DRUM SOUND CLASSIFICATION

Benjamin L Buentello

ECEG 478, Fall 2023

Introduction

This project aims to design a neural network that can accurately distinguish between different types of drum sounds such as kicks and snares, both electronic and acoustic. The plan is to classify each data sample by analyzing the spectrograms of each different type of sound. This project is for my Machine Learning (ECEG 478) course.

Dataset

Collection

My dataset is gathered from many different locations. The initial set is taken from the machine learning dataset-sharing website Kaggle; it contains 40 samples of four different types of acoustic drum sounds. The license is listed as unknown and was sourced from an IEEE publication, "Drum Instrument Classification using Machine Learning". It has been used periodically since it was posted and there are no indications that anything went wrong, so I assume it is safe to use. The other part of my initial dataset is the FL Studio default drum sample pack. FL is a digital audio workstation used for the recording and production of music and contains a few hundred electronic drum samples. While I wouldn't be able to publish the sounds themselves or any derivative works for monetary purposes, I can freely use them for this unpaid class project. The rest of my dataset was gathered from a large number of websites listing royalty-free or free-use drum sample packs. These websites include:

Wavbvkery (Vintage Ludwig drum set)

Producers Buzz (Tom drum samples and Clueless 2 Hip-Hop Drum Sample kit)

The Metal Kick Drum (Toms Pack)

Deadloops (Studio Toms Sample Pack)

Black Lotus Audio (STUDIO)

Indie Drums (Slingerland Drum Samples and Rock Kit)

Music Radar (SampleRadar: cymbal samples)

StayOnBeat Live (Free Cymbal Sounds)

SM Drums (For Kontakt)

Description

- The overall dataset contains .wav file samples of each of the four drum types
- At the end of preprocessing, each data item is an array of fixed size (2, 100, 100)

- There are 1,000 of each type of drum sample, a total of 4,000 with all four labels
- The .wav files are loaded and modified with the Python module Librosa
- Cleaning:
    - Each sample is checked for .wav compatibility in case there are any prooblemaic files (of which there were a few due to the variety and quality of sources)
    - Each sample is made to be mono and any extra silence is trimmed
    - Each sample is either clipped or padded to 2 seconds and normalized
- Once the data was cleaned, any problematic files were removed and each class was augmented to 1,000 by selecting random samples and adding 10% noise to them. This noise injection was also to help with the overfitting problem that my first classifier had
- Preprocessing (app. 1):
    - Each sample was renamed to a uniform numbered naming scheme
    - Each sample was log-compressed in time by a factor of 0.1 and remapped to emphasize the initial transient
    - Each sample was converted to a Mel spectrogram, which is a spectrogram that is scaled to match the way humans hear frequencies
    - Each sample was log-compressed again in frequency by a factor of 0.1 and remapped to emphasize and spread out the lower frequencies. This would help the machine tell the difference between low tom drums and kicks
    - These compressed Mel spectrograms were plotted and saved as .pngs
- The samples in each sound type are shuffled separately and then joined together in an alternating fashion (0, 1, 2, 3, 0, 1, 2, . . .)
- Once together, the images are downsampled from (4, 217, 558) to (4, 100, 100) using the Python Pillow module and saved as NumPy arrays. Then, the first and last of the four channels were stripped off due to problematic data results (app. 2), ending with a shape of (2, 100, 100)
- Because the images themselves were put together in an alternating order, the label vector can be created by repeating the sequence [0, 1, 2, 3]. Once the data had a matching label vector, they were separated into training, development, and testing sets. The final sizes of those were 3200, 320, and 480 respectively

Representation

This dataset includes a mix of both electronically generated and acoustically recorded drum samples from many genres including hip-hop, metal, jazz, and electronic dance music. The machine will be able to train on the general aural similarities between drum sounds without being restricted to any particular genre or "feel". There is a perfect 1:1:1:1 balance between each type of drum sound as well, so the dataset representation is sufficient for what I would like this project to be capable of.

Goal of the Project

I plan to implement a convolutional neural network for this project for the classification of audio files. This type of neural network uses convolution layers to look for recognizable patterns in the data. The inputs would be sound samples and the output would be what type of sound sample it is. The practical applications of this project would be further development into drum pattern recognition and extraction from whole song samples. If you can recognize different drum sounds on their own, then you can expand to recognize them within other larger audio files. This would allow you to map their placement relative to each other and thus you will be able to extract drum patterns for recreation or further analysis.

Computational Setup

Testing and development of the project took take place on a few different computer setups. All files were stored and run from my BisonNet drive, which I used to access and develop my project seamlessly from any computer I used. The installation and running of special Python modules was simple for every setup. I used some standard ones like NumPy but also specialized ones for audio and image processing, namely Librosa and Pillow. I chose Librosa because it had a lot of flexibility with the types of .wav files it was able to process. Some .wav files are encoded differently than others, which is a problem I ran into because of how varied the sources of my dataset were. Pillow was chosen because it was one of the first to show up while searching. Initial testing was done on ACET 225 and MakerE computers, which had some memory issues, mostly because this was done before I started downsampling the data. My solutions to this were to use a smaller dataset to start with, add "del" statements to delete variables and free up space between iterations, and manually clear stored variables between runs. This worked well enough for my very first tests, but further development required maximum efficiency. For the bulk of project development, I used nine to ten computers in the Dana 307 lab. Even using my full dataset, I encountered no memory errors, especially since at this time, I had been downsampling my data and had also cut off two channels, which halved the size of the data on its own.

Methods and Architecture

Due to my lack of experience with Python, much of my code, especially for preprocessing the data, was written with help from Chat-GPT. I had it explain every step, however and was able to catch some bugs before they happened. A lot of the processes of it, such as normalization and trimming were from my own experiences working with audio, but the lo-compressions and Mel

spectrogram specifically were inspired by a YouTube video series, “Deep Learning for Audio Cassification” by Seth Adams. My first minimum feasible system was a neural network based on the example code from class, “mnist_experiments02”. I used 3 ReLU activation layers of size 500, 300, and 100 before the softmax layer, and used a cross-entropy loss cost function, which wasn’t a change from the original code, and I used a minuscule version of my dataset, just the 160 Kaggle samples. The final code to run the model itself was based on the example code from class, “mnist_experiments04”. This was the convolutional neural network example. I took out some things, such as the comparison with the k-nearest neighbor classifier. After much testing, the architecture consisted of two convolutional layers, a linear layer, and a dropout layer before the softmax layer. The activation function for all of these was the ReLU function, and I used a batch size of 32 and a kernel size of 5 for each layer. The hyperparameters ended up not being very different than the example, but that’s just what the experiments resulted in.

