Audio Tagging Challenge

Machine Learning II

Professor: Amir Jafari

Madison Turano, Jingjing Xu

Outline

Introduction **3**

Description of the Project **3**

Dataset 3

Deep Learning Network and Training Algorithm 3

Experimental Setup **5**

Data for Training and Testing: 5

Results **6**

Time Domain 6

Frequency Domain 7

Summary and Conclusions **20**

References **22**

Appendix **23**

Introduction

Facial and audio recognition have become a large part of our lives. We use our voices to control smart devices, such as home appliances, and sometimes even the home. We get music recommendations at digital storefronts based on what our individual preferences are, while almost every social media platform automatically recognizes faces in photos. We are very curious to find out how those algorithms work, and after a semester of studying deep learning, see if we can develop a mechanism of our own.

So far what we have been working on is image tagging. However, in order to apply what we learned on a broader level, as well as deepen our understanding on the algorithm, we chose to challenge ourselves with a project about audio tagging. The greatest difference between the audio tagging and image tagging is that we have an extra dimension of time to consider. Therefore, we designed a network with LSTMs to train the network, and classify the audio clips with as high accuracy as possible.

Description of the Project

Dataset

The "Medley" dataset we use consists of 21571 3-second audio clips, each with a single label, distributed among 8 classes: clarinet, distorted electric guitar, female singer, flute, piano, tenor saxophone, trumpet, and violin. Each clip on the csv file which the dataset provides has already been prelabeled as training, validation, and test. The training set size is 5841, the validation set size is 3494, and the test set size is 12236.

Deep Learning Network and Training Algorithm

We designed our network with LSTMs since the data we extracted from the dataset is a time series data. We designed the network with Pytorch framework, and coded the model with nn module. Two layers of LSTMs were put into our model, with one layer of dropout 0.1 to prevent overfitting. One extra linear layer was used to get the output we need. We used cross entropy as our loss function because it combined softmax and NLL loss functions, and we hoped to get a more complete result from it. Adam, Adadelta, and SGD were all tried as optimizers, and we hoped to find the one with the highest accuracy.

The data we used was two fold. Since we needed to extract the data from the audio clip files, we first considered the extraction in the sense of time domain using wav file. To make the prediction more accurate, we also considered extracting other features in frequency domain. By using librosa package, we were able to extract the Mel frequency cepstral coefficients (MFCCs), spectral centroid and spectral contrast data from each clip. Theoretically, this data should train the network much better since the frequency domain captures the differences of the classes more distinctly.

To understand the concept of MFCCs, we need to have a basic knowledge of Fourier Transform. Simply put, a Fourier Transform extracts sin and cos waves with different features (i.e. amplitude, frequency, etc) from a composed wave, and expresses it in a frequency domain:

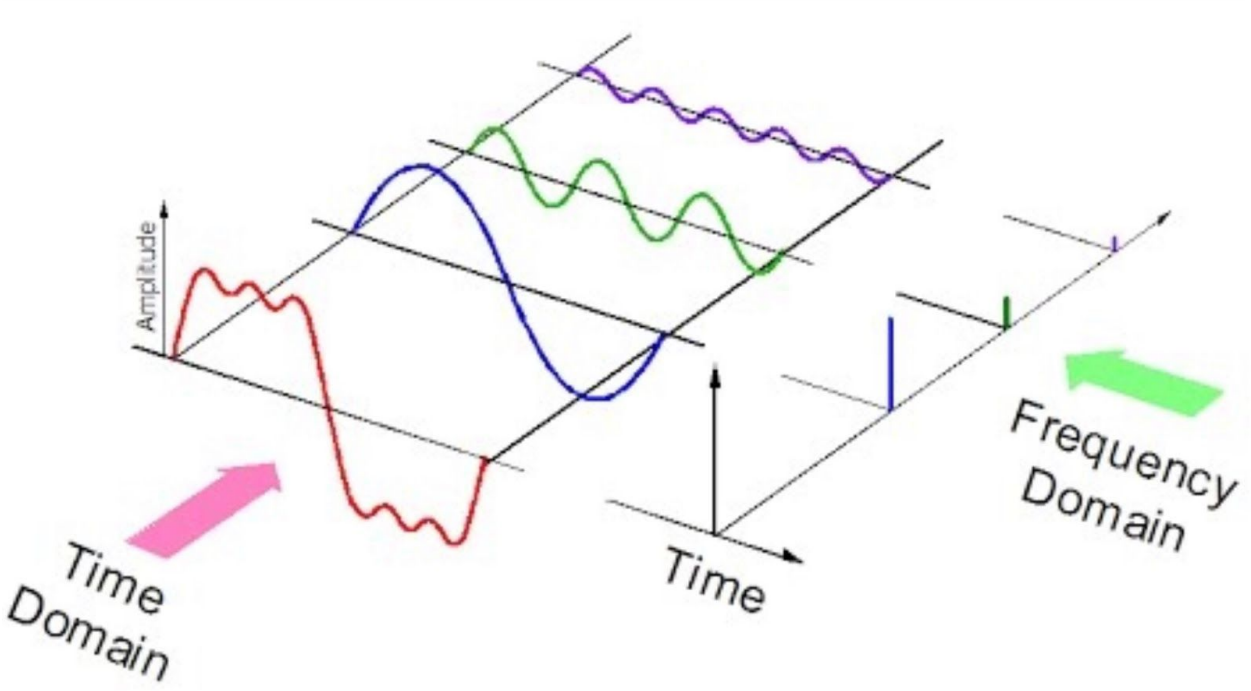

Figure 1: Fourier Transformation

It is used to achieve MFCCs:

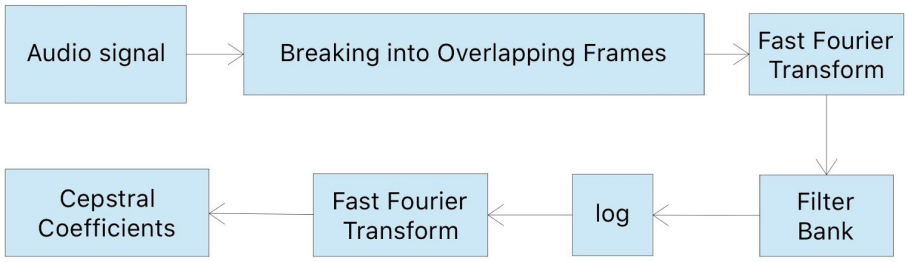

Figure 2: MFCC archiving

For the sake of simplicity, we can view the overlapping frames and filter bank as filters. We also utilize spectral centroid, which describes the "brightness" of the sound, and the spectral contrast, which measure the "level difference between the peak and valleys in the spectrum"(Nogueira et al., Optimization of a Spectral Contrast Enhancement Algorithm for Cochlear Implants Based on a Vowel Identification Model). Therefore, we use MFCCs, spectral centroid and spectral contrast as the data in frequency domain for our network to analyze.

Experimental Setup

Data for Training and Testing:

To train the tagging algorithm, we split the data into training, validation, and test subsets. In order to format and load the data to the tagging algorithm, we created two functions to load and split the audio files. We then created a MedleyDataset class to transform the data into a format that Dataloader and LSTM layer can process.

Once we loaded and transformed the data, we ran the training subset through the tagging algorithm. At first the algorithm had a small accuracy rating, so we re-ran the first data transformation through librosa to extract MFCC, spectral centroid and spectral contrast for better results.

We judged the performance of our network by the overall accuracy and accuracy of each class of the prediction on validation set, and the time used to achieve that.

We used mini batches to train our network, since we found out from exam 1 that it is a compromise between time and accuracy. As our training set size was 5841, we experimented around batch sizes of 30, 50, and 100. 100 seems to have a better result when we are doing the analysis in time domain. Hence, we use the same batch size on the analysis in frequency domain.

Results

Time Domain

When we first ran the algorithm, the algorithm began with a loss of 1.5 and ended with a loss around 0.01. After running the model multiple times, the loss (or performance index) decreased significantly. Below is a graph of loss over training time:

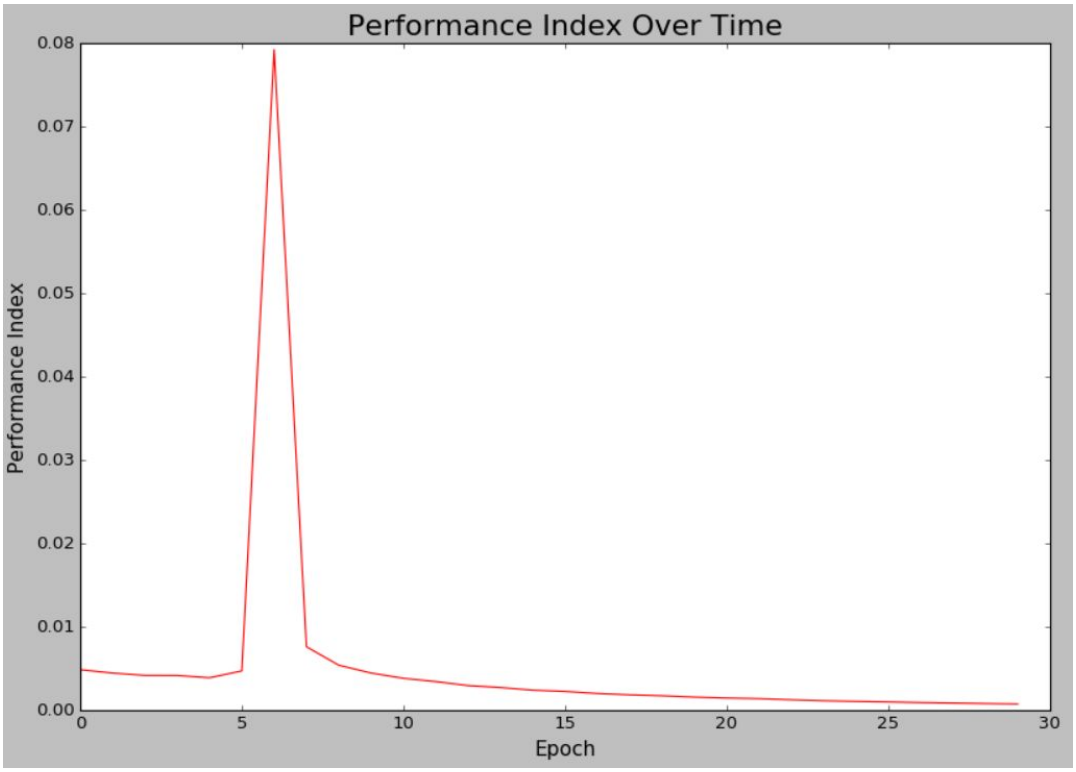

Figure 3: Loss over time

Based on the above graph, the loss started around 0.005 and ended near 0.00, or no error. However, there's a spike in loss value between the fifth and eight epoch, with loss almost 20 times that of previous epochs.

