Middlebury College

Class(ical) Registration

Utilizing deep learning to understand relationships between different periods of classical music

Christian Chiang

CSCI 0500

Prof. Jason Grant

5/16/19

The genre of classical music encompasses many periods, or subgenres, from a broad span of time, ranging from renaissance music in the 6th century AD to modern and contemporary classical music now. Many of these classical music periods do not have definite dates where they began and ended, but rather periods of classical music slowly developed from influences of previous compositions. For example, music from the baroque period shared similar music theory rules and structures as music from the classical period. Thus, the classical music periods can show clear relationships between each other, often sharing similar chord structures, instruments, and musical notation.

To explore possible relationships between composers in different classical music periods, Professor Grant and I worked this semester in our CS 500 Advanced Study to train a convolutional neural network to recognize, classify, and provide a hierarchy for different samples of classical music. Observing how the neural net processes information from different kinds of music could reveal similar patterns in classical compositions, suggesting how some composers could had been influenced by previous classical music pieces.

Generally, our project worked as follows: given a large dataset of different classical music songs and a general list of which classical composer belonged to which classical music period, we would create another dataset consisting of spectrograms, visual representations of the frequencies that occur throughout an audio file, and slice each spectrogram of a song into three-second samples. We would then generate a random selection of samples from our spectrogram dataset to train, test, and validate a neural network. Once our training is one, upon giving it a random image sample of a spectrogram, our neural net would return a hierarchy of what the sample is most likely classified as. This hierarchy would return a percentage breakdown

of how likely it is to belong to a certain class, providing insight to similar classical music periods and compositions. Since we were training the neural network with seven different classical music periods, we aimed to train a neural net to get a test accuracy rate as high as possible, but still achieve a rate that surpassed random guessing at 14%.

For our study, we decided to break up the semester into two parts: coding and setting up the technical parts, and training our neural net and analyzing our results. We anticipated that setting up our entire environment would take a little more than half of the semester (about seven to eight weeks), and we expected to train our neural net and try to optimize our results for the last four weeks of the semester. We stayed true to our schedule in terms of timing, but in our "training" part of the semester, we did not get to look at clustering algorithms with our results due to debugging and unforeseen complications with making our dataset compatible with our coding environment. Though it would have been nice to look at additional insights from our findings from our deep learning training, clustering algorithms were not as critical to our project as actually training and testing the neural net.

All of our scripts essential to both preprocessing our data and training our neural network were written in Python. We would test our scripts and code on small datasets in Python notebooks on personal machines, but our large datasets and big processes would take place in a working Conda environment on Gattaca, the Linux workstation belonging to Prof. Linderman and the CS department. Using Gattaca gave our project ample storage and RAM to run our preprocessing and training methods. Moreover, working with Gattaca throughout the semester also practiced good use of Linux terminal commands.

For actually training our neural network, we used Tensorflow, a multipurpose machine learning framework. We used a MobileNet model for our convolutional neural net training to make an image classifier with our spectrogram dataset. MobileNet is a small, versatile, open-source computer vision model for Tensorflow that can also meet parameters for object detection and facial recognition. Most of our processes for training using MobileNet were run through a combination of Python code and Linux terminal commands.

Besides Tensorflow, other packages critical to our project included Matplotlib, to create and plot the spectrograms we used as input for our model, and SciPy.io, to interpret and read in our initial .wav files. Our classical music dataset was pulled from two albums titled "24 Hours of Classical Music – The Perfect Start to Your Collection" (licensed by Naxos) and "The Classical Piano Collection" (licensed by ABC Australia). These collections were used for no purpose other than using their data to convert songs into three-second spectrogram samples.