Experimental Setup

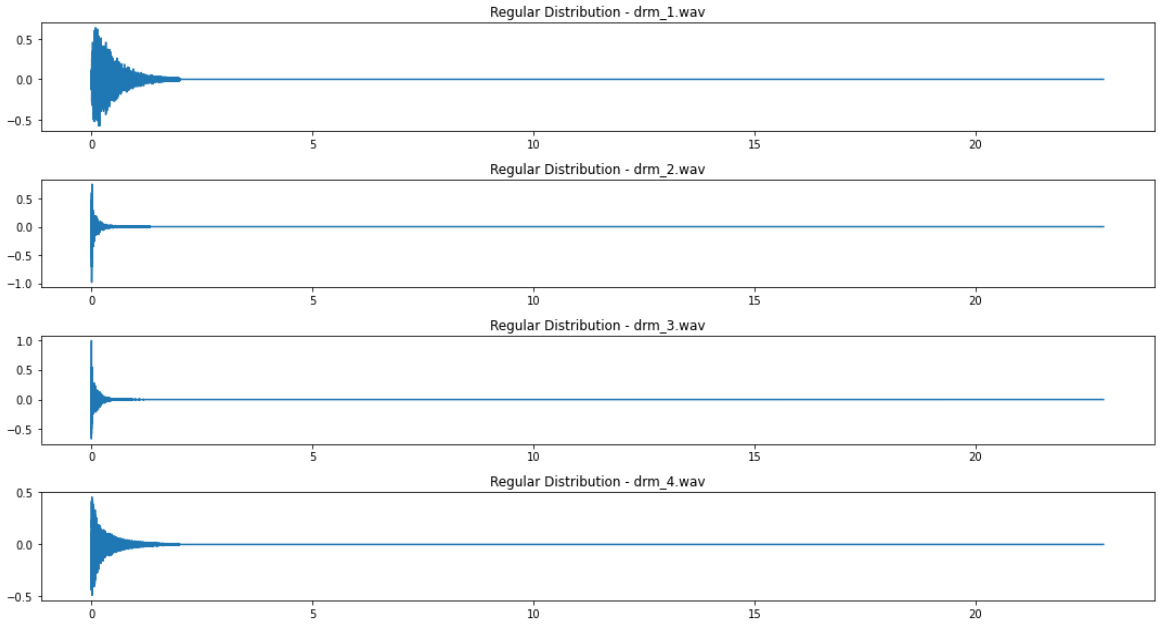
The initial experiment was done on my usual ACET 225 computer. The data preprocessing only consisted of taking the spectrogram and passing the intensities as a NumPy array. While the results were not great, the model did work. The dataset from Kaggle was too uniform, which caused massive overfitting.

After that, I spent a lot of time working on preprocessing my data so I could use more of it than just the 160 Kaggle samples. I ended up discovering and fixing some problems with how the data was encoded and was able to fix it (app. 2) by only using kicks and cymbals. Once this was done, I began testing other aspects. Using a 1000-sized dataset of just kicks and cymbals and 10

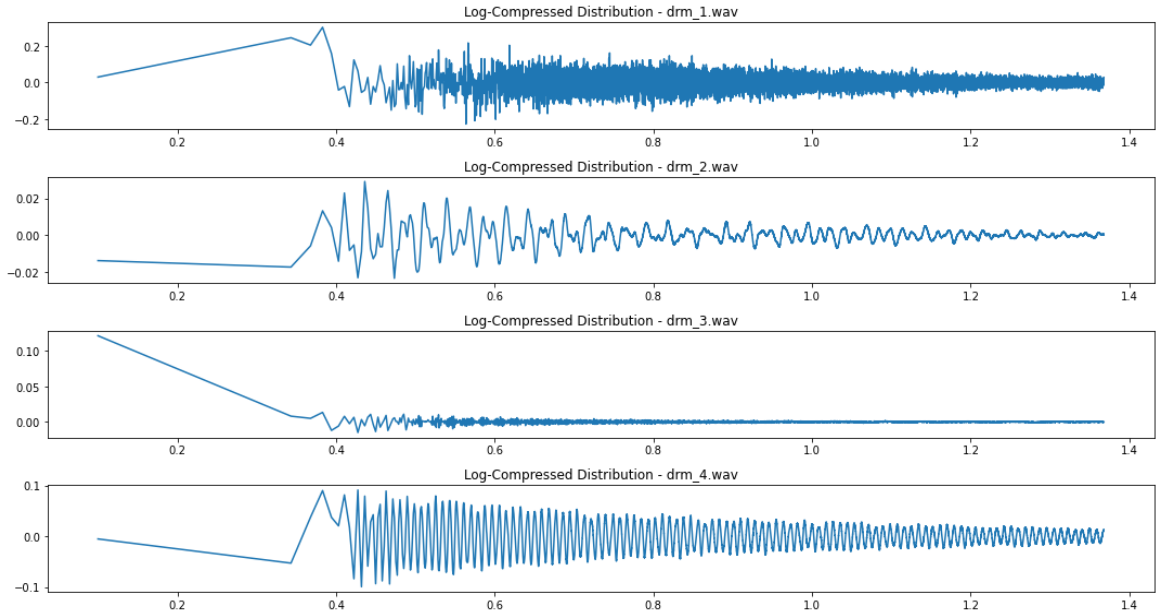
epochs, I tested out trying different activation functions. I ran tests with ReLU, Tanh, and Sigmoid activation functions using batch sizes of 32, 32, and 64 for each, with ReLU being the winner (app. 3). After that, I tried adjusting the batch size and added in the other two types of drums while keeping the total size of the dataset 1,000 samples. I tested batch sizes of 3, 8, 16, 20, 26, 32, 36, 40, 48, and 64 with the ReLU activation function. The batch size of 64 was the winner (app. 4). My next experiment was trying out some different kernel sizes for the convolution layers. I kept it the same for each and because I didn't know how they would change with batch size, I tested the entire 4,000 sample dataset with kernel sizes of 3, 5, and 7 and batch sizes of 32, 64, and 128. The three viable models were (3, 64), (3, 128), and (5, 32) (app. 5). For the final test, I retrained each of the three viable model settings with 20 epochs instead of 10. The only viable model come the end was the kernel of 5, batch size of 32 (app. 6).

