DNN Music Genre Classification
APM 598 Project

Mariam Joseph, Aravind Siddharth, Paritosh Hattyangdi, and Rohan Kapoor

May 1, 2024

Abstract

Sound, melody, and emotions; are all aspects of music that allow us to discover what genres or types of music we relate to or enjoy listening to. A music genre can be defined as a category that encompasses similar musical styles. However, sometimes, the task of music genre classification can be difficult as noted by Kumar et al. (2018), "many times the boundaries are not clearly defined and genres are overlapping" [1]. In this project, we utilize the GTZAN dataset which contains 1000 audio files split into 10 different music genres. Our goal for this project is to build a Convolution Neural Network (CNN) to classify the audio files in this dataset. We build a baseline linear model, a simple CNN, and a complex CNN for comparison purposes, the details of these will be defined in a later section. Although many adjustments and future works are necessary to verify the validity of our results we were able to demonstrate that what we note as our complex CNN was able to achieve a higher test accuracy when compared to the other models.

1 Introduction

In recent years there has been an increase in the utilization of machine learning techniques in a variety of real-world applications. Within this branch of machine learning there exists Deep Neural Networks, known as DNNs. As described by Joo et al. (2023), "A DNN consists of layers, including nodes and edges, that contain mathematical relationships" [2]. DNNs are widely used in tasks that involve classification (Halnaut et al. 2023) [3] through the use of building a model, training that model on a given dataset, and incorporating a test or validation set to discover the results. This project in particular uses a Convolution Neural Network (CNN) for image classification. One reason we chose to utilize a CNN model for this project is that CNNs are able to analyze image features with more detail when compared to other neural networks (Hung 2023) [4]; therefore, we decided that using a CNN would be an exciting and beneficial first dive into machine learning and image classification.

Now, our dataset consists of .wav audio files. At first, this seemed to pose a challenge for our team as we were unsure of how to use a neural network to classify audio files. We found that it is common practice to classify audio files using spectrograms, especially when using a CNN model (Doshi 2021) [5]. We noticed some researchers using the GTZAN dataset followed a similar structure of turning the audio files into spectrograms including Chowdry (2021) [6], who tried to use machine learning (ML) on the dataset in connection with music recommendation systems and Dutt (2022) [7], who worked on spectrogram and feature extraction of the audio files in the dataset in preparation for a CNN model. In the work of Doshi (2021) [8], he provided a very clear depiction of what audio classification can look like, seen in Figure 1 below, before beginning his work on the classification of the Urban Sound 8K dataset. After seeing these techniques used by the community in regards to audio classification it was clear that our journey would now begin.

In this paper we include a methods section which depicts the details of our feature extraction and spectrogram creation. We also discuss our linear model and various CNN models. It is important to note the distinction between what we call simple CNN and complex CNN. In this case, simple CNN refers to our CNN model with only 15,610 parameters; whereas complex CNN refers to our CNN model with 59,114 parameters. Next, we include a results section which will contain numerous figures and information depicting the results and accuracy comparisons of our models from one particular run. There will be a discussion section involving the analysis and review of our results. Lastly, a limitations and future works section discussing the various ways our models and results can be improved upon or explored.

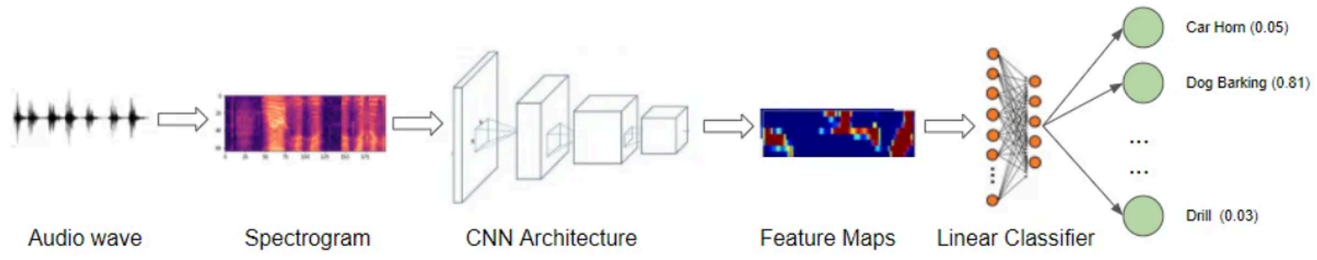


Figure 1: “Audio Classification Application” title and image by Doshi (2021) [8]

We provide a roadmap of our project in Figure 2 below. This describes the general structure of our project and the methods we utilized to compare the accuracies between a baseline linear model and our CNN model.

