Individual Report

Madison Turano

Introduction:

After learning about time-series data and LSTM models, we were interested in creating a machine learning algorithm incorporating time-series data and LSTM data. After searching for appropriate data, I found the Medley-solos-DB dataset, a collection of audio clips used for automatic musical instrument recognition. The dataset contains 21,000 records of 3-second audio clips in wav.wav format. The dataset also came with a metadata CSV file, which contains the unique identification number of the audio clip, the subset (test/validation/test), and classification for each datapoint.

For the project, we decided to create an audio-tagging algorithm like the image-tagging algorithm in exam 1. We developed functions and classes to load and pre-process the data, ran the algorithm using different optimizers and learning rate, and used accuracy rate to determine the effectiveness of the algorithm.

Individual Work:

For the project, I determined what data to use and what kind of modeling to use. Since we didn't have LSTM examples, I decided time-series data with LSTM model would help us better understand how time-series data and LSTM models work. I found the Medley-solos-DB dataset and debugged issues with downloading the zip files, then shared the data with my team.

Since the audio files were stored in a separate folder, I created a function to load the audio file. I figured that the audio file names (which aren't in the metadata file) are based on uuid4 (unique identification), subset (train/validation/test), and instrumend_id (label), and wrote the following code to get the full audio file name for loading:

```
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
```

Figure 1: Full name function

After reviewing my exam 1 code as a reference, I decided to transform the data into a torch dataset so we could use torch Dataset and Dataloader to load our data for the tagging algorithm. I used code from *PyTorch Dataset and DataLoader* as a reference and wrote the following code:

```
class DatasetMedley(Dataset):

  def __init__(self, file_path, transform=None):
    self.data = pd.read_csv(file_path)
    self.transform = transform

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

  def __getitem__(self, index):
    audio = self.data.iloc[index, 2:].values.astype(np.uint8).reshape((1, 131072))
    label = self.data.iloc[index, 1]
    subset = self.data.iloc[index, 0]

    if self.transform is not None:
      audio = self.transform(audio)

    return audio, label, subset
```

Figure 2: Dataset class code

Once we determined the best way to load and save the audio data, we used my DatasetMeldley code as a baseline for our final dataset class code.

After developing the name and loading functions, I wrote the first draft on the model and training function. Below are the first draft codes:

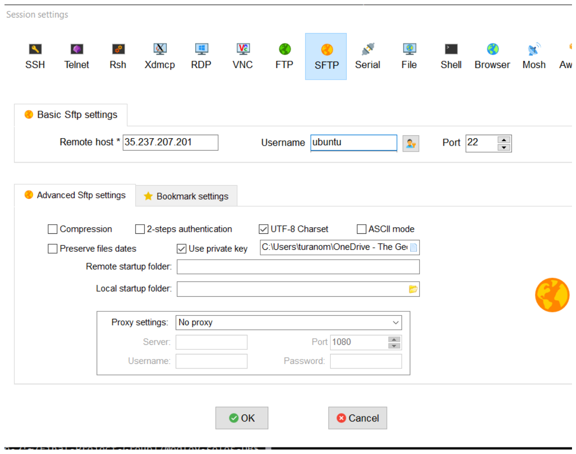
```
class Net(nn.Module):
  def __init__(self, input_size, hidden_size, num_classes):
    super(Net, self).__init__()
    self.fc1 = nn.LSTM(input_size, hidden_size)
    self.relu = nn.ReLU()
    self.fc2 = nn.Linear(hidden_size, num_classes)

  def forward(self, x):
    out = self.fc1(x)
    out = self.relu(out)
    out = self.fc2(out)
    return out
```

```
# In[60]:

net = Net(input_size, hidden_size, num_classes)
net.cuda()

criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adadelta(net.parameters(), rho = 0.8, eps = 1e-6, lr=learning_rate)

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

dtype = torch.float

for epoch in range(num_epochs):
    for i, (audio, labels) in enumerate(train_loader):
        # Convert torch tensor to Variable
        audio = Variable(audio)
        labels = Variable(label.cuda())

        # Forward + Backward + Optimize
        optimizer.zero_grad()  # zero the gradient buffer
        outputs = net(images)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

        if (i+1) % 100 == 0:
            print ('Epoch [%d/%d], Step [%d/%d], Loss: %.4f'
                   %(epoch+1, num_epochs, i+1, len(train_set)//batch_size, loss.item()))
```