However, based on the accuracy function, the algorithm has an accuracy rating of 24%. Further, the confusion matrix is filled with 0s:

```
[0. 0. 0. 0. 0. 0. 0. 0.]
[0. 0. 0. 0. 0. 0. 0. 0.]
[0. 0. 0. 0. 0. 0. 0. 0.]
[0. 0. 0. 0. 0. 0. 0. 0.]
[0. 0. 0. 0. 0. 0. 0. 0.]
[0. 0. 0. 0. 0. 0. 0. 0.]
[0. 0. 0. 0. 0. 0. 0. 0.]]
```

Figure 4: Confusion matrix

Given the low loss value during training, it seems odd that the algorithm would have a low accuracy rate and a confusion matrix filled with 0s. There is either an issue in our algorithm or an issue in our accuracy code, or possibly an issue with both.

Frequency Domain

When we were analyzing the time domain, we got extremely low loss (almost 0s) every epoch, but the overall accuracy hovered around 30%. Therefore, we believe there are three problems:

1. The train set is relative small
2. The information we extract is not very useful
3. The program might be over or under fitting.

We tried to adjust the dropout rate, learning rate and the number of neurons to prevent over or under fitting, and improved the second problem with the analysis in frequency domain. We also tried to redivide the size of train set to make it bigger, but it doesn't seem to have a significant impact on the time domain data.

Hence we started the analysis in the frequency domain.

The analysis in frequency domain gets much better results. It partially confirms that the reason why we have such low accuracy in time domain analysis is because we didn't extract enough useful data for the network to train.

While training the network, the loss starts from 1.39, and decreased to 0.19 over 50 epochs and batch number 100. The overall accuracy is 78% . We acquired the loss of training time image, which seems to have a clear trend of decreasing:

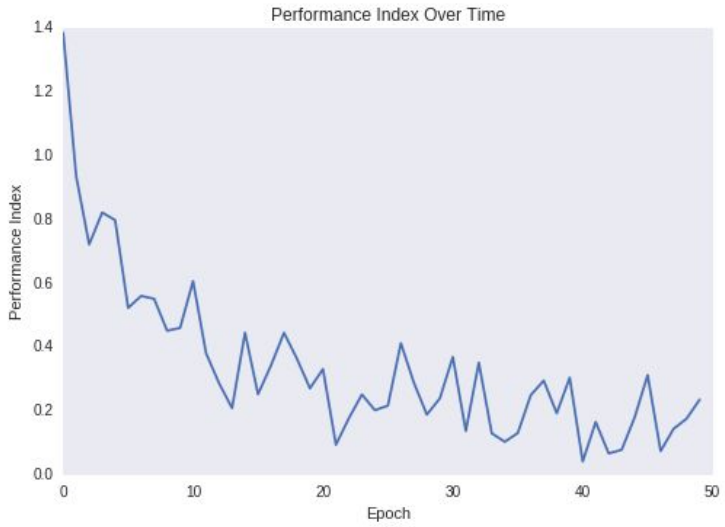

Figure 5-1. Epoch 50, Batch Size 100, Learning Rate 0.0001, Dropout 0.1, Adam, LSTM Layer 2

The confusion matrix (as rows represent the prediction, while columns represent the actual) is as follows:

```
Accuracy of the network on the 3494 validation audio clips: 78 %
Accuracy of clarinet : 48 %
Accuracy of distorted electric guitar : 83 %
Accuracy of female singer : 75 %
Accuracy of flute : 30 %
Accuracy of piano : 89 %
Accuracy of tenor saxophone : 24 %
Accuracy of trumpet : 84 %
Accuracy of violin : 86 %
```

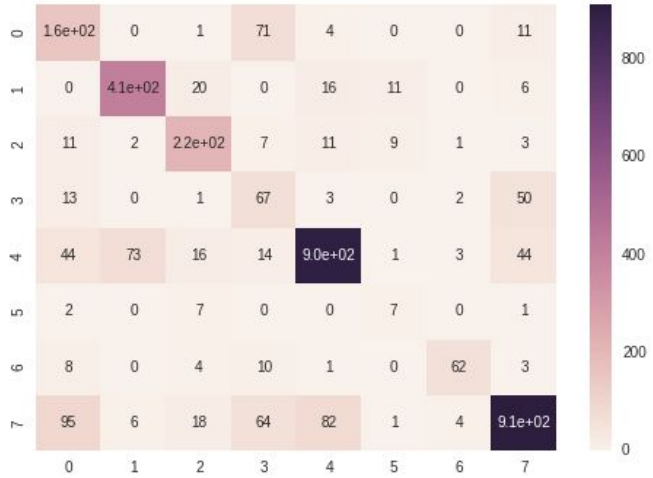

Figure 5-2. Epoch 50, Batch Size 100, Learning Rate 0.0001, Dropout 0.1, Adam, LSTM Layer 2

We can see that the confusion matrix confirms the lower category accuracy on flute, and saxophone. The network seem to be easily confuse flute with clarinet, and saxophone with guitar

or female singer. This could result from the fact that the frequency of those pairs are indeed quite close, or it could also com from the fact that the number of clips on those categories are much smaller than, say, the piano and violin clips. For better results, other data are possibly needed. To get the optimal training result from my network, we are using this set of training parameters as my base of comparison (epoch: 50, batch size: 100, learning : 0.0001, drop out: 0.1, LSTMs layer: 2), and then adjusted the different parameters to see how the code performs.

Batchsize:
Using the same epochs, we adjusted the batch size to 50, and get the following result:

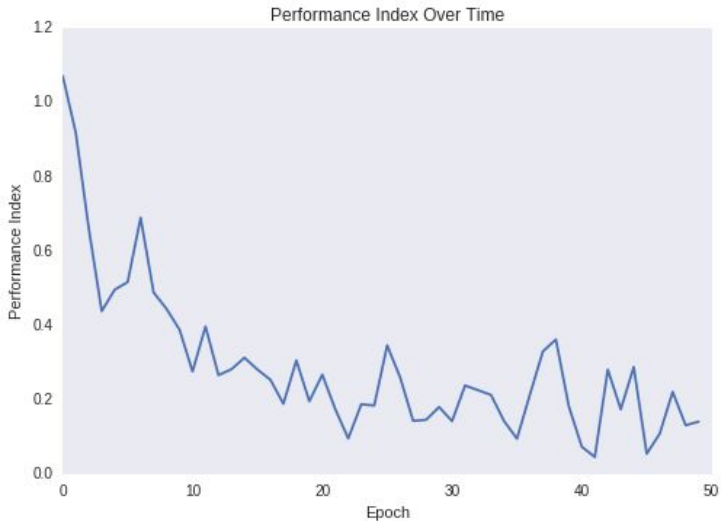

Figure 6-1. Epoch 50, Batch Size 50, Learning Rate 0.0001, Dropout 0.1, Adam, LSTM Layer 2

```
Accuracy of the network on the 3494 validation audio clips: 80 %
Accuracy of clarinet : 52 %
Accuracy of distorted electric guitar : 84 %
Accuracy of female singer : 73 %
Accuracy of flute : 33 %
Accuracy of piano : 88 %
Accuracy of tenor saxophone : 24 %
Accuracy of trumpet : 84 %
Accuracy of violin : 90 %
```

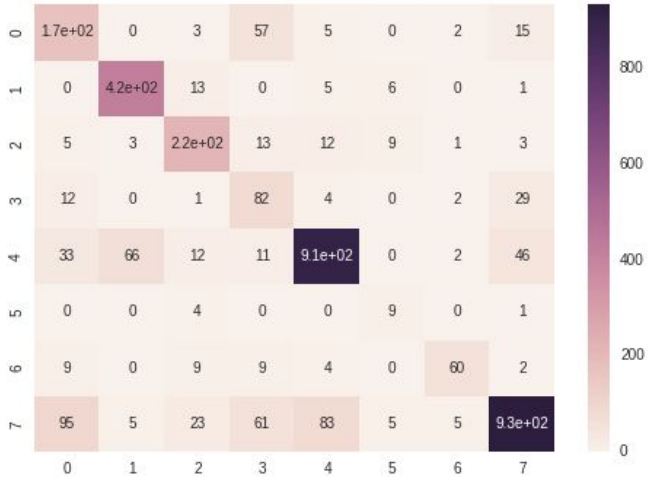

Figure 6-2. Epoch 50, Batch Size 50, Learning Rate 0.0001, Dropout 0.1, Adam, LSTM Layer 2

