INHA University

ICE4101 AI Application System

Final Project Report

Instrument Classification Using Pytorch

12161725 – Duckhyung Ryu (류덕형)

Table of Contents

Introduction

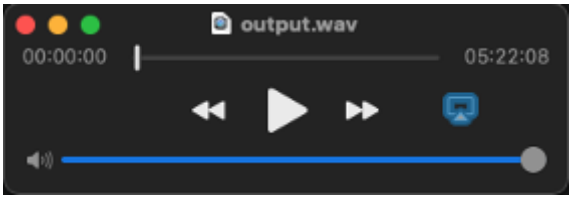
When listening to music, people had a curiosity of knowing what instrument made the sound they hear and sometimes it would be a game to bet. Moreover, it is important not just for fun but also for further music tasks such as identifying. However, as more and more varieties of instruments with digital instruments are used nowadays, it's getting harder and harder to win. AI models, especifically CNN architectured classifiers, are performing well on vision areas and showing extraordinarily high accuracy when classifying a bunch of images. I noticed the fact that every sound can be transformed into an image called spectrogram that can express all the features of the specific sound. Therefore I expected to be able to apply the fancy technologies used in the vision area to sounds as well, so I built a CNN architectured model to classify what instrument's sound is the input. As a result, if I put 2 seconds of a single instrument's sound in, then I can get an output which is the instrument's number among 174 MIDI virtual instruments list.

All codes are available from <https://github.com/doduck210/Instrument_Classifier>

Data Preprocessing

1. Gathering Data

It is realistically hard to get real sounds of instruments manually in a finite period, so I used MIDI standard virtual instruments. I got soundfont data of virtual instruments from MuseScore (MuseScore_General.sf3) and using the FluidSynth library, I played it and saved it as '.wav' format. In detail, I played 50 notes of 128 instruments (note number 38-87) and 46 drum sounds for 2 seconds each to make the model work regardless of sound's pitch. The source code I used for this is "gathering.cpp" and by running this with the soundfont data, I can get a long .wav file below.



.wav file which has all the virtual instrument's sounds in series.

The list of instruments is general MIDI 1-128 and MIDI 9th channel 35-81. The specific names of instruments are below.

Piano [edit]
- 1 Acoustic Grand Piano
- 2 Bright Acoustic Piano
- 3 Electric Grand Piano
- 4 Honky-tonk Piano
- 5 Electric Piano 1
- 6 Electric Piano 2
- 7 Harpsichord
- 8 Clavi

Chromatic Percussion [edit]
- 9 Celesta
- 10 Glockenspiel
- 11 Music Box
- 12 Vibraphone
- 13 Marimba
- 14 Xylophone
- 15 Tubular Bells
- 16 Dulcimer

Organ [edit]
- 17 Drawbar Organ
- 18 Percussive Organ
- 19 Rock Organ
- 20 Church Organ
- 21 Reed Organ
- 22 Accordion
- 23 Harmonica
- 24 Tango Accordion

Guitar [edit]
- 25 Acoustic Guitar (nylon)
- 26 Acoustic Guitar (steel)
- 27 Electric Guitar (jazz)
- 28 Electric Guitar (clean)
- 29 Electric Guitar (muted)
- 30 Overdriven Guitar
- 31 Distortion Guitar
- 32 Guitar Harmonics

Bass [edit]
- 33 Acoustic Bass
- 34 Electric Bass (finger)
- 35 Electric Bass (pick)
- 36 Fretless Bass
- 37 Slap Bass 1
- 38 Slap Bass 2
- 39 Synth Bass 1
- 40 Synth Bass 2

Strings [edit]
- 41 Violin
- 42 Viola
- 43 Cello
- 44 Contrabass
- 45 Tremolo Strings
- 46 Pizzicato Strings
- 47 Orchestral Harp
- 48 Timpani

