MACHINE LEARNING ENGINEER NANODEGREE

CAPSTONE PROJECT

Geoffrey Hung

Definition

Project Overview

Speech commands is becoming more common in smart home devices, for example, Amazon Echo[1], Google Home[2] and Apple Sira[3]. Speech command detection technology play a vital role here to receive and understand the commands from the users.

In this project, I used the speech commands dataset provided by Google in a Kaggle competition[3] to build a classifier model by deep learning to recognize the simple spoken commands such as “yes”, “no”, “up”, “down”. By improving the accuracy of the model, these types of technology can help IoT products to “listen”.

Project Statement

The problem can be reduced to multi-class classification, to train a classifier and classify audio commands to different class. The tasks involved are the following:

1. Download and preprocess the audio data
2. Split data into train-valid-test set and train classifier
3. Apply the trained model to classify the testing data

The final model is expected to recognize the audio data that isn't not from training set and output the predicted command. The model candidates can be range from comparing the correlation to deep learning (CNN or Sequence model). Former requires many human designed features for every command so we will take the deep learning approach to create features automatically. This way we don't need to consider every scenario and the solution can be trained to even more classes without extra efforts in feature engineering.

Metrics

Accuracy is a common metric for classification; it takes account for true positives and true negatives with equal weights.

$Accuracy$ = *(true positives + true negatives)/datasize*

This metric is used when evaluating the classifier, true positives result in classifying the spoken commands correctly, true negatives result in classifying the commands wrong. It works well for balanced dataset which we will explore in next part. Otherwise, we may also need F-score to adjust for imbalanced data.

Analysis

Data Exploration

Audio dataset[(4)] has thousands of samples for each spoken commands, they are 1 second long and collected from different settings (male, female, different background sounds) by Google; Data is simplified and only part of the data is used because original Kaggle dataset doesn't provide the labels for test data, we use training data to produce both training and testing data in this project. Properties of audio dataset will be discussed in later session.

We focus on the subset of dataset of this Kaggle competition (yes, no, up, down, left, right, on, off, stop, go commands) and also use them as training, validation and testing.

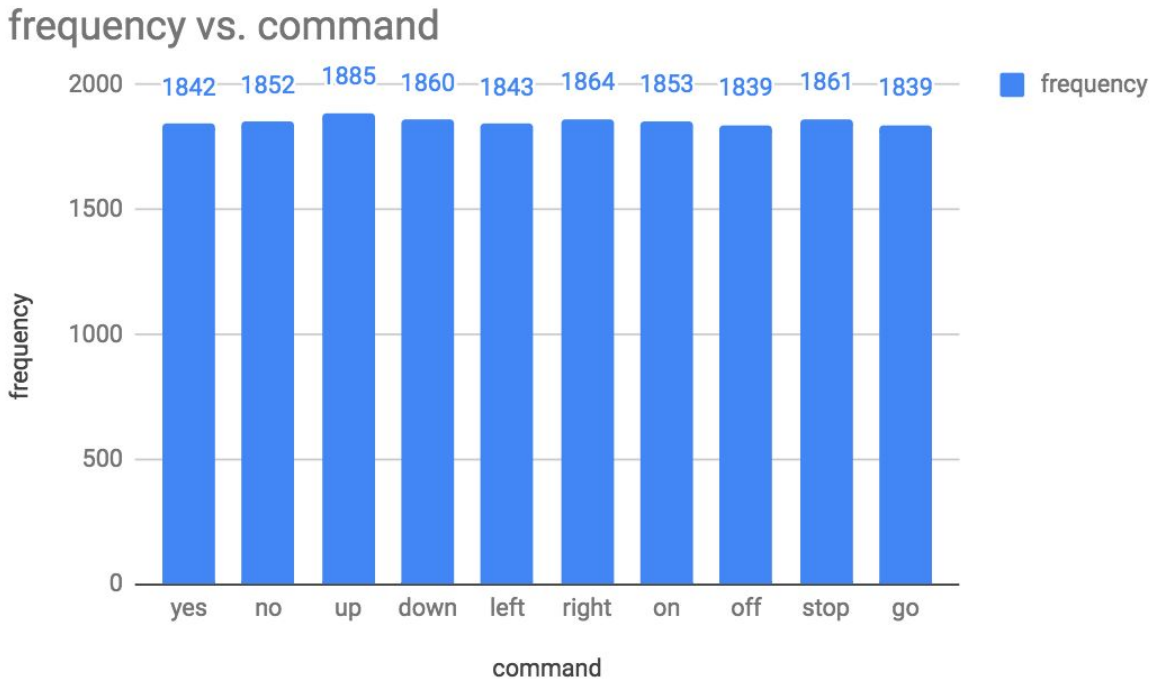
The project is structured by different folders by functions:

root -- data [train, valid, test]
-- model [different h5 checkpoint/trained models]
-- notebooks [exploration / model notebooks]
-- result [export csv]

