COMP 520: Music Genre Classification with RNNs

1st Ryan Chang
School of Science, Engineering, and Technology
The Pennsylvania State University - Harrisburg
Harrisburg, USA
rjc5887@psu.edu

2nd Joel Horne
School of Science, Engineering, and Technology
The Pennsylvania State University - Harrisburg
Harrisburg, USA
jth6046@psu.edu

3rd Alexander Hughes
School of Science, Engineering, and Technology
The Pennsylvania State University - Harrisburg
Harrisburg, USA
ajh7066@psu.edu

***Abstract*—This report provides a discussion on the challenge of Music Genre Classification in Machine learning. We discuss current works related to Music Genre Classification, which includes the use of Mel-Spectrograms as training data due to their ability to illustrate patterns of human perceptions in songs. The project in this report uses different types of Recurrent Neural Networks that are trained on the same dataset and then evaluated on the same validation data to compare accuracy and results.**

***Index Terms*—Machine Learning, Recurrent Neural Networks, Long Short-Term Memory, Gated Recurrent Unit, Artificial Intelligence, Music Genre Classification, Supervised Learning, Mel-Spectrograms**

I. Introduction

Music genre classification in machine learning is the challenge of analyzing audio features of songs to categorize them into their respective genres such as rock, jazz, hiphop, etc. In recent times, this analysis has been done by transforming the sound features of songs into numerical data such as spectrograms. Using these numerical representations of the audio data, studies have shown that genre classification can be achieved with accuracy of near 80% for models trained to handle the task using different types of network structures [1]. As of today, studies have shown that applying Recurrent Neural Networks (RNNs) to genre classification has promising results, specifically Long Short Term Memory (LSTM) RNNs. In this paper, we were curious to test another type of RNN called the Gated Recurrent Unit (GRU), a simpler version of the LSTM model, and compare it with LSTM to see if a simpler network can learn just as well as the more complex one. GRUs are typically used for smaller datasets to achieve faster training and efficiency, while LSTMs are typically used when training on large and complex datasets with long sequences.

II. Related Work

Most studies on the topic of Genre Classification today use Mel-Spectrograms as training samples during the training process. Mel Spectrograms compress audio into meaningful frequency bins that represent human perception of audio. The Mel Spectrograms are created based off of the Mel Scale which helps to capture harmonic structure, timbre, rhythmic energy, and other genre specific patterns that can be perceived in the spectrogram. Because the spectrograms are based off the Mel Scale, we are able to see distinct patterns for audio clips that fall under the same genre [2]. In our project, we also decide to take the Mel Spectrogram of the audio files in our training data and use these as input for training samples during the training phase.

We have found a common dataset used for the task of genre classification is the GTZAN dataset. This dataset contains 1000 30 second audio clips and 10 unique genre labels. It is a relatively small dataset compared to other datasets available online. Studies using this dataset were able to achieve 70-80% accuracy using different types of networks such as Convolutional Neural Networks (CNNs), LSTMs, and even hybrid models like Convolutional RNNs [1]. The training dataset we selected for this project, the FMA Genre Dataset, is about six times larger than GZTAN, and was selected to test if accuracy of genre classification can be improved using different sized datasets [3]. In this project, we decided to train both an LSTM and GRU model on the same dataset (as well as a subset of the dataset) to have a comparative baseline (LSTM) and an experimental model (GRU).

III. Methodology

A. Dataset Selection

For this project, we selected the fma-genre-classification dataset, which is publicly available on hugging face [3]. This dataset contains eight thousand samples total and uses a 80/20 split to create the training and validation datasets. Each sample in the dataset contains a 30 second mp3 audio clip and a genre label. The dataset has eight unique genre labels in total. As mentioned before, the dataset we selected is much larger than the commonly used GTZAN dataset, which contains only one thousand samples.

B. Pre-Processing

To preprocess the audio clips into usable training samples, we first had to load the dataset from hugging face using the datasets library [4]. Then once we have the dataset loaded, we need to decode the audio into waveforms. Our first attempt for this conversion used the default PyTorch audio decoder; however, this decoder was not robust enough to convert all of the audio clips in our dataset. Instead we decided to use the FFMPEG framework which includes a decoder that lets you transform audio clips into the desired waveform format [5].

The FFMPEG framework was used to convert byte data of the audio clips into a PCM int16 mono channel audio waveform with a sampling rate of 16kHz. The waveform was then normalized to contain values between [-1, 1]. Resampling to a uniform sampling rate was done to ensure consistent spectrogram shapes. Forcing a mono channel waveform also helps to reduce the waveform size and make it more scalable for training. For this project, we decided not to trim the audio files into smaller audio clips because we wanted to test the ability of an LSTM network to analyze and learn from large and complex data samples.