APPENDIX 1

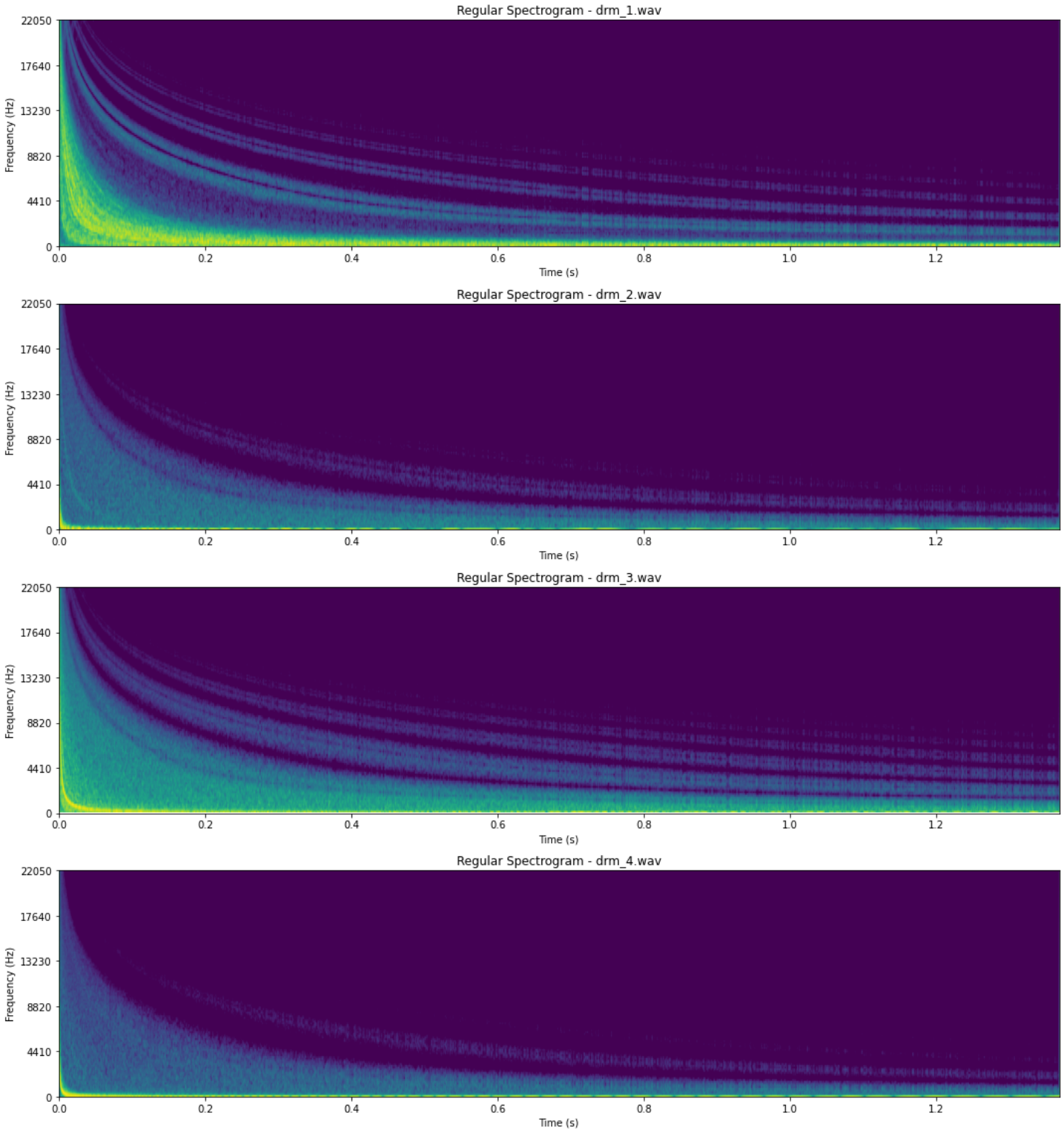
These images show some of the steps of cleaning and preprocessing the audio. The original trimmed length was 24 seconds and the naming scheme was originally that all samples would be named drm_#. The current version of the cleaning is trimmed to 2 seconds and renamed cym_#, kik_#, snr_#, and tom_# depending on the sample type.



