sequence of serially uncorrelated random variables with zero mean and finite variance. We use the np.random.randn function to generate a random sequence and add it with the voice signal. Considering that white noise should not overly affect the original signal, we give white noise a small coefficient (such as 0.05).

4.2 Environmental noise

In the previous assignment 3, we used 1200 environmental noise to train the model of audio denoising. We believe that adding environmental noise to the sound signal can make the model have the ability to resist environmental noise, making the model more versatile. In actual training, we will add a random environmental noise to each voice sample. Notice that the length of the environmental noise actually exceeds 1 second. We will randomly capture 1 second of audio from it, which makes it difficult to have two identical noises during training.

5. Deep Learning Based Model

Here we built three neural network models to classify each sound into one of eleven commands in the data set.

5.1 Fully connected neural network (FNN)

We use fully connected neural network as the baseline model. The fully connected neural network consists of 5 hidden layers and each hidden layer consists of 1024 units. ReLU is used as activation function in each hidden layer. The output layer is a prediction label.

5.2 Convolutional neural network (CNN)

Here we implement a simple 2D Convolutional Neural Network (CNN) model. Each audio file is regarded as a 2D image and the CNN model contains one convolution layer and pooling layer followed by dropout and dense processing.

5.3 Recurrent Neural Network (RNN)

The audio is time dependent domain data and thus has strong time dependency. Recurrent Neural Network is suitable to capture the time dependence feature in the data. Long Short Term Memory (LSTM) network model is implemented here to classify each audio file into different commands.

5.4 Loss function and Optimizer function

The loss function, categorical crossentropy, is used for the single label categorization.

$$L(y,\hat{y}) = -\sum_{j=0}^{M}\sum_{i=0}^{N}(y_{ij} * log(\hat{y}_{ij}))$$

Categorical crossentropy, where ŷ is the predicted value. Categorical crossentropy will compare the distribution of the predictions (the activations in the output layer, one for each class) with the true distribution, where the probability of the true class is set to 1 and 0 for the other classes. To put it in a different way, the true class is represented as a one-hot encoded vector, and the closer the model's outputs are to that vector, the lower the loss. The adaptive learning rate optimization algorithm "Adam" is used as the optimizer function during training neural networks.

6. Results and Discussion

6.1 Optimization of training epochs

In the three models, we perform 40 epochs across all the training set (20391 files). The improvement of prediction accuracy with increasing epoch is shown in Figure 6.1, it is found that after 15 epochs, the model becomes stable with limited improvement of prediction accuracy on training set and no improvement of prediction accuracy validation set. Therefore, we set the training epochs as 15.

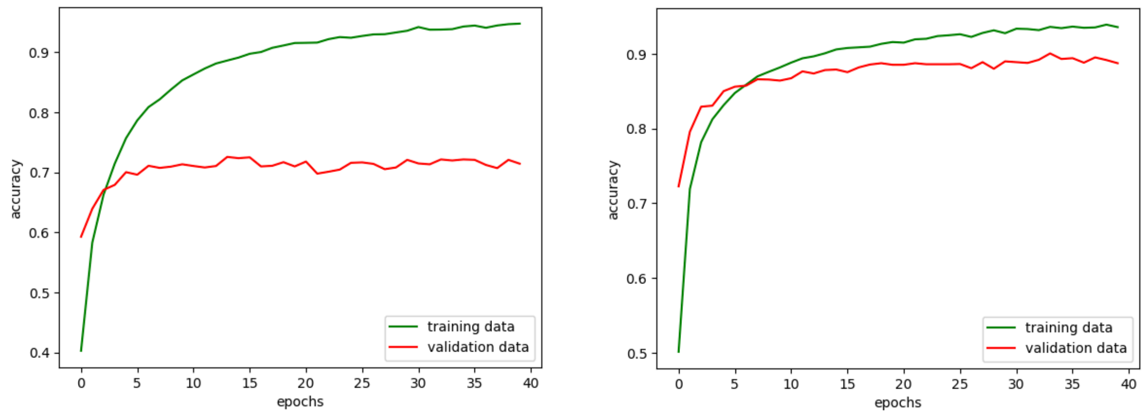


Figure 6.1 Improvement of prediction accuracy with increasing epoch. (Left: CNN model, Right: LSTM model)

6.2 Prediction accuracy on down sampled data (50%) and raw data

Table 1. Classification accuracy after 15 epochs training (without data augmentation)

Data	Model	Training set	Validation set	Test set
Raw data	**FNN**	**95.3%**	**73.7%**	**70.9%**
Down sampled data (50%)	**FNN**	**97.6%**	**71.0%**	**71.3%**
Raw data	**CNN**	**97.4%**	**74.2%**	**74.1%**
Down sampled data (50%)	**CNN**	**98.5%**	**65.0%**	**64.7%**
Raw data	**LSTM**	**94.7%**	**87.5%**	**86.9%**
Down sampled data (50%)	**LSTM**	**95.1%**	**84.3%**	**83.8%**

After down sampling the data to 50% as described in section 3, we re-run the three deep learning models to the data set, it is found that the prediction accuracy on the data set fluctuates very little. The LSTM model remains the highest prediction accuracy among the three models for down sampled data, showing the major features remain in the down sampled data and are "memorized" by LSTM model. Therefore, we apply data augmentation technique to add noise on the 50% down sampled data.