Figure 3: Model and training function

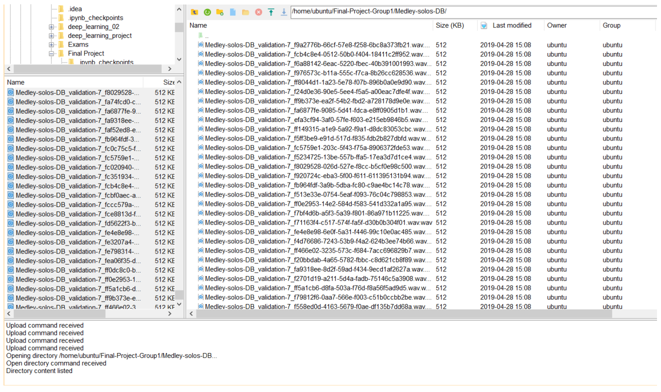
After we figured out the most efficient way to save and load the data, we adjusted the above the to fit the new data structure. However, we kept the loss function and basic structure of the model code and the epoch training code.

I also worked on loading the data to GitHub. I created an SFTP connection to my instance and added the audio files via the SFTP connection. Below are screenshots of the SFTP connection I created and used to load audio files to my vm instance.



```
Remote host * 35.237.207.201    Username ubuntu    Port 22

[ ] Compression    [ ] 2-steps authentication    [✓] UTF-8 Charset    [ ] ASCII mode
[ ] Preserve files dates    [✓] Use private key  C:\Users\turanom\OneDrive - The Ge
Remote startup folder:
Local startup folder:

Proxy settings: No proxy
Server:              Port 1080
Username:            Password:
```

