Intro to Machine Learning Project 3

Super Vector Machines and Neural Networks

By Cody Crane and Eric Crozier

Cody Crane
Eric Crozier

Description of Data Representation

For our three data representations we choose, MFCC coefficients, Spectrograms and the raw signal data.

We chose Spectrograms specifically so we could build a convolutional Neural Network using them, as we needed multidimensional data, preferably images for a convolutional network. To make our spectrograms we took the first 29 seconds of each of the supplied test mp3 and fed it through this code, modified from this code StackOverflow answer here.

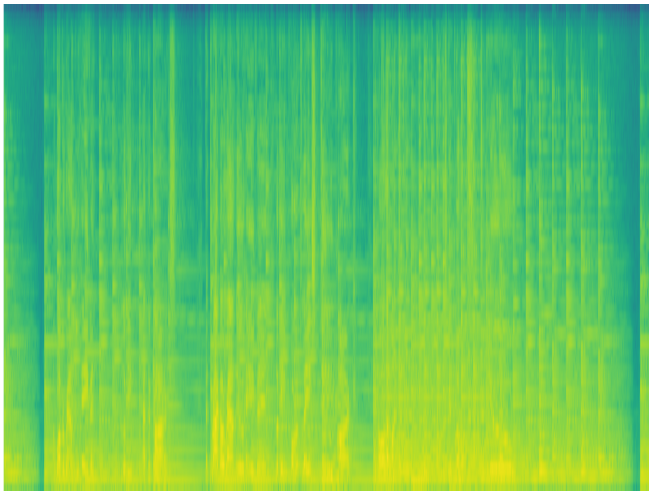
```
import librosa as lb
import os
import matplotlib.pyplot as plt


for fileName in os.scandir(os.path.abspath("./MusicData/test")):
    data, FS = lb.load(os.path.abspath(fileName), duration=29)
    fig, ax = plt.subplots(1)
    fig.subplots_adjust(left=0, right=1, bottom=0, top=1)
    ax.axis('off')
    pxx, freqs, bins, im = ax.specgram(x=data, Fs = FS, noverlap=0, NFFT=128)
    ax.axis('off')
    specFileName = os.path.abspath(os.path.abspath("./testSpecGrams") + '/' + fileName.name[:len(fileName.name)-3] + "png")
    fig.savefig(specFileName, dpi=300)
```

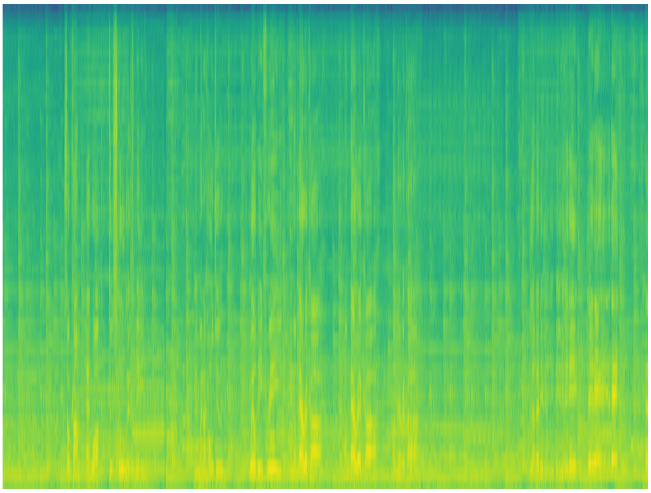
This gave us 1440 x 1980 color spectrogram images to use for our neural network. We chose this representation because for one it provided us with a challenge in data preprocessing that we had not been exposed to in this class before and because we thought the spatiality and locality provided by the spectrograms could provide more information and thus more accuracy for our machines.