We began the semester by reviewing an *Hacker Noon* article by Julien Despois.[1] In his study, Despois trained a neural network to distinguish a given music sample's genre of music. He only used four genres, or classes, for classification: rap, classical, metal, and EDM / dubstep. Notably, each one of the genres he used are vastly different in style, meter, and frequency. For example, a spectrogram for a modern-day hip hop song would most likely contain a lot of deep, 808-like bass sounds, while classical music would show higher frequencies for strings and horn sections in their spectrograms. Thus, Despois' audio classification project reached a 90% accuracy rate on its validation dataset. After reviewing this article, though, Professor Grant and I sought to explore how well a trained neural net could recognize different styles of music within

[1] Despois, Julien. "Finding the Genre of a Song with Deep Learning - A.I. Odyssey Part. 1." *Medium*, Medium, 21 Nov. 2016, medium.com/@juliendespois/finding-the-genre-of-a-song-with-deep-learning-da8f59a61194.

the classical music genre; we wanted to see if a neural net could detect and make distinction between minor details in frequency changes that are representative of different periods in classical music.

Our first step to preprocessing was creating a script that could convert any audio file in .wav format and converting it to one full picture of its spectrogram (we would handle the slicing later). In this stage of preprocessing, we used Matplotlib, SciPy.io, and the OS module for reading in our input files, convert them from stereo to mono, creating spectrograms of them, and saving them into their own named directories. We specifically wanted all our .wav files to be converted to mono instead of stereo, because stereo audio shows the spectrograms of two different audio files that are oftentimes almost identical. Once we were able to save the standard spectrogram of a mono audio file, we multiplied the frame rate by a factor of 3 and ran a for-loop that iterated through the spectrogram thrice as long, saving a picture of every three-second slice into a directory specific to its input file name. For example, a file called "Bach_1.wav" would output a directory called "Bach_1_spectrograms," in which each spectrogram would be named in order of appearance, "Bach_1_1.png," "Bach_1_2.png," etc. We initially named our files with hyphens "-", but we changed to underscores "_" so we could account for composers with hyphenated last names (Rimsky-Korsakov, Kats-Chernin) without breaking our Python script. Once we were sure that our code would run properly and correctly given any input, we concatenated both our datasets into one big directory called "MasterFolder" and ran a Parallel command to run the Python script on any file ending in .wav in the directory. This process enabled us to utilize our dataset to its fullest, creating far more spectrograms for training and testing instead of just using one spectrogram per classical music composition. During this

process, since we now had a finite list of which composers we would be looking at for data, we created seven .txt files, one for each classical music period, and did simple research on which classical composer in the dataset belonged to which classical music period. Though small, these files were critical to labeling our data correctly within the proper music period.

Once we had all of our spectrogram data created, we needed to come up with an algorithm that would help choose random yet even samplings of spectrograms from our dataset. Our objective was to take an even number of spectrograms from each individual composer in our dataset, removing those with only one composition. When iterating through the artists in our dataset, we specifically did not include composers with only one composition; if they were not a prominent enough composer with many pieces, it would be difficult to claim their music representative of any classical music period. We wanted an even number of spectrograms from each composer regardless of which music period they belonged to so we could utilize most of our dataset on each iteration of this algorithm. We did acknowledge that taking spectrograms from each composer regardless of music period would make our training and testing datasets a bit lopsided in favor of the Romantic period because the largest category of composers that we had was from the Romantic period, but we decided that if there were any large error in our test accuracy rate, it probably would not be because we gave it too much information on what Romantic-era classical music looked like.

The way our initial algorithm generated our datasets worked as follows: for each artist in the dataset, considering all of their songs, our code randomly iterated through the list of songs, choosing one spectrogram per song. Once a spectrogram from each song was selected, the code would randomly iterate through the list of songs again in a different order, choosing one more

spectrogram per song. This process was completed until we reached the amount of spectrograms we wanted from each composer. We called this method of getting data the "100 spectrograms model," because we would request an arbitrary amount of spectrograms from each composer (oftentimes 100), run this code to randomly generate the dataset, and move certain percentages of our entire selection into a training set, testing set, and validation set. The percentage breakdown of our generated dataset was as follows: 70% to training, 10% to testing, and 20% to validation.