6.3 Prediction accuracy on noise-free data and noise-added data

Here we summarize the classification prediction accuracy after 15 epochs training noise-free data and noise-added data in tables below. Table 2 shows the prediction accuracy on 50% down sampled data with augmenting only training data by adding random noise as described in section 4. Table 3 shows the prediction accuracy on 50% down sampled data with augmenting all data.

Table 2. Classification accuracy after 15 epochs training (with data augmentation on training data only)

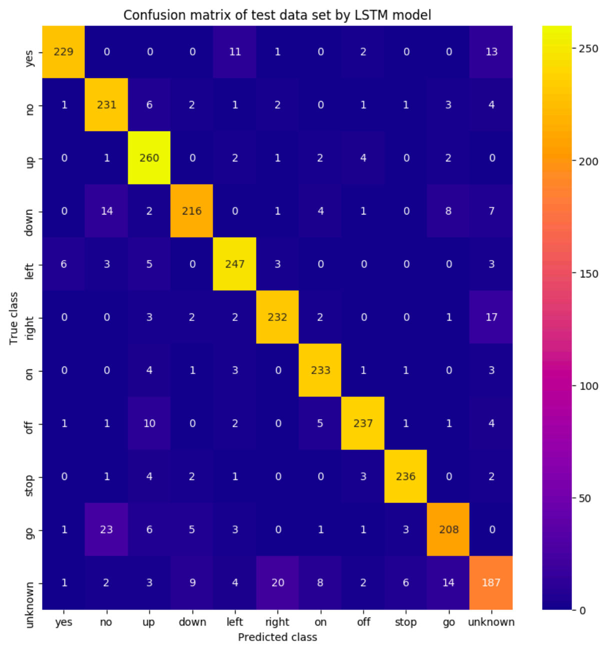
Model	Training set	Validation set	Test set
FNN	97.7%	71.0%	70.7%
CNN	93.5%	64.5%	62.9%
LSTM	94.3%	84.1%	84.0%

Table 3. Classification accuracy after 15 epochs training (with data augmentation on all data)

Model	Training set	Validation set	Test set
FNN	97.6%	69.5%	69.7%
CNN	94.4%	62.4%	61.7%
LSTM	93.9%	84.3%	82.5%

Based the prediction accuracy metric in the tables, it is found that LSTM performs best among the three models, which achieves 83.8%, 84.0%, 82.5% across three data conditions. The classification accuracy reduces when more noise added to the audio. This is because the model captures the noise to the model weights during training the model. Since the noise is randomly added to training data and test data, the prediction accuracy goes down when there is new noise that is not captured in the training process.

Figure 6.2 shows the grid of confusion matrix of test data set by LSTM model. It is found that the major files of 10 commands except “unknown” are classified accurately while the “unknown” command has many false positives. Since “unknown” files are curated from different categories of commands that may not be covered by training data, the model sometime does not predict the audio commands very well.



	yes	no	up	down	left	right	on	off	stop	go	unknown
yes	229	0	0	0	11	1	0	2	0	0	13
no	1	231	6	2	1	2	0	1	1	3	4
up	0	1	260	0	2	1	2	4	0	2	0
down	0	14	2	216	0	1	4	1	0	8	7
left	6	3	5	0	247	3	0	0	0	0	3
right	0	0	3	2	2	232	2	0	0	1	17
on	0	0	4	1	3	0	233	1	1	0	3
off	1	1	10	0	2	0	5	237	1	1	4
stop	0	1	4	2	1	0	0	3	236	0	2
go	1	23	6	5	3	0	1	1	3	208	0
unknown	1	2	3	9	4	20	8	2	6	14	187

Figure 6.2 Confusion matrix of test data set by LSTM model (without data augmentation).

7. References:

[1] TensorFlow Speech Recognition Challenge, https://www.kaggle.com/c/tensorflow-speech-recognition-challenge

[2] Google AI, "Launching the Speech Commands Dataset", https://ai.googleblog.com/2017/08/launching-speech-commands-dataset.html

[3] https://voice.mozilla.org/en

[4]. Panayotov, G. Chen, D. Povey, and S. Khudanpur, "Librispeech: an ASR corpus based on publicdomain audio books," inProceedings of the International Conference on Acoustics, Speech and SignalProcessing (ICASSP). IEEE, 2015.

[5] (2018) The 5th chime speech separation and recognition challenge. http://spandh.dcs.shef.ac.uk/chime_challenge/data.htm

[6] https://arxiv.org/pdf/1303.5778.pdf