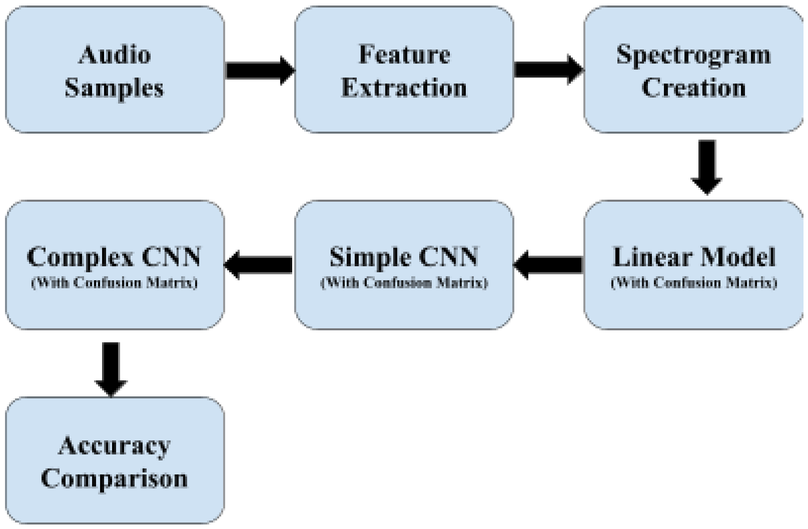


Figure 2: Project Roadmap

2 Methods

In this project, we utilize the GTZAN dataset, downloaded from HuggingFace [9]. This dataset contains 1,000 audio files split into 10 different genres of music: blues, classical, country, disco, hip-hop, jazz, metal, pop, reggae, and rock. It should be noted that each of these audio files is 30 seconds long. We simplified the dataset used in this project to only include the 100 .wav audio files per genre. However, during data processing, we encountered an error with one jazz audio file which will be discussed further in the next section. We utilized techniques and suggestions from our course and our professor, Dr. Sebastien Motsch, to create the different model types. Some of the main packages we used for this project include the librosa python package for audio and music analysis [10], the Scikit-learn library for machine learning which provides us tools for data preprocessing and other instruments for working with models [11][12], and Pytorch, a tensor library for deep learning [13][14].

2.1 Feature Extraction and Spectrogram Creation

To begin we needed to first extract spectrograms from the given audio files. We created a function that is initialized with an FFT window length of 2048, 512 samples between frames, and 128 mel bands. These all seemed to be the standard input as demonstrated by the librosa Mel spectrogram documentation [15]. Within this function we load the audio data, collect the spectrograms, and convert them into a dB scale all using librosa built-in functionalities. Next, we try to format the spectrograms so that they all have the same general target shape of 128 by 1291. However, as we will find out later this size was too large for our models and we eventually reshaped the inputs within each model as described in the below sections. After this, we go through and extract spectrograms for every file in our genres dataset and convert our X (spectrograms) and y (genre labels) lists into NumPy arrays. Lastly, we use the train_test_split function from Scikit-learn [16][12] in order to create a stratified train-test split of 80% for training and 20% for testing. We used the stratify feature to ensure there was an even split amongst the genres. Now, due to the error with one of the jazz files our model uses 799 files for training (79 from jazz and 80 from the rest of the genres) and 200 files for testing (20 from each genre). The figures below depict examples of two spectrograms we are now working with. Figure 3 is from the train set under the jazz genre and Figure 4 is from the test set under the blues genre. It is interesting to see the visual differences between the spectrograms, as there are notable distinctions between the two.