Then we adjusted the batch size to 20 and get:

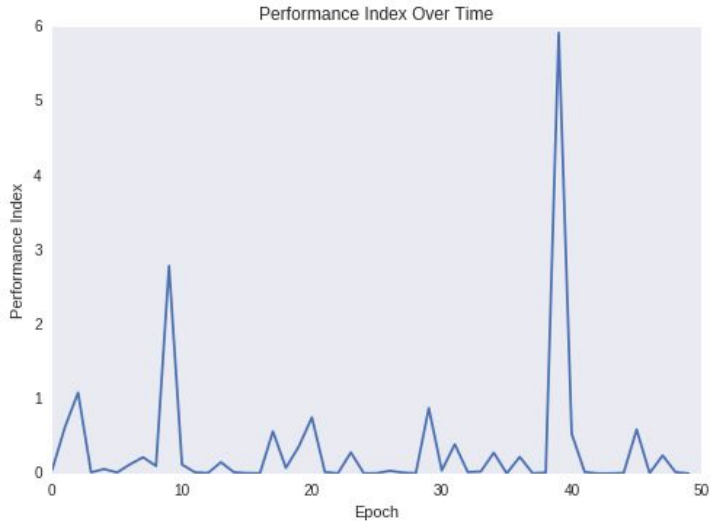

Figure 7-1. Epoch 50, Batch Size 20, Learning Rate 0.0001, Dropout 0.1, Adam, LSTM Layer 2

```
Accuracy of the network on the 3494 validation audio clips: 78 %
Accuracy of clarinet : 44 %
Accuracy of distorted electric guitar : 71 %
Accuracy of female singer : 75 %
Accuracy of flute : 29 %
Accuracy of piano : 89 %
Accuracy of tenor saxophone : 17 %
Accuracy of trumpet : 77 %
Accuracy of violin : 92 %
```

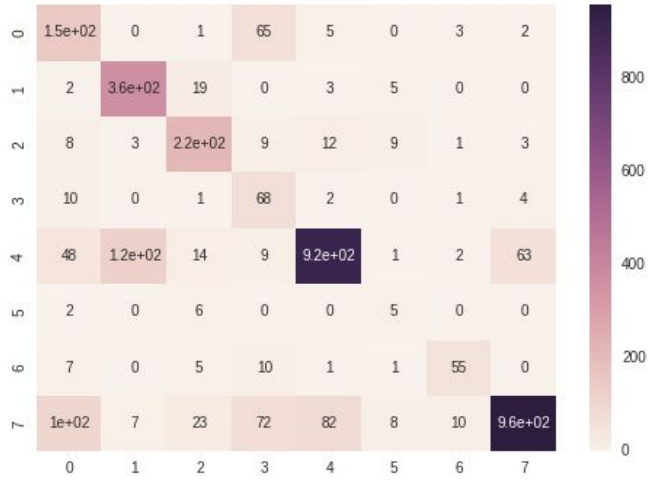

Figure 7-2. Epoch 50, Batch Size 20, Learning Rate 0.0001, Dropout 0.1, Adam, LSTM Layer 2

The performance index behaves very strangely when batch size is 20, it seems to be constantly very low, however, there are a few pikes occurring, the biggest one happens around epoch 10 and 40. It could due to the fact that the batch size is too low, and therefore it affects the training of the network, as the calculation of the gradient is closer to the individual file when batch size is smaller. The performance index of batch size 50 is quite similar to 100, with a downward trend. It seems that the accuracy peaks in the middle, where the batch size equals 50, although the difference is not that significant. It could be due to the fact that while using mini-batch, we are compromising between stochastic and gradient descent, and there would be a optimal point in the middle. We suppose 50 is closer to that optimal point.

Learning rate:

We are now changing the batch size back to 100, and trying to adjust the learning rate from 0.001. The training loss changes from 0.88 to 0.40 over 50 epochs. We can see the loss hovers over that number, this suggests that the learning rate is too high, as a result, every step the weight and bias getting updated, they changes way too much.

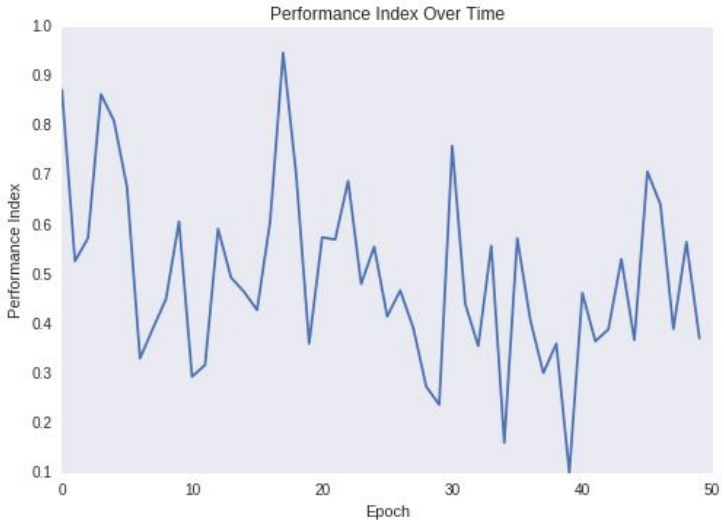

Figure 8-1. Epoch 50, Batch Size 100, Learning Rate 0.001, Dropout 0.1, Adam, LSTM Layer 2

```
Accuracy of the network on the 3494 validation audio clips: 75 %
Accuracy of clarinet : 35 %
Accuracy of distorted electric guitar : 79 %
Accuracy of female singer : 77 %
Accuracy of flute : 14 %
Accuracy of piano : 86 %
Accuracy of tenor saxophone : 17 %
Accuracy of trumpet : 69 %
Accuracy of violin : 93 %
```

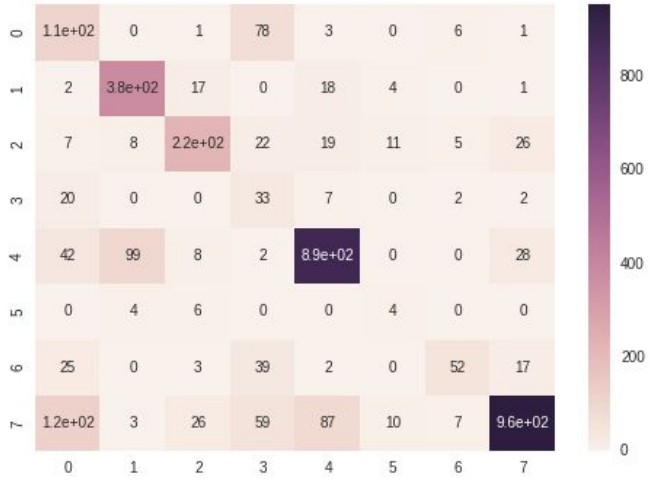

Figure 8-2. Epoch 50, Batch Size 100, Learning Rate 0.001, Dropout 0.1, Adam, LSTM Layer 2

The overall accuracy is nicely at 75%, but it could be random. Now we adjust the learning rate to 0.00001. The following result suggests the overall accuracy is at 67%, while it's actually a little bit worse than the result we acquire from the learning rate 0.001, and certainly not as good as the learning rate 0.0001. The performance index seem to enter a rapid ocillation after epoch 10 compare to the result of 0.001.

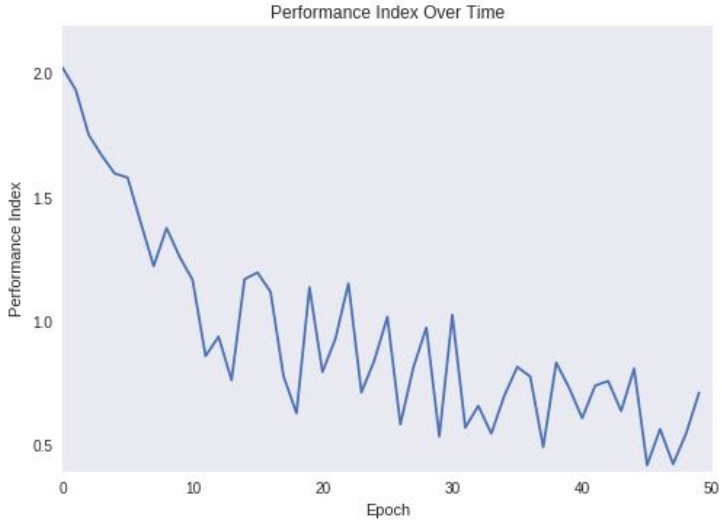

Figure 9-2. Epoch 50, Batch Size 100, Learning Rate 0.00001, Dropout 0.1, Adam, LSTM Layer 2

```
Accuracy of the network on the 3494 validation audio clips: 67 %
Accuracy of clarinet : 11 %
Accuracy of distorted electric guitar : 58 %
Accuracy of female singer : 45 %
Accuracy of flute :  0 %
Accuracy of piano : 88 %
Accuracy of tenor saxophone :  0 %
Accuracy of trumpet : 13 %
Accuracy of violin : 95 %
```

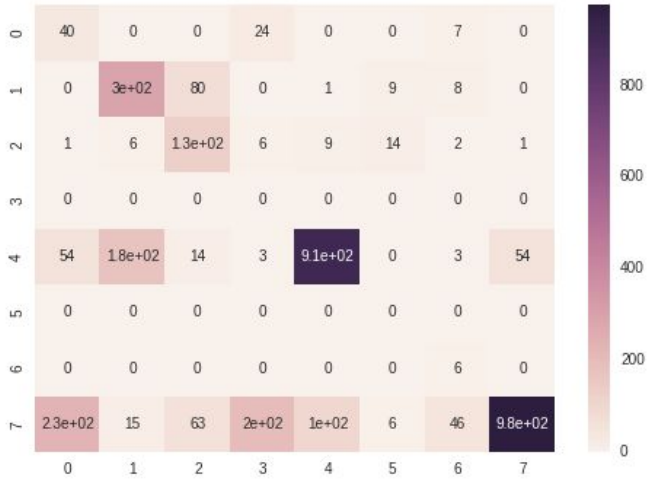

Figure 9-2. Epoch 50, Batch Size 100, Learning Rate 0.00001, Dropout 0.1, Adam, LSTM Layer 2

We can see that the learning too fast or too slow have an impact on our result; learning too fast, then there is a problem with over fitting, the loss hover around a range, and has trouble to decrease; learning too slow, it doesn't reach the result as fast.