Exploratory Visualization

The plot below shows the distribution of different spoken commands, this is helpful to evaluate how balance the data is, this information can help to judge whether we need to use different metrics other than accuracy to reflect the potential data imbalance or apply some data augmentation techniques to increase and balance the dataset.

Fig 1. The distribution of different commands



From this distribution (see Figure 1), we can see the data size of each category is approximately the same, we don't need to consider the data skewness for the moment and accuracy as metric will be good enough for our case.

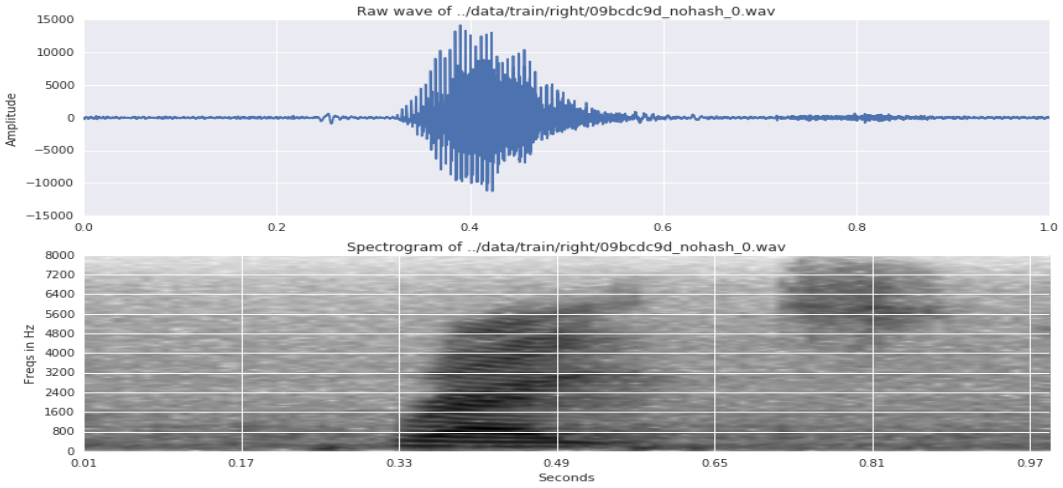
Next, although we humans can hear and recognize the commands, we do need to represent the audio data in the objective way such that computer can also handle it. We borrow python package scipy to convert the audio signal to spectrogram[5] and show some examples below.

