Speech Commands Recognition

Xi Deng (xd487)*   Zulong Ye (zy2010)†

1 Introduction

1.1 Problem Description and Motivation

Nowadays, with the popularization of smart devices, speech recognition has become a promising way to reduce people's screen addiction. Speech recognition provides an effective approach for us to hands-free interact with all those screens, and the most important factor of voice recognition is its accuracy. It is never easy to transcript audio speech because human voices vary a lot from person to person in terms of accents, tones, paces and so on. In this project, we plan to develop an algorithm based on CNN or RNN with LSTM(Long Short-term Memory) that understands simple spoken commands. We will use the Speech Commands Dataset from Google TensorFlow, which includes 65,000 utterances by thousands of different people. We need to predict the exact command labels in test data "yes", "no", "up", "down", etc. after training.

1.2 Literature Survey

We have reviewed three paper performing experiments on the TIMIT phoneme recognition dataset. In *Convolution Neural Networks for Speech Recognition*, the authors show us how to implement CNN on speech input and they manage to reach the average error rate of 20%. While the authors achieve a lower error rate of 17.7% by applying deep bidirectional RNN with LSTM in *Speech Recognition with Deep Recurrent Neural Networks*. Both have noticeable performance improvements over traditional DNNs on the same dataset. In *Research and System Design of Speech Recognition Based on Improved CNN*, another author has done the comparison between CNN with GPU acceleration and RNN with LSTM. As a result, the improved CNN can not only save the running time by 20%, but also reduce the recognition error rate by 15%.

*Tandon School of Engineering, New York University. Email: xd487@nyu.edu.
†Tandon School of Engineering, New York University. Email: zy2010@nyu.edu.

2 Technical Details

2.1 Datasets

The Speech Commands Dataset, released by Google TensorFlow, is a set of one-second wav audio files, which contains 65,000 samples of human voices. In this dataset, files are organized into folders based on the words spoken (labels). There are 30 kinds of short words including core command words such as “yes”, “no”, "up", "down", "left", "right", "on", "off", "stop", "go", from "zero" to"nine" and auxiliary words such as "bed", "bird", "cat", "dog", "happy", etc. Since the size of the dataset is very large, we plan to extract a part of the data as an experiment to test whether our model works. To validate our model’s performance, we will use 80% of the whole dataset for training and 20% of it for testing.
In terms of these .wav file speech information, we use 2 methods to read them. Firstly, we use wavfile.read() function to get the spectrogram with 16000Hz and normalize it. As it takes too long a time to get all data, we store the data information to a .txt file which includes all .wav files information. Then, we can use numpy arrays to read the .txt file to get all the data with this second method, which improves the reading speed greatly.

2.2 Models

Input: For wav audio files, we can use the spectrogram to analyze them. All files can be extracted to a numpy array with a sample rate of 16000Hz. Then, it can be regarded as 16000 * 1 input features for each sample.
Model implementation: In the convolution neural network, we can set the input dimension to be 16000 * 1. Next, we can construct a few convolution 1D layers and some fully connected layers. In the convolution layer, we can take “same” padding mode and “relu” as the activation function. In the fully connected layer, we will also take “relu” as the activation function. In the output layer, “softmax” is the activation function.
In the LSTM model, we can also set the input dimension of 16000 * 1. We implement 2 layers of LSTM. All parameters need to be adjusted for better accuracy.
Output: The output result is the probability of each class representative, and we will select the largest probabilities as the predicted label. For actual values computed by the label of each file folder, we will match each text label with a number. Then, each label can be matched with a class representative.

2.3 Algorithms

Firstly, we denoise the speech data using a high pass filter and resample the input so that it can be taken by CNN and RNN. Then we train and modify the model so that it can make the right prediction of ten commands “yes”, “no”, etc., with a high accuracy. Finally, we apply both models to test data and compare the results of both networks (pure CNN and CNN with LSTM). The main flow chart diagram of our project is shown in Figure 1.

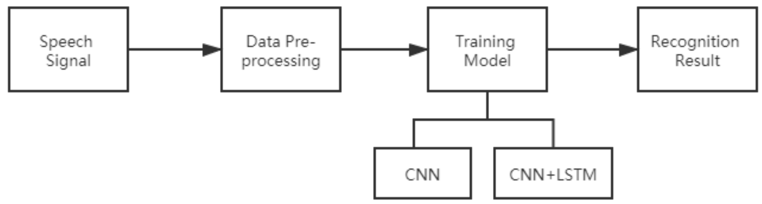


FIGURE 1: Flow chart diagram

In terms of data pre-processing, we choose "yes", "no", "up", "down", "left", "right", "on", "off", "stop", "go" commands as the target for recognition. By using the "LabelEncoder" to change each text label into specific numbers, we regard the value of 1-10 numbers as ten classes representative.