Dropout:

Up to this point we have been using the dropout rate 0.1, and now we change it to 0.05. The result is as follows:

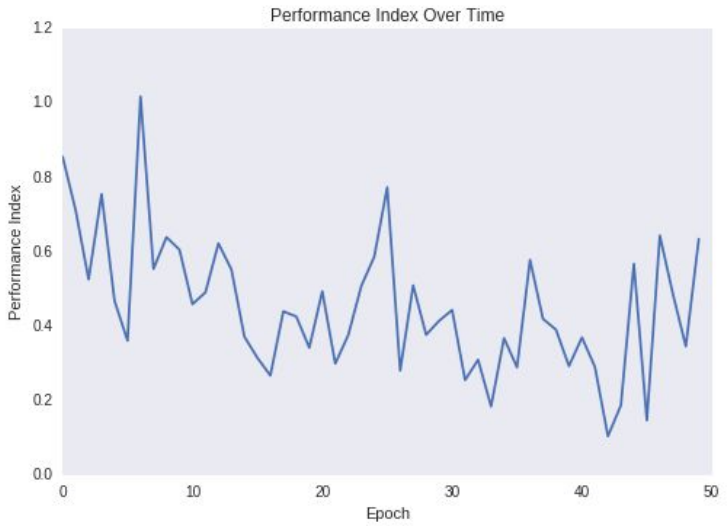

Figure 10-1. Epoch 50, Batch Size 100, Learning Rate 0.0001, Dropout 0.05, Adam, LSTM Layer 2

```
Accuracy of the network on the 3494 validation audio clips: 75 %
Accuracy of clarinet : 39 %
Accuracy of distorted electric guitar : 76 %
Accuracy of female singer : 61 %
Accuracy of flute : 27 %
Accuracy of piano : 85 %
Accuracy of tenor saxophone : 17 %
Accuracy of trumpet : 76 %
Accuracy of violin : 94 %
```

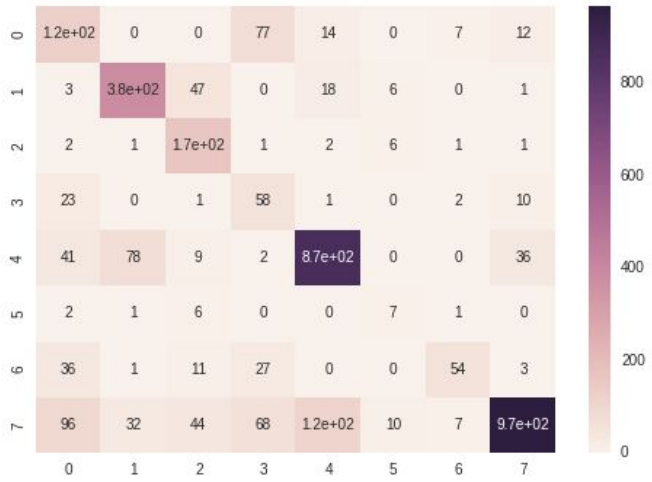

Figure 10-2. Epoch 50, Batch Size 100, Learning Rate 0.0001, Dropout 0.05, Adam, LSTM Layer 2

We can see from the confusion matrix that the code seems to mistaken the pair of flute and clarinet, saxophone and guitar/female singer, also it seems that the network confuses clarinet with piano sometimes. This agrees with the lack of significant trend in performance index. But the overall accuracy is quite good, at 75%. Then we have the results from changing the dropout rate to 0.5:

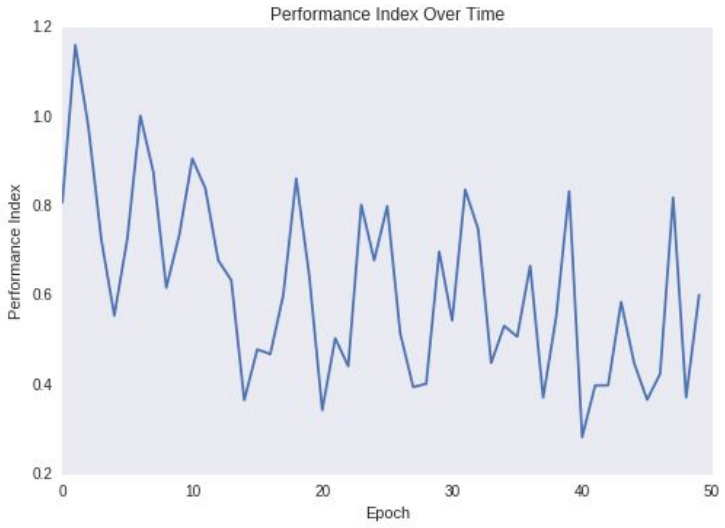

Figure 11-1. Epoch 50, Batch Size 100, Learning Rate 0.0001, Dropout 0.5, Adam, LSTM Layer 2

```
Accuracy of the network on the 3494 validation audio clips: 70 %
Accuracy of clarinet : 23 %
Accuracy of distorted electric guitar : 68 %
Accuracy of female singer : 50 %
Accuracy of flute : 15 %
Accuracy of piano : 86 %
Accuracy of tenor saxophone :  3 %
Accuracy of trumpet : 59 %
Accuracy of violin : 88 %
```

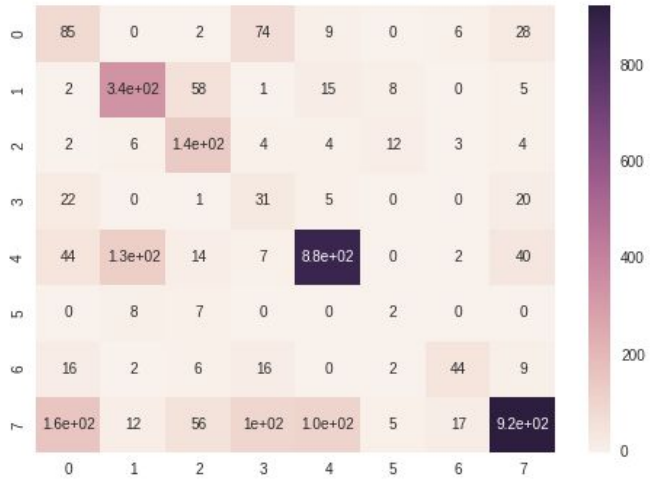

Figure 11-2. Epoch 50, Batch Size 100, Learning Rate 0.0001, Dropout 0.5, Adam, LSTM Layer 2

We see a slight decrease in overall accuracy, at 70%, but the network behaves badly not only on flute and saxophone, but also several other classes. The only 2 classes having a good result is piano and violin. Both of the dropout rate works worse than our dropout at 0.1. And if we take a look at the performance index images, we can see that there is no significant trend of loss throughout time. This might due to over or underfitting.

Optimizer:
We are currently using Adam as my optimizer. Switching to SGD gives the result:

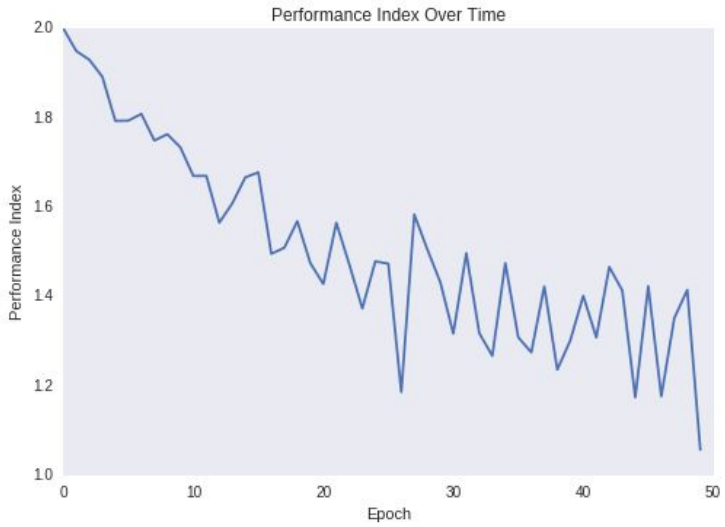

Figure 12-1. Epoch 50, Batch Size 100, Learning Rate 0.0001, Dropout 0.1, SGD, LSTM Layer 2

```
Accuracy of the network on the 3494 validation audio clips: 52 %
Accuracy of clarinet :  0 %
Accuracy of distorted electric guitar :  0 %
Accuracy of female singer :  0 %
Accuracy of flute :  0 %
Accuracy of piano : 87 %
Accuracy of tenor saxophone :  0 %
Accuracy of trumpet :  0 %
Accuracy of violin : 90 %
```

