**Team name:** Moving Pictures

**Members:** Anirban Bhattacharya, Bhaskar Mishra, Hao-Yu Liao, Ryon Kennedy

1. (20 points) Progress made in Milestone
2. Implementation of a baseline classifier on the training set.

We use the librosa library to transfer each song into the dB spectrogram with logarithmic scale as shown in Figure 1, and we retrieve 1 minute of each song from the middle points. For example, if a song has 3 minutes, the spectrogram will plot from 1:30 to 2:30. After retrieving the songs into dB spectrogram images, we resize the images into 200x200 sizes as shown in Figure 2. The resized images will be inputted into the model, and the output will be category labels such as Nonprog or Prog. The Prog images are labeled as 1, and the Nonprog images are labeled as 0. We have 253 songs made of 155 Nonprog and 98 Prog. 80% of the dataset is for training, and 20% of the dataset is for testing. The input is resized image, and the output is the labeled values (0 or 1). The process of implementation is shown in Figure 3. Currently, we train a convolution neural network (CNN ) as the classifier for our project by using Pytorch. The architecture of CNN is shown in Figure 4. We use the ReLU activation for each layer, and the last layer is Sigmoid activation. Also, we apply the batch normalization and max-pooling in the CNN. The proposed CNN has 5 convolution layers and one fully connected layer. The optimizer is stochastic gradient descent (SGD) with a 0.001 learning rate and 0.9 momentum. The loss function is binary cross entropy. The batch size for each iteration is 5.We take 10,000 epochs to train the CNN model.

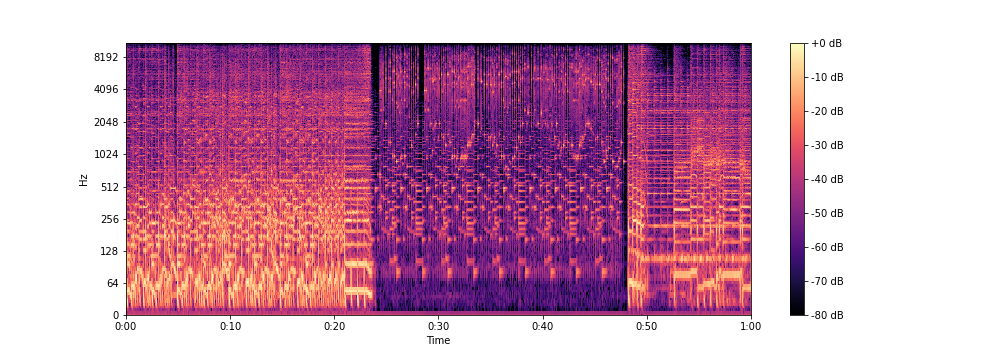


Figure 1 Transfer the song into dB spectrogram with logarithmic scale (Example of -04- Knots.mp3).

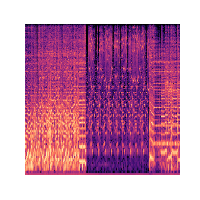


Figure 2 Resize dB spectrogram into 200 by 200 (Example of -04- Knots.mp3).