After training our neural network several times with the "100 spectrograms model," we felt as if our selection algorithm wasn't getting enough data, so our second algorithm, called the "all spectrogram model," used the same random selection process but did not stop running until all available spectrograms we had in the dataset were placed in either the testing, training, or validation datasets. In other words, our second selection algorithm ran the same process, but exhausted all of our available data of around 24,000 spectrograms. Using all 24,000 spectrograms in our dataset ensured us that if any problem were to arise with our test accuracy rates, it would not be because we did not offer our neural network too little data. This was the algorithm that we used for our final model, and we also added a feature that, when running our final script, our dataset would be organized into subdirectories based on their music period i.e. seven directories named 20th century, 21st century, baroque, classical, impressionist, renaissance, and romantic. Creating subdirectories made it easier to deliver our dataset to MobileNet. Figure 1 in the appendix gives a visual of our final preprocessing pipeline in our project.

Before we decided to use a MobileNet model, we initially tried to familiarize ourselves with a basic classification tutorial on the Tensorflow website[2], assuming that we could just run the tutorial in our own way but just using our own data. However, the author of the tutorial used the standard fashion MNIST dataset to just show how the basic classifier works. Our data was not completely similar to how they imported their data, so we found a lot of trouble in simply trying to get our data in a compatible form for Tensorflow's basic classification tutorial. Our biggest concern at the time was that once we thought we were able to run the basic image classifier, we would get a returned test accuracy value that was always the same down to the decimal points at 42.857143%. We figured that if we were getting the same, exact result every time we tried training our neural network, even with different datasets, we either needed to reevaluate how we were inputting our training data and which model we wanted to use for our advanced study.

After our initial trouble with Tensorflow's basic classifier documentation, we decided to use MobileNet in a tutorial called "Tensorflow-for-Poets," a Google-sponsored demo on how to build a basic image classifier using the MobileNet model. For our MobileNet model, we used a pre-trained neural network and added one final layer to the network trained with our classical music spectrograms. In the interest of time during this semester, we did not want to build and train our own neural network from scratch but rather use a pre-trained model to understand the difficulties in the classical music recognition problem. We trained our neural net for 6000 steps, or iterations, to make sure our neural net reached the highest train accuracy rate it can achieve. Each time we trained our neural net, we would often get different final test accuracy ratings, so

[2] Chollet, François. "Train Your First Neural Network: Basic Classification | TensorFlow Core | TensorFlow." *TensorFlow*, 2017, www.tensorflow.org/tutorials/keras/basic_classification.

we ran twenty different trials with different spectrogram datasets and recorded our neural net accuracy ratings.

Our results were as follows: figure 2 shows a plotting of train accuracy rates every 100 steps to show how our model trended in accuracy through its training process, figure 3 gives insights to our results after recording our final test accuracy rates for twenty trials, and figure 4 and figure 5 show examples of our neural net model correctly and incorrectly (respectively) guessing the music period of two different samples.

A quick note for our results and figures: for figure 1, it's important to note that, although our neural net train accuracy rate trended upward the more steps it was trained, we often saw a few large dips in training accuracy, dropping about 10 to 20%. This phenomenon is also shown in figure 2, where some of our trials gave us final test accuracy ratings in the mid-30s (and a minimum of 35.2%), though accuracy ratings were trending upward around 50% during our neural network training. Overall, though, our final test results were rather good.