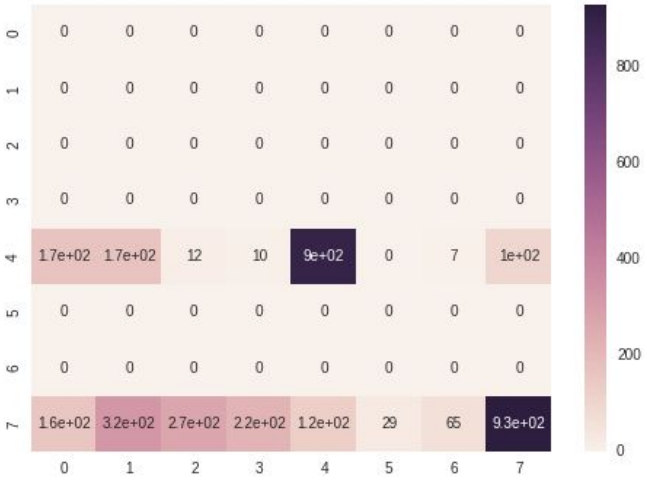

Figure 12-2. Epoch 50, Batch Size 100, Learning Rate 0.0001, Dropout 0.1, SGD, LSTM Layer 2

The overall accuracy is about 52%, and the confusion matrix suggests that the category of piano and violin are most likely predicted. The prediction result seems bad, however there is a clear trend on the loss over time, it just vibrates at the end of the epochs. Adadelta optimizer with Rhode = 0.8 and eps = 1e-6 give:

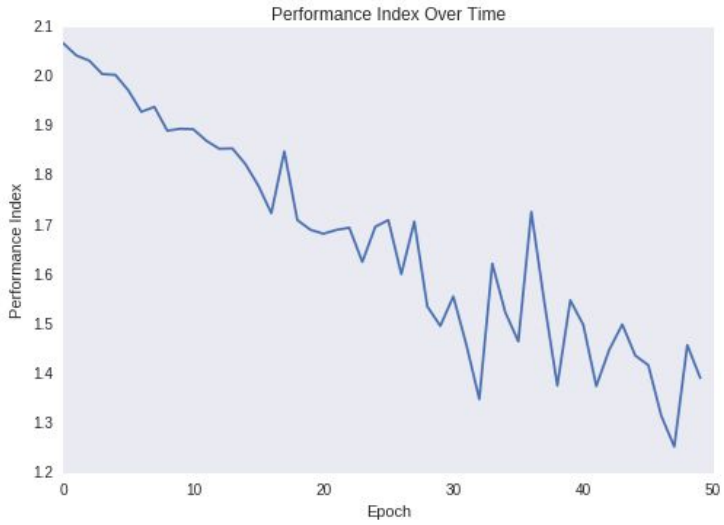

Figure 13-1. Epoch 50, Batch Size 100, Learning Rate 0.0001, Dropout 0.1, Adadelta, LSTM Layer 2

```
Accuracy of the network on the 3494 validation audio clips: 49 %
Accuracy of clarinet :  0 %
Accuracy of distorted electric guitar :  0 %
Accuracy of female singer :  0 %
Accuracy of flute :  0 %
Accuracy of piano : 93 %
Accuracy of tenor saxophone :  0 %
Accuracy of trumpet :  0 %
Accuracy of violin : 75 %
```

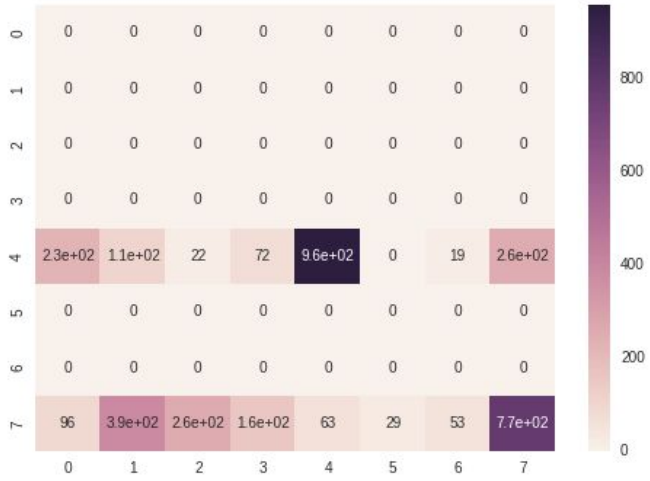

Figure 13-2. Epoch 50, Batch Size 100, Learning Rate 0.0001, Dropout 0.1, Adadelta, LSTM Layer 2

It produces 49% of overall accuracy, which is quite close to SGD optimizer. Still the network seems to think everything is either piano or violin, although we have a clear trend of loss over time. We can see from the results that Adam outperforms SGD and Adadelta.

Extra Layer of LSTMs:

Now we are adding another layer of LSTMs in the nn module. The result is as follows:

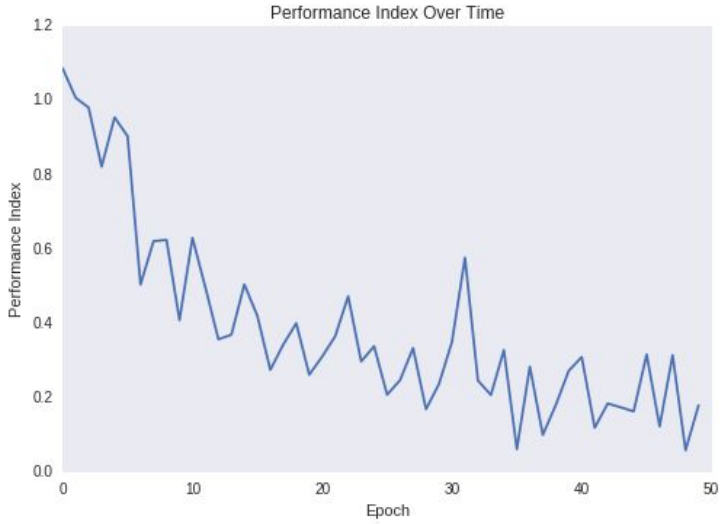

Figure 14-1. Epoch 50, Batch Size 100, Learning Rate 0.0001, Dropout 0.1, Adam, LSTM Layer 2+1

```
Accuracy of the network on the 3494 validation audio clips: 79 %
Accuracy of clarinet : 45 %
Accuracy of distorted electric guitar : 92 %
Accuracy of female singer : 74 %
Accuracy of flute : 30 %
Accuracy of piano : 88 %
Accuracy of tenor saxophone : 55 %
Accuracy of trumpet : 79 %
Accuracy of violin : 91 %
```

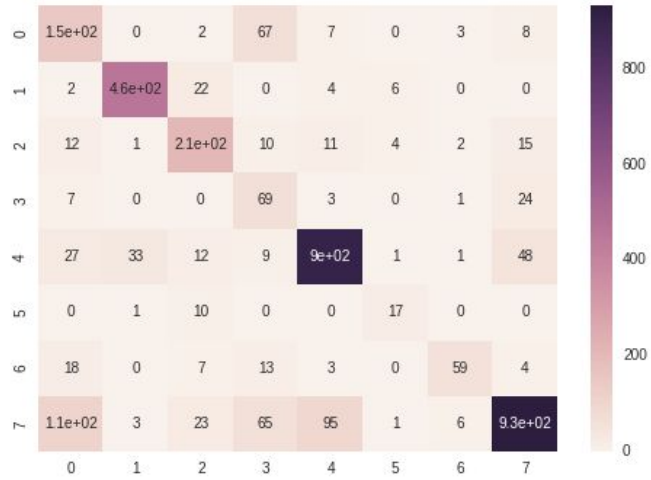

Figure 14-2. Epoch 50, Batch Size 100, Learning Rate 0.0001, Dropout 0.1, Adam, LSTM Layer 2+1

From the result we can see that the extra layer of LSTMs has an overall accuracy of 79%. The performance index oscillates similarly to our comparison parameters. The addition of the LSTMs layer does not change the result much. We now use the previously tested optimal training parameters: batch size =50, training rate = 0.0001, dropout = 0.1, optimizer = Adam, to predict the test set, and get the result as follows:

Figure 15-1: Performance Index over time for test set

```
Accuracy of the network on the 3494 validation audio clips: 53 %
Accuracy of clarinet : 29 %
Accuracy of distorted electric guitar : 83 %
Accuracy of female singer : 69 %
Accuracy of flute :  6 %
Accuracy of piano : 98 %
Accuracy of tenor saxophone :  6 %
Accuracy of trumpet : 50 %
Accuracy of violin : 56 %
```

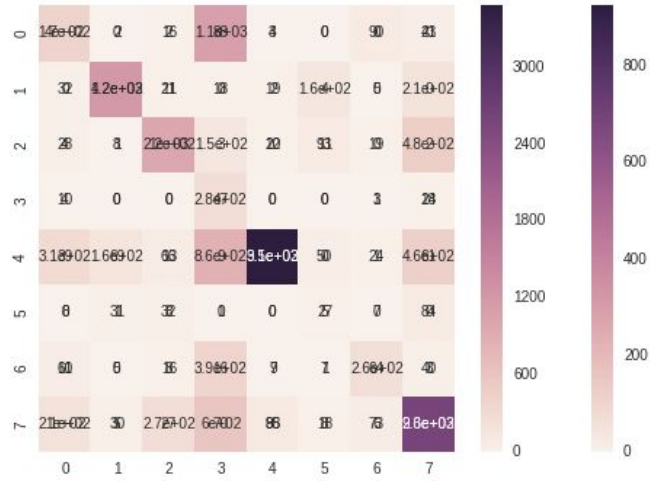

Figure 15-2. Epoch 50, Batch Size 50, Learning Rate 0.0001, Dropout 0.1, Adam, LSTM Layer 2; Test Set

We can see that the overall accuracy for the test set is 53%, not as high as the validation set. Again, from the confusion matrix (noted that it's in the scientific mode) we know the major problem is that the network has trouble distinguishing the flute and clarinet pair, and saxophone and female singer pair.

