Neural Networks for Song Genre Prediction

Nicholas Livingstone, Calvin Stahoviak
University of New Mexico

May 6th, 2024

1 Introduction

Neural Networks currently dominate the field of machine learning. They offer effective solutions to complex problems in a wide variety of domains. In a previous project, we utilized logistic regression to classify the genre of songs. In this report, we continue work on the song genre problem by using neural networks for the classification model. We implement three different types of neural network architectures and discuss the results of each: a multilayer perceptron, a convolutional neural network, and transfer learning.

2 Data

The data used in this report is comprised of 900 audio samples in the ".au" file format. Each file is a 30 second clip of a song with accompanying knowledge of the song's genre. There are 10 genres to classify: blues, classical, country, disco, hiphop, jazz, metal, pop, reggae, and rock. Each genre has 90 samples each, so the dataset has perfect class balance.

3 MLP with Extracted Audio Features

3.1 MLP Architecture

In this model, a traditional multilayer perceptron network was developed using features extracted from the audio waveform. The extracted features are statistical descriptions of a variety of different audio features at different window sizes e.g. *Mean of the Spectral Centroid w/ Window Size of 2048*. We use the full set of features and window sizes as described in our previous paper (Livingstone and Stahoviak 2024) and encourage the reader to reference the section on feature extraction (see Livingstone and Stahoviak 2024, sec. 3) for further details.

The model is comprised of one hidden layer with a width of the mean of the input and output size, an input layer matching the feature size, and an output layer. Between each layer is a ReLu activation followed by a dropout to prevent over fitting.

3.2 MLP Results

This model had surprisingly good accuracy with little tuning, achieving up to 83% accuracy on an unknown dataset. This suggests that the features extracted represent the data well. We then considered modifying batch size, number of epochs, and learning rate to see if it would result in any improvement. We tested these changes using 5-fold Stratified Shuffling with a validation size of 0.2. We found that increasing the learning rate larger than the recommended 0.0001 only decreased the accuracy of the model. Modifying the batch size did not have drastic impacts in accuracy, but did have impacts on training time. This implies that the error landscape is not incredibly complex, and we should utilize large batch sizes when the option is available. Lastly, a sufficiently large epoch size did result in some very slight over fitting i.e. an increase in validation loss while maintaining low training loss. This is to be expected, as any sufficiently complex model is likely to overfit given enough training iterations. But the magnitude of the loss itself was insignificant and thus did not have drastic impacts to final validation accuracy. Overall, any modification to the model or fine tuning did not result in much greater gains beyond $\sim$ 80% accuracy. We hypothesized that a traditional MLP combined with summarized features did not allow for a model complex enough to perfectly label the instances and that the subsequent models discussed here would give better performance.

4 Spectrogram CNN

Convolution Neural Networks are designed specifically to handle images and large input formats. A typical fully-connected neural network's parameter space is unreasonable to operate on for standard sized images. In this section, we will discuss

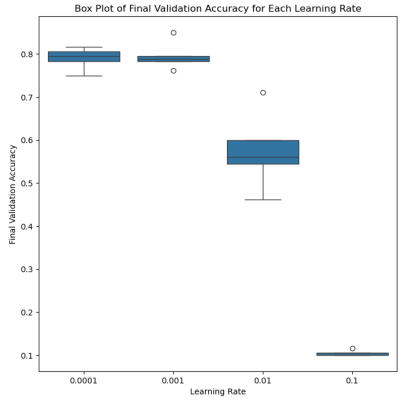

Figure 1: A box plot of final validation accuracy against learning rate from MLP.

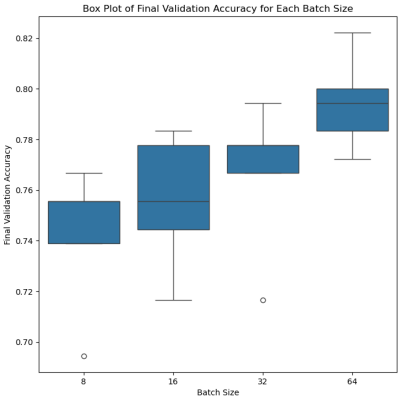

Figure 2: A box plot of final validation accuracy against batch size from MLP.