Since we aim to predict the probability of the label, we would choose “cross entropy” as the loss function, and “stochastic gradient descent” as an optimizer for CNN; “cross entropy” as the loss function and “adam” as an optimizer for LSTM. Through the loss function and “softmax” algorithm, we can calculate the probability for each speech command, and choose the largest probability as the predictive value.

For the convolutional 1D layer, we set the ‘Relu’ as the activation function. Therefore, we have all neuron activations in each layer can be represented in the following matrix form:

$$o^{(l)} = \phi(o^{(l-1)}W^{(l)})(l = 1, 2, ..., L - 1),$$

where $\phi(x) = max(0, x)$. The loss function shows as follows.

$$L_{CE} = -\sum_{i=1}^{n} t_i log(p_i),$$

for n classes, where t_i is the actual label and p_i is the Softmax probability for the i^{th} class. And the softmax probability is given by:

$$p_i = \frac{exp(y_i)}{\sum_{j=1}^{n} exp(y_j)}$$

2.4 Training Methods

Training Models (CNN):

We finalize our CNN model including 10 convolution 1D layers and 1 fully connected layers to achieve the best training result. We set our parameters as follows. From the first to the ninth layer, there are 8, 16, 32... 2048 hidden units and a 9*9-size filter for each layer. Besides, we apply Batch Normalization to these layers, and we set the max pooling size to 2*2 for each layer. When it comes to the tenth convolution layer, we use 1024 hidden layers and there is a fully connected dense layer with 1024 units. We choose ‘same’ padding mode

```
max_pool_level_2 (MaxPooling (None, 2000, 32)          0
_________________________________________________________________
conv_level_3 (Conv1D)        (None, 2000, 64)          18496
_________________________________________________________________
batch_level_3 (BatchNormaliz (None, 2000, 64)          256
_________________________________________________________________
max_pool_level_3 (MaxPooling (None, 1000, 64)          0
_________________________________________________________________
conv_level_4 (Conv1D)        (None, 1000, 128)         73856
_________________________________________________________________
batch_level_4 (BatchNormaliz (None, 1000, 128)         512
_________________________________________________________________
max_pool_level_4 (MaxPooling (None, 500, 128)          0
_________________________________________________________________
conv_level_5 (Conv1D)        (None, 500, 256)          295168
_________________________________________________________________
batch_level_5 (BatchNormaliz (None, 500, 256)          1024
_________________________________________________________________
max_pool_level_5 (MaxPooling (None, 250, 256)          0
_________________________________________________________________
conv_level_6 (Conv1D)        (None, 250, 512)          1180160
_________________________________________________________________
batch_level_6 (BatchNormaliz (None, 250, 512)          2048
_________________________________________________________________
max_pool_level_6 (MaxPooling (None, 125, 512)          0
_________________________________________________________________
conv_level_7 (Conv1D)        (None, 125, 1024)         4719616
_________________________________________________________________
batch_level_7 (BatchNormaliz (None, 125, 1024)         4096
_________________________________________________________________
max_pool_level_7 (MaxPooling (None, 63, 1024)          0
_________________________________________________________________
conv_level_8 (Conv1D)        (None, 63, 2048)          18876416
_________________________________________________________________
batch_level_8 (BatchNormaliz (None, 63, 2048)          8192
_________________________________________________________________
max_pool_level_8 (MaxPooling (None, 32, 2048)          0
_________________________________________________________________
Last_level (Conv1D)          (None, 32, 1024)          2098176
_________________________________________________________________
global_max_pooling1d (Global (None, 1024)              0
_________________________________________________________________
Dense_level (Dense)          (None, 1024)              1049600