Summary and Conclusions

To better understand time-series data and modules, we used Medley-solos audio data and pytorch LSTM layers. To prepare the data, we transformed the audio files into numpy arrays and loaded the arrays as a torch dataset. After extracting, transforming, and loading the data, we set up the parameters for training the time domain and frequency domain. We then adjusted parameters, including size of training dataset and learning rate, to increase accuracy rate.

From the result, we can see that the change of the training parameters changes the performance of the network drastically. We need to find the optimal set of parameters to prevent issues like over or underfitting, learning too fast or slow, and find compromise between accuracy and running time.

We didn't get great results on the test set. This is majorly due to the fact that the network keeps confuse flute with clarinet, and saxophone with female singer or guitar. Since this behavior appears in all of the results of validation set as well. We think a better set of useful data extraction, or a more balance dividing of clips among the classes is needed.

We could also extract more features using librosa, for instance, using librosa.chrome could be helpful to make the prediction more accurate. Using a bigger sized train data on frequency domain analysis is also a way to train the network better.

In addition to these improvements, we can also adjust our code to better represent our results. We can add seeds in lines of code to see what data is being processed when there are spikes in loss over epochs, allowing us to better understand why loss increased. We can also use log-based scale in performance index graphs to better represent change in loss over time.

References

1. Kaggle.com. (2019). *PyTorch Dataset and DataLoader | Kaggle*. [online] Available at: https://www.kaggle.com/pinocookie/pytorch-dataset-and-dataloader [Accessed 23 Apr. 2019].
2. Pytorch.org. (2019). *Torch.NN* [online]. Available at: https://pytorch.org/docs/stable/nn.html [Accessed 23 Apr. 2019].
3. YouTube.com. *Time domain and frequency domain*. Retrieved from https://m.youtube.com/watch?v=tMPDe7z7ERE.
4. Nair, Pratheeksha. (2018). *The dummy's guide to MFCC*. Retrieved from https://medium.com/prathena/the-dummys-guide-to-mfcc-aceab2450fd.
5. Lostanlen, Vincent; Cella, Carmine-Emanuele; Bittner, Rachel; Essid, Slim., Medley solos-DB: A Cross-Collection Dataset for Musical Instrument Recognition. Retrieved from https://zenodo.org/record/2582103#.XMTeI4opDmq.
6. Jafari, Amir., FasionMINST Project. Retrieved from https://github.com/amir-jafari/Deep-Learning/blob/master/Pytorch_/Mini_Project/FashionMNIST.py;
7. Shaikh, Faizan., Getting Started with Audio Data Analysis using Deep Learning (with case study). Retrieved from https://www.analyticsvidhya.com/blog/2017/08/audio-voice-processing-deep-learning/.
8. Hagan, Matin T., Neural Network Design, Chapter 25: Case Study 3: Pattern Recognition.
9. uohoruotsi., Music genre classification with LSTM Recurrent Neural Nets. Retrieved from https://github.com/ruohoruotsi/LSTM-Music-Genre-Classification.
10. Guthrie, Robert., Sequence Models and Long-Short Term Memory Networks. Retrieved from https://seba-1511.github.io/tutorials/beginner/nlp/sequence_models_tutorial.html.
11. Zafar., Beginner's Guide to Audio Data. Retrieved from https://www.kaggle.com/fizzbuzz/beginner-s-guide-to-audio-data.
12. Petrou, Ted., Selection Subsets of Data in Pandas: Part 1. Retrieved from https://medium.com/dunder-data/selecting-subsets-of-data-in-pandas-6fcd0170be9c.
13. Chilamcurthy, Sasank., Data Loading and Processing Tutorial. Retrieved from https://pytorch.org/tutorials/beginner/data_loading_tutorial.html.
14. Nogueira W., Rode T., Büchner A. (2016) Optimization of a Spectral Contrast Enhancement Algorithm for Cochlear Implants Based on a Vowel Identification Model. In: van Dijk P., Başkent D., Gaudrain E., de Kleine E., Wagner A., Lanting C. (eds) Physiology, Psychoacoustics and Cognition in Normal and Impaired Hearing. Advances in Experimental Medicine and Biology, vol 894. Springer, Cham.

Appendix

Time domain analysis:
Main(time).py:

```
import numpy as np
import pandas as pd
from scipy.io import wavfile
from torch.autograd import Variable
import time
import IPython.display as ipd
import matplotlib.pyplot as plt
import os, glob

import torch
import torch.nn as nn
from torch.utils.data import Dataset, DataLoader
import torch.nn.functional as F
import torch.optim as optim
import torchvision.transforms as transforms
from sklearn.metrics import confusion_matrix

#%%
# Initial data sizes
input_size = 131072       #Number of inputs (splited because of the LSTM model)
hidden_size1 = 500      #Number of neurons
hidden_size2 = 30     #Number of neurons
num_classes = 8        #8 classes/labels
num_epochs = 30     #Number of epochs
batch_size = 100       #Number of audio clips to ran through 1 iteration
learning_rate = 0.001
confusion_m = np.zeros((8,8))

#%%
#Classes

classes = ['clarinet', 'distorted electric guitar', 'female singer', 'flute',
           'piano', 'tenor saxophone', 'trumpet', 'violin']

#%%
```

```
#Spit data into test, train, validation sets

print("Loading CSV...")
Medley = pd.read_csv("Medley-solos-DB_metadata.csv")

train = Medley.iloc[12236:18077]
train.index = range(len(train.index))

validation = Medley.iloc[18077:]
validation.index = range(len(validation.index))

test = Medley.iloc[0:12236]
test.index = range(len(test.index))

print("Number of audios=", Medley.shape[0], "  Number of classes=", len(Medley.instrument.unique()))
print()
print(Medley.instrument.unique())


print("Train set size: ", len(train))
print("Validation set size: ", len(validation))
print("Test set size: ", len(test))

#%%
#Load audio file linked to the uuid
def full_name(file):
    correspding_row = Medley.loc[Medley['uuid4'] == file].iloc[0]
    subset = str(correspding_row.loc['subset'])
    instrument_id = str(correspding_row.loc['instrument_id'])
    parts = ['Medley-solos-DB_', str(subset), '-', str(instrument_id), '_', file, '.wav.wav']
    s = ''
    file_name = s.join(parts)
    return file_name

def load_file(file):
    file_name = full_name(file)
    path = '/home/ubuntu/Final-Project-Group1/Medley-solos-DB/'
    parts = [path, file_name]
```

```
    s = ''
    link = s.join(parts)
    return link

#%%

#%%
#Process Dataset

class MedleyDataset(Dataset):
    def __init__(self, dataset_csv, transform=None):
        self.dataset_frame = dataset_csv
        self.transform = transform

    def __len__(self):
        return len(self.dataset_frame)

    def __getitem__(self, index):
        uuid4 = self.dataset_frame.iloc[index, 4]
        instrument_list = self.dataset_frame.iloc[index, 2]
        instrument_id = int(instrument_list)
        link = load_file(uuid4)

        fs, audio = wavfile.read(link)
        audio = audio.astype('float')

        sample = {'audio': audio, 'label': instrument_id}

        if self.transform:
            sample = self.transform(sample)

        return sample
#%%
transform = transforms.Compose([transforms.ToTensor(),
                                transforms.Normalize((0.5,), (0.5,))])
#%%

audio_dataset_train = MedleyDataset(dataset_csv= train)
audio_dataset_validation = MedleyDataset(dataset_csv= validation)
```

```
audio_dataset_test = MedleyDataset(dataset_csv= test)

#%%
#LSTM Model

class Net(nn.Module):
    def __init__(self, input_size, hidden_size1, hidden_size2, num_classes):
        super(Net, self).__init__()
        self.LSTM1 = nn.LSTM(input_size, hidden_size1, batch_first = True, num_layers = 2, dropout = 0.1).cuda()
        self.LSTM2 = nn.LSTM(hidden_size1, hidden_size2, batch_first = True).cuda()
        self.lstm2tag = nn.Linear(hidden_size2, num_classes).cuda()

    def forward(self, x):
        #print(x.shape)
        s = x.shape[0]
        #print(s)
        x = x.reshape(s, 1, input_size)
        out, states = self.LSTM1(x).cuda()
        out, states = self.LSTM2(out).cuda()
        #print("Afterlstm")
        #print(out.shape)
        out = out[:,0,:]
        out = self.lstm2tag(out)
        return out

#%%
#Train LSTM

model = Net(input_size, hidden_size1, hidden_size2, num_classes)

#%%
criterion = nn.CrossEntropyLoss()

#optimizer = torch.optim.Adadelta(model.parameters(), rho = 0.8, eps = 1e-6, lr=learning_rate)
#optimizer = torch.optim.SGD(params = model.parameters(), lr=learning_rate, momentum=0, dampening=0, weight_decay=0, nesterov=F
optimizer = torch.optim.Adam(params = model.parameters(), lr=learning_rate, betas=(0.9, 0.999), eps=1e-08, weight_decay=0, amsg
```

```
train_loader = torch.utils.data.DataLoader(dataset = audio_dataset_train, batch_size = batch_size, shuffle = True)
validation_loader = torch.utils.data.DataLoader(dataset = audio_dataset_validation, batch_size = batch_size, shuffle = False)
test_loader = torch.utils.data.DataLoader(dataset = audio_dataset_test, batch_size = batch_size, shuffle = False)

#%%
start_time = time.time()

device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")

epochs = np.array([])
loss_index = np.array([])

for epoch in range(num_epochs):
    for i, data in enumerate(test_loader):

        model.zero_grad()

        audios = data['audio']
        labels = data['label']

        audios = audios.type(torch.FloatTensor)
        audios = Variable(audios.cuda())

        output = model(audios)

        labels = labels.type(torch.LongTensor)
        labels = Variable(labels.cuda())

        loss = criterion(output, labels).cuda()
        loss.backward()
        optimizer.step()
        optimizer.zero_grad()

        if (i + 1) % 10 == 0:
            print('Epoch [%d/%d], Step [%d/%d], Loss: %.4f'
                  % (epoch + 1, num_epochs, i + 1, len(test) // batch_size, loss.data[0]))

    epochs = np.append(epochs, epoch)
```

```
188         loss_index = np.append(loss_index, loss.item())
189 
190     end_time = time.time()
191     elapsed_time = end_time - start_time
192     print("Elapsed time:", elapsed_time)
193 
194     #%%
195 
196     plt.figure(figsize = (12,8))
197     plt.plot(epochs, loss_index, 'r')
198     plt.xlabel("Epoch", fontsize = 14)
199     plt.ylabel("Performance Index", fontsize = 14)
200     plt.title("Performance Index Over Time", fontsize = 20)
201     plt.show()
202 
203     #%%
204     #Test accuracy on validation set
205 
206     correct = 0
207     total = 0
208 
209     for i, data in enumerate(validation_loader):
210         audios = data['audio']
211         labels = data['label']
212 
213         audios = audios.type(torch.FloatTensor)
214         audios = Variable(audios.cuda())
215 
216         outputs = model(audios)
217 
218         labels = labels.type(torch.LongTensor)
219         labels = Variable(labels.cuda())
220 
221         _, predicted = torch.max(outputs.data, 1)
222 
223         total += labels.size(0)
224         #correct += (predicted == labels).sum()
225         for j in range(len(predicted)):
```

```
            if predicted[j] == labels[j]: correct += 1
        # print(total)
        # print(correct)


    #Confusion matrix
    results = confusion_matrix(labels, predicted)
    confusion_m = confusion_m + results

    print(confusion_m)
    print('Accuracy of the network on the 3494 validation audio clips: %d %%' % (100 * correct / total))

    #%%
    #Test accuracy of each class on Validation set

    class_correct = list(0. for i in range(8))
    class_total = list(0. for i in range(8))

    for data in test_loader:
        audios = data['audio']
        labels = data['label']

        audios = audios.type(torch.FloatTensor)
        audios = Variable(audios.cuda())

        outputs = model(audios)

        labels = labels.type(torch.LongTensor)
        labels = Variable(labels.cuda())

        _, predicted = torch.max(outputs.data, 1)

        labels = labels.cpu().numpy()
        c = (predicted.cpu().numpy() == labels)
        s = labels.shape[0]
        for i in range(s):
            label = labels[i]
            class_correct[label] += c[i]
            class_total[label] += 1
             class_correct[label] += c[i]
             class_total[label] += 1

    #%%

    for i in range(8):
        if (class_total[i] == 0):
            print('Accuracy of %5s : %2d %%' % (classes[i], 0))
        else:
            print('Accuracy of %5s : %2d %%' % (classes[i], 100 * class_correct[i] / class_total[i]))

    #%%
    torch.save(model.state_dict(), 'model_time.pkl')
```