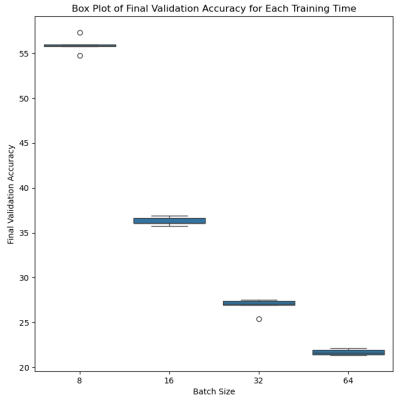

Figure 3: A box plot of training time against batch size from MLP.

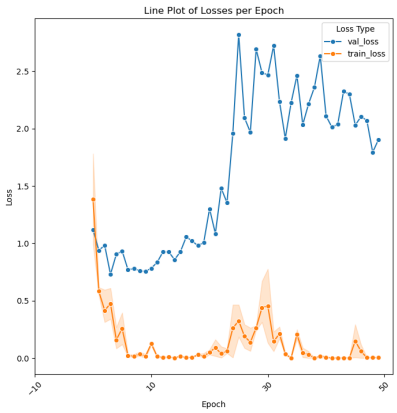

Figure 4: Training loss and validation loss over epochs for MLP.

the creation of spectrograms for songs to serve as the input in a convolutional neural network.

4.1 Spectrogram Creation

Audio files are commonly visualized through spectrograms, the most common being the amplitude-frequency spectrogram. For our given data we converted each audio file into several types of spectrograms. All spectrograms are generated with a window size and fast Fourier transform sample size of 1024, hop size of 512, and sample rate of 22.05kHz which was retrieved from the audio clips. All audio clips were cropped to the size of the smallest audio clip. This, in addition to the constants above, allows for all spectrograms to be of constant width. The extraction process and information gained from each spectrogram is detailed below, with the y-axis units in parenthesis:

- Decibel-Scaled Spectrogram (Hz)
- Power Decibel-Scaled Spectrogram (Hz)
- Phase-Shift Spectrogram (Hz)
- Chromagram (Hz)
- Constant-Q Chromagram (Pitch Class)
- Mel-Frequency Cepstral Coefficients
- Mel-Scaled Spectrogram (Hz)
- Constant-Q Spectrogram (Hz)
- Tempogram (BPM)
- Tonal Centroid Features (BPM)

The spectrograms that were generated by first calculating the Short-time Fourier transform are *Decibel-Scaled Spectrogram*, *Power Decibel-Scaled*

Spectrogram, *Phase-Shift Spectrogram*, and *Chromagram*. The short-time Fourier transform extracts the amplitude and phase-shift values for each frequency component, which is then uniquely scaled and converted by each spectrogram generation method. The remaining spectrograms are generated with other prepossessing steps such as finding the constant-Q transform or estimating BPM.

After spectrograms are generated, they are normalized and converted into single-channel grey-scale images for ease of use in the CNN. All of these items together have the potential to fully represent the differences and nuances of each genre.

4.2 CNN Architecture

A single-channel CNN can only train on a single type of spectrogram, limiting the complexity and available information. However, it still performs adequately in classifying. The computational path from generated spectrograms to a classification goes as follows.

For prepossessing the data, the training dataset is loaded into memory after an initial transformation. This transformation resizes images, converts them to single-channel grey-scale, and loads them as tensors. From this data, the standard deviation and mean is calculated. Next the data set is loaded into memory again with the previous transformation plus normalization using the mean and standard deviation. Notice that the test set was not used to calculate the mean and standard deviation, since our model is only trained on the training set, we don't want any outside-information entering that system.