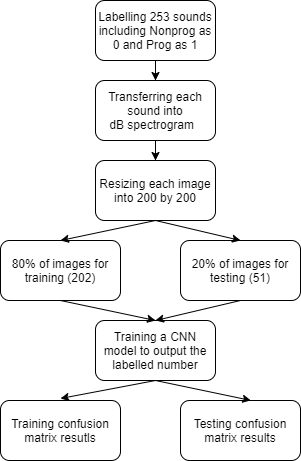


Figure 3 Flowchart of the implementation

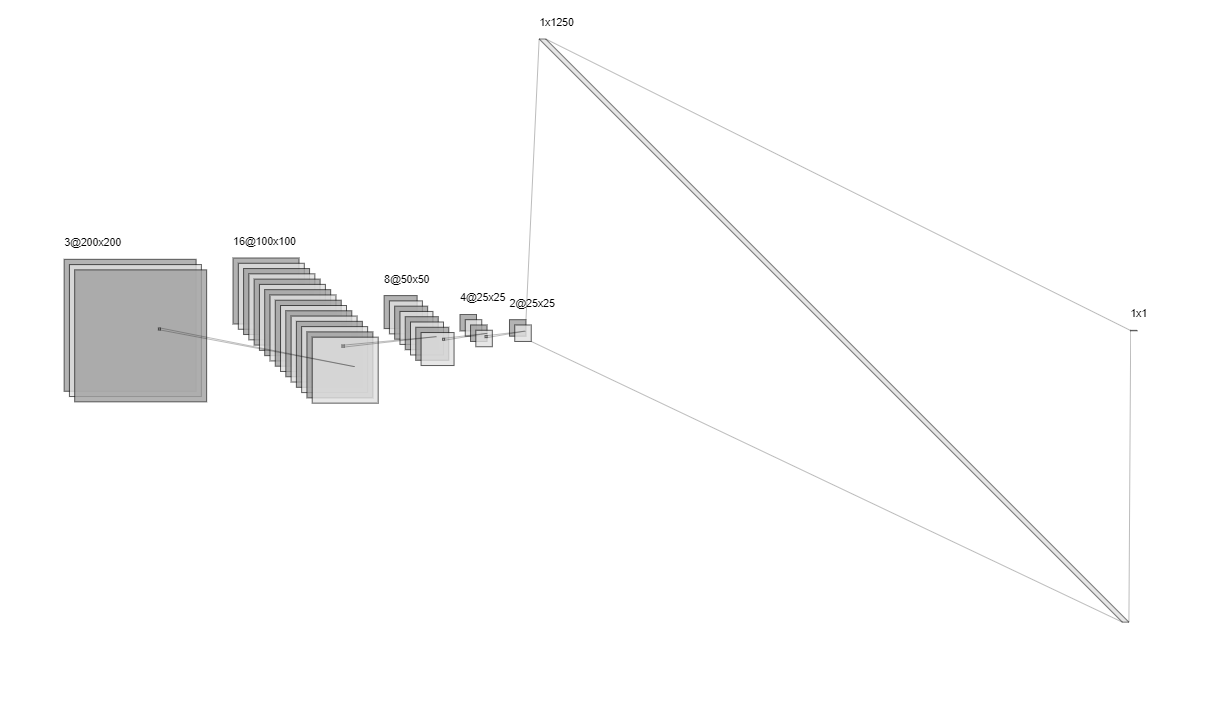


Figure 4 The architecture of CNN in the project.

1. (30 points) Performance of the classifiers on the Non-Prog and Prog training sets.
2. (10 points) Discussion of the techniques underpinning your classifiers.

The project is a binary classification problem because of only two categories such as Nonprog and Prog. Based on the problem, the techniques of our proposed model are following. First, we take the ReLU function for each layer except for the last layer. Because the project is a binary problem, after taking the ReLU function, each pixel will be transferred from 0 to a positive number. If some information in the feature maps of the convolution layer is not important, the information will be computed as 0 based on the ReLU function. The important information will be counted as a positive number. For the last layer with Sigmoid function because the labeled values are 0 or 1. The sigmoid function will output from 0 to 1. It is suitable for binary problems. Second, we take the binary cross entropy as the loss function because the classification problem is binary. Third, we also take the batch normalization in our architecture. The batch normalization can speed up the training process and enhance the training performance. Finally, the number of channels in the current layer is 2 times of the next layers before connecting a fully connected layer. Because the output value is 0 or 1 and the last layer is fully connected, the 2 times will be appropriate for the classification problem. The detail of the proposed model is shown:

Sequential(

(0): Conv2d(3, 16, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1))

(1): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

(2): ReLU(inplace=True)

(3): MaxPool2d(kernel\_size=2, stride=2, padding=0, dilation=1, ceil\_mode=False)

(4): Conv2d(16, 8, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1))

(5): BatchNorm2d(8, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

(6): ReLU(inplace=True)

(7): MaxPool2d(kernel\_size=2, stride=2, padding=0, dilation=1, ceil\_mode=False)

(8): Conv2d(8, 4, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1))

(9): BatchNorm2d(4, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

(10): ReLU(inplace=True)

(11): MaxPool2d(kernel\_size=2, stride=2, padding=0, dilation=1, ceil\_mode=False)

(12): Conv2d(4, 2, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1))

(13): BatchNorm2d(2, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

(14): ReLU(inplace=True)

(15): Flatten(start\_dim=1, end\_dim=-1)

(16): Linear(in\_features=1250, out\_features=1, bias=True)

(17): Sigmoid()

)

1. (20 points) Discussion and explanation of the performance of the classifiers on the training set.

Figures 5 and 6 show the proposed CNN training results of the confusion matrix. 202 dB spectrogram images are used in training the CNN model. The training results are very good with 1.0 accuracy. Figure 6 shows the normalization confusion matrix. It means that if we input 100 Nonprog songs to the CNN model, the model can classifier correctly with 100 percent. Because of the training results is super good. We can expect that the model is overfitting as we will discuss in testing results.

|  |  |
| --- | --- |
| Chart  Description automatically generated  Figure 5 The training confusion matrix results of CNN with 100% accuracy. | Chart, treemap chart  Description automatically generated  Figure 6 The training normalization confusion matrix results of CNN with 100% accuracy |

1. (50 points) Performance of the classifiers on the test set.
2. (20 points) Demo of performance of classifier on test set.