Frequency domain analysis:

music_feature_loader:

```
#This file extracts the music feature (mfcc, spectral centroid, spectral contrast) using librosa, and save it as.npy numpy
#------------------------------------------------------------------------------------------------------------
#Load audio file linked to the uuid
def full_name(file):
    correspding_row = Medley.loc[Medley['uuid4'] == file].iloc[0]
    subset = str(correspding_row.loc['subset'])
    instrument_id = str(correspding_row.loc['instrument_id'])
    parts = ['Medley-solos-DB_', str(subset), '-', str(instrument_id), '_', file, '.wav.wav']
    s = ''
    file_name = s.join(parts)
    return file_name

def find_file(file):
    file_name = full_name(file)
    path = '/home/ubuntu/Machine-Learning/Medley-solos-DB/'
    parts = [path, file_name]
    s = ''
    link = s.join(parts)
    return link

#------------------------------------------------------------------------------------------------------------
#Generating music data ndarray in frequency domain using librosa

def get_music_features(dataset):
    timeseries_length = 3
    audio = np.zeros((len(dataset), timeseries_length, 21), dtype=np.float64)

    for i in range(len(dataset)):
        row = dataset.loc[i]
        uuid4_name = str(row.loc['uuid4'])
        link = find_file(uuid4_name)

        y, sr = librosa.load(link)

        mfcc = librosa.feature.mfcc(y=y, sr=sr, n_mfcc=13)
        spectral_center = librosa.feature.spectral_centroid(y=y, sr=sr)
        spectral_contrast = librosa.feature.spectral_contrast(y=y, sr=sr)

        audio[i, :, 0:13] = mfcc.T[0:timeseries_length, :]
        audio[i, :, 13:14] = spectral_center.T[0:timeseries_length, :]
        audio[i, :, 14:21] = spectral_contrast.T[0:timeseries_length, :]


        if ((i + 1) % 100 == 0):
            print("Extracted features audio clip %i of %i." % (i + 1, len(dataset)))

    return audio

train_audio_data = get_music_features(train)
validation_audio_data = get_music_features(validation)
test_audio_data = get_music_features(test)

np.save('train_audio_data.npy', train_audio_data)
np.save('validation_audio_data.npy', validation_audio_data)
np.save('test_audio_data.npy', test_audio_data)
```

Main(Frequency):

```
#The commented part is the same as the music_feature_loader
#Need to run music_feature_loader first to generate the 3 .npy file, to run this code.


import numpy as np
import pandas as pd
from torch.autograd import Variable
import time
import librosa

import torch
import torch.nn as nn
from torch.utils.data import Dataset, DataLoader

import matplotlib
import matplotlib.pyplot as plt
import seaborn as sns; sns.set()

device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")

# ----------------------------------------------------------------------------------------------------
# Initial data sizes
input_size = 63  # Number of inputs (splited because of the LSTM model) using Librosa
hidden_size1 = 500  # Number of neurons
hidden_size2 = 30  # Number of neurons
num_classes = 8  # 8 classes/labels
num_epochs = 300  # Number of epochs
batch_size = 100  # Number of audio clips to ran through 1 iteration
learning_rate = 0.001
confusion_m = np.zeros((8, 8))
# ----------------------------------------------------------------------------------------------------
# Classes

classes = ['clarinet', 'distorted electric guitar', 'female singer', 'flute', 'piano', 'tenor saxophone', 'trumpet',
           'violin']

# ----------------------------------------------------------------------------------------------------
# Spit data into test, train, validation sets

print("Loading CSV...")
```

```
Medley = pd.read_csv("Medley-solos-DB_metadata.csv")

train = Medley.iloc[12236:18077]
train.index = range(len(train.index))

validation = Medley.iloc[18077:]
validation.index = range(len(validation.index))

test = Medley.iloc[0:12236]
test.index = range(len(test.index))

print("Number of audios=", Medley.shape[0], "  Number of classes=", len(Medley.instrument.unique()))
print(Medley.instrument.unique())

print("Train set size: ")
print(len(train))
print("Validation set size: ")
print(len(validation))
print("Test set size: ")
print(len(test))

"""""
#-----------------------------------------------------------------------------------------------------
#Load audio file linked to the uuid
def full_name(file):
    correspding_row = Medley.loc[Medley['uuid4'] == file].iloc[0]
    subset = str(correspding_row.loc['subset'])
    instrument_id = str(correspding_row.loc['instrument_id'])
    parts = ['Medley-solos-DB_', str(subset), '-', str(instrument_id), '_', file, '.wav.wav']
    s = ''
    file_name = s.join(parts)
    return file_name

def find_file(file):
    file_name = full_name(file)
    path = '/home/ubuntu/Machine-Learning/Medley-solos-DB/'
    parts = [path, file_name]
    s = ''
    link = s.join(parts)
    return link
```

```
81
82  #-------------------------------------------------------------------------------------------------------
83  #Generating music data ndarray in frequency domain using librosa
84
85  def get_music_features(dataset):
86      timeseries_length = 3
87      audio = np.zeros((len(dataset), timeseries_length, 21), dtype=np.float64)
88
89      for i in range(len(dataset)):
90          row = dataset.loc[i]
91          uuid4_name = str(row.loc['uuid4'])
92          link = find_file(uuid4_name)
93
94          y, sr = librosa.load(link)
95
96          mfcc = librosa.feature.mfcc(y=y, sr=sr, n_mfcc=13)
97          spectral_center = librosa.feature.spectral_centroid(y=y, sr=sr)
98          spectral_contrast = librosa.feature.spectral_contrast(y=y, sr=sr)
99
100         audio[i, :, 0:13] = mfcc.T[0:timeseries_length, :]
101         audio[i, :, 13:14] = spectral_center.T[0:timeseries_length, :]
102         audio[i, :, 14:21] = spectral_contrast.T[0:timeseries_length, :]
103
104
105         if ((i + 1) % 100 == 0):
106             print("Extracted features audio clip %i of %i." % (i + 1, len(dataset)))
107
108     return audio
109
110 train_audio_data = get_music_features(train)
111 validation_audio_data = get_music_features(validation)
112 test_audio_data = get_music_features(test)
113
114 np.save('train_audio_data.npy', train_audio_data)
115 np.save('validation_audio_data.npy', validation_audio_data)
116 np.save('test_audio_data.npy', test_audio_data)
117 """
118 # -------------------------------------------------------------------------------------------------------
119 # load npy music feature matrix data
120 train_audio_data = np.load('train_audio_data.npy')
```

```
validation_audio_data = np.load('validation_audio_data.npy')
test_audio_data = np.load('test_audio_data.npy')


# ----------------------------------------------------------------------------------------------------
# Process Dataset

class MedleyDataset(Dataset):
    def __init__(self, dataset_csv, transform=None):
        self.dataset_frame = dataset_csv
        self.transform = transform

    def __len__(self):
        return len(self.dataset_frame)

    def __getitem__(self, index):
        uuid4 = self.dataset_frame.iloc[index, 4]
        instrument_list = self.dataset_frame.iloc[index, 2]
        instrument_id = int(instrument_list)
        set = self.dataset_frame
        if (set.shape == train.shape):
            feature_matrix = train_audio_data
        if (set.shape == validation.shape):
            feature_matrix = validation_audio_data
        if (set.shape == test.shape):
            feature_matrix = test_audio_data

        audio = feature_matrix[index, :, :]

        audio = audio.astype('float')

        sample = {'audio': audio, 'label': instrument_id}

        if self.transform:
            sample = self.transform(sample)

        return sample


audio_dataset_train = MedleyDataset(dataset_csv=train)
```

```
audio_dataset_validation = MedleyDataset(dataset_csv=validation)
audio_dataset_test = MedleyDataset(dataset_csv=test)


# ------------------------------------------------------------------------------------------------------
# LSTM Model

class Net(nn.Module):
    def __init__(self, input_size, hidden_size1, hidden_size2, num_classes):
        super(Net, self).__init__()
        self.LSTM1 = nn.LSTM(input_size, hidden_size1, batch_first=True, num_layers=2, dropout=0.1).cuda()
        #self.LSTM2 = nn.LSTM(hidden_size1, hidden_size2, batch_first = True).cuda()
        self.lstm2tag = nn.Linear(hidden_size1, num_classes).cuda()

    def forward(self, x):
        #print(x.shape)
        s = x.shape[0]
        x = x.reshape(s, 1, input_size)
        out, states = self.LSTM1(x)
        #out, states = self.LSTM2(out)
        out = out[:, 0, :]
        out = self.lstm2tag(out)
        return out


# ------------------------------------------------------------------------------------------------------
# Train LSTM

model = Net(input_size, hidden_size1, hidden_size2, num_classes)
criterion = nn.CrossEntropyLoss()


#optimizer = torch.optim.Adadelta(model.parameters(), rho = 0.8, eps = 1e-6, lr=learning_rate)
#optimizer = torch.optim.SGD(params = model.parameters(), lr=learning_rate, momentum=0, dampening=0, weight_decay=0, nesterov=F
optimizer = torch.optim.Adam(params=model.parameters(), lr=learning_rate, betas=(0.9, 0.999), eps=1e-08, weight_decay=0,amsgrad

train_loader = torch.utils.data.DataLoader(dataset=audio_dataset_train, batch_size=batch_size, shuffle=True)
validation_loader = torch.utils.data.DataLoader(dataset=audio_dataset_validation, batch_size=batch_size, shuffle=True)
test_loader = torch.utils.data.DataLoader(dataset = audio_dataset_test, batch_size = batch_size, shuffle = True)

