Introduction to Artificial Intelligence

Homework Report #3

Junghun Lee
Economics / BigData Science / Japanese Culture
December 4, 2022

Convolutional Recurrent Neural Network

The *Convolutional Recurrent Neural Networks* is the combination of two of the most prominent neural networks. The **CRNN** (convolutional recurrent neural network) involves **CNN**(convolutional neural network) followed by the **RNN**(Recurrent neural networks). The proposed network is similar to the CRNN but generates better or optimal results especially towards audio signal processing.

In this project, I've classified voice data called **ESC-50**. The ESC-50 data consists of approximately 2000 data, 50 classes of dog, baby crying, rain, water, and so on. To classify these data, we need to first look at the characteristics of the data. I took a quick look at the data before constructing a neural network model.

```
metadata.head()
```

	filename	fold	target	category	esc10	src_file	take
0	1-100032-A-0.wav	1	0	dog	True	100032	A
1	1-100038-A-14.wav	1	14	chirping_birds	False	100038	A
2	1-100210-A-36.wav	1	36	vacuum_cleaner	False	100210	A
3	1-100210-B-36.wav	1	36	vacuum_cleaner	False	100210	B
4	1-101296-A-19.wav	1	19	thunderstorm	False	101296	A

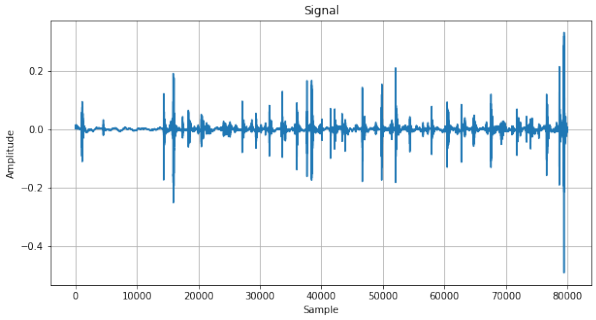
The table shown above shows information about the data contained in the ESC-50. The file name, target, category, and other information are recorded. In order to take a good look at the category, I checked the unique values of classes in the following way.

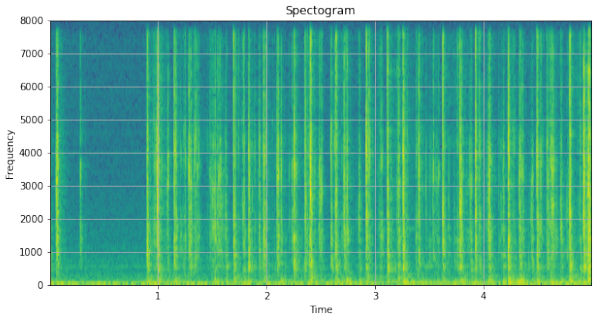
```
metadata['category'].unique()
```

```
array(['dog', 'chirping_birds', 'vacuum_cleaner', 'thunderstorm',
       'door_wood_knock', 'can_opening', 'crow', 'clapping', 'fireworks',
       'chainsaw', 'airplane', 'mouse_click', 'pouring_water', 'train',
       'sheep', 'water_drops', 'church_bells', 'clock_alarm',
       'keyboard_typing', 'wind', 'footsteps', 'frog', 'cow',
       'brushing_teeth', 'car_horn', 'crackling_fire', 'helicopter',
       'drinking_sipping', 'rain', 'insects', 'laughing', 'hen', 'engine',
       'breathing', 'crying_baby', 'hand_saw', 'coughing',
       'glass_breaking', 'snoring', 'toilet_flush', 'pig',
       'washing_machine', 'clock_tick', 'sneezing', 'rooster',
       'sea_waves', 'siren', 'cat', 'door_wood_creaks', 'crickets'],
      dtype=object)
```

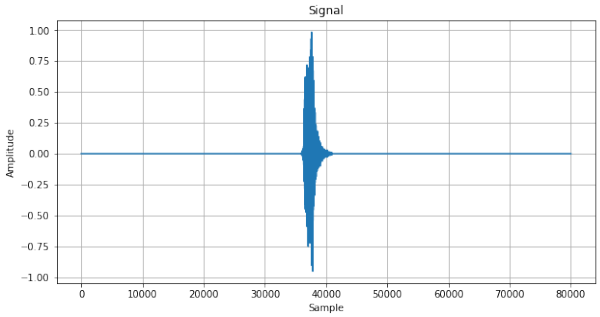
Unique values of category

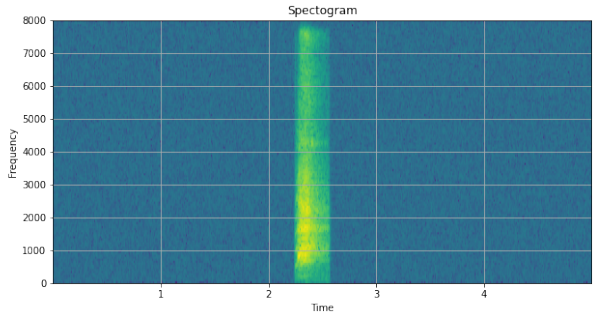
In this way, we were able to see what classes were in the data. Use a few samples to verify whether the sound source data is stored well.