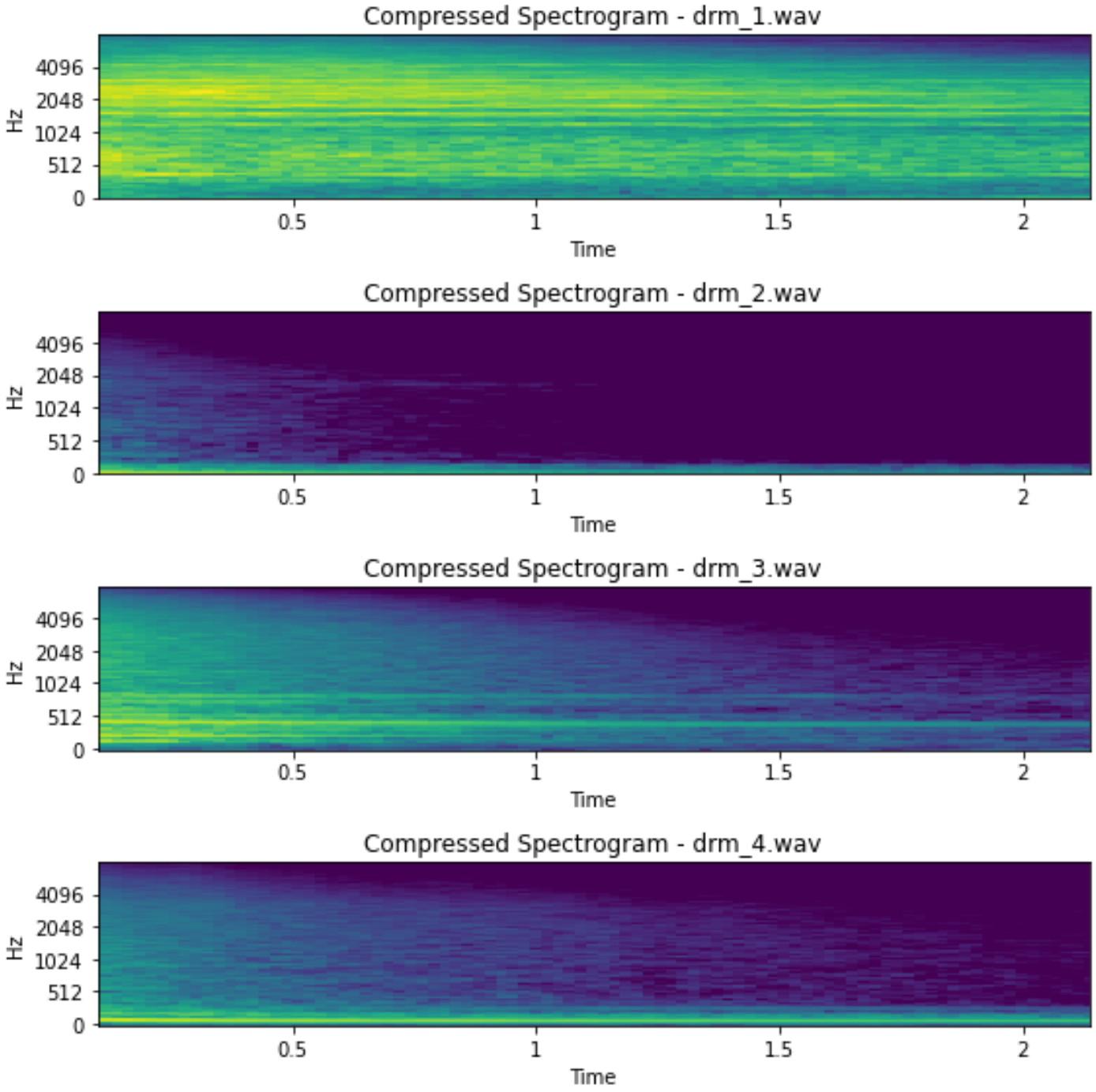
Cleaned audio



Log-compression in time



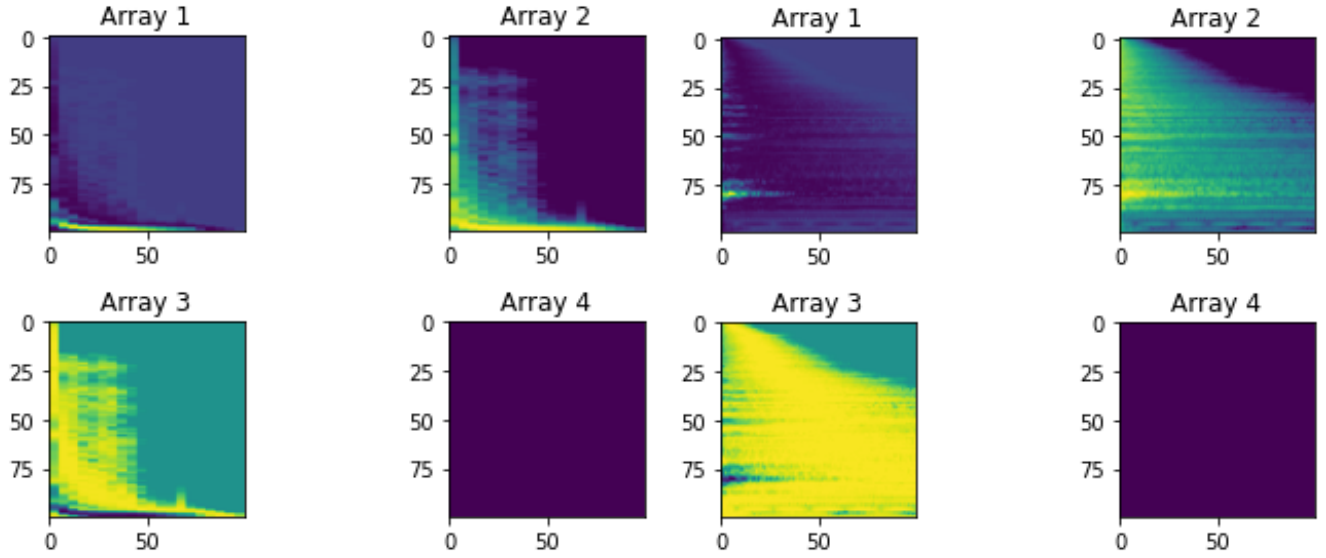
Mel Spectrogram



Log-compression in frequency

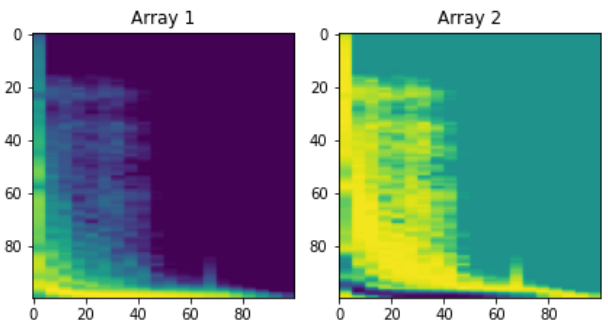
APPENDIX 2

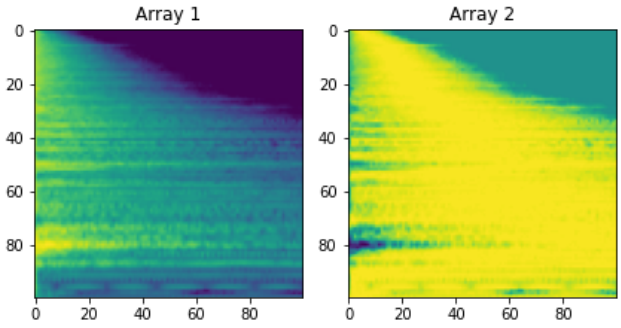
The data in the arrays are comprised of three dimensions (4, 100, 100). The two 100-length components represent the frequency and time bins, and all but one channel represents intensity bins. I don't know what the fourth channel represents, but it seems to be blank. Additionally, the highest intensity bin seems to show only the pitch of each drum hit. This would have been problematic in telling apart most of the drums because, without the extra noise and timbre, the drums looked too similar in shape. This was discovered while testing with only cymbals and kicks, which sound the most dissimilar of all four samples and therefore should look that way.