```

```
201
202 start_time = time.time()
203
204 epochs = np.array([])
205 loss_index = np.array([])
206
207 for epoch in range(num_epochs):
208     for i, data in enumerate(train_loader):
209
210         model.zero_grad()
211
212         audios = data['audio']
213         labels = data['label']
214
215         audios = audios.type(torch.FloatTensor)
216         audios = Variable(audios.cuda())
217
218         output = model(audios)
219
220         labels = labels.type(torch.LongTensor)
221         labels = Variable(labels.cuda())
222
223         loss = criterion(output, labels).cuda()
224         loss.backward()
225         optimizer.step()
226         optimizer.zero_grad()
227
228
229         if (i + 1) % 10 == 0:
230             print('Epoch [%d/%d], Step [%d/%d], Loss: %.4f'
231                   % (epoch + 1, num_epochs, i + 1, len(train) // batch_size, loss.data[0]))
232
233 epochs = np.append(epochs, epoch)
234 loss_index = np.append(loss_index, loss.item())
235
236 end_time = time.time()
237 elapsed_time = end_time - start_time
238 print("Elapsed time:", elapsed_time)
239
240
```

```
241
242 fig, ax = plt.subplots()
243 ax.plot(epochs, loss_index)
244
245 ax.set(xlabel='Epoch', ylabel='Performance Index', title='Performance Index Over Time')
246 ax.grid()
247 fig.savefig("test.png")
248 plt.show()
249
250
251 # ----------------------------------------------------------------------------------------------------
252 # Test accuracy on validation set
253
254 correct = 0
255 total = 0
256
257 for i, data in enumerate(validation_loader):
258
259     audios = data['audio']
260     labels = data['label']
261
262     audios = audios.type(torch.FloatTensor)
263     audios = Variable(audios.cuda())
264
265     outputs = model(audios)
266
267     labels = labels.type(torch.LongTensor)
268     labels = Variable(labels.cuda())
269
270     _, predicted = torch.max(outputs.data, 1)
271
272     total += labels.size(0)
273     correct += (predicted == labels).sum()
274
275
276     leng = predicted.shape[0]
277
278     a = predicted.cpu().detach().numpy()
279     b = labels.cpu().detach().numpy()
280
```

```
281
282     for j in range(leng):
283         k1 = a[j]
284         k2 = b[j]
285         confusion_m[k1,k2] += 1
286
287 #confusion matrix
288 confusion_m.astype(int)
289 ax = sns.heatmap(confusion_m, annot=True)
290 print(confusion_m)
291 print('Accuracy of the network on the 3494 validation audio clips: %d %%' % (100 * correct / total))
292 # ------------------------------------------------------------------------------------------
293 # Test accuracy of each class on Validation set
294
295 class_correct = list(0. for i in range(8))
296 class_total = list(0. for i in range(8))
297
298 for data in validation_loader:
299     audios = data['audio']
300     labels = data['label']
301
302     audios = audios.type(torch.FloatTensor)
303     audios = Variable(audios.cuda())
304
305     outputs = model(audios)
306
307     labels = labels.type(torch.LongTensor)
308     labels = Variable(labels.cuda())
309
310     _, predicted = torch.max(outputs.data, 1)
311
312     labels = labels.cpu().numpy()
313     c = (predicted.cpu().numpy() == labels)
314     s = labels.shape[0]
315     for i in range(s):
316         label = labels[i]
317         class_correct[label] += c[i]
318         class_total[label] += 1
319
320 # ------------------------------------------------------------------------------------------
```

```
321
322 for i in range(8):
323     if (class_total[i] == 0):
324         print('Accuracy of %5s : %2d %%' % (classes[i], 0))
325     else:
326         print('Accuracy of %5s : %2d %%' % (classes[i], 100 * class_correct[i] / class_total[i]))
327 # ----------------------------------------------------------------------------------------------
328 torch.save(model.state_dict(), 'model_time.pkl')
329
330 # ----------------------------------------------------------------------------------------------
331 # Test accuracy on test set
332
333 correct = 0
334 total = 0
335
336 for i, data in enumerate(test_loader):
337
338     audios = data['audio']
339     labels = data['label']
340
341     audios = audios.type(torch.FloatTensor)
342     audios = Variable(audios.cuda())
343
344     outputs = model(audios)
345
346     labels = labels.type(torch.LongTensor)
347     labels = Variable(labels.cuda())
348
349     _, predicted = torch.max(outputs.data, 1)
350
351     total += labels.size(0)
352     correct += (predicted == labels).sum()
353
354
355     leng = predicted.shape[0]
356
357     a = predicted.cpu().detach().numpy()
358     b = labels.cpu().detach().numpy()
359
360
```

```
    for j in range(leng):
        k1 = a[j]
        k2 = b[j]
        confusion_m[k1,k2] += 1

#confusion matrix
confusion_m.astype(int)
ax = sns.heatmap(confusion_m, annot=True)
print(confusion_m)
print('Accuracy of the network on the 3494 validation audio clips: %d %%' % (100 * correct / total))
# ------------------------------------------------------------------------------------------
# Test accuracy of each class on test set

class_correct = list(0. for i in range(8))
class_total = list(0. for i in range(8))

for data in test_loader:
    audios = data['audio']
    labels = data['label']

    audios = audios.type(torch.FloatTensor)
    audios = Variable(audios.cuda())

    outputs = model(audios)

    labels = labels.type(torch.LongTensor)
    labels = Variable(labels.cuda())

    _, predicted = torch.max(outputs.data, 1)

    labels = labels.cpu().numpy()
    c = (predicted.cpu().numpy() == labels)
    s = labels.shape[0]
    for i in range(s):
        label = labels[i]
        class_correct[label] += c[i]
        class_total[label] += 1

# ------------------------------------------------------------------------------------------

for i in range(8):
    if (class_total[i] == 0):
        print('Accuracy of %5s : %2d %%' % (classes[i], 0))
    else:
        print('Accuracy of %5s : %2d %%' % (classes[i], 100 * class_correct[i] / class_total[i]))
# ------------------------------------------------------------------------------------------
torch.save(model.state_dict(), 'model_time.pkl')
```