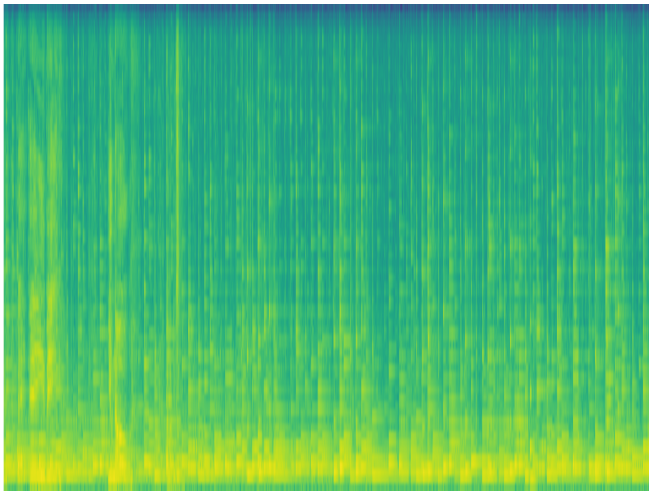
We choose MFCC coefficients for our dimensionally reduced data representation as they were unique to audio processing and were mentioned in the PDF. We choose these specifically for our fully connected neural network as we could not use the raw data due to computational constraints. The coefficients gave us usable data that could be computed in a reasonable amount of time for our fully connected neural network.

Finally we chose the raw signal data from each of the mp3's because we wanted to use something that had a minimal amount of preprocessing in it and see what we could come up with. We chose to use this representation with our super vector machine due to computation constraints given the size of the raw data and the number of instances we need to calculate. SVM's provide themselves well to this task as they are relatively easy to compute and work very well with numerical data like this. The two neural networks code is based on code from Martin Gorner, and the TensorFlow website, here, here, and here.

Here are some examples of the spectrograms we pulled from the testing data.







Cody Crane
Eric Crozier

Description of Classifiers

We choose to train a super vector machine, a fully connected neural network and a convolutional neural network for our three classifiers.

Support Vector Machine

We chose to use the raw signal data from the mp3 files to train the SVM. This was done easily using the Librosa package for python, which specializes in music and audio analysis. Thankfully Librosa can load in an mp3 and return the raw signal data for a given file, along with the sampling of the raw signal data.

From there we can get the raw signal data from all 2400 mp3 files of the training data set, along with the correct classifications for each training mp3. We used the package sklearn which has functions for training and creating an SVM. The SVM can be created, and then fit with the signal data from the training files, and the correct classification for each file. Then the SVM can predict an mp3 file based on the signal data from the training set.

Then the signal data can be extracted from the testing set into a list, and the entire list can be predicted with the SVM, and a list of 1200 classifications is output.

Accuracy Report: Support Vector Machine

The accuracy of the classifications from the testing set on our first try was around 15%, and then increased to 19% with some minor adjustments. I believe this number is low because we used the raw signal data to train the set, which was the most basic of data with no transformations done. So, the predictions were not the most accurate, but still much more accurate than guessing

randomly. The whole process of extracting the raw signal data from the training/testing sets, training the SVM, and predicting the testing set took around 90 – 120 minutes.

Fully Connected Neural Network

For the fully connected Neural Network we used the MFCC coefficients of the first 29 seconds of each mp3 file as our data. To do this we simply read in the signal data from each file, stacked them and saved them to one large .npz that we would load as our dataset, these can be seen in the fullyConnected[test/train]DataToNPZ.py. Within our model we used 4 layers, with 64, 16, 4, and 6 neurons respectively. The final layer uses softmax for our predictions and the rest use relu. We chose this as we did not want to go too deep and our input dimension was about 64,000 and thus thought 64 was a good place to start.