Figure 4: SFTP connection



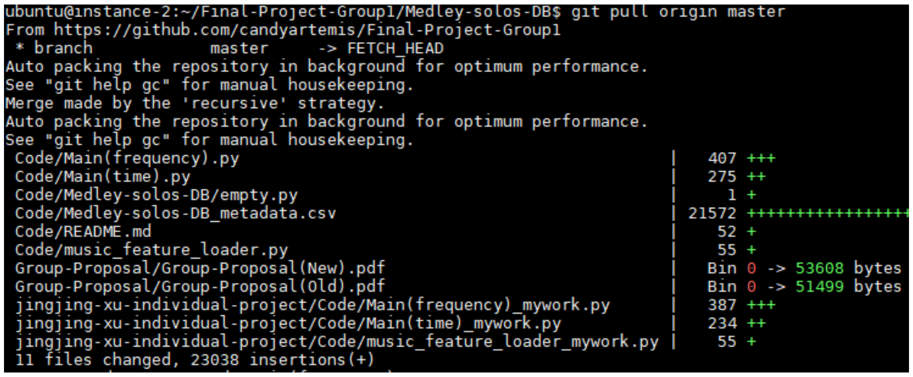
```
.idea
.ipynb_checkpoints
deep_learning_02
deep_learning_project
Exams
Final Project
  .ipynb_checkpoints

Name                                         Size
Medley-solos-DB_validation-7_f8029528-...    512 KE
Medley-solos-DB_validation-7_fa74fcd0-c...   512 KE
Medley-solos-DB_validation-7_fa6877fe-9...   512 KE
Medley-solos-DB_validation-7_fa9318ee-...    512 KE
Medley-solos-DB_validation-7_faf52ed8-e...   512 KE
Medley-solos-DB_validation-7_fb964fdf-3...   512 KE
Medley-solos-DB_validation-7_fc0c75c5-f...   512 KE
Medley-solos-DB_validation-7_fc5759e1-...    512 KE
Medley-solos-DB_validation-7_fc020940-...    512 KE
Medley-solos-DB_validation-7_fc351934-...    512 KE
Medley-solos-DB_validation-7_fcb4c8e4-...    512 KE
Medley-solos-DB_validation-7_fcbf0aec-a...   512 KE
Medley-solos-DB_validation-7_fccc579a-...    512 KE
Medley-solos-DB_validation-7_fce8813d-f...   512 KE
Medley-solos-DB_validation-7_fd5622f3-b...   512 KE
Medley-solos-DB_validation-7_fe4e8e98-...    512 KE
Medley-solos-DB_validation-7_fe3207a4-...    512 KE
Medley-solos-DB_validation-7_fe798314-...    512 KE
Medley-solos-DB_validation-7_fea06f35-d...   512 KE
Medley-solos-DB_validation-7_ff0dc8c0-b...   512 KE
Medley-solos-DB_validation-7_ff0e2953-1...   512 KE
Medley-solos-DB_validation-7_ff5a1cb6-d...   512 KE
Medley-solos-DB_validation-7_ff9b373e-e...   512 KE
Medley-solos-DB_validation-7_ff466e02-3...   512 KE
```

```
/home/ubuntu/Final-Project-Group1/Medley-solos-DB/

Name                                                                     Size (KB)  Last modified     Owner   Group
..
Medley-solos-DB_validation-7_f9a2776b-66cf-57e8-f258-6bc8a373fb21.wav....  512  2019-04-28 15:08  ubuntu  ubuntu
Medley-solos-DB_validation-7_fcb4c8e4-0512-50b0-f404-18411c2ff952.wav....  512  2019-04-28 15:08  ubuntu  ubuntu
Medley-solos-DB_validation-7_f6a88142-6eac-5220-fbec-40b391001993.wav...   512  2019-04-28 15:08  ubuntu  ubuntu
Medley-solos-DB_validation-7_f976573c-b11a-555c-f7ca-8b26cc628536.wav...   512  2019-04-28 15:08  ubuntu  ubuntu
Medley-solos-DB_validation-7_ff8044d1-1a23-5e78-f07b-896b0a0e9d90.wav....  512  2019-04-28 15:08  ubuntu  ubuntu
Medley-solos-DB_validation-7_f24d0e36-90e5-5ee4-f5a5-a00eac7dfe4f.wav....  512  2019-04-28 15:08  ubuntu  ubuntu
Medley-solos-DB_validation-7_ff9b373e-ea2f-54b2-fbd2-a728178d9e0e.wav....  512  2019-04-28 15:08  ubuntu  ubuntu
Medley-solos-DB_validation-7_fa6877fe-9085-5d41-fdca-e8ff0905d1b1.wav....  512  2019-04-28 15:08  ubuntu  ubuntu