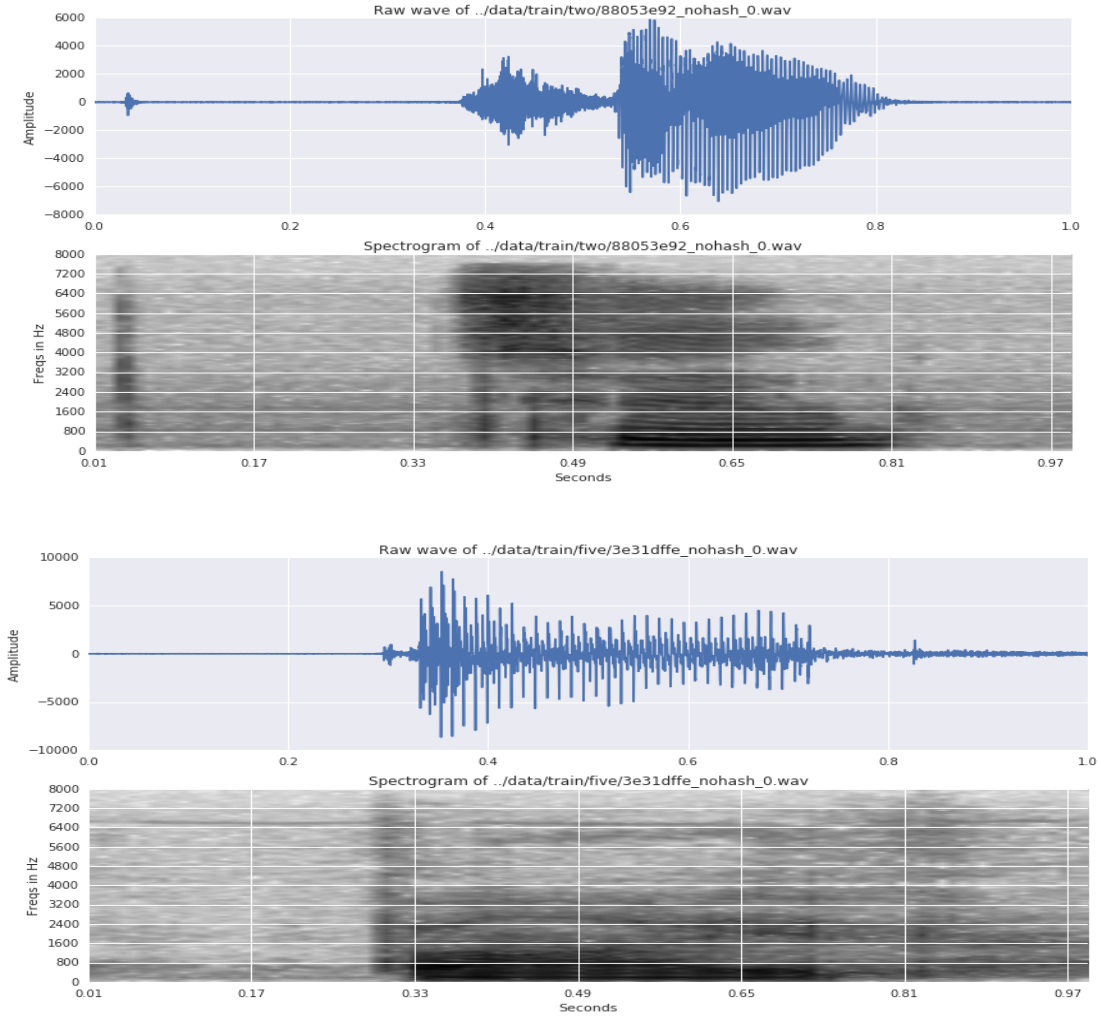
During the exploration, we found that the data comes from different people (eg: male, female, children) and they speak the commands under different background noise (eg: silence, home, office), we need the model to really learn the main patterns other than overfitting to noise

Frequencies of audio data is between 0 and 8000 Hz according to Nyquist theorem (the principles to allow us to digitize analog signal (see Figure 2), We have around 160 features for each frame and one frame corresponds to 50 Hz. To reduce the size of each sample, we apply resampling technique (change sample rate to 8000) to discard information that is not important in terms of audio (means we can still hear it well)

Regardless of the horizontal axis (which represents the time horizon, since every spoken command doesn't last for one whole second, we can see different starting points), we see different commands do have different signal shapes and spectrogram. In last 2 graphs below (see figure 2), it seems there are some noises, so the challenge to us isn't noise existence but can we still have a machine learning model to recognize the key patterns from the noises.

Fig 2. signal and spectrogram of some spoken commands to visualise how raw wave is converted to spectrogram, spectrogram represents the intensity and darker means dense





Algorithm and Techniques

In above visualization, we can reduce the problem to recognize the patterns in spectrogram which are generated from our audio data. The classifier we will apply in this case is Convolutional Neural Network, which is state-of-art algorithm for most image classification task and becomes the standard solution of visual area since 2012[(6)]. Regardless of a large amount of training data are needed compared with hand-designed algorithms, it results in more generalised solution and saves the efforts to design features for each application.