Figures 7 and 8 show the testing confusion matrix results. 51 dB spectrogram images are used to evaluate the CNN model. The accuracy of the testing results is 72% and the results are overfitting because of 100% accuracy in training results. These figures show the performance of the CNN model.

|  |  |
| --- | --- |
| Figure 7 The testing confusion matrix results of CNN with 72% accuracy. | Figure 8 The testing normalization confusion matrix results of CNN with 72% accuracy |

1. (30 points) Discussion and explanation of the performance of the classifiers on the test set.

We can see that the model is overfitting when we compare the training and testing results. In the testing results, the accuracy is 72%. It means that if we have 100 unknown labeled songs, 72 songs will be classified correctly. Based on the performance as shown in the confusion matrix, we see that the CNN model as the classifier has more performance on classing Nonprog music. From the normalization matrix, if we have 100 Nonprog songs with unknown labels to the model, 80 Nonprog songs will be classified correctly. However, the CNN model has lower performance on classing the Prog songs as shown in Figure 8. If we have 100 Prog songs, only 56 Prog songs will be classified correctly. It seems that it is hard to class Prog songs than the Nonprog songs as observing the confusion matrix. Although the training results are very good, the CNN model will need more improvement on the testing results. Also, to improve the overfitting problem, the main way is to increase the number of datasets. If we have more songs, the testing results will be improved.

**Appendix**

1. The extracted dB spectrogram images can be found by the link (<https://uflorida-my.sharepoint.com/:f:/g/personal/haoyuliao_ufl_edu/Eq1jMHKs_7NChCSS93zoFUABzzUw2SKT6pfKPT7YrLk7sA?e=z75RrF>). In the link, we don’t put any music because of copyright. Only images and the programs such as ExtractFeatures.ipynb and CNN2\_BCELoss.ipynb can be found in the link.
2. ExtractFeatures.ipynb: Extract each song into dB spectrogram.

#!/usr/bin/env python

# coding: utf-8

# In[37]:

#https://librosa.org/doc/main/auto\_examples/plot\_display.html

import librosa, audioread

import librosa.display

import matplotlib.pyplot as plt

import pandas as pd

import numpy as np

import os, cv2, warnings

from pydub import AudioSegment

folders=['Progressive\_Rock\_Songs','Not\_Progressive\_Rock/Other\_Songs','Not\_Progressive\_Rock/Top\_Of\_The\_Pops']

saveFolders = ['ExtractDataset/prog/', 'ExtractDataset/nonprog/']

#1 for prog; 0 for nonprog

database = {'idx':[],'ReName':[],'Style':[],'Label':[],'StartDur (Sec)':[],'EndDur (Sec)':[],'Duration (Sec)':[]

,'OriName':[], 'DbMax':[],'DbMin':[]}

saveArr = []

saveImgArr = []

saveImgArrLog=[]

labels = []

filesName = []

for folder in folders:

if folder == 'Progressive\_Rock\_Songs':

saveFolder = saveFolders[0]

reName = 'Prog-'

style = 'Prog'

idx = 0

label = 1

if folder == 'Not\_Progressive\_Rock/Other\_Songs':

saveFolder = saveFolders[1]

reName = 'NonProgOther-'

style = 'NonProg'

idx = 0

label = 0

if folder == 'Not\_Progressive\_Rock/Top\_Of\_The\_Pops':

reName = 'NonProgTopPops-'

label = 0

filesNames = os.listdir('./Dataset/'+folder)

for i in range(len(filesNames)):

fileName = filesNames[i]

print(i)

print(fileName)

audio\_path = './Dataset/'+folder+'/'+fileName

formalName = fileName.split('.')[-1]

FilExt = fileName.split('.')[-1]

if FilExt != 'mp3':

flac\_audio = AudioSegment.from\_file('./Dataset/'+folder+'/'+fileName, FilExt)

fileName = fileName.replace(fileName.split('.')[-1],'')+'mp3'

audio\_path = './FlatToMP3/'+folder+'/'+fileName

flac\_audio.export(audio\_path, format="mp3")

print(fileName)

saveName = reName+str(idx)

totalDur = audioread.audio\_open(audio\_path).duration

plt.cla()

plt.clf()

plt.close('all')

plt.figure(1, figsize=(14, 5))

x , sr = librosa.load(audio\_path, sr=22050, offset=totalDur/2-30, duration=60.0)