Kick sample (all channels ↑, two channels ↓)

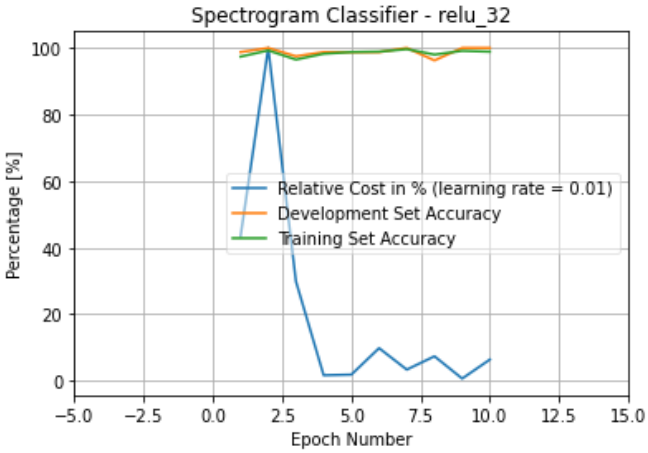
Cymbal sample (all channels ↑, two channels ↓)

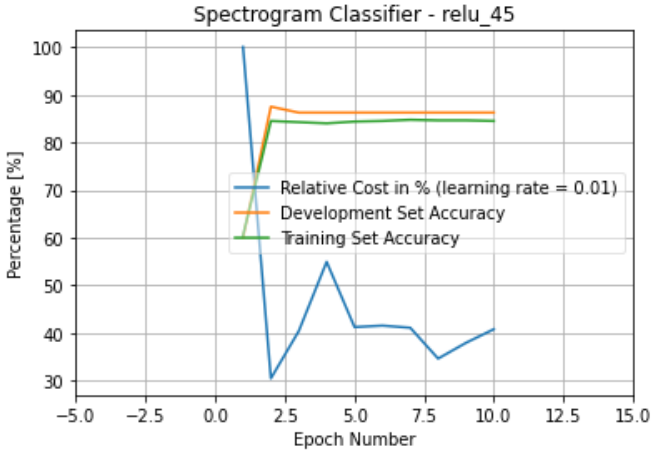


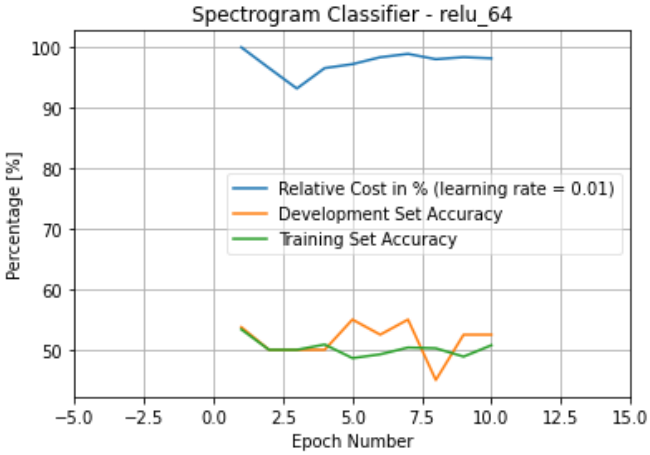


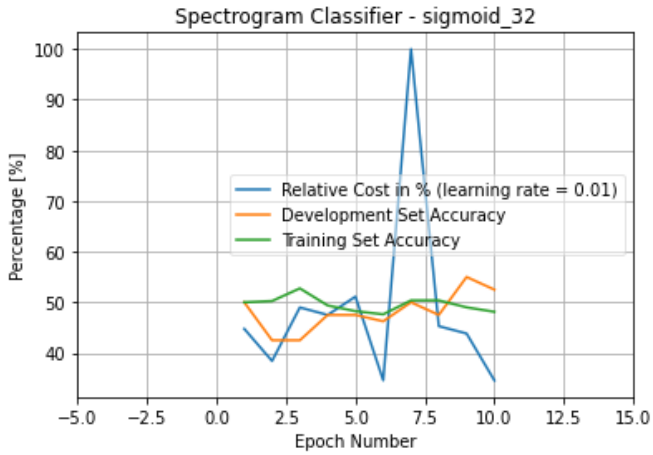
APPENDIX 3

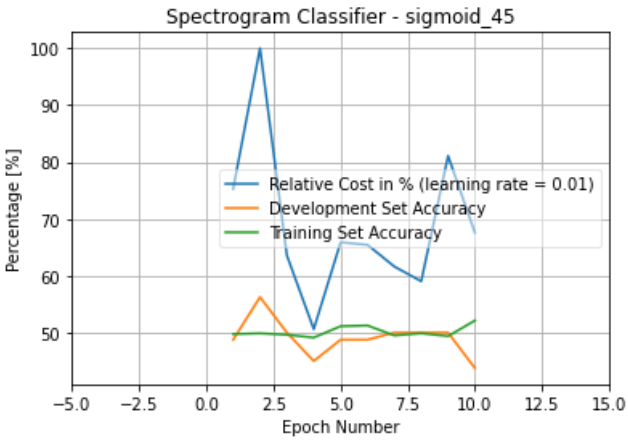
The titles show the activation function and batch size