_________________________________________________________________
dropout (Dropout)            (None, 1024)              0
_________________________________________________________________
cls (Dense)                  (None, 10)                10250
=================================================================
Total params: 28,343,982
Trainable params: 28,335,804
Non-trainable params: 8,178
_________________________________________________________________
```

```
Epoch 41/300
400/400 [==============================] - 27s 69ms/step - loss: 0.0077 - accuracy: 0.9974 - val_loss: 0.2433 - val_accuracy: 0.9479
Epoch 42/300
400/400 [==============================] - 28s 69ms/step - loss: 0.0080 - accuracy: 0.9976 - val_loss: 0.2018 - val_accuracy: 0.9510
Epoch 43/300
400/400 [==============================] - 28s 69ms/step - loss: 0.0081 - accuracy: 0.9974 - val_loss: 1.2482 - val_accuracy: 0.7307
Epoch 44/300
400/400 [==============================] - 28s 69ms/step - loss: 0.0099 - accuracy: 0.9973 - val_loss: 0.2337 - val_accuracy: 0.9465
Epoch 45/300
400/400 [==============================] - 28s 69ms/step - loss: 0.0073 - accuracy: 0.9977 - val_loss: 0.2389 - val_accuracy: 0.9425
Epoch 46/300
400/400 [==============================] - 28s 69ms/step - loss: 0.0070 - accuracy: 0.9977 - val_loss: 0.2490 - val_accuracy: 0.9383
Epoch 47/300
400/400 [==============================] - 28s 69ms/step - loss: 0.0109 - accuracy: 0.9968 - val_loss: 0.3399 - val_accuracy: 0.9254
Epoch 48/300
400/400 [==============================] - 28s 69ms/step - loss: 0.0058 - accuracy: 0.9982 - val_loss: 0.2036 - val_accuracy: 0.9496
Epoch 49/300
400/400 [==============================] - 28s 69ms/step - loss: 0.0055 - accuracy: 0.9983 - val_loss: 0.3800 - val_accuracy: 0.9134
Epoch 50/300
400/400 [==============================] - 27s 69ms/step - loss: 0.0070 - accuracy: 0.9979 - val_loss: 0.3862 - val_accuracy: 0.9120
Epoch 51/300
400/400 [==============================] - 27s 69ms/step - loss: 0.0154 - accuracy: 0.9953 - val_loss: 0.3762 - val_accuracy: 0.9125
Epoch 52/300
400/400 [==============================] - 27s 69ms/step - loss: 0.0083 - accuracy: 0.9981 - val_loss: 0.2273 - val_accuracy: 0.9444
Epoch 53/300
400/400 [==============================] - 27s 69ms/step - loss: 0.0066 - accuracy: 0.9983 - val_loss: 0.3263 - val_accuracy: 0.9259
Epoch 54/300
400/400 [==============================] - 28s 69ms/step - loss: 0.0061 - accuracy: 0.9988 - val_loss: 0.2663 - val_accuracy: 0.9409
Epoch 55/300
400/400 [==============================] - 27s 69ms/step - loss: 0.0062 - accuracy: 0.9979 - val_loss: 0.2114 - val_accuracy: 0.9484
Epoch 56/300
400/400 [==============================] - 28s 69ms/step - loss: 0.0123 - accuracy: 0.9963 - val_loss: 0.2562 - val_accuracy: 0.9369
Epoch 57/300
400/400 [==============================] - 27s 69ms/step - loss: 0.0074 - accuracy: 0.9980 - val_loss: 0.2898 - val_accuracy: 0.9287
Epoch 58/300
400/400 [==============================] - 28s 69ms/step - loss: 0.0076 - accuracy: 0.9975 - val_loss: 0.2600 - val_accuracy: 0.9390
Epoch 59/300
400/400 [==============================] - 28s 69ms/step - loss: 0.0093 - accuracy: 0.9969 - val_loss: 0.3300 - val_accuracy: 0.9214
Epoch 60/300
400/400 [==============================] - 27s 69ms/step - loss: 0.0135 - accuracy: 0.9975 - val_loss: 0.2345 - val_accuracy: 0.9463
Epoch 61/300
400/400 [==============================] - 27s 69ms/step - loss: 0.0066 - accuracy: 0.9984 - val_loss: 0.2830 - val_accuracy: 0.9362
Epoch 62/300
400/400 [==============================] - 27s 69ms/step - loss: 0.0082 - accuracy: 0.9976 - val_loss: 0.2534 - val_accuracy: 0.9418
Epoch 63/300
400/400 [==============================] - 27s 69ms/step - loss: 0.0059 - accuracy: 0.9982 - val_loss: 0.2223 - val_accuracy: 0.9477
Epoch 64/300
400/400 [==============================] - 28s 69ms/step - loss: 0.0059 - accuracy: 0.9978 - val_loss: 0.7075 - val_accuracy: 0.8419
Epoch 65/300
400/400 [==============================] - 27s 69ms/step - loss: 0.0087 - accuracy: 0.9968 - val_loss: 0.2824 - val_accuracy: 0.9334
Epoch 00065: early stopping
```