Ensemble [edit]
- 49 String Ensemble 1
- 50 String Ensemble 2
- 51 Synth Strings 1
- 52 Synth Strings 2
- 53 Choir Aahs
- 54 Voice Oohs
- 55 Synth Voice
- 56 Orchestra Hit

Brass [edit]
- 57 Trumpet
- 58 Trombone
- 59 Tuba
- 60 Muted Trumpet
- 61 French Horn
- 62 Brass Section
- 63 Synth Brass 1
- 64 Synth Brass 2

Reed [edit]
- 65 Soprano Sax
- 66 Alto Sax
- 67 Tenor Sax
- 68 Baritone Sax
- 69 Oboe
- 70 English Horn
- 71 Bassoon
- 72 Clarinet

Pipe [edit]
- 73 Piccolo
- 74 Flute
- 75 Recorder
- 76 Pan Flute
- 77 Blown bottle
- 78 Shakuhachi
- 79 Whistle
- 80 Ocarina

Synth Lead [edit]
- 81 Lead 1 (square)
- 82 Lead 2 (sawtooth)
- 83 Lead 3 (calliope)
- 84 Lead 4 (chiff)
- 85 Lead 5 (charang)
- 86 Lead 6 (voice)
- 87 Lead 7 (fifths)
- 88 Lead 8 (bass + lead)

Synth Pad [edit]
- 89 Pad 1 (new age)
- 90 Pad 2 (warm)
- 91 Pad 3 (polysynth)
- 92 Pad 4 (choir)
- 93 Pad 5 (bowed)
- 94 Pad 6 (metallic)
- 95 Pad 7 (halo)
- 96 Pad 8 (sweep)

Synth Effects [edit]
- 97 FX 1 (rain)
- 98 FX 2 (soundtrack)
- 99 FX 3 (crystal)
- 100 FX 4 (atmosphere)
- 101 FX 5 (brightness)
- 102 FX 6 (goblins)
- 103 FX 7 (echoes)
- 104 FX 8 (sci-fi)

Ethnic [edit]
- 105 Sitar
- 106 Banjo
- 107 Shamisen
- 108 Koto
- 109 Kalimba
- 110 Bag pipe
- 111 Fiddle
- 112 Shanai

Percussive [edit]
- 113 Tinkle Bell
- 114 Agogô
- 115 Steel Drums
- 116 Woodblock
- 117 Taiko Drum
- 118 Melodic Tom
- 119 Synth Drum
- 120 Reverse Cymbal

Sound Effects [edit]
- 121 Guitar Fret Noise
- 122 Breath Noise
- 123 Seashore
- 124 Bird Tweet
- 125 Telephone Ring
- 126 Helicopter
- 127 Applause
- 128 Gunshot