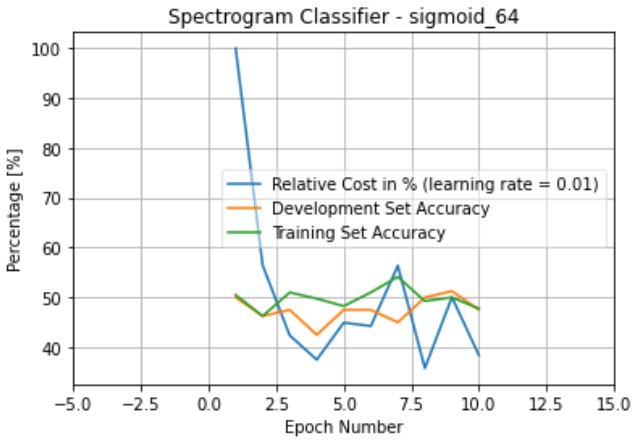


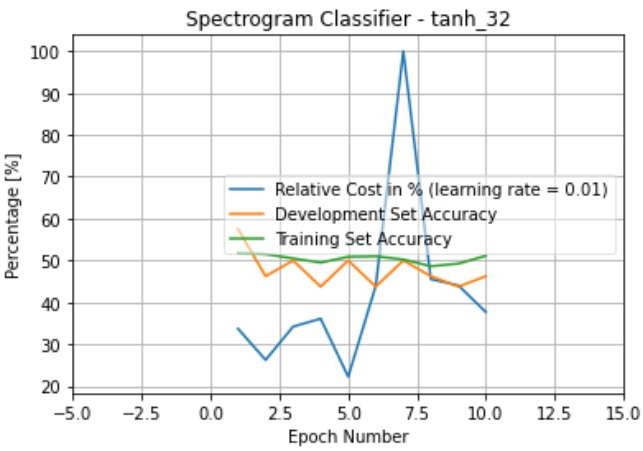


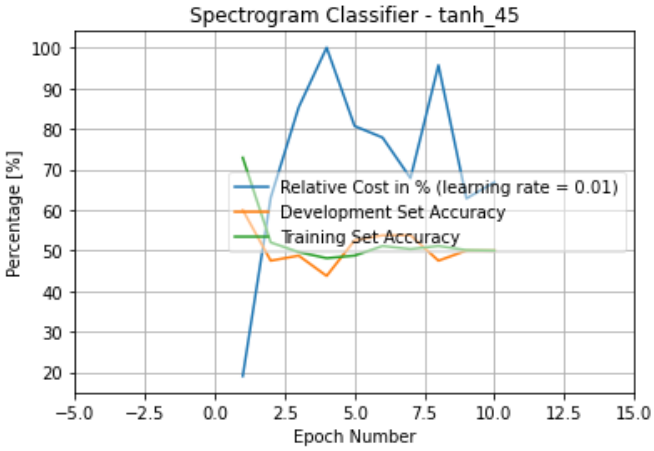


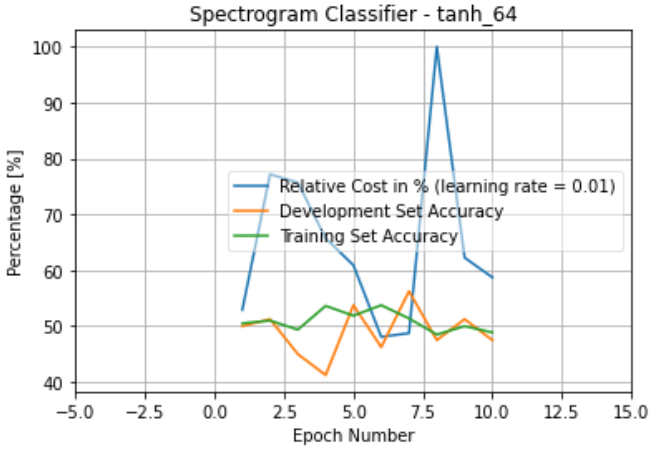






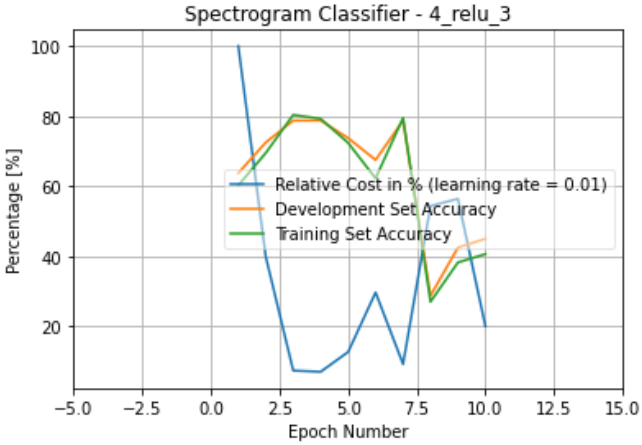

Spectrogram Classifier - tanh_32
Relative Cost in % (learning rate = 0.01)
Development Set Accuracy
Training Set Accuracy
Percentage [%]
Epoch Number

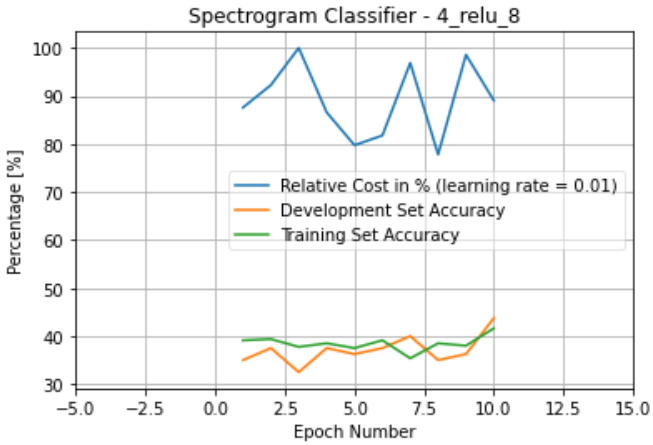

Spectrogram Classifier - tanh_45
Relative Cost in % (learning rate = 0.01)
Development Set Accuracy
Training Set Accuracy
Percentage [%]
Epoch Number

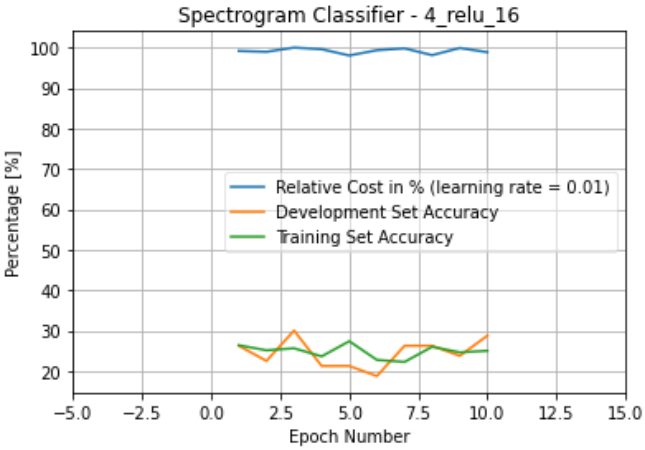

Spectrogram Classifier - tanh_64
Relative Cost in % (learning rate = 0.01)
Development Set Accuracy
Training Set Accuracy
Percentage [%]
Epoch Number

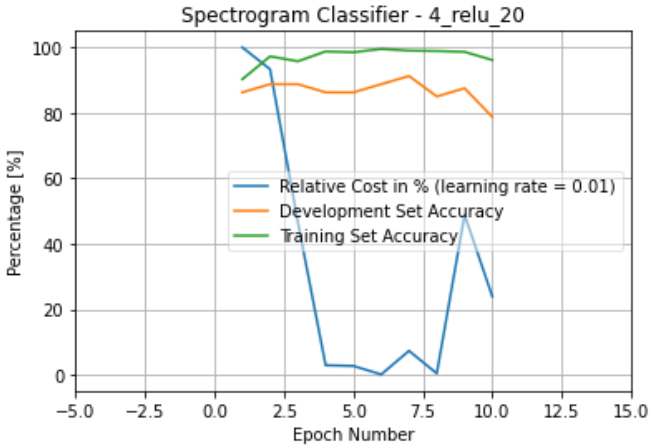
APPENDIX 4

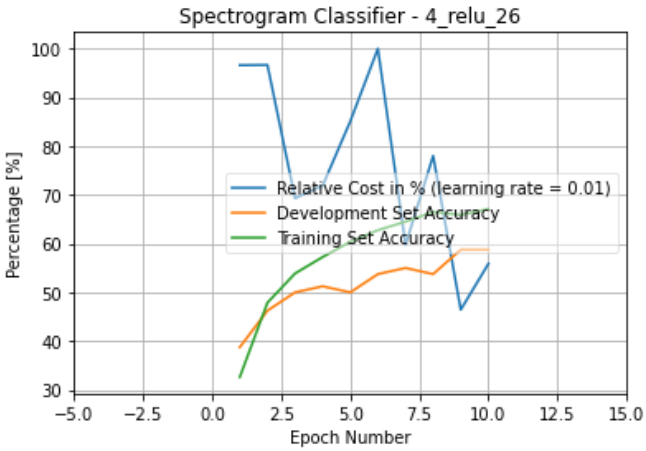
The titles show the batch size at the end

