Spoken Number Recognition

Yicong Liu (yl3975)

Description

The main purpose project is to build a classifier to recognize a single spoken number (0 ~ 9). To be specific, as one speaks a number (0 ~ 9), the program will recognize the correct number and shows as the output. The input comes from a microphone.

Related Work

This project is actually a small of speech recognition, which is very popular in last few years.

As far as I've leaned from the Internet, two techniques used for speech recognition in the past were DTW (dynamic time warping) and HMM (Hidden Markov models).

In past few years, neural networks, especially deep networks, are more and more widely used in speech recognition. In contrast to HMMs, neural networks make no assumptions about feature statistical properties and have several qualities making them attractive recognition models for speech recognition.

The most commonly used models are RNN (recurrent neural network) or models on it. I've seen some projects that use RNNs or a combination including RNN. For example, https://github.com/lucko515/speech-recognition-neural-network.

Another commonly used technique is CTC (Connectionist Temporal Classification). For example, https://github.com/philipperemy/tensorflow-ctc-speech-recognition.

Since we have learned CNN (convolution neural network) in class, so I wonder if the problem can be solved by CNN. We know that CNN performs well in image recognition, so I wonder if an audio can be described as an image. I looked for similar work using CNN, and it turns out that CNN work, according to reference [2][3]. They provide me with the basic idea.

Solution

1. Basic Idea

I looked for similar work using CNN, and it turned out that the idea actually works. Most of them use MFCCs as features to plot certain images, and trained these images using CNN. Base on this, I came up with the basic ideas.

Draw the Mel spectrogram of each *.wav* file, and save as an image. In this way, the speech recognition problem is transferred into an image recognition problem.

Use CNN to build a classifier for the dataset. The CNN model includes 2 Dense (fully

connected) layers and 5 Convolution layers, with Max-Pooling and Batch Normalization layers in it.

2. Dataset

The dataset can be downloaded from the link

http://pannous.net/files/spoken_numbers_pcm.tar

The dataset includes 2850 *.wav* files of 15 different people (male and female) speaking number 0 - 9. Besides, 400 .wav files recorded by me and my roommate are added to the dataset.

I pick 500 audios as the test set, and the remaining as the training data.

Another dataset I may recommend is called FSDD. It is an open dataset, which means it will grow over time as data is contributed. But there are only recordings of 4 male so far, so it isn't an optimal dataset for this project. The link of dataset is

https://github.com/Jakobovski/free-spoken-digit-dataset/tree/master/recordings

3. Python Libraries Required

keras	library for neural network
tensorflow-gpu	backend for keras
h5py	to store huge amount data (to save the trained model)
librosa	to compute mel spectrogram of each audio file
pyaudio	to get input from the microphone
numpy	common library for array computation
matplotlib	common library for plotting

4. Plot Mel Spectrogram

Explore all the *.wav* files in the dataset. I use *librosa.feature.melspectrogram()* method in to compute the mel spectrogram of each file, and then use *librosa.display.specshow()* method to plot it.

Finally, save all the mel spectrogram images to generate a new dataset, only containing images. One sample image is shown in Figure 1.

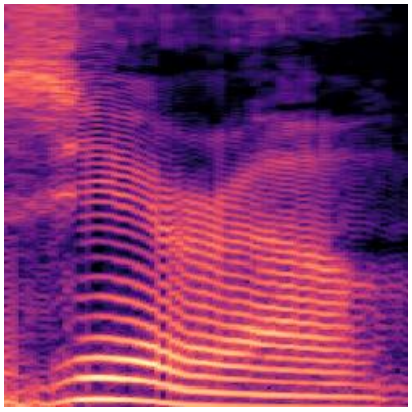

Figure 1. Mel spectrogram of '*0_Alex_60.wav*'

5. Build Classifier

Since we've got the dataset, let's build the classifier. It is the key to the project.

I use CNN to build the classifier [4]. The network consists of 6 blocks of 2D convolution, with ReLU nonlinearity, 2D max pooling and batch normalization.

The reason to apply batch normalization to each block is that the training speed can be significantly increased, according to many papers.

Finally, I use 1 fully connected layer before the output block along with softmax activation and 50% dropout. Since the number of target classes is 10, the number of channels of output layer is 10. The parameters of each layer are shown in Table 1.

Table 1. Parameters of CNN model

Block No.	Type	Channels	Kernel Size	Stride
0	Input	1		
1	Conv2D	16	7 * 7	1
	ReLU	16		
	MaxPooling	16	3 * 3	2
	BatchNormalization	16		
2	Conv2D	32	5 * 5	1
	ReLU	32		
	MaxPooling	32	3 * 3	2
	BatchNormalization	32		
3	Conv2D	64	3 * 3	1
	ReLU	64		
	MaxPooling	64	3 * 3	2
	BatchNormalization	64		
4	Conv2D	128	3 * 3	1
	ReLU	128		
	MaxPooling	128	3 * 3	2
	BatchNormalization	128		
5	Conv2D	128	3 * 3	1
	ReLU	128		
	MaxPooling	128	3 * 3	2
	BatchNormalization	128		
6	Conv2D	256	3 * 3	1
	ReLU	256		
	MaxPooling	256	3 * 3	2
	BatchNormalization	256		
7	Dense	1024		
	ReLU	1024		
	BatchNormalization	1024		
	Dropout	1024		
8	Dense	10		
	Softmax	10		

The summary of the model in python is shown in Figure 2.

Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 200, 200, 16)	2368
max_pooling2d_1 (MaxPooling2	(None, 100, 100, 16)	0
batch_normalization_1 (Batch	(None, 100, 100, 16)	64
conv2d_2 (Conv2D)	(None, 100, 100, 32)	12832
max_pooling2d_2 (MaxPooling2	(None, 50, 50, 32)	0
batch_normalization_2 (Batch	(None, 50, 50, 32)	128
conv2d_3 (Conv2D)	(None, 50, 50, 64)	18496
max_pooling2d_3 (MaxPooling2	(None, 25, 25, 64)	0
batch_normalization_3 (Batch	(None, 25, 25, 64)	256
conv2d_4 (Conv2D)	(None, 25, 25, 128)	73856
max_pooling2d_4 (MaxPooling2	(None, 13, 13, 128)	0
batch_normalization_4 (Batch	(None, 13, 13, 128)	512
conv2d_5 (Conv2D)	(None, 13, 13, 128)	147584
max_pooling2d_5 (MaxPooling2	(None, 7, 7, 128)	0
batch_normalization_5 (Batch	(None, 7, 7, 128)	512
conv2d_6 (Conv2D)	(None, 7, 7, 256)	295168
max_pooling2d_6 (MaxPooling2	(None, 4, 4, 256)	0
batch_normalization_6 (Batch	(None, 4, 4, 256)	1024
flatten_1 (Flatten)	(None, 4096)	0
dense_1 (Dense)	(None, 1024)	4195328
batch_normalization_7 (Batch	(None, 1024)	4096
dropout_1 (Dropout)	(None, 1024)	0
dense_2 (Dense)	(None, 10)	10250

Total params: 4,762,474
Trainable params: 4,759,178
Non-trainable params: 3,296

Figure 2. Summary of CNN model

Then I use *ImageDataGenerator()* method in keras to generate the training set and test set.

Finally, I use Adam optimizer with the learning rate 0.0005. After training the model, I use *model.save()* method to save the model.

6. Recorder

Now we've already got my trained model. So, at this step, I implement it in real-time, which means the program will do number recognition while you are speaking through input devices. The python package *pyaudio* will be used to obtain speech data from your input devices.

First, I define a function with a threshold. The program will keep obtaining data

from the input device, and in the meantime, determine whether there is valid speech input (above the threshold). Once the speech input is detected, it starts recording, and stops when certain number of inputs are determined to be invalid (below the threshold).

Then the program will plot the mel spectrogram of the input audio, and use *model.predict()* method to calculate the probabilities of each class.

Finally, the problem will print the class with highest probability, but return "error" if the highest probability is less than 10%.

7. More Tips

- Use microphones, for better performance. The recorder starts recording when the input reaches the threshold. So, it won't work if there is too much noise.
- The program named *'trimmer.py'* is provided, which extends a *.wav* file to 1 second, in case that you want all the audio to have the same length. If you do that, remember to modified *'recorder.py'* so that the program always records 1 second of your speech.
- I use RGB images in all programs. You can try grayscale images, and I think it will also work. The training speed may be increased in this way.

Evaluation

I record additional 50 audios (5 audios for each number) as the test set, and the others as the train set (3250 images). After 20 epochs, the accuracy reaches 100%, and the validate accuracy reaches 98%. Very Nice!

Part of the training step for last epoch is shown in Figure 3.

```
 90/101 [=========================>....] - ETA: 0s - loss: 1.7278e-04 - acc: 1.0000
 91/101 [=========================>...] - ETA: 0s - loss: 1.7384e-04 - acc: 1.0000
 92/101 [=========================>...] - ETA: 0s - loss: 1.7615e-04 - acc: 1.0000
 93/101 [=========================>...] - ETA: 0s - loss: 1.7539e-04 - acc: 1.0000
 94/101 [=========================>...] - ETA: 0s - loss: 1.7559e-04 - acc: 1.0000
 95/101 [=========================>..] - ETA: 0s - loss: 1.7387e-04 - acc: 1.0000
 96/101 [=========================>..] - ETA: 0s - loss: 1.7264e-04 - acc: 1.0000
 97/101 [=========================>..] - ETA: 0s - loss: 1.7117e-04 - acc: 1.0000
 98/101 [==========================>.] - ETA: 0s - loss: 1.6981e-04 - acc: 1.0000
 99/101 [==========================>.] - ETA: 0s - loss: 1.6898e-04 - acc: 1.0000
100/101 [==========================>.] - ETA: 0s - loss: 1.6868e-04 - acc: 1.0000
102/101 [==============================] - 9s 88ms/step - loss: 1.6790e-04 - acc: 1.0000 - val_loss: 0.0356 - val_acc: 0.9800
```

Figure 3. Part of last epoch

Then I test the program in real-time, which means I speak 10 numbers through microphone continuously, and count the error rate. The performance is great!

The output of the program is shown in Figure 4. We can see that no error occurs. I test the program for multiple times, the error rate keeps low. However, the error rate may increase if there is some background noise.

```
Found 1 images    Found 1 images
0                 5
Found 1 images    Found 1 images
1                 6
Found 1 images    Found 1 images
2                 7
Found 1 images    Found 1 images
3                 8
Found 1 images    Found 1 images
4                 9
```

Figure 4. Output of the program

References

[1] https://en.wikipedia.org/wiki/Speech_recognition

[2] https://github.com/libphy/which_animal

[3] https://github.com/pannous/tensorflow-speech-recognition

[4] https://yerevann.github.io/2016/06/26/combining-cnn-and-rnn-for-spoken-language-identification/