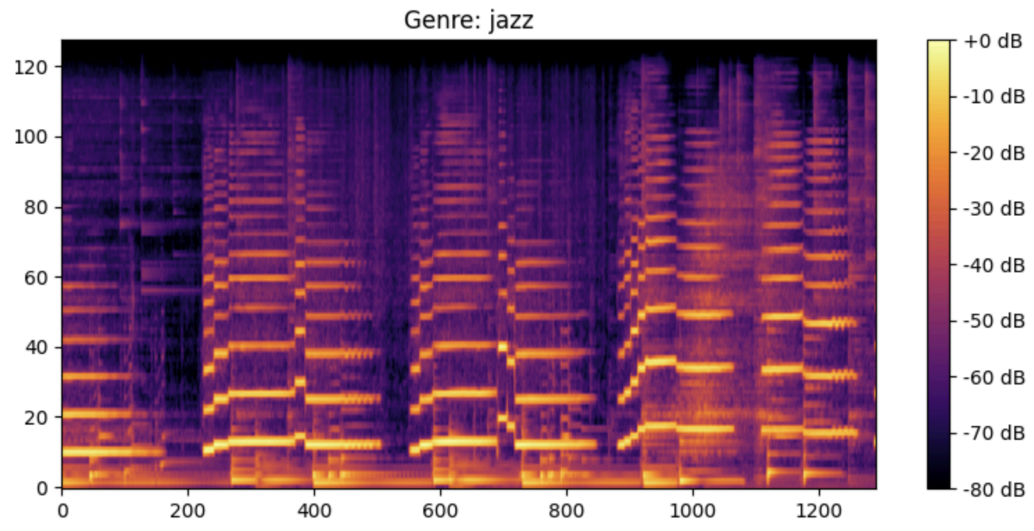


Figure 3: Spectrogram example from the training set

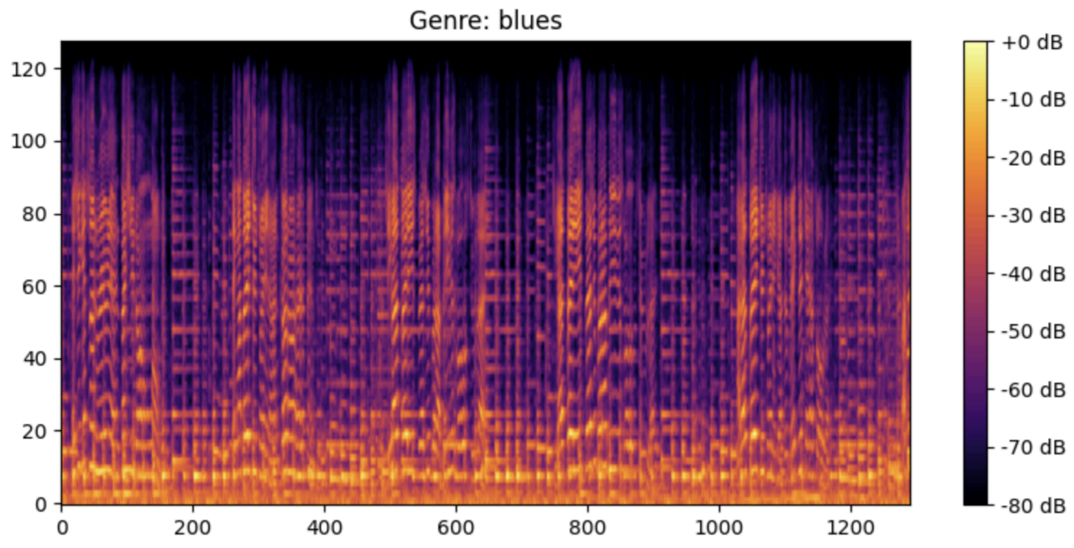


Figure 4: Spectrogram example from the test set

2.2 Linear Model

We define a Linear Model using Pytorch in order to formulate a baseline model to which we will compare our CNN model with at the end of this report. The linear model takes in an input size and the number of classes which will represent our output size. In this case, we have 10 classes to represent each of the genres provided by the dataset. We also include a forward pass in order to reshape or flatten the input before passing it through the linear layer. We define a *train_model* function which keeps track of train and test losses/accuracies. During training, we run for 150 epochs with a batch size of 128 giving us a total of 7 batches per epoch. We utilize the *CrossEntropyLoss* function alongside the *Adam* optimizer with a learning rate of 0.0001. These were chosen as we felt they were standard amongst the tasks we were trying to complete. We also learned about these from our homework assignments and course lectures. Evaluation on the test set is done within the training function in order to track test loss and accuracy as the model is running. This is done with a batch size of 64 giving us a total of 4 batches per epoch. Prior to feeding the data into the model we first conduct a few transformations, in what we describe as data preprocessing. First, we utilize a label encoder to encode the genre labels into numerical values. Then, we resize the input data to 64 by 100 in order to lessen the computational cost of running inputs of the size 128 by 1291. We also normalize the input data with a mean of -42.88 and a standard deviation of 15.50. Lastly, we plot our training and test loss/accuracy graphs. It is important to note that the linear model consisted of 64,010 trainable parameters.

2.3 Simple CNN

For the sake of consistency we utilized the same code structure in terms of training, evaluation, and data preprocessing in the Simple CNN model as we did in the Linear Model. Thus similarly we utilize the same *train_model* and *evaluate_model* to keep track of the train and test losses/accuracies. This is also run for 150 epochs with the same batch sizes, loss function, optimizer, learning rate, normalization, and data input size of 64 by 100, all for comparison purposes. However, here we run our Simple CNN model. With the Simple CNN, we have 3 convolution layers each with increasing output channels, followed by batch normalization, ReLU activation, and a max pooling layer after each successive layer in order to reduce some of the complexity of the model. We also include a dropout layer in an attempt to reduce some overfitting that might occur. The model also utilizes two fully connected layers which take in the modified input size, as after each convolution block the dimensions are reduced and finally flattened into a vector that passes through those layers. It is also important to note that our number of trainable parameters for the Simple CNN model is 15,610.

2.4 Complex CNN

Similarly, for the sake of consistency we follow the same code structure and data preprocessing as before. This model is almost identical to the Simple CNN other than the fact that we utilize a higher number of trainable parameters, that being, 59,114. Thus, the only changes here involve the output sizes of the convolution and linear layers. We double those values. For example, the first convolution layer of the Simple CNN takes in 1 channel and produces 8. In our Complex CNN model, we take in 1 channel but produce 16. Thus, at the end of our third convolution layer, we have 64 output channels whereas the Simple CNN had 32. Lastly, our fully connected layers utilize 256 nodes in comparison to the 128 nodes of the Simple CNN. These changes were made to see what the effects of increased parameters are on the accuracy of the model. The number of parameters here are closer to that of the ones used in the linear model.

3 Results

As stated earlier we are using Pytorch to form our systems architecture. The coding for this project was done using Jupyter Notebooks either through Google Colab or through the given software. Throughout the project, we produced confusion matrices for each model as well as train and test loss/accuracy graphs. It is important to note that, due to time constraints, the results presented here are that of one particular run. The limitations of this will be discussed in a later section. For this run, we found that the Complex CNN outperformed the Simple CNN and Linear models when it came to final test accuracy, as seen in Figure 5. However, an analysis and breakdown of the results, including overfitting, will be addressed in the discussion section.