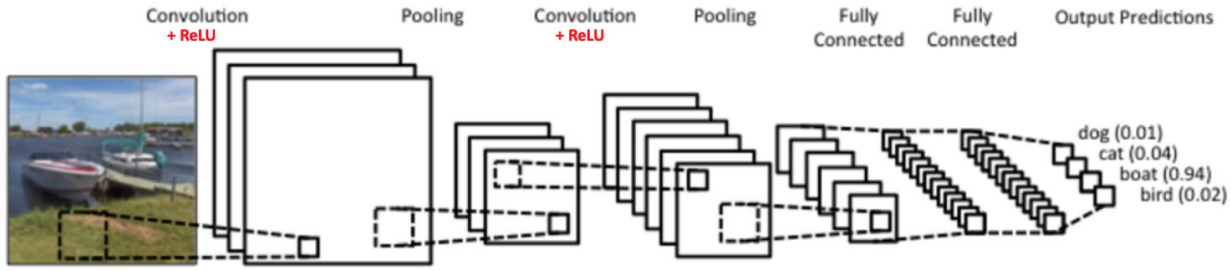
Other options include traditional way to capture information for each audio command by designing the features from us but this approach is too time consuming and not scalable (means we need to design for every case). The result isn't even better than deep learning approach as well, it can be considered if we have less data. Sequence model in deep learning is also an option but since we have spectrogram images, this can be solved by existing mature models in CNN.

The deep learning model we will use for spectrogram images is CNN, a subset of neural network with convolution layers that helps to summarize pixel nearby (means computer don't look at one pixel but also nearby pixels to make prediction) [see figure 3].

Convolution layers projects the original data to many layers (means many parameters) and then apply non-linear transformation (activation function, eg: ReLU, tanh) to output of each layer. To reduce the parameters size, we use pooling to group the nearby data on convolution layers and then feed to next layers. At last, fully connect the layers to few prediction class.

At output prediction part, it will compute loss for every epoch, by backpropagation algorithm, it feeds the information back to each layer to adjust the weight to optimize the model. The process will repeat many times (epoches) and hopefully the model will converge at certain point.

Fig. 3 example of how image information pass via many layers to prediction



The following key parameters can be tuned to optimize the classifiers:

Training parameters:

-Epochs (How many epochs for the training?)
-Batch size (Data size for one batch?)
-Learning rate (How fast to learn?)
-Optimization algorithms (momentum / RMS to include previous steps?)
-Dropout rate (percentage of weight that set to zero randomly to avoid overfitting)

Neural network architecture:

-Number of layers
-Layer types (Convolutional, fully-connected, pooling etc)

During model training, training and validation sets are loaded into RAM and use GPU to process using mini-batch gradient descent. After the training, we can use the fitted models to do prediction, continued training or fine tuning depends on the needs.

Benchmark

Since there are 10 classes in total, if we guess randomly, we will get 10% accuracy as the baseline, the accuracy of model has to be bigger than 10% to be considered as valid.

From Google's tutorial[7], although the data used are not exact match, they achieve around 85% accuracy for their advanced final model in the tutorial. Since their data include 2 more classes (silence and noise) and 85% can be the achievable benchmark for this project.

Methodology

Data Preprocessing

The preprocessing steps in model notebook [model-exploration-revised.ipython] consists of the following steps:

1. Split the data into train, valid and test set

From original Google dataset, it has a text file that we can follow to split the data to ratio of approximate 6:1:1 which is reasonable split ratio.

2. Force audio to present one full second

See chop_audio and pad_audio functions in [see: model-exploration-revised.ipython], they help to ensure the data is exactly one second even the original data is longer/shorter.

3. Transform training / valid data to spectrogram (shape: 99 x 161, see Figure 4)

We use scipy to read wavfile and re-adjust the sample rate to 16000 to convert the data [see: model-exploration-revised.ipython] as we discussed in exploration session

Fig 4. train and validation split before model training

```
print x_train.shape
print y_train.shape
print x_valid.shape
print y_valid.shape
```

```
(18538, 99, 161, 1)
(18538, 10)
(2577, 99, 161, 1)
(2577, 10)
```

Implementation

The implementation process is mainly on the classifier training stage, the classifier was trained on the preprocessed training data in notebook [model-exploration-revised.ipython] by steps below:

1. Load both training and validation spectrogram into memory
2. Define network architecture and training parameters
3. Define the loss function and accuracy
4. Train the network and log the process
5. Adjust learning rate when the model is stable
6. Save the trained network

The model is consisted of 98250 parameters using ReLU as activation layer. RMS is used as optimization algorithm and cross entropy is used for cost function. In high level, we repeat partition (convoluted layer [3x3 ReLU] -> dropout [0.2] -> batch normalization -> max pooling [3x3]) 4 times followed by a dense layer [64] and prediction class [10] [see figure 5]