FIGURE 2: CNN model and training details

```
Model: "model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 16000, 1)]        0
_________________________________________________________________
conv1d (Conv1D)              (None, 2000, 256)         8448
_________________________________________________________________
max_pooling1d (MaxPooling1D) (None, 125, 256)          0
_________________________________________________________________
dropout (Dropout)            (None, 125, 256)          0
_________________________________________________________________
lstm_1 (LSTM)                (None, 125, 256)          525312
_________________________________________________________________
lstm_2 (LSTM)                (None, 125, 256)          525312
_________________________________________________________________
flatten (Flatten)            (None, 32000)             0
_________________________________________________________________
dense (Dense)                (None, 64)                2048064
_________________________________________________________________
dropout_1 (Dropout)          (None, 64)                0
_________________________________________________________________
dense_1 (Dense)              (None, 10)                650
=================================================================
Total params: 3,107,786
Trainable params: 3,107,786
Non-trainable params: 0
_________________________________________________________________
```

```
Epoch 21/100
100/100 [==============================] - 5s 51ms/step - loss: 0.2012 - accuracy: 0.9288 - val_loss: 0.3808 - val_accuracy: 0.8897
Epoch 22/100
100/100 [==============================] - 5s 51ms/step - loss: 0.1911 - accuracy: 0.9321 - val_loss: 0.3984 - val_accuracy: 0.8893
Epoch 23/100
100/100 [==============================] - 5s 51ms/step - loss: 0.1856 - accuracy: 0.9317 - val_loss: 0.3641 - val_accuracy: 0.8921
Epoch 24/100
100/100 [==============================] - 5s 50ms/step - loss: 0.1654 - accuracy: 0.9422 - val_loss: 0.3707 - val_accuracy: 0.8919
Epoch 25/100
100/100 [==============================] - 5s 51ms/step - loss: 0.1632 - accuracy: 0.9428 - val_loss: 0.3792 - val_accuracy: 0.8970
Epoch 26/100
100/100 [==============================] - 5s 51ms/step - loss: 0.1677 - accuracy: 0.9387 - val_loss: 0.3762 - val_accuracy: 0.8837
Epoch 27/100
100/100 [==============================] - 5s 51ms/step - loss: 0.1553 - accuracy: 0.9439 - val_loss: 0.3747 - val_accuracy: 0.8935
Epoch 28/100
100/100 [==============================] - 5s 51ms/step - loss: 0.1529 - accuracy: 0.9437 - val_loss: 0.3635 - val_accuracy: 0.9022
Epoch 29/100
100/100 [==============================] - 5s 51ms/step - loss: 0.1292 - accuracy: 0.9518 - val_loss: 0.3915 - val_accuracy: 0.8970
Epoch 30/100
100/100 [==============================] - 5s 51ms/step - loss: 0.1259 - accuracy: 0.9548 - val_loss: 0.4025 - val_accuracy: 0.8987
Epoch 31/100
100/100 [==============================] - 5s 51ms/step - loss: 0.1311 - accuracy: 0.9554 - val_loss: 0.4305 - val_accuracy: 0.9012
Epoch 32/100
100/100 [==============================] - 5s 51ms/step - loss: 0.1331 - accuracy: 0.9532 - val_loss: 0.3616 - val_accuracy: 0.8982
Epoch 33/100
100/100 [==============================] - 5s 51ms/step - loss: 0.1319 - accuracy: 0.9528 - val_loss: 0.3633 - val_accuracy: 0.8928
Epoch 34/100
100/100 [==============================] - 5s 51ms/step - loss: 0.1262 - accuracy: 0.9554 - val_loss: 0.3672 - val_accuracy: 0.9034
Epoch 35/100
100/100 [==============================] - 5s 51ms/step - loss: 0.1172 - accuracy: 0.9585 - val_loss: 0.3613 - val_accuracy: 0.8975
Epoch 36/100
100/100 [==============================] - 5s 51ms/step - loss: 0.1037 - accuracy: 0.9616 - val_loss: 0.4023 - val_accuracy: 0.9043
Epoch 37/100
100/100 [==============================] - 5s 51ms/step - loss: 0.1115 - accuracy: 0.9605 - val_loss: 0.4116 - val_accuracy: 0.8991
Epoch 38/100
100/100 [==============================] - 5s 51ms/step - loss: 0.1007 - accuracy: 0.9621 - val_loss: 0.4211 - val_accuracy: 0.9017
Epoch 39/100
100/100 [==============================] - 5s 51ms/step - loss: 0.1017 - accuracy: 0.9628 - val_loss: 0.4406 - val_accuracy: 0.8991
Epoch 40/100
100/100 [==============================] - 5s 51ms/step - loss: 0.1095 - accuracy: 0.9601 - val_loss: 0.4580 - val_accuracy: 0.9034
Epoch 41/100
100/100 [==============================] - 5s 51ms/step - loss: 0.1067 - accuracy: 0.9628 - val_loss: 0.4247 - val_accuracy: 0.8949
Epoch 42/100
100/100 [==============================] - 5s 51ms/step - loss: 0.1055 - accuracy: 0.9643 - val_loss: 0.4019 - val_accuracy: 0.8912
Epoch 43/100
100/100 [==============================] - 5s 51ms/step - loss: 0.1129 - accuracy: 0.9628 - val_loss: 0.4125 - val_accuracy: 0.9031
Epoch 44/100
100/100 [==============================] - 5s 51ms/step - loss: 0.0898 - accuracy: 0.9690 - val_loss: 0.4642 - val_accuracy: 0.8968
Epoch 45/100
100/100 [==============================] - 5s 51ms/step - loss: 0.1053 - accuracy: 0.9629 - val_loss: 0.4244 - val_accuracy: 0.9048
Epoch 00045: early stopping
```