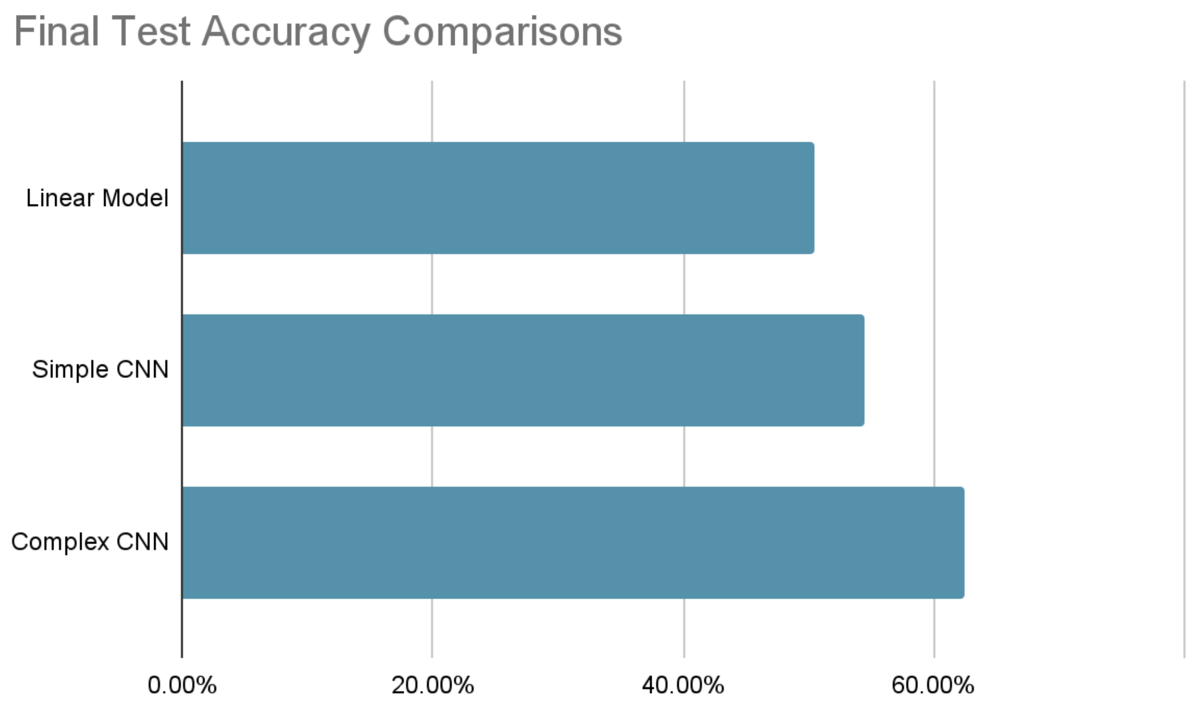


Figure 5: Model Accuracy Comparisons

3.1 Linear Model

The results of the Linear Model training/test loss and accuracy graphs can be seen in Figure 6. The top graph represents Training and Test Loss; this is helpful in order to visualize how the loss changed for both the training and test sets over each epoch. The bottom graph represents Training and Test Accuracy; which is also helpful as we are able to track how the accuracy changes over each epoch. For the Linear Model on epoch 150, we received a train loss of 0.3108 and a train accuracy of 98.25%. We received a test loss of 1.2655 and a test accuracy of 50.50%. It is evident that there is large overfitting going on with this model. The training set seems to reach almost perfect accuracy; however, the test accuracy is substantially lower. This seems to indicate that the model may be memorizing the training data yet failing to generalize the new data.

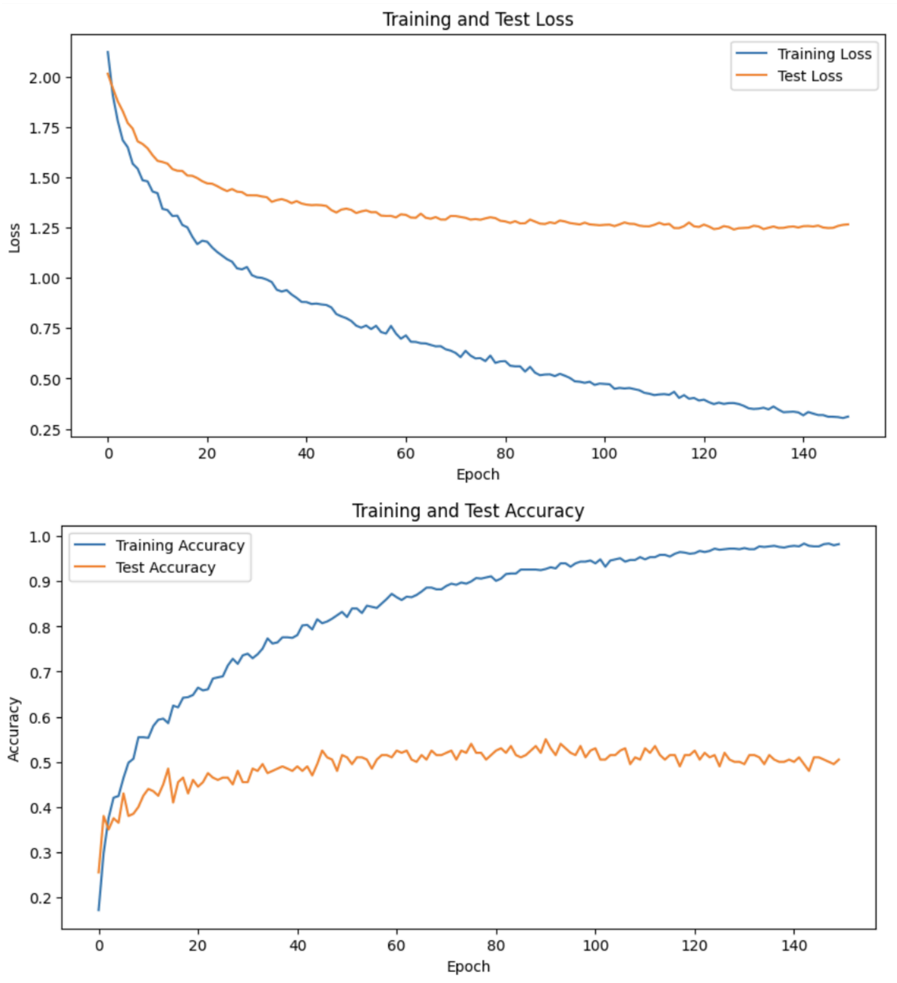


Figure 6: Linear Model Results

3.2 Simple CNN

The results of the Simple CNN can be seen in Figure 7. For the Simple CNN model on epoch 150, we received a train loss of 1.0657 and a train accuracy of 65.21%. We received a test loss of 1.4456 and a test accuracy of 54.50%. This model has substantially less parameters than our other two which may attest to the lessened overfitting that we see. It is interesting to see how close the training accuracy and test accuracy are to one another after 150 epochs. However, it is important to note that this model does take longer to converge when compared to the Linear Model above; thus further investigation may be necessary in order to accurately assess the results.