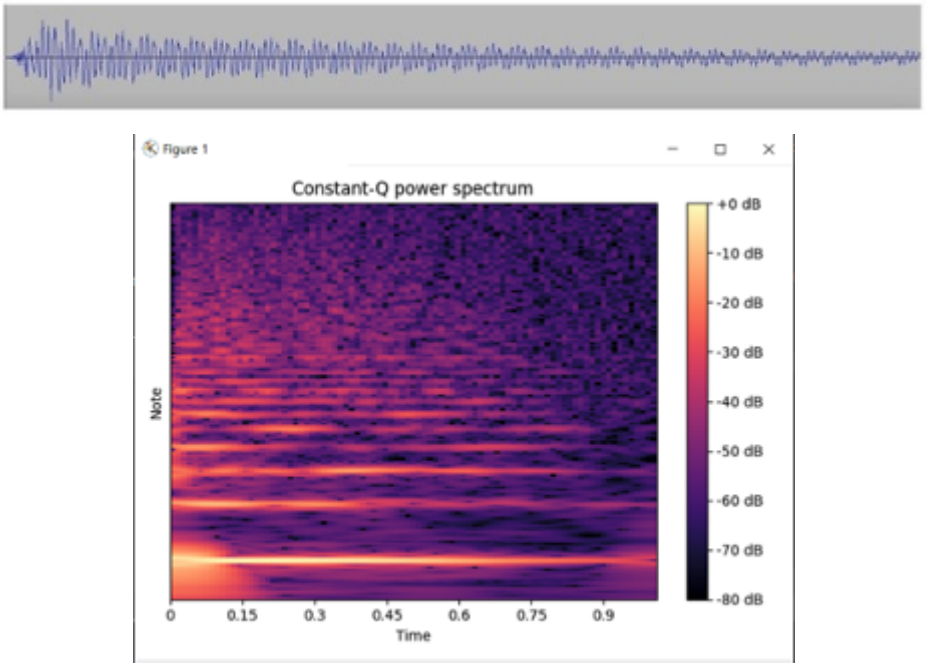
Midi Key	DRUM SOUND	Midi Key	DRUM SOUND
35	Acoustic Bass Drum	59	Ride Cymbal 2
36	Bass Drum 1	60	Hi Bongo
37	Side Stick	61	Low Bongo
38	Acoustic Snare	62	Mute Hi Conga
39	Hand Clap	63	Open Hi Conga
40	Electric Snare	64	Low Conga
41	Low Floor Tom	65	High Timbale
42	Closed Hi-Hat	66	Low Timbale
43	High Floor Tom	67	High Agogo
44	Pedal Hi-Hat	68	Low Agogo
45	Low Tom	69	Cabasa
46	Open Hi-Hat	70	Maracas
47	Low-Mid Tom	71	Short Whistle
48	Hi-Mid Tom	72	Long Whistle
49	Crash Cymbal 1	73	Short Guiro
50	High Tom	74	Long Guiro
51	Ride Cymbal 1	75	Claves
52	Chinese Cymbal	76	Hi Wood Block
53	Ride Bell	77	Low Wood Block
54	Tambourine	78	Mute Cuica
55	Splash Cymbal	79	Open Cuica
56	Cowbell	80	Mute Triangle
57	Crash Cymbal 2	81	Open Triangle
58	Vibraslap		

reference : https://en.wikipedia.org/wiki/General_MIDI

128 instruments (left), 46 drums (right)

2. Data Preprocessing

It is hard to analyze the sounds by their waveform directly. Also, because I tried to build a CNN architecture that works well with vision tasks to win this sound classification, I transformed all the sounds to the spectrogram images using the library called librosa. The data sound is independent every 2 seconds, so I made independent spectrograms for every 2sec of the data. This source code is “toSpectro.py”.



Original Waveform (up) and Spectrogram (Down) of the same piano sound

Spectrogram is an image that expresses both waveform in the time domain and the spectrum in the frequency domain at the same time. Simply, X-axis represents the time, Y-axis represents the frequency, the colour(‘s concentration) means the amplitude in X time at the Y frequency.
Above the images are a waveform and spectrogram of the same piano sound, but obviously, it is hard to get meaningful characteristics from the original waveform image. In Contrast, the spectrogram shows us a lot more information so the sounds are able to be analyzed only with this image.

Additionally, I made a data argument here, when transforming data as a spectrogram, I added some noisy versions white noise. I used list of white noise whose amplitude is (0,1e-4,1e-3) for 128 instruments and (0,1e-5,1e-4,1e-3) * range(7) for 46 drums. I argumented more about the drum sounds because of their relative lack of quantity caused by absence of notes.
Furthermore, to deal with this image conveniently in programming, I saved its data as a numpy array with labeling. The label has its instrument number and note, but the instrument number here is just a sequence of numbers from 0-173 which is the list of the instruments in order.

As a result, the total number of data I made spectrograms in the numpy array format is 20,488. And each image size is [168][87].

$$num\ of\ data\ =\ (instruments\ *\ notes\ *\ argumentation)\ +\ (drums\ *\ argumentation)$$
$$=\ (128\ *\ 50\ *\ 3)\ +\ (46\ *\ 28)\ =\ 20,488$$

Model Design

Designing a CNN Model that can discern instruments was the main work of this project. I built it from scratch without using well known models in image classification, so had to go through countless malfunctions and trials. The final model’s architecture is below, the code file is “instrument_classifier.ipynb” and this code also includes training and result parts.