Considering our seven classical music periods, our trained model far surpassed random guessing, averaging a 52.1% test accuracy rate. But again, the results over our twenty trials showed not much trend or consistency. Our trials ranged from 35.2% to 63.9% in accuracy rating, with not much precision overall. We have a few hypotheses on why our neural net wasn't giving consistent results. Our first hypothesis is rooted in what kind of music we're feeding into our neural net as input data. After listening to some of our samples in the "The Complete Piano Collection" and "24 Hours" datasets, we noticed that the instrumentation widely varied from piece to piece. Some songs had full orchestra, some only had piano, and some had a mixture of both. We don't know how big of a roll instrumentation played in the role of training our neural

net, and in future work, we would want to make our dataset uniform in instrumentation by making every composition either all orchestra or all piano. That way, our neural net would not be misguided by thinking certain musical periods only had a full orchestra instrumentation while some only had solo piano ballads or pieces.

Our second hypothesis for our inconsistent test accuracy rates is because of how new and unfounded the problem we're trying to solve is. As we mentioned before, MobileNet is a neural network pre-trained using images from the "ImageNet Large Visual Recognition Challenge dataset." In other words, most neural nets are trained to see vast differences in images, like classifying different kinds of flowers, but may have trouble recognizing differences in spectrogram details. Additionally, Julien Despois' project for wider music genre recognition compared genres that were completely different in frequencies (like rap vs. classical music). Thus, the dependency of our neural net noticing details and differences in spectrogram frequencies between different kinds of classical music may be more difficult of a problem to solve than just any other kind of image classification.

If we had more time, we would try to control inconsistencies in our datasets, while trying to look at clustering methods to see the relationships between the classical music periods. Looking at clustering algorithms to organize our results can show what classical music periods share the most similarities, which was one of our initial goals when conducting this study. Capitalizing on the hierarchies the neural network finds on given music samples and clustering those samples is something we would look at in the future to draw conclusions to our project goals at large. Additionally, more time could enable our study to look at relationships between individual composers instead of just classical music periods.

Figure 1:

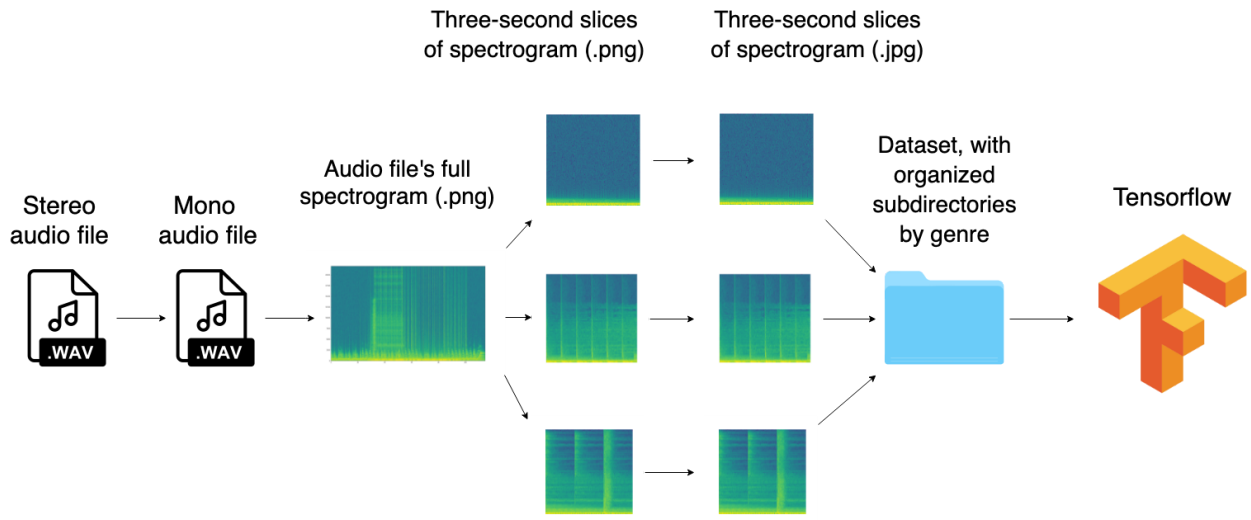

Figure 2:

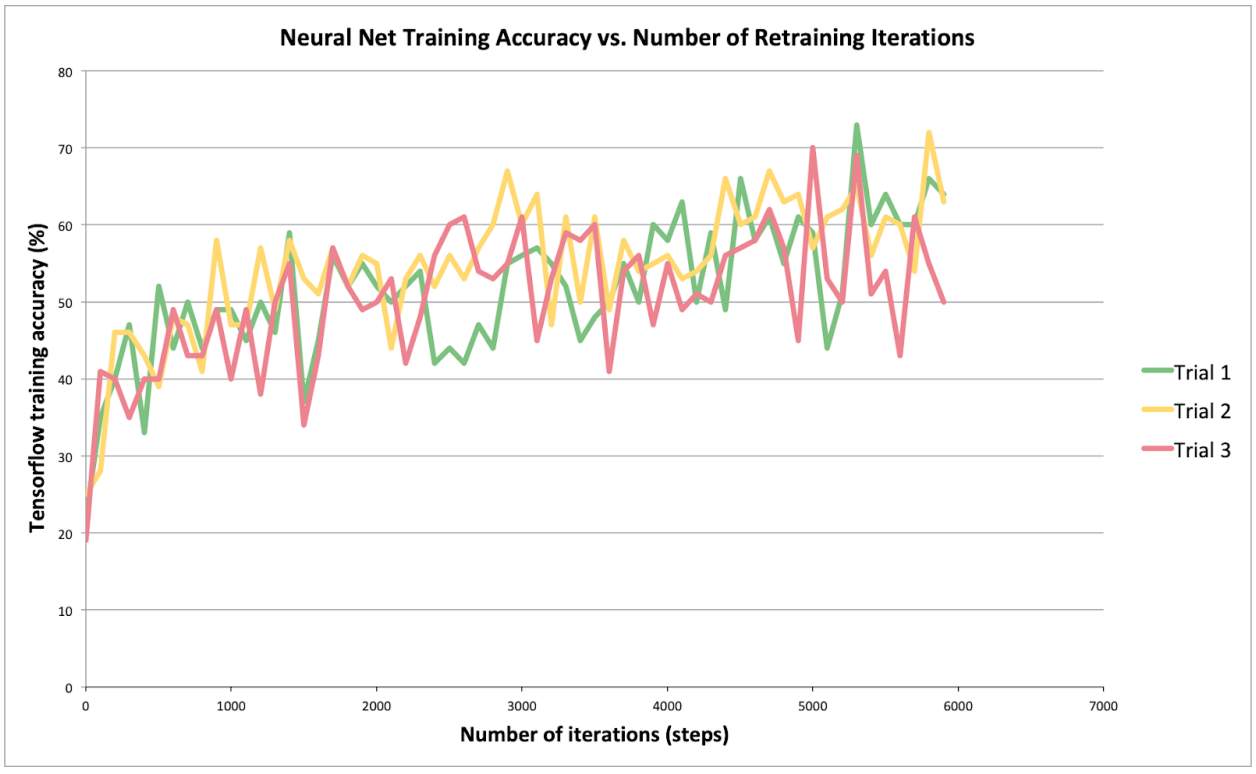

Figure 3:

Final Test Accuracy Ratings (20 Trials)

Measurement	Accuracy
Mean	52.1%
Median	55.0%
Minimum	35.2%
Maximum	63.9%

Figure 4:

```
Evaluation time (1-image): 0.472s

20thcentury (score=0.96422)
21stcentury (score=0.02586)
romantic (score=0.00562)
renaissance (score=0.00355)
(cs500-env) crchiang@gattaca:/home/jg
```

Figure 5:

```
Evaluation time (1-image): 0.471s

20thcentury (score=0.61108)
renaissance (score=0.21500)
romantic (score=0.12829)
classical (score=0.03912)
impressionist (score=0.00369)
(cs500-env) crchiang@gattaca:/home/jg
```