FIGURE 3: CNN+LSTM model and training details

for every conv1D layer, and we use dropout with 0.2 to reduce overfit. In addition, a dense layer with 10 units is added for label classifications in output. Last but not least, we plan to train the model for 300 epochs with "early stopping" callback function. Model and training details are shown in Figure 2.

Training models (CNN + LSTM):

We finalize our CNN and LSTM model with 1 convolution 1D layer with 256 hidden units and a 32*32-size filter using the "same" padding mode, as well as 2 LSTM layers with 256 units connected to the first convolution layer. Next, there is 1 fully connected layer with 64 units. We use dropout with 0.2 to reduce overfit. Lastly, a dense layer with 10 units is added for label classifications. Model and training details are shown in Figure 3.

2.5 Evaluation Metrics

We are using training accuracy, validation accuracy, testing accuracy and confusion matrix to evaluate the performance of our CNN and LSTM models. The index of confusion matrix is given by the number of correct predicted labels divided by the number of actual labels. The vertical line is the predicted labels and horizontal line is the actual label.

3 Results

3.1 Data Visualization

We can open a speech command file through the "wavfile.read()" function and convert it into a numpy array. To extract its features, we plot its wave spectrum and spectrogram using the "log_spectgram()" function. For example, we take a .wav file from "left" commands, then plot its wave spectrum and spectrogram in Figure 4.

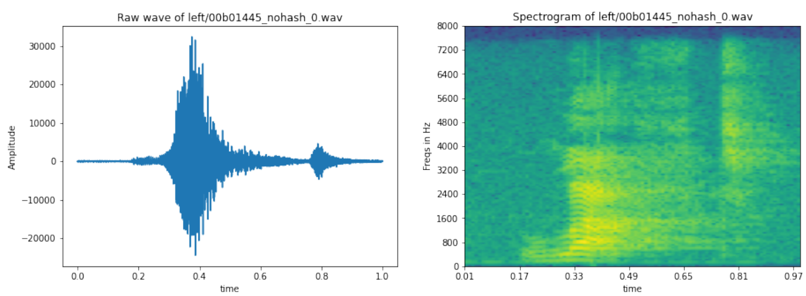
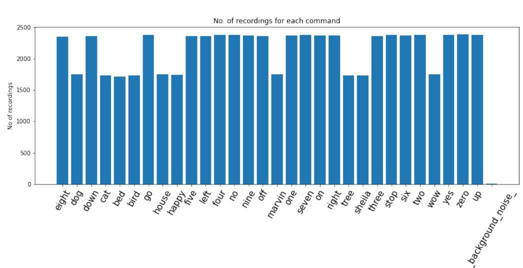


FIGURE 4: Wave spectrum and spectrogram of a speech command

Next, we will count the number of audio files in each label, as shown in the left part of Figure 5. Then, we split the whole dataset as training data, validation data, and test data with a ratio of 0.2 test size. The right part of Figure 5 represents the result of data split.