Fig 5. Neural Network Architecture

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 99, 161, 32)       320
_________________________________________________________________
dropout_1 (Dropout)          (None, 99, 161, 32)       0
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 33, 53, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 33, 53, 64)        18496
_________________________________________________________________
batch_normalization_1 (Batch (None, 33, 53, 64)        256
_________________________________________________________________
dropout_2 (Dropout)          (None, 33, 53, 64)        0
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 11, 17, 64)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 11, 17, 64)        36928
_________________________________________________________________
batch_normalization_2 (Batch (None, 11, 17, 64)        256
_________________________________________________________________
dropout_3 (Dropout)          (None, 11, 17, 64)        0
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 3, 5, 64)          0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 3, 5, 64)          36928
_________________________________________________________________
batch_normalization_3 (Batch (None, 3, 5, 64)          256
_________________________________________________________________
dropout_4 (Dropout)          (None, 3, 5, 64)          0
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 1, 1, 64)          0
_________________________________________________________________
flatten_1 (Flatten)          (None, 64)                0
_________________________________________________________________
dense_1 (Dense)              (None, 64)                4160
_________________________________________________________________
dropout_5 (Dropout)          (None, 64)                0
_________________________________________________________________
dense_2 (Dense)              (None, 10)                650
=================================================================
Total params: 98,250
Trainable params: 97,866
Non-trainable params: 384
```

Refinement

Fig 6a. Graph of training/validation losses

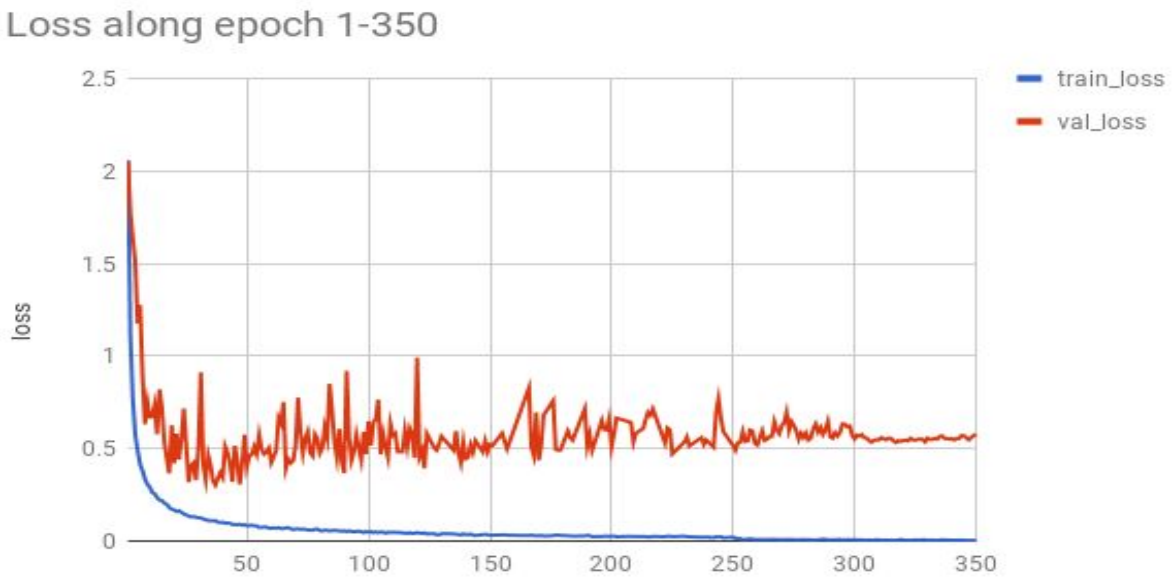
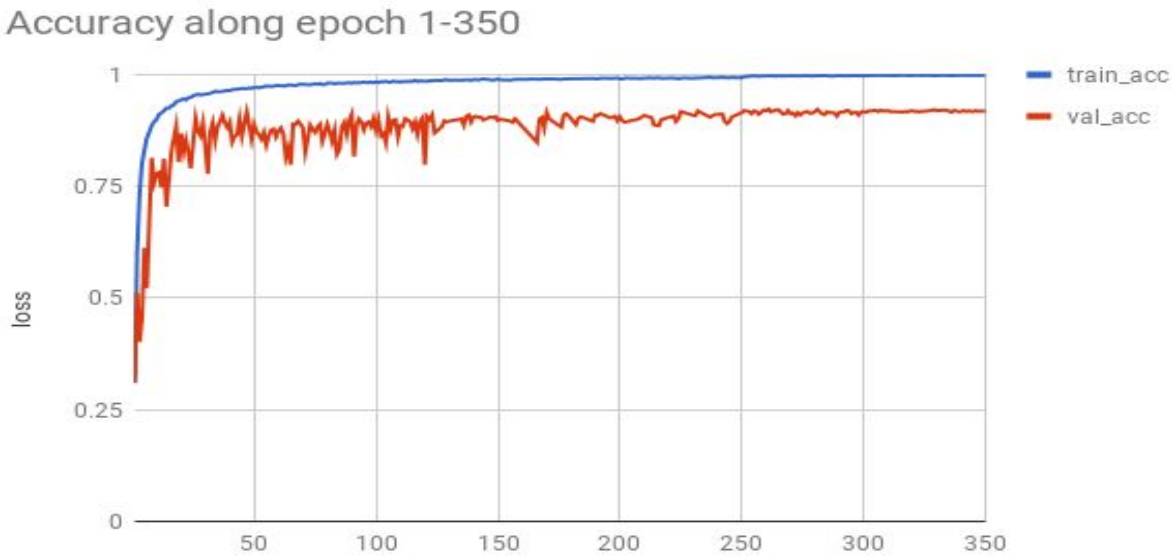


Fig 6b. Graph of training/validation accuracy



We can see flat loss and accuracy trend which suggests the model attains its bottleneck, to further improve it, we may need to either collect more data or use models with stronger representation power.

Also, I did 2 **learning rate reduction** at epoch 250 [lr = 0.00025] and epoch 300 [lr = 0.00005] from [lr = 0.001] to avoid inappropriate learning rate (too big). We can see loss and accuracy tends to be more stable after epoch 250 and 300, it suggests that the original learning rate maybe too big, rather than performance improvement, it makes the results unstable between consecutive epochs.

Although learning reduction doesn't make huge improvement [improve around 0.5%], it helps the model to become more robust. The final model has accuracy of 92 % for validation data and 99.9 % for training data. Obviously, we can't tell whether the model is good or bad because it maybe overfitting, in next part, we will use test set to evaluate the model performance.

Results

Model Evaluation and Validation

As mentioned above, the final model effectively classify the spoken commands to its related word in training and validation data, this part will try to use test set which wasn't used for model training.