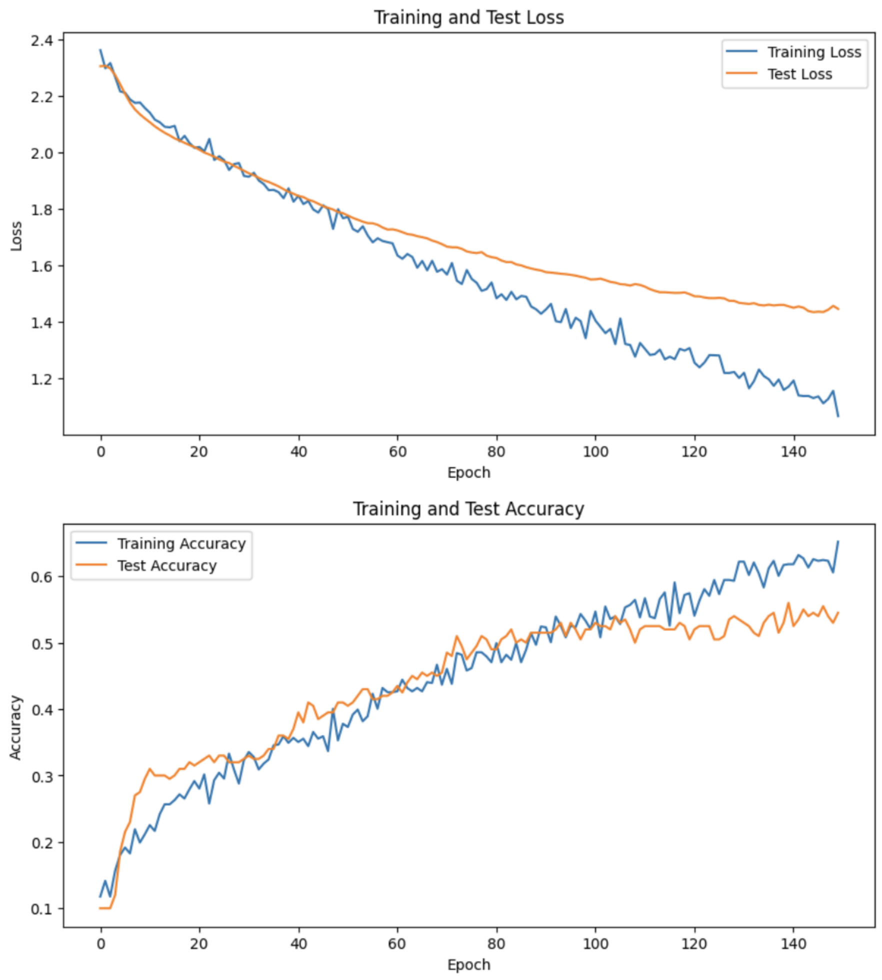


Figure 7: Simple CNN Results

3.3 Complex CNN

The results of the complex CNN can be seen in Figure 8. For the Complex CNN model on epoch 150, we received a train loss of 0.2804 and a train accuracy of 95.62% We received a test loss of 1.1579 and a test accuracy of 62.50%. With the increase of parameters, we do see an increase in the amount of overfitting going on when compared to the Simple CNN. This is very interesting and leads us to believe that a closer look needs to be taken in regards to the convergence points of the three models.