(a) Each label

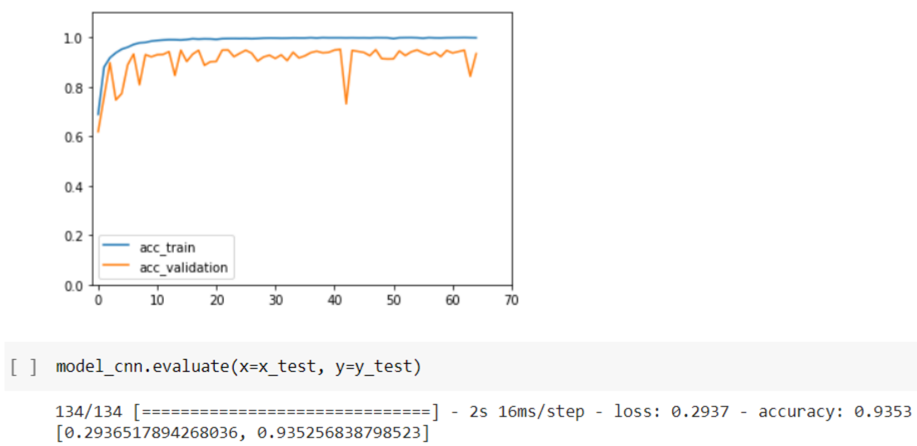
```
training data shape: (12786, 8000, 1)
validation data shape: (4263, 8000, 1)
test data shape: (4263, 8000, 1)
```

(b) Data size

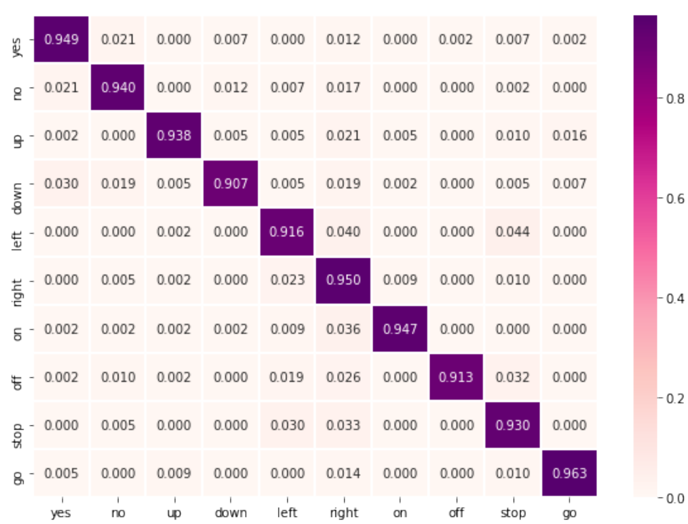
FIGURE 5: Number of samples in each label and training, validation, testing data size

3.2 Train Results

Training Models (CNN):
After a number of attempts on adjusting the CNN model structure and coefficients, we manage to reach a training accuracy of 99%, a validation accuracy of 94% and a test accuracy of 93%. The curve of training accuracy is smoother and converges at about 15 epochs, while the curve of validation accuracy bounces a lot. From the confusion matrix on the right, we can see that the accuracy of predicting each label is higher than 90%, while the highest one goes above 96%.