Medley-solos-DB_validation-7_efa3cf94-3af0-57fe-f603-e215eb9846b5.wav....  512  2019-04-28 15:08  ubuntu  ubuntu
Medley-solos-DB_validation-7_ff149315-a1e9-5a92-f9a1-d8dc83053cbc.wav....  512  2019-04-28 15:08  ubuntu  ubuntu
Medley-solos-DB_validation-7_f5ff3be9-e91d-517d-f835-fdb2b827dbfd.wav.w... 512  2019-04-28 15:08  ubuntu  ubuntu
Medley-solos-DB_validation-7_fc5759e1-203c-5f43-f75a-8906372fde53.wav....  512  2019-04-28 15:08  ubuntu  ubuntu
Medley-solos-DB_validation-7_f5234725-13be-557b-ffa5-17ea3d7d1ce4.wav....  512  2019-04-28 15:08  ubuntu  ubuntu
Medley-solos-DB_validation-7_f8029528-026d-527e-f8cc-b5cf0e98c500.wav....  512  2019-04-28 15:08  ubuntu  ubuntu
Medley-solos-DB_validation-7_f920724c-eba3-5f00-f611-611395131b94.wav....  512  2019-04-28 15:08  ubuntu  ubuntu
Medley-solos-DB_validation-7_fb964fdf-3a9b-5dba-fc80-c9ae4bc14c78.wav....  512  2019-04-28 15:08  ubuntu  ubuntu
Medley-solos-DB_validation-7_f513e33e-0754-5eaf-f093-76c04c798853.wav....  512  2019-04-28 15:08  ubuntu  ubuntu
Medley-solos-DB_validation-7_ff0e2953-14e2-584d-f583-541d332a1a95.wav....  512  2019-04-28 15:08  ubuntu  ubuntu
Medley-solos-DB_validation-7_f7bf4d6b-a5f3-5a39-f801-86a971b11225.wav....  512  2019-04-28 15:08  ubuntu  ubuntu
Medley-solos-DB_validation-7_f71163f4-c517-574f-fa5f-d30b0b304f01.wav.wav  512  2019-04-28 15:08  ubuntu  ubuntu
Medley-solos-DB_validation-7_fe4e8e98-6e0f-5a31-f446-99c10e0ac485.wav....  512  2019-04-28 15:08  ubuntu  ubuntu
Medley-solos-DB_validation-7_f4d76686-7243-53b9-f4a2-624b3ee74b66.wav...   512  2019-04-28 15:08  ubuntu  ubuntu
Medley-solos-DB_validation-7_ff466e02-3235-573c-f684-7acc696829b7.wav....  512  2019-04-28 15:08  ubuntu  ubuntu
Medley-solos-DB_validation-7_f20bbdab-4a65-5782-fbbc-c8d621cb8f89.wav....  512  2019-04-28 15:08  ubuntu  ubuntu
Medley-solos-DB_validation-7_fa9318ee-8d2f-59ad-f434-9ecd1af2627a.wav....  512  2019-04-28 15:08  ubuntu  ubuntu
Medley-solos-DB_validation-7_f2701d19-a211-5d4a-fadb-75146c5a3908.wav...   512  2019-04-28 15:08  ubuntu  ubuntu
Medley-solos-DB_validation-7_ff5a1cb6-d8fa-503a-f76d-f8a56f5ad9d5.wav.w... 512  2019-04-28 15:08  ubuntu  ubuntu
Medley-solos-DB_validation-7_f79812f6-0aa7-566e-f003-c51b0ccbb2be.wav....  512  2019-04-28 15:08  ubuntu  ubuntu
Medley-solos-DB_validation-7_f558ed0d-4163-5679-f0ae-df135b7dd68a.wav....  512  2019-04-28 15:08  ubuntu  ubuntu
```

```
Upload command received
Upload command received
Upload command received
Upload command received
Opening directory /home/ubuntu/Final-Project-Group1/Medley-solos-DB...
Open directory command received
Directory content listed
```