Next, our CNN is initialized with a robust set of parameters. Our CNN is also setup to dynamically configure the amount of convolution layers based on the input size of the resized dimensions of the images, and the kernel parameters, and the desired size the pooling layers will converge to. For example, with the sample parameters listen in Table 1, there will be a total of 4 convolution, ReLU and pooling layers until it reaches it's "Final Convolution Dimension" threshold. After convolution, fully connected linear layer is built, which connects to another linear layer half of it's size, then to an output layer which classifies.

$$lr = lr_0 \times \gamma^{\lfloor \frac{n}{m} \rfloor} \tag{1}$$

During the fitting process learning rate scheduling is utilized to find a precise local minimum. The formula for current learning rate given an initial learning rate lr_0, decay factor γ, epoch n, and step size m, can be seen in Equation 1.

Early stopping is also implemented. If the validation loss does not improve after "Patience" amount of epochs then the model stops and reverts to its best weights.

Table 1: Sample Model Parameters

Parameter	**Value**
Data Parameters	
Batch Size	128
Resize Dimensions	(512, 512)
Spectrogram Type	Amplitude
Fit Parameters	
Optimizer Function	Adam
Number of Epochs	2
Learning Rate	0.001
Learning Rate Decay Factor	0.1
Learning Rate Decay Step	10
Patience	5
Minimum Delta	0.001
CNN Parameters	
Input Channels	1
Kernel Size	3
Stride	1
Padding	1
Final Convolution Dimension	(4, 4)

4.3 CNN Results

Our CNN performs adequately. Although our accuracy never reached a value comparable to MLP or transfer learning, it at least managed to get past random chance. Seen in the bottom half of Figure 8, the validation accuracy plateaus while the training loss continues to decrease. This indicates the the model may be over-fitting. However, a measure is taken to stop early if over fitting occurs, and that threshold is not being broken because the validation loss continues to decrease but at a very slow rate. Seen in the top half of Figure 8, our model only reaches a maximum accuracy of about 50% for the validation set.

5 VGGish Transfer Learning

For this model, we utilized transfer learning, a technique which involves using an existing pre-trained model as the foundation of a new model trained on different data. For the pre-trained model, we selected VGGish.