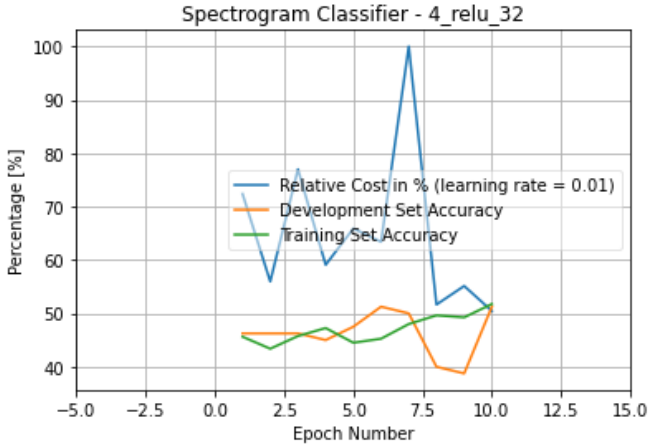




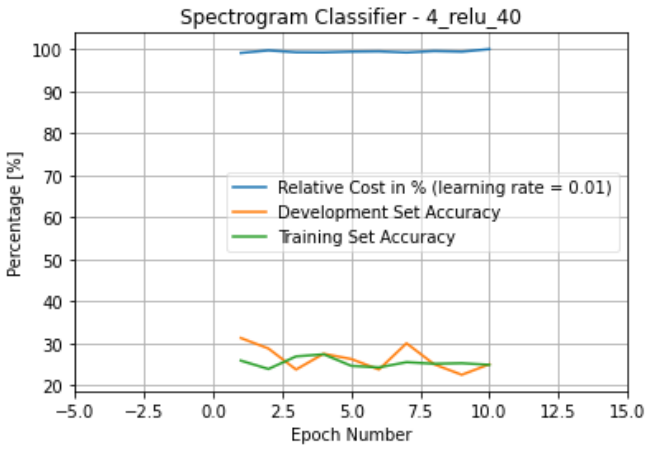


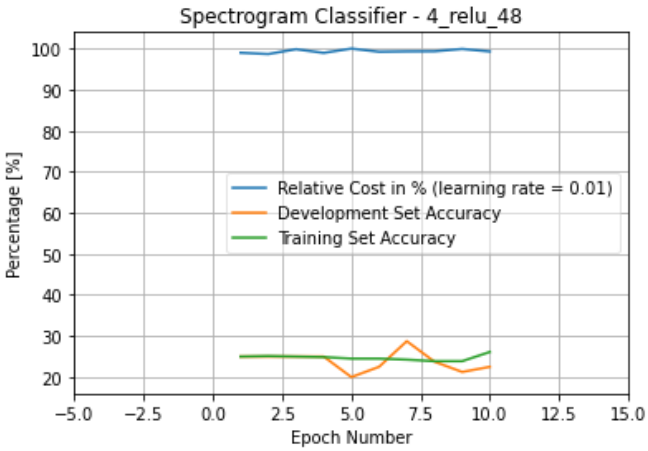


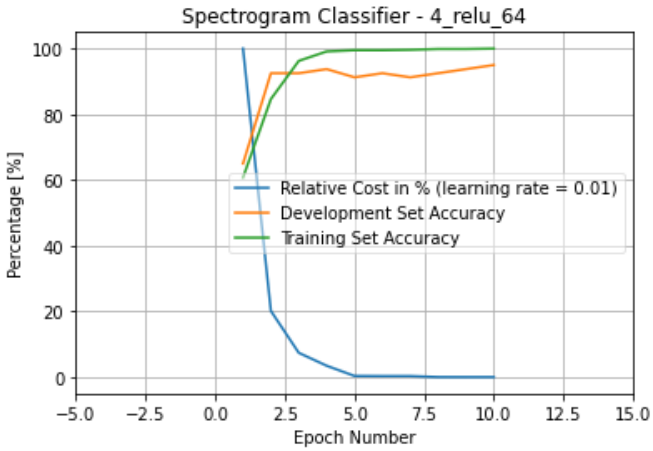





Spectrogram Classifier - 4_relu_36
Percentage [%]
Epoch Number
Relative Cost in % (learning rate = 0.01)
Development Set Accuracy
Training Set Accuracy

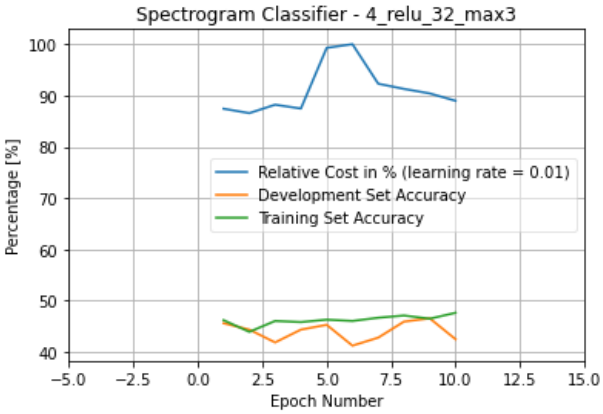

Spectrogram Classifier - 4_relu_40
Percentage [%]
Epoch Number
Relative Cost in % (learning rate = 0.01)
Development Set Accuracy
Training Set Accuracy