We have **2567 audio commands** as the test set and every class has approximately the size (range from 240 to 290). The accuracy of test data is **93.9%** (2410/2567) which is even higher than validation accuracy and when we do error analysis, we can't see strong bias (all of them are less than 10 / 5% of each class). we can make a conclusion that the model **can effectively classify** spoken commands even for new data! Also, from [figure 6] we can see loss and accuracy are almost flat curve at final stage of epoches, it suggests that our model fits the model in stable way.

As the nature of CNN is to learn the image data layer by layer, from low level information [sound, frequency etc] to high level information [meaning, words etc]. When we finish the model training, the model will be robust because the low level information will not change a lot, even we need to finetune for an application, we can reuse large part of the model.

Compare the results with benchmarks that we set, we obviously exceed the baseline (random guess, 10%) significantly, the model also performs better than Google tutorial (85%, note the data is little bit different, their model to the same dataset can be as good as our final model)

Conclusion

Conclusion abstract

In this project, we tried to classify the spoken commands and we first found ways to convert audio data to spectrogram by scipy. The problem is then reduced to classify spoken commands from spectrogram images which we can utilize mature CNN model in deep learning by breaking images to many layers to do classification. We first did 250 epoches (lr = 0.001) and then extra 50 epoches for each (lr = 0.00025) & (lr = 0.00005) to refine the model. At last, we use the model to predict a dataset that isn't in training process and achieve around 94% accuracy without strong error bias [See figure 7].

Fig 7. Classification matrix

SUM of helper label	*prediction* down	go	left	no	off	on	right	stop	up	yes	Grand Total
down	238	2	2	5	1	2	2		1		253
go	4	220	1	4	7	2	4	1	8		251
left			252		3	1	4	1	3	3	267
no	3	4	3	230		1	5	2	4		252
off	2	1	2		249	2	1	1	4		262
on				1	5	233	5	1	1		246
right			1	1		1	253	1	2		259
stop	1	1	1		5	1		233	7		249
up	1		1		8		1	1	260		272
yes	4	2			2		2	1	3	242	256
Grand Total	**253**	**230**	**263**	**241**	**280**	**243**	**277**	**242**	**293**	**245**	**2567**

Using audio dataset with CNN classifier can effectively classify the spoken commands into words, work in this area shows important potential in the future/current IoT as tech giant such as Amazon, Apple and Google are pushing smart devices using voice control. Similar solutions (of course in much bigger and complicated scale) can be applied to not only single commands but conversations, this allows the machines have the capability to "listen" and accumulate dataset so that performance can be further improved and build the foundation for NLP research.

Reflection

After project completion, few things I learned the most:

-Understood that data engineering is much more important than algorithms alone, compare pipeline process with time spent on solution design, former took me majority of

time whereas the later one took me the least efforts. Even a small function (eg: turn audio data to spectrogram) isn't as easy as I thought from the beginning and I spent much more time on how to run everything smoothly.

-Learned how to use Datalabs (GCP)[8] for GPU and moving data in/out between instance and storage buckets. The process of moving/unzip the data in GB levels (original data) is much more difficult than few csv with help of GUI. I did need to use terminal only to control the data in/out. To solve this problem, I did need to study some key concepts of cloud platforms[9], read documents and do operations in cloud (since I need to use GPU there), these two are hidden details and efforts in the project.

Improvement