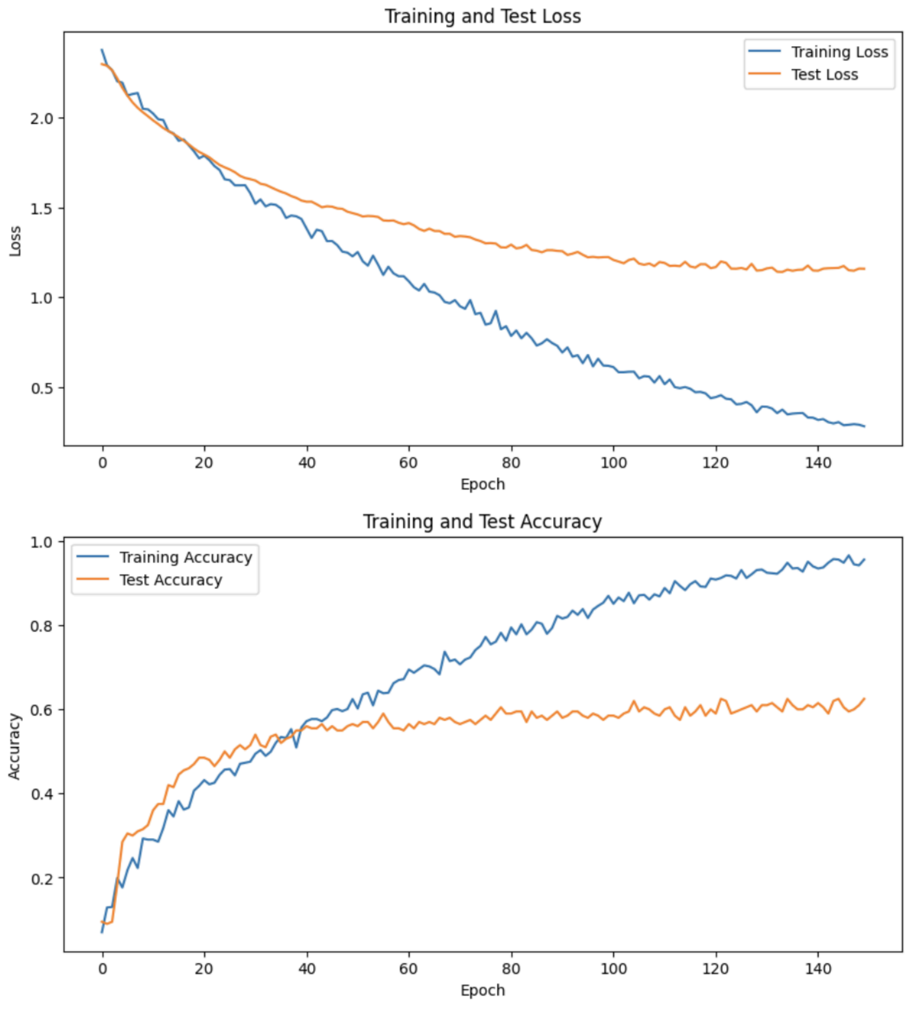


Figure 8: Complex CNN Results

3.4 Overall Comparison

The primary purpose of this report is to compare the linear baseline model and our CNN model. Thus the following graphs presented in Figure 9 showcase the results of all 3 models in comparison with one another. This way we can visualize their behaviors and points of convergence in comparison to one another.

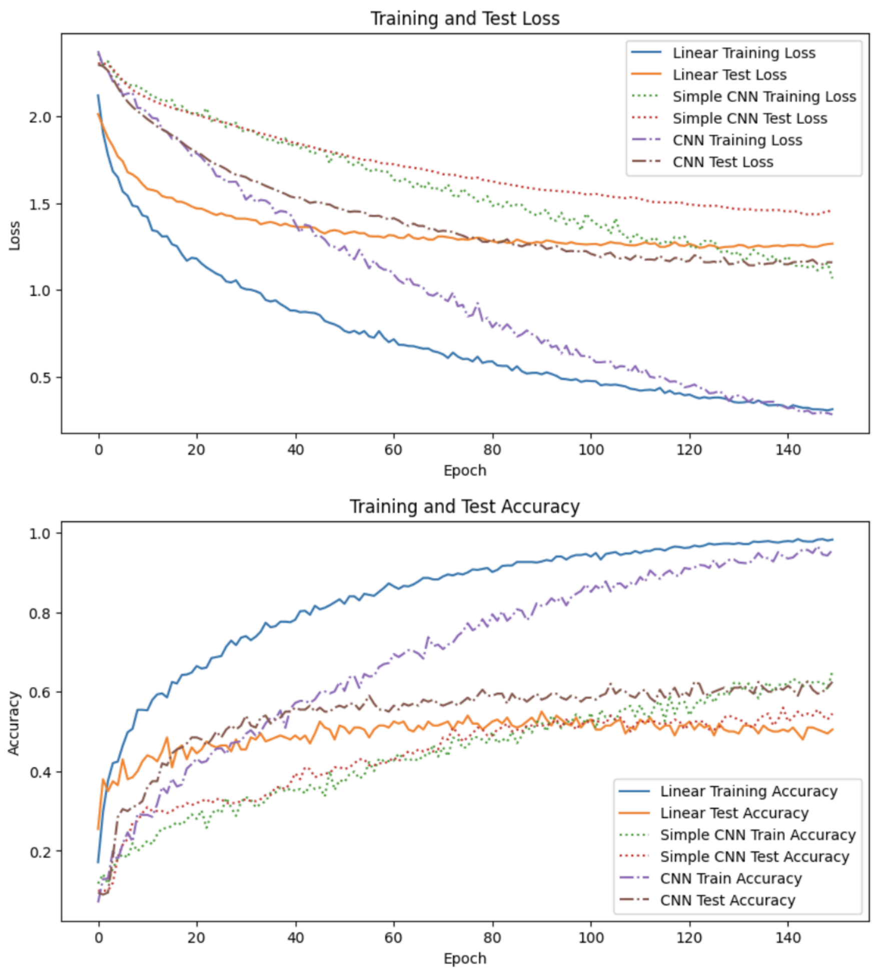
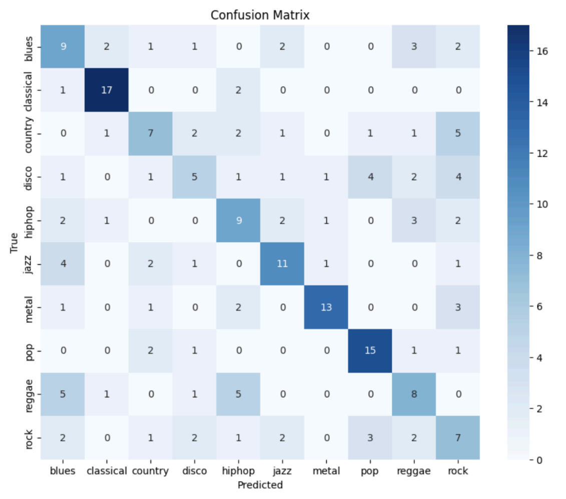