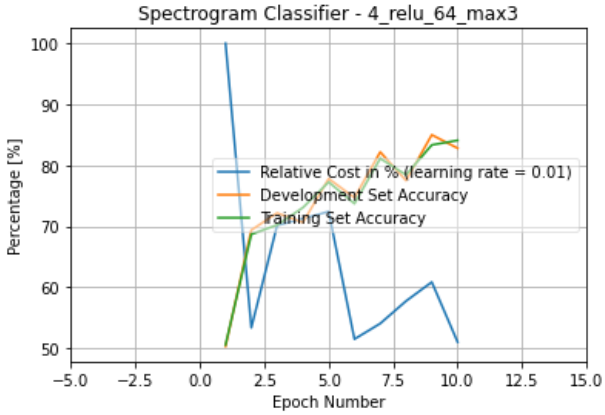

Spectrogram Classifier - 4_relu_48
Percentage [%]
Epoch Number
Relative Cost in % (learning rate = 0.01)
Development Set Accuracy
Training Set Accuracy

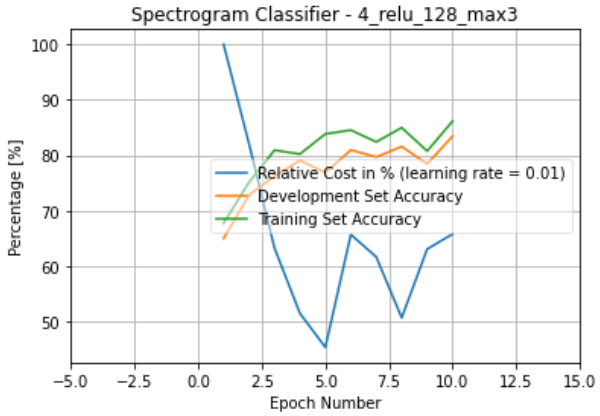

Spectrogram Classifier - 4_relu_64
Percentage [%]
Epoch Number
Relative Cost in % (learning rate = 0.01)
Development Set Accuracy
Training Set Accuracy

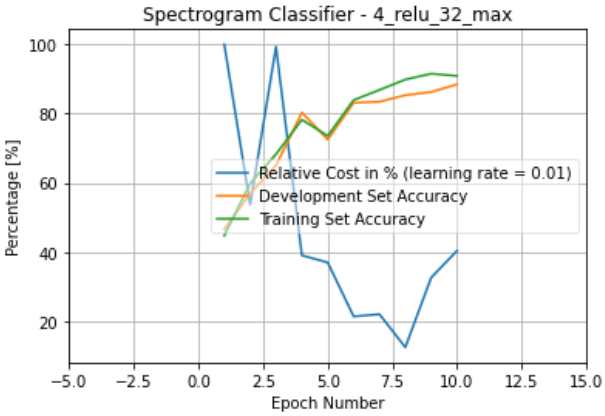
APPENDIX 5

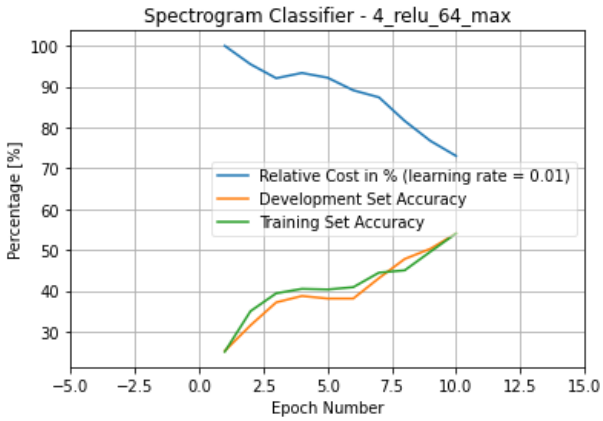
The titles show the kernel at the end, except the kernel of 5, which doesn't have a number

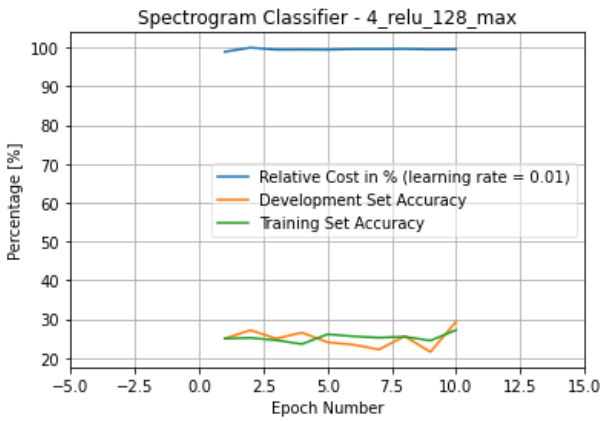


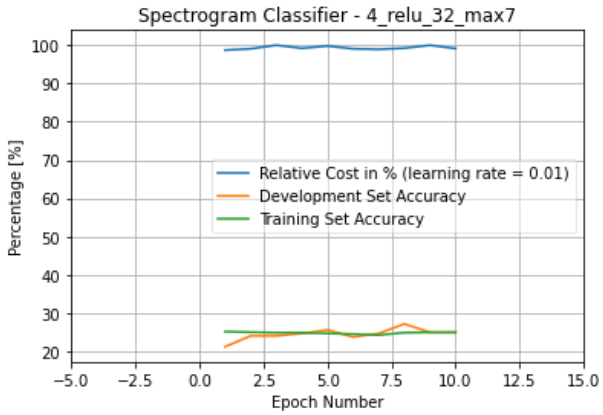


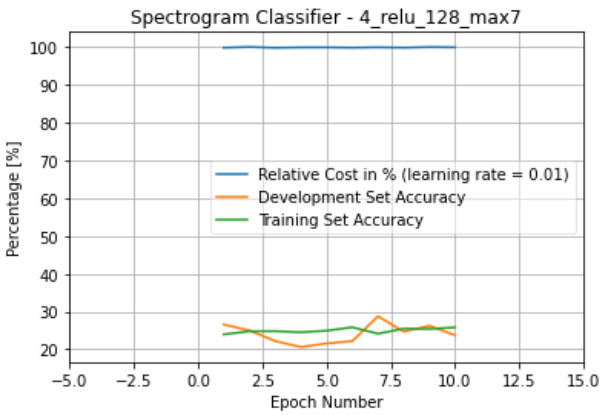









Spectrogram Classifier - 4_relu_32_max7
Relative Cost in % (learning rate = 0.01)
Development Set Accuracy
Training Set Accuracy
Percentage [%]
Epoch Number


Spectrogram Classifier - 4_relu_128_max7
Relative Cost in % (learning rate = 0.01)
Development Set Accuracy
Training Set Accuracy
Percentage [%]
Epoch Number

APPENDIX 6

The titles are formatted 4_relu_(batch size)_max(kernel size)_20