Some areas can be explored to achieve even better results.

Algorithm

Here we only use straightforward CNN to achieve almost 94% model already, but there are more architectures available to explore, for example, ResNet architecture[10] is common way to increase number of layers so the spectrogram can be even further decomposed to basic shape with stronger representation. Sequence model will be another direction although I didn't explore, audio in nature is sequence, there are series of sequence model with different mechanisms such as attention can be used.

Data

We didn't apply data augmentation[11] but it is possible to include some noise to original data that may make the model becomes more robust against noise and increase the data size at the same time.

If this needs to be **production level**, below are some extra considerations I can foresee outside of the scope of this project

New data

In this project we use the defined data source to do the things, in production environment, we do need to have mechanism to include new words. In that sense, we at least need to include few more steps:

1. Another model to recognize the new words
2. Label the detected words to old/new set of words
3. Solve data skewness problem because we can't control the appearance frequency
4. Design unified mechanism regardless of languages

Model update

As we accumulate more data, it is normal to fine tune the model with new data so it can become more generalised and up-to-date, for example, original dataset may not cover specific group of users with accent of specific region[12].

Prediction

In production usage, it is impossible to have some instances to open Jupyter notebook for prediction especially usually people expect almost real time response, we need to have solution that can scale the prediction when needed. Also, when it may have multiple models co-exist to do some A/B test to evaluate the performance, it is not uncommon to replace the old models with updated models, hence learning cloud solution[13] for prediction is needed especially for online jobs.

References

0. Image of front page

http://techomebuilder.com/emagazine-articles-1/5-ways-to-say-hello-multifamily-voice-control

1. Amazon Echo, in room device and able to control by voice

https://www.amazon.com/dp/B075RWFCHB/

2. Google Home, similar product by Google

https://store.google.com/gb/product/google_home

3. Apple siri

https://www.apple.com/hk/en/ios/siri/

4. Data of Tensorflow Speech Recognition Challenge

https://www.kaggle.com/c/tensorflow-speech-recognition-challenge/data

5. Techniques to represent audio data in image

https://www.kaggle.com/davids1992/speech-representation-and-data-exploration

6. Alex Krizhevsky and Geoffrey Hinton showed practical CNN feasibility in ImageNet 2012

https://www.cs.toronto.edu/~kriz/imagenet_classification_with_deep_convolutional.pdf

7. Tutorial by Google for some benchmarking

https://www.tensorflow.org/versions/master/tutorials/audio_recognition

8. Datalab in Google Cloud offers GPU option and it is my GPU resource for this project

https://cloud.google.com/datalab/

9. GCP on coursera offered me lots of insights and understanding to cloud computing

https://www.coursera.org/specializations/gcp-data-machine-learning

10. Kaiming He's contribution on ResNet makes very deep model possible

https://arxiv.org/pdf/1512.03385.pdf

11. Technique to do audio data augmentation

https://www.kaggle.com/CVxTz/audio-data-augmentation

12. Fine tuning techniques summary by fast.ai

http://wiki.fast.ai/index.php/Fine_tuning

13. Google Machine learning engine and Amazon sagemaker

https://cloud.google.com/ml-engine/

https://aws.amazon.com/tw/sagemaker/