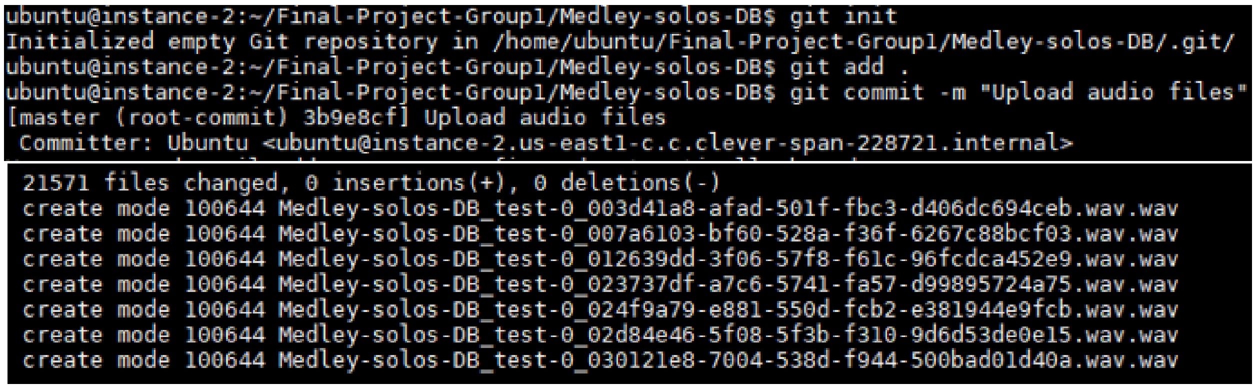
Figure 5: Load audio files

After I loaded the audio files to my account, I used git terminal commands to load the data to the project GitHub account. Since I need the most recent version of the repo to push any new changes, I pulled all changes to the repo. I then set my audio files folder as my local directory and added all files, and then committed the changes. Below are screenshots of the steps I performed:



```
ubuntu@instance-2:~/Final-Project-Group1/Medley-solos-DB$ git pull origin master
From https://github.com/candyartemis/Final-Project-Group1
 * branch            master     -> FETCH_HEAD
Auto packing the repository in background for optimum performance.
See "git help gc" for manual housekeeping.
Merge made by the 'recursive' strategy.
Auto packing the repository in background for optimum performance.
See "git help gc" for manual housekeeping.
 Code/Main(frequency).py                                         |   407 +++
 Code/Main(time).py                                              |   275 ++
 Code/Medley-solos-DB/empty.py                                   |     1 +
 Code/Medley-solos-DB_metadata.csv                               | 21572 ++++++++++++++++
 Code/README.md                                                  |    52 +
 Code/music_feature_loader.py                                    |    55 +
 Group-Proposal/Group-Proposal(New).pdf                          |   Bin 0 -> 53608 bytes
 Group-Proposal/Group-Proposal(Old).pdf                          |   Bin 0 -> 51499 bytes
 jingjing-xu-individual-project/Code/Main(frequency)_mywork.py   |   387 +++
 jingjing-xu-individual-project/Code/Main(time)_mywork.py        |   234 ++
 jingjing-xu-individual-project/Code/music_feature_loader_mywork.py |  55 +
 11 files changed, 23038 insertions(+)
```

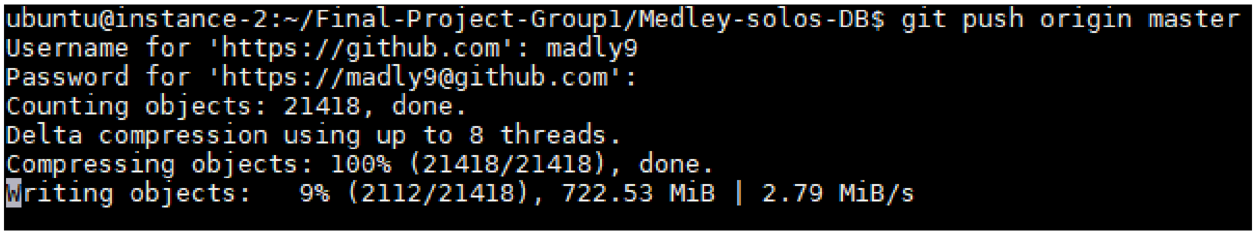
Figure 6: Pull most recent repo version



```
ubuntu@instance-2:~/Final-Project-Group1/Medley-solos-DB$ git init
Initialized empty Git repository in /home/ubuntu/Final-Project-Group1/Medley-solos-DB/.git/
ubuntu@instance-2:~/Final-Project-Group1/Medley-solos-DB$ git add .
ubuntu@instance-2:~/Final-Project-Group1/Medley-solos-DB$ git commit -m "Upload audio files"
[master (root-commit) 3b9e8cf] Upload audio files
 Committer: Ubuntu <ubuntu@instance-2.us-east1-c.c.clever-span-228721.internal>
```

```
 21571 files changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 Medley-solos-DB_test-0_003d41a8-afad-501f-fbc3-d406dc694ceb.wav.wav
 create mode 100644 Medley-solos-DB_test-0_007a6103-bf60-528a-f36f-6267c88bcf03.wav.wav
 create mode 100644 Medley-solos-DB_test-0_012639dd-3f06-57f8-f61c-96fcdca452e9.wav.wav
 create mode 100644 Medley-solos-DB_test-0_023737df-a7c6-5741-fa57-d99895724a75.wav.wav
 create mode 100644 Medley-solos-DB_test-0_024f9a79-e881-550d-fcb2-e381944e9fcb.wav.wav
 create mode 100644 Medley-solos-DB_test-0_02d84e46-5f08-5f3b-f310-9d6d53de0e15.wav.wav
 create mode 100644 Medley-solos-DB_test-0_030121e8-7004-538d-f944-500bad01d40a.wav.wav
```