Final Model's architecture

It has 2 layers using convolution filters and only one FC layer at the end to make an output class list. Each convolution layer consisted of a convolution filter, batch normalization, ReLU activation function, and max pool filter. 10 channels of (23x4) conv filters are used on the first layer, 3 channels of(6*3) conv filters are used on the second layer.

At first, I referred to many other CNN classification projects and tried to follow their general architecture that used more than 50 channels of conv filters and had 3 conv layers. But that didn't work with my data. The loss didn't go down at all, it seemed it was because of its complexity, the gradient flow was stuck. Therefore, I started to simplify the model so much. I tried to change the number of channels and the architecture heuristically by changing something and trying training 5 epochs, and changing something again. And at the end, I got my best heuristic architecture above.

Convolution filter sizes are not square, which is different from common shapes in image classification models. The reason is on the input data which is spectrogram images. If I see one of the spectrogram images in the data preprocessing section, it shapes like it's stretched out horizontally. That means it has little information about changes on the x-axis so no need to be sensitive about x-axis. Therefore, I edit the filter shape that is long horizontally, and could see the increase of training speed and accuracy.

Training and Result

At first, I shuffled the data and split it into 8:2 ratios a and b. Also, I used mini batches that bind 256 data each.

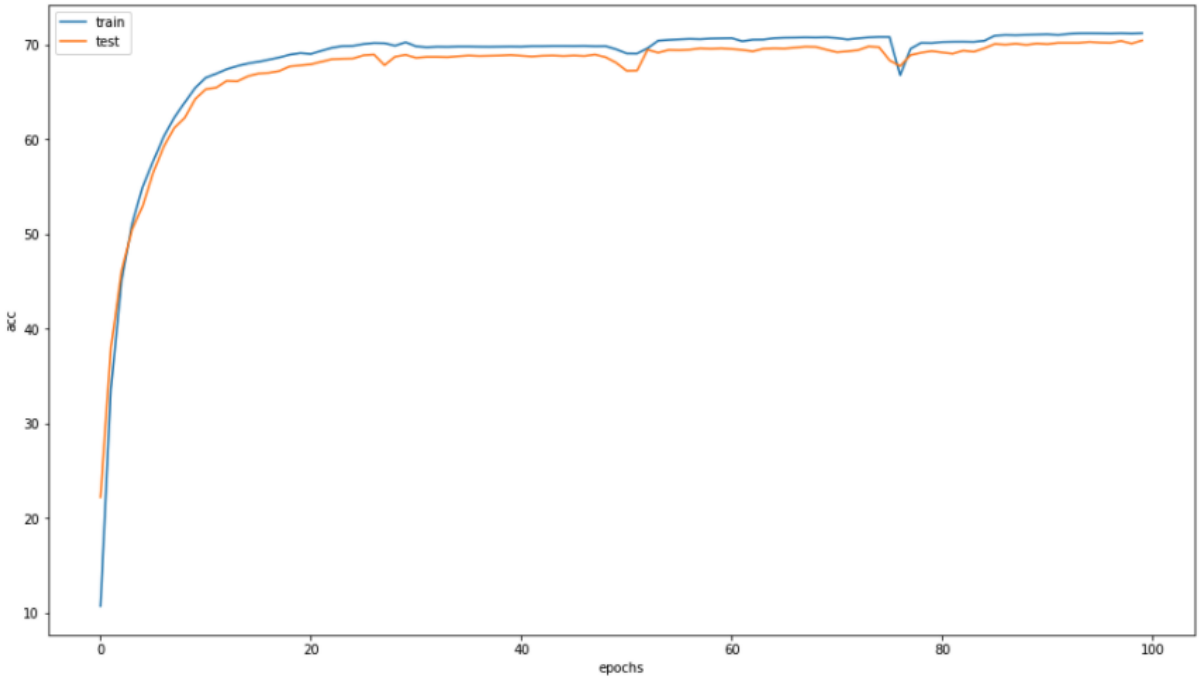
```
ridx = list(range(len(x))) # x is data
random.shuffle(ridx)
```

```
test_size=len(ridx)//5 # train:test=8:2
test_ridx, train_ridx=ridx[:test_size],ridx[test_size:]
```