#display Spectrogram

X = librosa.stft(x, hop\_length=256, n\_fft=4096)

Xdb = librosa.amplitude\_to\_db(abs(X), ref=np.max)

print(Xdb.shape)

if Xdb.shape[1] != 5168:

continue

#librosa.display.specshow(Xdb, hop\_length=256, sr=sr, x\_axis='time', y\_axis='hz', cmap='gray\_r') #, vmin=-50, vmax=50,

librosa.display.specshow(Xdb, hop\_length=256, sr=sr, x\_axis='time', y\_axis='hz') #, vmin=-50, vmax=50,

cb=plt.colorbar(format="%+2.f dB")

plt.savefig(saveFolder+saveName+'-withAix.png')

plt.axis('off')

cb.remove()

plt.draw() #update plot

plt.savefig(saveFolder+saveName+'.png')

#plt.show()

###Resize each img size into 200x200 for CNN training

readImg1 = cv2.imread(saveFolder+saveName+'.png')

resizeImg1 = cv2.resize(readImg1,(200,200))

cv2.imwrite(saveFolder+saveName+'-Resize.png', resizeImg1)

#########################################################

###Log img###############################################

plt.cla()

plt.clf()

plt.close('all')

plt.figure(2, figsize=(14, 5))

librosa.display.specshow(Xdb, hop\_length=256, sr=sr, x\_axis='time', y\_axis='log') #, vmin=-50, vmax=50,

cb=plt.colorbar(format="%+2.f dB")

plt.savefig(saveFolder+saveName+'-LogwithAix.png')

plt.axis('off')

cb.remove()

plt.draw() #update plot

plt.savefig(saveFolder+saveName+'-Log.png')

#plt.show()

###Resize each img size into 200x200 for CNN training

readImg2 = cv2.imread(saveFolder+saveName+'-Log.png')

resizeImg2 = cv2.resize(readImg2,(200,200))

cv2.imwrite(saveFolder+saveName+'-LogResize.png', resizeImg2)

##############################################################

##############################################################

saveArr.append(Xdb)

labels.append(label)

filesName.append(fileName)

saveImgArr.append(resizeImg1)

saveImgArrLog.append(resizeImg2)

database['idx'].append(idx)

database['ReName'].append(saveName)

database['OriName'].append(fileName)

database['Style'].append(style)

database['Label'].append(label)

database['StartDur (Sec)'].append(totalDur/2-30)

database['EndDur (Sec)'].append(totalDur/2-30+60)

database['Duration (Sec)'].append(totalDur)

database['DbMax'].append(Xdb.max())

database['DbMin'].append(Xdb.min())

idx+=1

#break

np.save('./ExtractDataset/DBHzarray.npy', saveArr)

np.save('./ExtractDataset/Imgages.npy', saveImgArr)

np.save('./ExtractDataset/LogImages.npy', saveImgArrLog)

np.save('./ExtractDataset/Labels.npy', labels)

np.save('./ExtractDataset/FilesName.npy', filesName)

DatabaseToPd = pd.DataFrame(data=database)

DatabaseToPd.to\_excel('Database.xlsx', index=True)

1. CNN2\_BCELoss.ipynb: The CNN classifier to class Nonprog and prog songs.

#!/usr/bin/env python

# coding: utf-8

# In[32]:

#https://pytorch.org/tutorials/beginner/blitz/cifar10\_tutorial.html#training-on-gpu

import torch, torchvision, os, cv2

from torch.utils.data import random\_split, Dataset

from torch.nn import Sigmoid, Tanh, Linear, ReLU, Sequential, Conv2d, MaxPool2d, Sigmoid, BatchNorm2d, Flatten, ConvTranspose2d

import torch.nn as nn

import torch.optim as optim

import numpy as np

import matplotlib.pyplot as plt

images = np.load('LogImages.npy') #Rade the trianing data.

images = np.moveaxis(images, -1, 1) #Reshape channeL from [B, H, W, C] to [B, C, H, W]

labels = np.load('Labels.npy') #Rade the trianing data.

labels = labels.reshape(labels.shape[0],1)