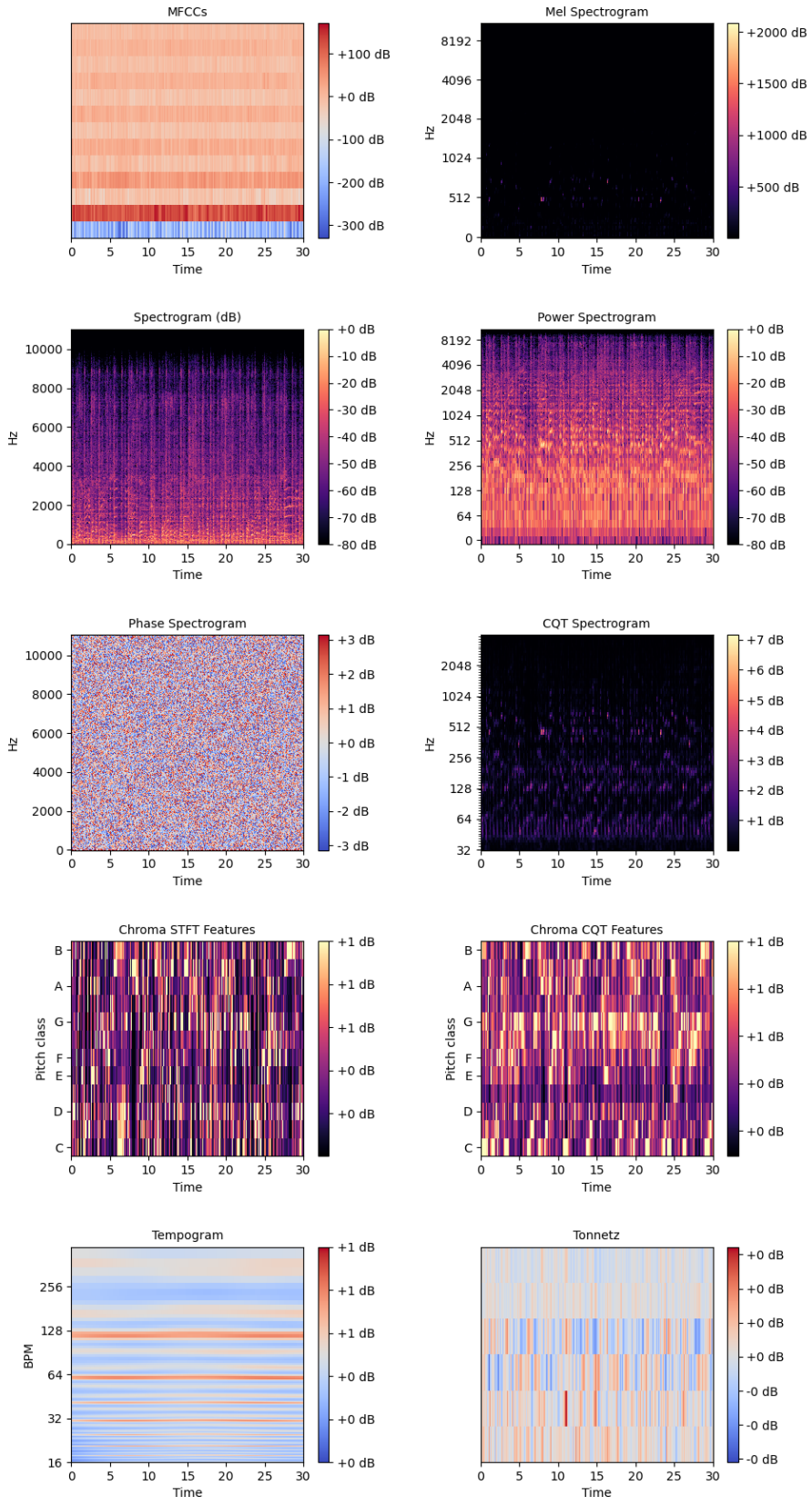

Figure 5: Spectrogram features of the audio sample "blues.00000.au".

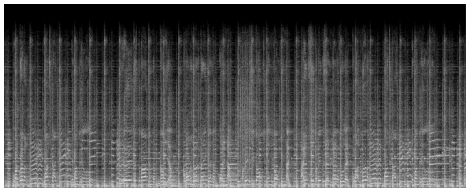

Figure 6: Amplitude spectrogram of the audio sample "blues.00000.au" before it's handed to the CNN. Has a size of 1290 x 513 pixels.

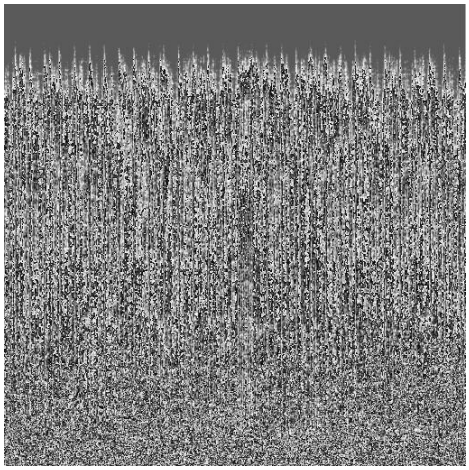

Figure 7: Amplitude spectrogram of the audio sample "blues.00000.au" after it has gone through resizing, grey-scaling, and normalization. This is the kind of image that the CNN will train on. Has a size of 512 x 512 pixels.