(a) Training result

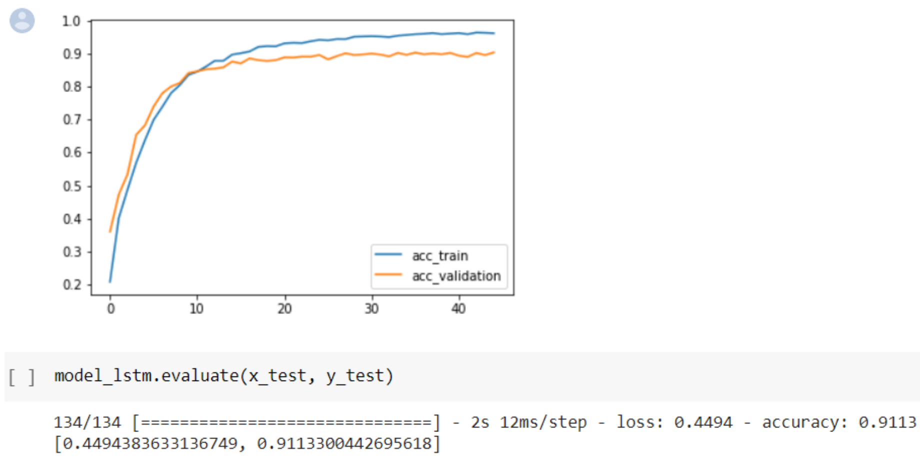


(b) Confusion matrix

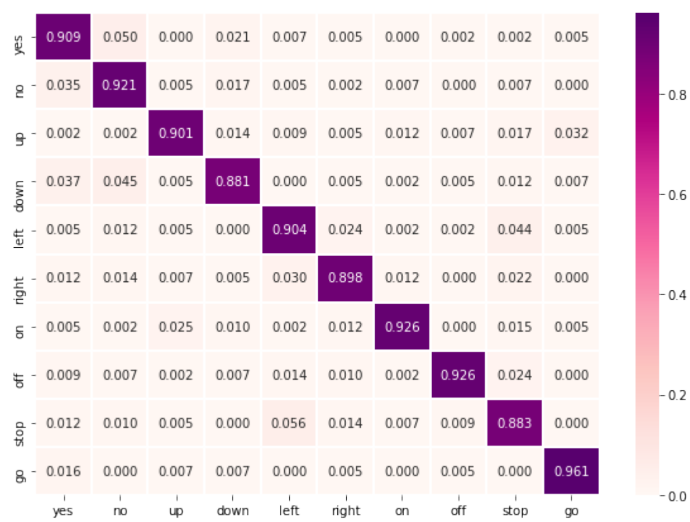
FIGURE 6: CNN training results

Training Models (CNN+LSTM):
We obtain a training accuracy of 96%, a validation accuracy of 90%, and a test accuracy

of 91%. The curve of both training and validation accuracy are smooth, but validation accuracy seems to converge earlier than training accuracy. From the confusion matrix on the right, we can tell that the prediction accuracy of each label is around 90%, and the highest one is 96% while the lowest one is 88%.



(a) Training result



(b) Confusion matrix

FIGURE 7: CNN+LSTM training results

The following table demonstrates the comparison between the performance of some models that we have tested and the performance of our finalized model in CNN and CNN+LSTM.

TABLE 1: Training Results of various network structures

	Network Structure	Average Test Accuracy	Number of Training Parameters	Average Running Time
1	CNN (2 conv1D, 1 dense, input size = 8000)	59.39%	7.351M	8m 43s
2	CNN (4 conv1D, 2 dense, input size = 16000)	78.14%	1.0996M	22m 59s
3	CNN (10 conv1D, 2 dense, input size = 16000)	93.5%	28.336M	45m 21s
4	CNN+LSTM (1 conv1D, 1 LSTM, 1 dense, input size = 8000)	86.61%	14.676M	3m 54s
5	CNN+LSTM (1 conv1D, 2 LSTM, 1 dense, input size = 16000)	90.17%	3.108M	3m 37s
6	CNN+LSTM (1 conv1D, 4 LSTM, 1 dense, input size = 16000)	89.96%	4.158M	5m 27s