Figure 7: Move to audio files folder and add files to GitHub



```
ubuntu@instance-2:~/Final-Project-Group1/Medley-solos-DB$ git push origin master
Username for 'https://github.com': madly9
Password for 'https://madly9@github.com':
Counting objects: 21418, done.
Delta compression using up to 8 threads.
Compressing objects: 100% (21418/21418), done.
Writing objects:   9% (2112/21418), 722.53 MiB | 2.79 MiB/s
```

Figure 8: Push changes to repo

To visualize our results, I wrote code to record and plot performance index/loss over time. I created numpy arrays to store the epochs and loss/performance index at each epoch, which is more efficient then appending lists. I then plotted the numpy arrays to show performance index/loss over time. Below is a sample of the plotting code I added to the training and test algorithms:

```
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)
  loss_index = np.append(loss_index, loss.item())
```

Figure 9: Storing performance index over time

```
plt.figure(figsize = (12,8))
plt.plot(epochs, loss_index, 'r')
plt.xlabel("Epoch", fontsize = 14)
plt.ylabel("Performance Index", fontsize = 14)
plt.title("Performance Index Over Time for Training Set", fontsize = 20)
plt.show()
```

Figure 10: Plotting performance index over time

Results:

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:

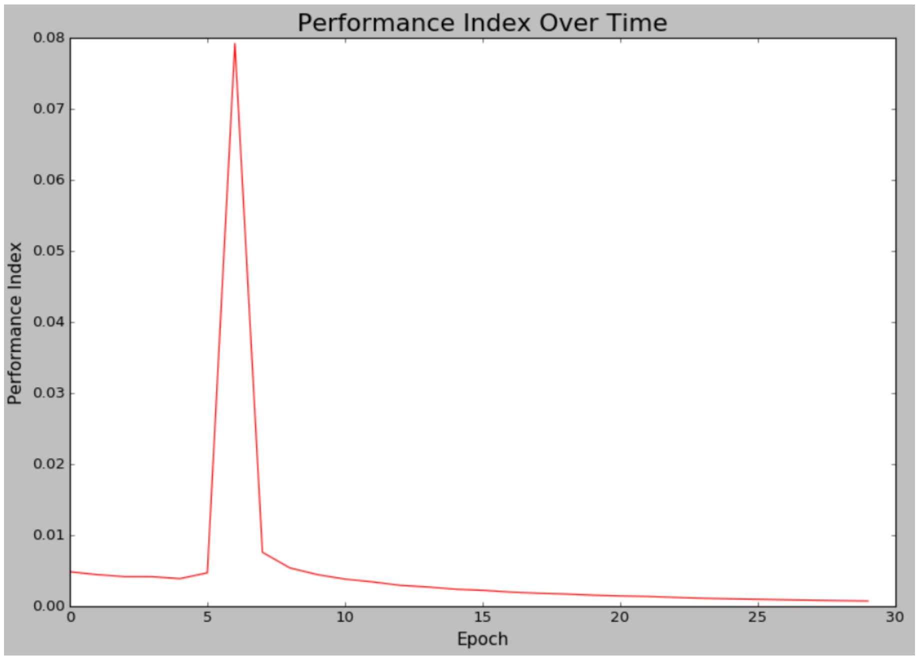
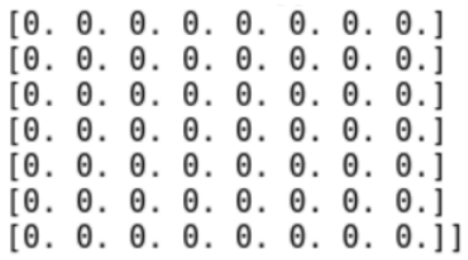


Figure 11: Loss over time

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 22%. Further, the confusion matrix is filled with 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. 0.]]
```

Figure 12: Confusion matrix

Given the low loss value at the end of 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.

Since we had a low accuracy rating, we decided to look at frequency domain. We used librosa to extract MFCC and spectral centroid for better results. After adjusting parameters and training the algorithm, we ran into a similar problem with the time domain data; the algorithm had low loss values but low accuracy rate around 30%.

The low loss–low accuracy issue may have three causes:

1. The train set is relatively 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 re-divide the size of train set to make it bigger, but it doesn't seem to have a significant impact on the time domain data.