Accuracy Report: Fully Connected Neural Network

Unfortunately we did not have the time to quite tune our machines, or get much useful information out of them, I will repeat this below but the lack of completeness on the fully connected neural net and the convolutional neural net is my(Cody Crane) fault and I take full responsibility and only ask that it should not reflect poorly on my partners grade. Below is the screenshot of the fully connected neural nets readout after 10000 epochs. Loss and accuracy while high are probably misleading as I did not have time to set up proper training and evaluation sets. This net produced useless predictions, they were all predicted as 4's, this problem rings of a similar problem I had when doing our logistic regression and is most likely indicative of bad normalization of our data, as I did no normalization on the MFCC coefficients.

```
75/75 [==============================] - 0s 2ms/step - loss: 1.7924 - accuracy: 0.1104
Epoch 9987/10000
75/75 [==============================] - 0s 2ms/step - loss: 1.7924 - accuracy: 0.1104
Epoch 9988/10000
75/75 [==============================] - 0s 2ms/step - loss: 1.7924 - accuracy: 0.1104
Epoch 9989/10000
75/75 [==============================] - 0s 2ms/step - loss: 1.7924 - accuracy: 0.1104
Epoch 9990/10000
75/75 [==============================] - 0s 1ms/step - loss: 1.7924 - accuracy: 0.1104
Epoch 9991/10000
75/75 [==============================] - 0s 1ms/step - loss: 1.7924 - accuracy: 0.1104
Epoch 9992/10000
75/75 [==============================] - 0s 1ms/step - loss: 1.7924 - accuracy: 0.1104
Epoch 9993/10000
75/75 [==============================] - 0s 2ms/step - loss: 1.7924 - accuracy: 0.1104
Epoch 9994/10000
75/75 [==============================] - 0s 2ms/step - loss: 1.7924 - accuracy: 0.1104
Epoch 9995/10000
75/75 [==============================] - 0s 2ms/step - loss: 1.7924 - accuracy: 0.1104
Epoch 9996/10000
75/75 [==============================] - 0s 2ms/step - loss: 1.7924 - accuracy: 0.1104
Epoch 9997/10000
75/75 [==============================] - 0s 2ms/step - loss: 1.7924 - accuracy: 0.1104
Epoch 9998/10000
75/75 [==============================] - 0s 1ms/step - loss: 1.7924 - accuracy: 0.1104
Epoch 9999/10000
75/75 [==============================] - 0s 1ms/step - loss: 1.7924 - accuracy: 0.1104
Epoch 10000/10000
75/75 [==============================] - 0s 1ms/step - loss: 1.7924 - accuracy: 0.1104
(base) cranec@ganymede:~/MLProj3$
```

Convolutional Neural Network

For this network we used 1440x1920 color spectrogram images of each of the mp3's supplied. We used an 80-20 split of the training data for training and validation. Our architecture consisted of 5 convolutional layers and 4 dense layers. We used the relu activation function throughout our hidden layers and softmax for our prediction layer.

First layer uses 8x8 kernels and a single step size to produce two channels of information giving us a 1140x1980x2 tensor.

The second layer uses a 5x5 kernel with a step size of 2 in both directions and produces 3 channels of output giving us a 720x960x3 tensor.

The third layer uses a 3x3 kernel with step size 2 and 5 output channels. This produces a 360x480x5 tensor.

The fourth layer uses a 2x2 kernel, step size 2, and 8 output channels. This provides a 180x240x8 tensor.

The final convolutional layer uses again a 2x2 kernel with step size 2 but produces 13 output channels. This gives us a tensor of 90x120x13, which we then flatten into a vector of length 140,400.

We then feed this vector into our first dense layer that has 10,000 neurons, the second dense layer contains 1,000 neurons, the third 200 and our final layer contains 6 neurons for each of our classes.

Our batch size is 32 and our epochs 10,000, we used relu and softmax for our activation functions as they are the industry standard for classification problems. We also employed a dropout of 25% on each hidden dense layer.We choose our architecture in such a way as to use the fibonacci sequence in both our kernel size and output channels. Possible improvements for this function would be to reduce the size of our images as we end up with very, very large tensors. This however does come at the cost of information in our spectrograms.

Accuracy Report: Convolutional Neural Network

Here's where things get tricky. I(Cody Crane, the work done and not done on the convolution network is my responsibility) was way too ambitious in my architecture design of the convolutional network, keeping the images as full 1440 by 1920 images was a massive oversight on my part. As such I was not able to complete the training of the machine and thus I do not have any accuracy to report. I can make excuses about the time available to us and my inexperience with tensorflow, keras and python in general but in the end I did not give myself enough time to

work through this and as a result was not able to complete it to the best of my ability. I apologize for my lack of foresight and my procrastination and ask only that my mistakes do not reflect poorly on my partner's grade as he was counting on me and I let him down as well.