5.1 Source Model Background

VGGish is a CNN developed by google (Diwakar and Gupta 2023) which was based on the VGG image classification model (Simonyan and Zisserman 2014). VGGish, however, was used to classify audio from YouTube videos. The VGGish model is roughly of comprised two components: an embedding component and a classification component. The embedding component takes the input sample and creates a latent representation, the classification component then takes this representation and determines the labels. The model was trained on 30k labels, which covered a vast set of audio identifiers like "trumpet", "lecturer", and "motorcycle". A single model input is a 975ms audio sample, which is passed through a processing pipeline to generate an array of 96x64 defined by the VGGish model, then given to this embedding input layer. The model outputs likelihoods of every class. This is useful in the case where a sample will have multiple sounds occurring at once.

This model was chosen for a few reasons. First,

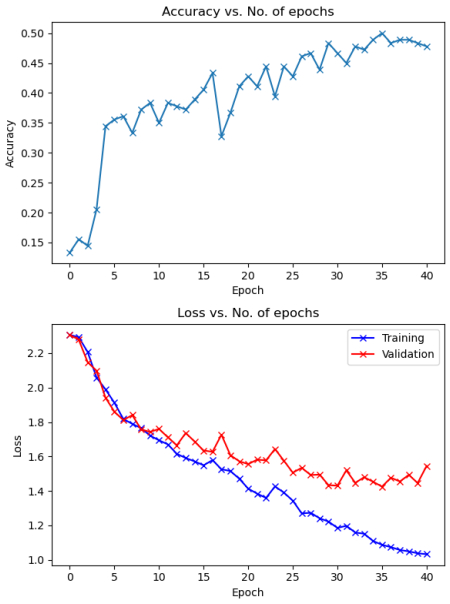

Figure 8: Performance from our CNN. On the top, accuracy on validation set of epochs. On the bottom, training and validation loss over epochs.

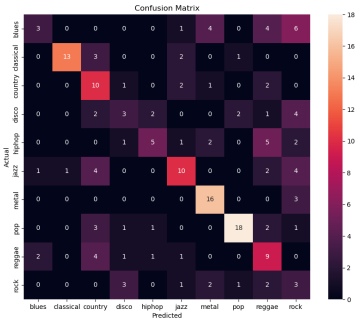

Figure 9: Performance from our CNN. A confusion matrix on the validation set.

the model was trained on audio data, this would likely give better performance than using a model from a different domain, like image detection. The VGGish model was trained on 20 billion samples, so it will hopefully be robust. Lastly, there is the abstract hypothesis that if a model can identify different types of instruments, it likely can associate that with different genres e.g. a country song is much more likely to have banjo than a disco song.

5.2 Transfer Architecture

There were a few considerations necessary for designing the transfer architecture. The first was the problem of input and output formats. The

VGGish model expects a 975ms audio sample and gives a multi-label output, however, we're attempting to give a *single* label to a 30s song instance. The solution was pseudo ensemble learning. Each sample was split into 975ms windows, giving 31 examples per song. The model was trained as if each example was a separate instance. For a final prediction, we use the majority vote of each example of a song. For example, if 25 out of the 31 windows were labeled as Rock, then the song was predicted to be rock. We did consider averaging every probability, then choosing the majority, but it was believed that doing so might result in votes being skewed by outliers. Consider a rock song which had a small portion of windows which the model had a very 99% probability of labeling it as blues, but every other window had only a 60% chance of being rock, while still being the highest label, it would likely mislabel it due to a loud minority.

The next design decision was to determine which portion(s) of the source model we needed to keep. Since the target space of the VGGish model was much larger than our own, i.e. only 10 genres vs 30k labels, We kept the embedding component and replaced the classification component with our own set of new layers. The extracted VGGish layers were frozen and the new model was trained with the dataset.