3.3 Predict

Here is a list of predictions of a couple of speech command samples using both our models. We can see that both our models make predictions very precisely, producing almost no mistake.

```
Audio: go
Text (CNN): go
Text (LSTM): go
Audio: on
Text (CNN): on
Text (LSTM): on
Audio: on
Text (CNN): on
Text (LSTM): on
Audio: up
Text (CNN): up
Text (LSTM): up
Audio: down
Text (CNN): down
Text (LSTM): down
Audio: left
Text (CNN): left
Text (LSTM): left
Audio: yes
Text (CNN): yes
Text (LSTM): yes
Audio: go
Text (CNN): go
Text (LSTM): go
Audio: go
Text (CNN): go
Text (LSTM): go
Audio: no
Text (CNN): no
Text (LSTM): no
Audio: go
Text (CNN): stop
Text (LSTM): off
Audio: yes
Text (CNN): yes
Text (LSTM): yes
Audio: right
Text (CNN): right
Text (LSTM): right
Audio: down
Text (CNN): down
Text (LSTM): down
Audio: right
Text (CNN): right
Text (LSTM): down
```

```
Audio: on
Text (CNN): on
Text (LSTM): on
Audio: no
Text (CNN): no
Text (LSTM): no
Audio: left
Text (CNN): left
Text (LSTM): left
Audio: up
Text (CNN): up
Text (LSTM): up
Audio: down
Text (CNN): down
Text (LSTM): down
Audio: yes
Text (CNN): yes
Text (LSTM): yes
Audio: go
Text (CNN): go
Text (LSTM): go
Audio: on
Text (CNN): on
Text (LSTM): on
Audio: up
Text (CNN): up
Text (LSTM): up
Audio: stop
Text (CNN): stop
Text (LSTM): stop
Audio: no
Text (CNN): no
Text (LSTM): no
Audio: no
Text (CNN): no
Text (LSTM): no
Audio: stop
Text (CNN): stop
Text (LSTM): stop
Audio: right
Text (CNN): right
Text (LSTM): right
Audio: on
Text (CNN): on
Text (LSTM): on
```

```
Audio: off
Text (CNN): off
Text (LSTM): off
Audio: go
Text (CNN): go
Text (LSTM): go
Audio: no
Text (CNN): no
Text (LSTM): no
Audio: left
Text (CNN): left
Text (LSTM): stop
Audio: stop
Text (CNN): stop
Text (LSTM): stop
Audio: no
Text (CNN): no
Text (LSTM): no
Audio: go
Text (CNN): go
Text (LSTM): go
Audio: yes
Text (CNN): yes
Text (LSTM): yes
Audio: yes
Text (CNN): yes
Text (LSTM): yes
Audio: no
Text (CNN): no
Text (LSTM): no
Audio: down
Text (CNN): down
Text (LSTM): down
Audio: yes
Text (CNN): yes
Text (LSTM): yes
Audio: no
Text (CNN): no
Text (LSTM): no
Audio: stop
Text (CNN): stop
Text (LSTM): off
Audio: left
Text (CNN): left
Text (LSTM): left
```

```
Audio: stop
Text (CNN): stop
Text (LSTM): stop
Audio: up
Text (CNN): up
Text (LSTM): up
Audio: stop
Text (CNN): up
Text (LSTM): right
Audio: up
Text (CNN): up
Text (LSTM): up
Audio: go
Text (CNN): go
Text (LSTM): go
Audio: up
Text (CNN): up
Text (LSTM): off
Audio: off
Text (CNN): off
Text (LSTM): off
Audio: no
Text (CNN): no
Text (LSTM): no
Audio: up
Text (CNN): up
Text (LSTM): up
Audio: go
Text (CNN): go
Text (LSTM): go
Audio: off
Text (CNN): off
Text (LSTM): off
Audio: right
Text (CNN): right
Text (LSTM): right
Audio: off
Text (CNN): up
Text (LSTM): up
Audio: stop
Text (CNN): stop
Text (LSTM): stop
Audio: on
Text (CNN): on
Text (LSTM): on
```

FIGURE 8: Predict results

4 Conclusions and Source Code

4.1 Conclusions

In this project, we manage to implement a deep convolutional neural network and a combination of CNN with LSTM network, as well as apply both networks on the Speech Commands Dataset from Google TensorFlow. Now we have accomplished the feature extractions of audio files, and increased the speed of audio data processing greatly. Most importantly, we have acquired very high recognition precisions on both finalized networks, which shows a significant improvement on what we did in the project update. As a result, the complex CNN model has slightly better performance compared to the CNN with LSTM model, while the CNN with LSTM model runs much faster than the complex CNN model. Furthermore, we found that CNN performance is sensitive to padding mode and pooling size, rather than filter size. In addition, CNN performance can be enhanced by adding more and more layers to make the network very deep but LSTM cannot.

4.2 Future Work

We plan to apply our system on some large-vocabulary speech dataset, or try some other feature extraction approaches like Mel Frequency Cepstrum Coefficient(MFCC) in the future. We may also try some new models such as Transformers/BERT, and we can compare the performance of these different models.

4.3 Source Code

Github: https://github.com/yzleaf/speech_commands_recognition
Colab (Access by NYU account): Click here

References

1. O. Abdel-Hamid, A. Mohamed, H. Jiang, L. Deng, G. Penn and D. Yu, "Convolutional Neural Networks for Speech Recognition," in IEEE/ACM Transactions on Audio, Speech, and Language Processing, vol. 22, no. 10, pp. 1533-1545, Oct. 2014, doi: 10.1109/TASLP.2014.2339736.
2. A. Graves, A. Mohamed and G. Hinton, "Speech recognition with deep recurrent neural networks," 2013 IEEE International Conference on Acoustics, Speech and Signal Processing, Vancouver, BC, Canada, 2013, pp. 6645-6649, doi: 10.1109 /ICASSP.2013.6638947.
3. L. Xu, "Research and System Design of Speech Recognition Based on Improved CNN," Master, Donghua University, Shanghai, China, 2019.
4. https://www.tensorflow.org/datasets/catalog/speech_commands
5. https://keras.io/api/layers/recurrent_layers/lstm/
6. Davids, "Speech representation and data exploration", https://www.kaggle.com/davids1992/speech-representation-and-data-exploration (accessed May 2021)