Code to shuffle and split data

I ran 100 epochs of training and tests and checked its performance based on test accuracy and if there is overfitting based on the difference between training and test accuracy. I thought 100 epochs would be enough to train this model because I could see no meaningful progress after 50-60 epochs.

In the training section, I used Adam optimizer to train models by changing inside weights, and printed loss 7 times per epoch. Because I used a softmax in the outcome of the model, I used cross entropy loss as a criterion. At the end of the epoch, it prints accuracy by calculating the number of correct answers divided by the number of data. I did the same thing in the test section too but there is just no optimizer cause the test data is rather for validation purposes not for further training.



Training and Test Accuracy during 100 Epochs

```
e :  99 loss :  tensor(4.4774, device='cuda:0', grad_fn=<NllLossBackward>)
train acc :  0.7122201207979989
test acc :  0.704417866731755
```

Last Epoch's Loss, Train Acc and Test Acc

As a result, I got 70% accuracy on test data and during 100 epochs of training, there was about 1-2% of difference between train and test accuracy which means there was no overfitting problem. At the very early part of the graph it seems test accuracy is a little bit

higher than training's, but it is just because training accuracy is measured during training but test accuracy is done after training at the epoch.

Reflection:

The final result of my model was 70% to discern an instrument from 174. Even though it's not very precise, if I consider the number of classes which is 174, it works well. Because generally, tutorial level classification problems deal with less than 20 classes so they started from more than 5% of acc, but this project started from 1/174 which is about 0.5-0.6%.

When building the model, I could try some fancy techniques. At first, batch normalization was a brilliant game changer, it makes training faster and raises acc up about 10%. I've tried using tanh as an activation function and Xavier initialization instead of using ReLU. That doesn't make a big difference in acc though, I just ended up using ReLU again because the ReLU function is more simple to calculate. I've tried using dropout too but because I had no overfitting problem, it dropped accuracy, so I got rid of it. At the very beginning when I made the model, I tried using only neural networks without convolution filters but that was meaningless at all because I could get less than 5% of accuracy. So I could see when dealing with images(spectrogram), CNN architecture is almost necessary.

To make further performance increase, I think analyzing the result using a confusion matrix can be helpful. Because errors can happen lots in a limited group, analyzing results and knowing where errors happen mainly can make further performance increase by dealing with that group separately for example.
One more suggestion is trying using LSTM. Only to use CNN, I should have done the preprocessing that converts sounds to spectrogram images. Even though it seems there was no problem, basically more processing stages means more possibilities to occur errors. Therefore using LSTM and classifying sounds by raw sounds data can be worth to try I guess.

References:

[1] K. Choi, "Voice/Music Signal + Machine Learning Guide for Beginners", Personal Web, Accessed May, 2021 [Online], Available: <http://keunwoochoi.blogspot.com/2016/01/blog-post.html>

[2] My Big-O is log-x (personal nickname), "Classification of instrument sounds using deep learning", Personal Web, Accessed on: May, 2021 [Online], Available: <https://bab2min.tistory.com/642>

[3] Prof. Sungeun. Hong, "ICE4101 AI Application System Lectures", INHA School of Engineering, Accessed Mar-Jun 2021

[4] K. Sanjeevan, "crnn-audio-classification", Personal Web, Accessed May, 2021 [Online], Available: <https://github.com/ksanjeevan/crnn-audio-classification>