# images = images.astype(np.float32)

# labels = labels.astype(np.int)

print(labels.shape)

labels2D = np.zeros((labels.shape[0],2))

for i in range(len(labels)):

lab = labels[i]

if lab == 0:

labels2D[i,0] = 1

if lab == 1:

labels2D[i,1] = 1

lengths = [round(len(images)\*0.8), round(len(images)\*0.2)]

print(lengths)

trainImg, testImg = random\_split(images, lengths ,generator=torch.random.manual\_seed(42)) #Shuffle data with random seed 42 before split train and test

trainLab, testLab = random\_split(labels, lengths ,generator=torch.random.manual\_seed(42)) #Shuffle data with random seed 42 before split train and test

print(trainImg[0].shape)

print(trainLab[25])

trainData = []

for i in range(len(trainImg)):

trainData.append([torch.tensor(trainImg[i], dtype=torch.float32), torch.tensor(trainLab[i],dtype=torch.float32)])

trainLoader = torch.utils.data.DataLoader(trainData, shuffle=True, batch\_size=5)

testData = []

for i in range(len(testImg)):

testData.append([torch.tensor(testImg[i], dtype=torch.float32), torch.tensor(testLab[i],dtype=torch.float32)])

testLoader = torch.utils.data.DataLoader(testData, shuffle=False, batch\_size=5)

# In[33]:

net = Sequential(

# Defining 1st 2D convolution layer

Conv2d(3, 16, kernel\_size=3, stride=1, padding=1), #200@3

BatchNorm2d(16),

ReLU(inplace=True),

MaxPool2d(kernel\_size=2, stride=2),

# Defining 2nd 2D convolution layer

Conv2d(16, 8, kernel\_size=3, stride=1, padding=1), #100@3

BatchNorm2d(8),

ReLU(inplace=True),

MaxPool2d(kernel\_size=2, stride=2),

# Defining 3rd 2D convolution layer

Conv2d(8, 4, kernel\_size=3, stride=1, padding=1), #50@3

BatchNorm2d(4),

ReLU(inplace=True),

MaxPool2d(kernel\_size=2, stride=2),

# Defining 4th 2D convolution layer

Conv2d(4, 2, kernel\_size=3, stride=1, padding=1), #25@3

BatchNorm2d(2),

ReLU(inplace=True),

Flatten(),

Linear(2 \* 25 \* 25, 1),

Sigmoid()

)

net = net.cuda()

#criterion = nn.CrossEntropyLoss()

criterion = nn.BCELoss()

optimizer = optim.SGD(net.parameters(), lr=0.001, momentum=0.9)

print(net)

# In[34]:

#%%time

n\_epoch = 10000

for epoch in range(n\_epoch): # loop over the dataset multiple times

#epoch = 0

#while True:

train\_loss = 0.0

for i, data in enumerate(trainLoader, 0):

# get the inputs; data is a list of [inputs, labels]

inputs, labels = data[0].cuda(), data[1].cuda() #Reg

# zero the parameter gradients

optimizer.zero\_grad()

# forward + backward + optimize

outputs = net(inputs)

labels = torch.reshape(labels, (-1,))

outputs = torch.reshape(outputs, (-1,))

loss = criterion(outputs, labels)

loss.backward()

optimizer.step()

# print statistics

train\_loss += loss.item()

logStr = 'Train epoch: %d, Loss: %.10f' % (epoch + 1, train\_loss/ (i+1))

print(logStr)

epoch += 1

print('Finished Training')

# In[46]:

# PATH = './CNN2\_SGDlr0.001\_BCEL\_EP10000\_72%.pth'

# torch.save(net.state\_dict(), PATH)

# In[44]:

correct = 0

total = 0

nb\_classes = 2

confusion\_matrix = torch.zeros(nb\_classes, nb\_classes)

with torch.no\_grad():

for data in trainLoader:

images, labels = data[0].cuda(), data[1].cuda()

outputs = net(images)

predicted = torch.round(outputs)

total += labels.size(0)

correct += (predicted == labels).sum().item()

for t, p in zip(labels.view(-1), predicted.view(-1)):

confusion\_matrix[t.long(), p.long()] += 1

print('Accuracy of the network on the %s test images: %d %%' % (total,

100 \* correct / total))

print(confusion\_matrix)

# In[48]:

correct = 0

total = 0

nb\_classes = 2

confusion\_matrix = torch.zeros(nb\_classes, nb\_classes)

with torch.no\_grad():

for data in testLoader:

images, labels = data[0].cuda(), data[1].cuda()

outputs = net(images)

predicted = torch.round(outputs)

total += labels.size(0)

correct += (predicted == labels).sum().item()

for t, p in zip(labels.view(-1), predicted.view(-1)):

confusion\_matrix[t.long(), p.long()] += 1

print('Accuracy of the network on the %s test images: %d %%' % (total,

100 \* correct / total))

print(confusion\_matrix)