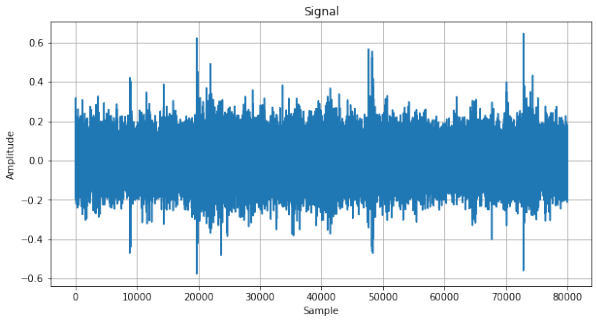


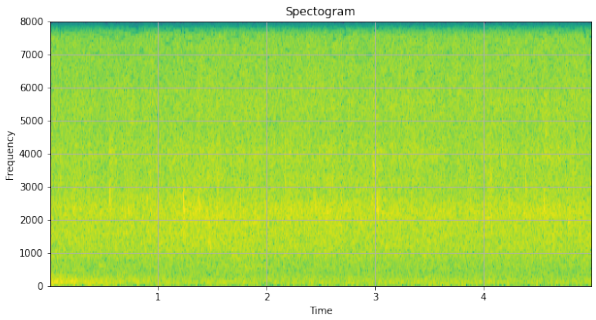
Find wav file as file name “1-137-A-32.wav"





Find wav file using index ‘dog’





Find wav file using index ‘rain’

```
In [26]: import librosa
         import librosa.display
         import IPython.display as display

         display.Audio(fname)
```

Out[26]: ▶ -0:05

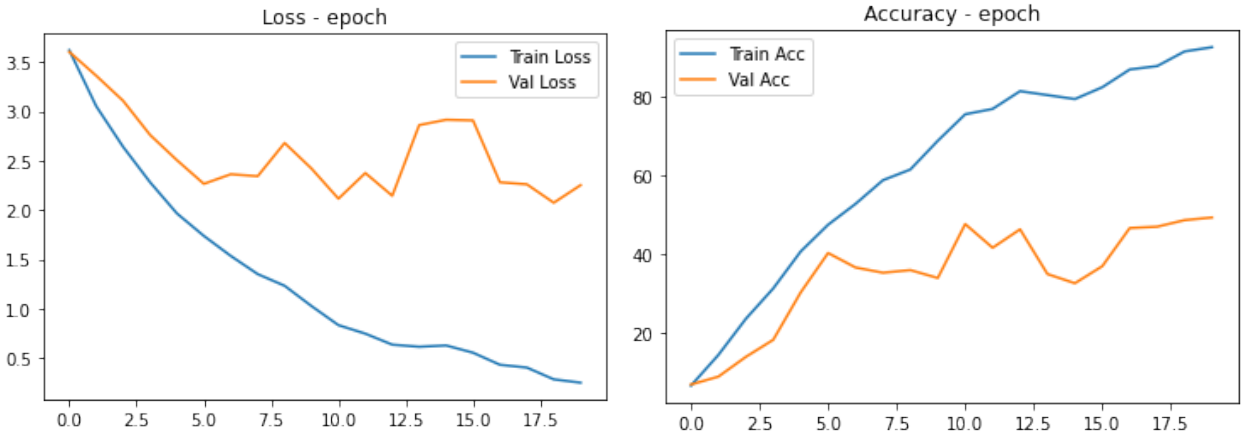
As a result of checking the **spectogram** as above, it was confirmed that it was printed normally. Now that we've confirmed that the data is normal, let's go make a neural network now!

Using *librosa library*, we can play what is voice data

Model Construction

For voice classification, I envisioned a **CRNN model** using **CNN** and **GRU**. I thought about which one to use between LSTM and GRU, but chose GRU because I thought it would be better to use a network not covered in class. CNN networks have built seven layers for better learning. In addition, the result of CNN went straight to the input of the GRU layer, so I implemented CRNN with only a single Class.

Next, I will talk about the *hyper-parameter* that I decided. The learning rate was set to 0.001, and the epoch was set to 20. There were 2,000 datasets, and even less than 1,000 learning data except for the validation set, so I was inevitably cautious about increasing the learning speed. I've supplemented the shortcomings by increasing the epoch. Batch size is specified as 64. Following graph is loss, accuracy of train, validation datas with each epochs. I found that the loss is constantly decreasing for the learning data, but at some point, the loss is not decreasing the validation data.



Finally, I saved the model and measured the performance for the test set, and the accuracy was 50.33%. That's more than 50% of the benchmark, which is a pretty good model!