5.3 Transfer Model Results

The transfer model produced the best accuracy out of the three models here. Achieving 89% accuracy on unseen data. The biggest impact to training was batch size. Due to how we were generating the data, a batch size of β actually was $\beta \times 31$ samples being passed into the model, since each sample was split into 31 frames. Initially, it was considered to preprocess the data and randomly shuffle the frames of the entire dataset together. However, we decide to forgo this option for two reasons. First, implementation would require a significant overhaul of the current model. Secondly, we had the belief that doing so would cause strange training behavior, i.e. the model wouldn't be able to learn consistently enough considering data the would become quite varied. Instead, all of a sample's frames would be in one batch. This had a noticeable impact on training. As the batch size decreased, the training loss became very sporadic, see Figure 10. We can consider that for a small batch size, although they are many "different" samples to the model, they are ultimately the same. i.e. a single blues song will still pass 31 instances of blues to the model. So if the model then sees a different genre, it will only be used to "seeing" blues. This then suggests that our previous hypothesis of grouping samples might be the wrong approach. A large batch size does smooth out the training loss, but at the cost of plateauing.

For analysis, we will consider a trained instance of the transfer model presented as a confusion matrix Figure 11. This model achieved 86% accuracy on the benchmark unseen dataset and had a final validation accuracy of 90%. The model was trained using a 80/20 train/validation stratified split. We can note a few things. First that blues, classical, and metal samples are all correctly classified. This follows the notion that they are arguably the most distinctive genres out of the set. We can also note a few reasonable mistakes such as confusing disco or reggae for hip hop and country for blues. Additionally, rock was one of the most misclassified labels. One can argue that even for humans, the line between genres can be blurry, especially for the ones it assumed incorrectly: blues, country, and metal. Potentially, we can say this model has the personality of a music snob; a snob that might argue "this song isn't rock, it has too much twangy guitar and is therefore modern country". Nonetheless, it's clear that this model exhibits a deep understanding of the data with further tuning would likely exhibit exemplary performance.

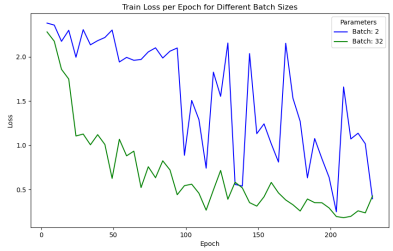

Figure 10: Training loss of batch sizes [2,32] on the transfer learning model.

6 Conclusion

This report examined three neural network architectures for song genre classification: a multilayer perceptron (MLP) using extracted audio features, a spectrogram-based convolutional neural network (CNN), and a VGGish transfer learning model. The MLP achieved moderate accuracy with extracted features, but performance plateaued due to limited complexity. The spectrogram CNN, though utilizing visual audio representations and convolutional layers, suffered from overfitting and high data dimensionality. Further research will

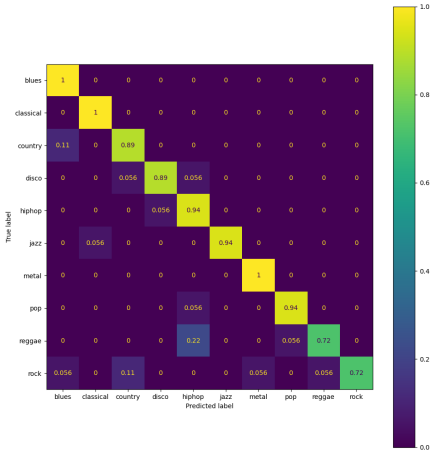

Figure 11: Confusion matrix of one trained instance of the transfer model.

be required to produce effective results. The VGGish transfer learning model was the most successful, achieving up to 89% accuracy. By leveraging pre-trained embeddings, it captured comprehensive audio features with minimal additional training. In summary, this work underscores the significance of architecture choice and pre-trained models in genre classification. While traditional models provide a solid baseline, transfer learning holds promise for improving predictive accuracy.

References

Diwakar, Mandar and Brijendra Gupta (2023). "The robust feature extraction of audio signal by using VGGish model." In.

Livingstone, Nick and Calvin Stahoviak (2024). "Using Logistic Regression to Predict the Genre of Songs". In.

Simonyan, Karen and Andrew Zisserman (2014). "Very deep convolutional networks for large-scale image recognition." In.