Figure 9: Comparison of the Results

3.5 Confusion Matrices

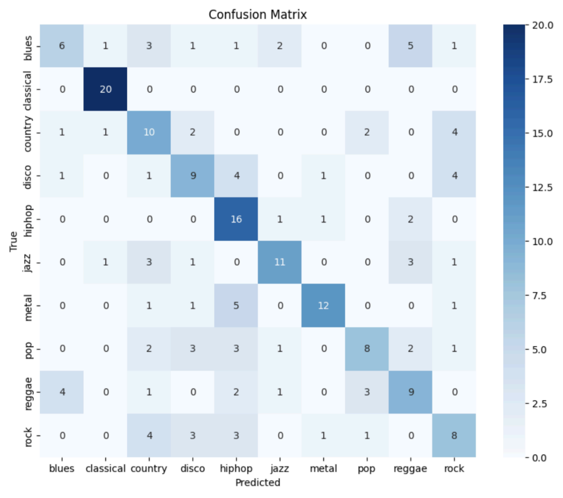
As mentioned earlier we built confusion matrices for each of the models in order to better understand how the test set was being classified. The results of these can be seen in Figure 10 below. This figure also demonstrates the increase in test accuracy as we go from Linear to Complex CNN. For example, if we look at the Disco category, the Linear Model correctly classified 5 out of the 20 files in that genre, the Simple CNN classified 9, and the Complex CNN classified 10. However, it is interesting to note that in the Pop category, the Linear Model correctly classified 15 files; whereas the Simple CNN only correctly classified 8 files, and the Complex CNN correctly classified 10. This clearly demonstrates some of the difficulties with classifying music genres; which again, can be due to the similar nature that some genres, such as Pop, can have when compared to other music genres.

Linear Model



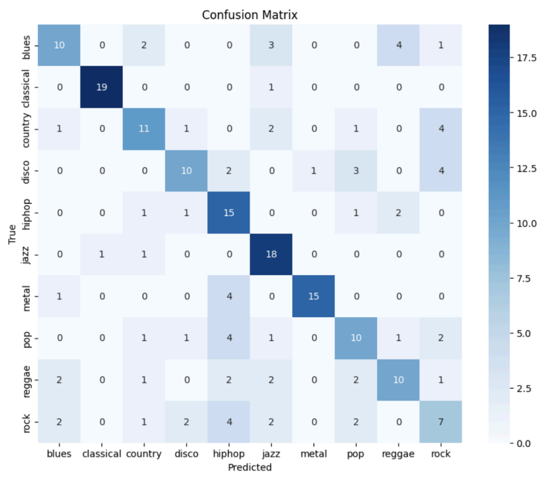
True \ Predicted	blues	classical	country	disco	hiphop	jazz	metal	pop	reggae	rock
blues	9	2	1	1	0	2	0	0	3	2
classical	1	17	0	0	2	0	0	0	0	0
country	0	1	7	2	2	1	0	1	1	5
disco	1	0	1	5	1	1	1	4	2	4
hiphop	2	1	0	0	9	2	1	0	3	2
jazz	4	0	2	1	0	11	1	0	0	1
metal	1	0	1	0	2	0	13	0	0	3
pop	0	0	2	1	0	0	0	15	1	1
reggae	5	1	0	1	5	0	0	0	8	0
rock	2	0	1	2	1	2	0	3	2	7

Simple CNN



True \ Predicted	blues	classical	country	disco	hiphop	jazz	metal	pop	reggae	rock
blues	6	1	3	1	1	2	0	0	5	1
classical	0	20	0	0	0	0	0	0	0	0
country	1	1	10	2	0	0	0	2	0	4
disco	1	0	1	9	4	0	1	0	0	4
hiphop	0	0	0	0	16	1	1	0	2	0
jazz	0	1	3	1	0	11	0	0	3	1
metal	0	0	1	1	5	0	12	0	0	1
pop	0	0	2	3	3	1	0	8	2	1
reggae	4	0	1	0	2	1	0	3	9	0
rock	0	0	4	3	3	0	1	1	0	8

Complex CNN



True \ Predicted	blues	classical	country	disco	hiphop	jazz	metal	pop	reggae	rock
blues	10	0	2	0	0	3	0	0	4	1
classical	0	19	0	0	0	1	0	0	0	0
country	1	0	11	1	0	2	0	1	0	4
disco	0	0	0	10	2	0	1	3	0	4
hiphop	0	0	1	1	15	0	0	1	2	0
jazz	0	1	1	0	0	18	0	0	0	0
metal	1	0	0	0	4	0	15	0	0	0
pop	0	0	1	1	4	1	0	10	1	2
reggae	2	0	1	0	2	2	0	2	10	1
rock	2	0	1	2	4	2	0	2	0	7

Figure 10: Confusion Matrices per Model

4 Discussion

From the results gathered above, we can perform comparisons of a linear baseline model, a simple CNN model, and a Complex CNN model. We can come to a few generalized conclusions based on the results above. Looking at Figure 9, the Training and Test Loss graph, we can see that the linear training and test loss start with less loss in the beginning when compared to the other models. It is also important to note that the test loss has a very fast convergence. However, as epochs continue we see the loss in the test set of the linear model ends up being a little higher when compared to the convergence point of the Complex CNN test loss. This can be seen around the 80th epoch as the Complex CNN begins to have a lower test loss than the Linear. The Simple CNN test loss has a very slow convergence and has the highest loss when compared to the other models. Lastly, both Complex CNN and Linear Models have very large issues of overfitting, it is difficult to decipher which one has more based on their points of convergence.

In regards to Figure 9, the Training and Test Accuracy graph, we can see that similarly, the Linear test accuracy converges very quickly and is only higher at the beginning for a small number of epochs. Soon we see that the Complex CNN overtakes in accuracy. The Simple CNN is slower to gain accuracy but ends up being slightly higher than the Linear Model. In terms of overfitting, at the 150 epoch mark, it seems that Linear has the largest amount, then Complex CNN, and lastly Simple CNN. However as mentioned before it would be interesting to conduct more analysis on their points of convergence, as in that case, it is difficult to tell the difference in overfitting between the Linear and Complex CNN models.

Overall, the Complex CNN achieved the highest test accuracy whereas the Simple CNN provides a fair compromise between accuracy and robustness by balancing model complexity with generalization performance. Further knowledge regarding how successfully the models perform on particular musical genres may be found in the confusion matrices. Even though the Complex CNN performs better than the other models most of the time, there are some cases where classification is difficult for specific genres. The Pop category, for instance, shows variations in classification accuracy between the models, underlining the difficulties in differentiating between related musical genres. In addition, we created a simple SVM model for further comparison of models. This gave us a train accuracy of 99.87% and a test accuracy of 57%. However, we believe that the SVM model needs further study and testing thus those results are not analyzed for the purposes of this report.

5 Limitations and Future Works

Due to the time constraints of the project, we believe there are a few notes that must be made about our results. Our dataset is very small and we are using very high numbers of parameters for each of these models. Thus, results can fluctuate very easily based on a number of factors. This could include initial conditions set by the *random_state* variable or even changes in the number of parameters per model. This report only analyzes the results of one run; thus, for validation of the results, it is imperative that in the future we run some sort of cross-validation techniques. This could include running multiple trials for various *random_state* variables. This means that we would be able to assess patterns in results fairly and accurately since we would not solely rely on one particular run. It would also be very helpful to conduct a deeper analysis into CNN and SVM architectures in order to compare models with greater detail. For example, we saw that the Simple CNN takes the most epochs to converge and has very little overfitting. It would be interesting to analyze where that dynamic is specifically coming from, whether that is from the model structure, number of parameters, or related to our choice of dataset. Lastly, it would be useful to investigate the use of pre-trained models on the given dataset. Therefore future works could also include the employment of a ResNet pre-trained model, as time did not allow for that in the given report. Overall, this was a very exciting first dive into machine learning for our team, we learned a lot, and hope to further our knowledge with future learning.

References

[1] Kumar, A., Rajpal, A., & Rathore, D. (2018, November). Genre classification using feature extraction and deep learning techniques. In 2018 10th International conference on knowledge and systems engineering (KSE) (pp. 175-180). IEEE. doi: 10.1109/KSE.2018.8573325.

[2] Joo, C., Kwon, H., Kim, J., Cho, H., & Lee, J. (2023). Machine-learning-based optimization of operating conditions of naphtha cracking furnace to maximize plant profit. In Computer Aided Chemical Engineering (Vol. 52, pp. 1397-1402). Elsevier. https://doi.org/10.1016/B978-0-443-15274-0.50222-5.

[3] Halnaut, A., Giot, R., Bourqui, R., & Auber, D. (2023). Compact visualization of DNN classification performances for interpretation and improvement. In Explainable Deep Learning AI (pp. 35-54). Academic Press. https://doi.org/10.1016/B978-0-32-396098-4.00009-0.

[4] Hung, C. L. (2023). Deep learning in biomedical informatics. In Intelligent Nanotechnology (pp. 307-329). Elsevier. https://doi.org/10.1016/B978-0-323-85796-3.00011-1.

[5] Doshi, K. (2021, February 18). Audio deep learning made simple (Part 2): why mel spectrograms perform better. Towards Data Science. https://towardsdatascience.com/audio-deep-learning-made-simple-part-2-why-mel-spectrograms-perform-better-aad889a93505

[6] Chowdhry, A. (2021, May 7). Music genre classification using CNN. Clairvoyant. https://www.clairvoyant.ai/blog/music-genre-classification-using-cnn

[7] Dutt, N. (2022, May 19). Music genre classification using CNN: Part 1- feature extraction. Medium. https://medium.com/@namratadutt2/music-genre-classification-using-cnn-part-1-feature-extraction-b417547b8981

[8] Doshi, K. (2021, March 18). Audio deep learning made simple: Sound classification, step-by-step. Towards Data Science. https://towardsdatascience.com/audio-deep-learning-made-simple-sound-classification-step-by-step-cebc936bbe5

[9] Marsyas (Music Analysis, Retrieval and Synthesis for Audio Signals). GTZAN Music Genre Dataset. Hugging Face Datasets. https://huggingface.co/datasets/marsyas/gtzan

[10] McFee, B., Matt McVicar, Daniel Faronbi, Iran Roman, Matan Gover, Stefan Balke, Scott Seyfarth, Ayoub Malek, Colin Raffel, Vincent Lostanlen, Benjamin van Niekirk, Dana Lee, Frank Cwitkowitz, Frank Zalkow, Oriol Nieto, Dan Ellis, Jack Mason, Kyungyun Lee, Bea Steers, … Waldir Pimenta. (2023). librosa/librosa: 0.10.1 (0.10.1). Zenodo. https://doi.org/10.5281/zenodo.8252662

[11] Getting started. scikit learn. (n.d.). https://scikit-learn.org/stable/getting_started.html

[12] Scikit-learn: Machine Learning in Python, Pedregosa et al., JMLR 12, pp. 2825-2830, 2011.

[13] PyTorch documentation. PyTorch documentation - PyTorch 2.3 documentation. (n.d.). https://pytorch.org/docs/stable/index.html

[14] J., Rodríguez, Zaurín., Pavol, Mulinka. (2023). pytorch-widedeep: A flexible package for multimodal deep learning. The Journal of Open Source Software, doi: 10.21105/joss.05027

[15] Librosa.feature.melspectrogram. librosa.feature.melspectrogram - librosa 0.10.2dev documentation. (n.d.-b). https://librosa.org/doc/main/generated/librosa.feature.melspectrogram.html

[16] Sklearn.model_selection.train_test_split. scikit learn. (n.d.). https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html