The waveform produced by the FFMPEG decoder is then inputted into our feature extractor. The feature extractor we decided to use is the ASTFeatureExtractor [6] [7]. This extractor was recommended to be used on the fma-genre-classification dataset's hugging face page. The ASTFeatureExtractor expects normalized waveforms and extracts the Mel-Spectrogram for the waveforms that are inputted. These Mel-Spectrograms are outputted as tensors by the ASTFeatureExtractor. Once we have our Mel-Spectrogram tensors, we are ready to begin the training process. Once we have preprocessed our full dataset, we saved the full dataset as well as a subset of the dataset that is about one fourth of the original dataset's size. We use both this small dataset and the full dataset to compare the results of LSTM and GRU networks being trained on datasets of different sizes.

C. Network

A core part of our engineering process involved testing different kinds of Recurrent Neural Networks with our data set. Since the mel spectrogram extracted from the samples was time-dependent we decided early on that an RNN would be best used to understand spectrogram features over time. An RNN is designed to handle series dependent data, such as words in a text sample or frames of a video. Its original design dates back to the classical era of machine learning, but more recent iterations of the model have lead to design iterations such as Long-Short Term Memory (LSTM) and Gated Recurrent Unit (GRU) RNNs. The basic RNN uses one or more hidden layers that are updated at each iteration using the next input in the series for each sample. In PyTorch's RNN implementation, the hidden layer is calculated as such: $h_t = \tanh(x_t W_{ih}^T + b_{ih} + h_{t-1} W_{hh}^T + b_{hh})$ where the activation function may be $\tanh$ or ReLU [8]. The weights and biases labeled ih are the input layer and the weights/biases labeled hh are the hidden layer, which maintains its own state using the previous hidden layer. Thus over successive inputs the hidden layer of the RNN attempts to encapsulate the features of the sample.

Basic RNNs such as the Ellman RNN suffer from problems such as the vanishing gradient problem, which causes all values in a normalized layer to shrink to zero over time due to repeated multiplication. The vanishing gradient problem is specifically addressed by another kind of RNN tested by our team, the Long-Short Term Network [9]. The LSTM uses a memory cell and a set of three gates to control what information goes into, stays, and is released from the cell into the hidden layer. These gates are called the input, forget, and output gates respectively. By using the gates to select relevant information to hold over the whole training period the LSTM attempts to learn when information may be needed as the sequence progresses. We also test an implementation of a Gated Recurrent Unit RNN [10], a more simplified version of LSTM. Like LSTM, the GRU uses gates to control what information is added to the hidden layer, but lacks the memory cell and the output gate from the cell. This results in a tradeoff between the advantage in memory from the cell with the size of the model, with GRU having much fewer parameters to train.

D. Training

To train our network we started by taking the preprocessed mel spectrogram data and feeding it into PyTorch's RNN module. In later tests, this was easily swapped with PyTorch's GRU and LSTM modules. This step built the recurrent network from each step in the sample. The output of the of the RNN is then normalized using PyTorch's LayerNorm class, which uses the variance of the values in the layer's nodes to compress those values into a range between 0 and 1. This is done to increase the speed of the network and reduce vanishing/exploding gradients. The output of the normalization layer is fed into a linear layer as an attention mechanism. Softmax is applied to this layer and its weights are then summed with the output of the RNN to find weighted values of the RNN output. A final classification layer is used to map the output from the weighted sum to the genre prediction. The loss function we decided to minimize during training was Cross Entropy Loss. We decided upon this loss function because it is efficient in classification problems, especially when classifying using indices, which we do in our project [11].

During the training phase of our project, we trained a number of different models using different hyperparameter values. We also tested training on the dataset using a different number of epochs. For the scope of this project, we decided to train multiple models using one hidden layer, two hidden layers, and one test of sixteen hidden layers. The goal was to test the effect of the number of hidden layers on the training

of the model, but due to the time constraints of the project and diminishing returns, we decided to limit these values. In addition, we also tested the effect of the hidden layer size. We trained models with a hidden layer size of 64 and a hidden layer size of 128 to test if increasing the hidden layer size would improve training results. The final item we tested was the number of linear layers. For the large portion of our tests, we only used one. However, towards the end of our project we ran some tests using 2 linear layers. In order to test whether or not our model would overfit to the dataset, we also trained models for a different number of epochs and analyzed at what point we see diminished returns using increased epochs. Finally, we tested the learning rate to see if we could train for different amounts of epochs without overfitting or underfitting.

For a detailed analysis of our methodology, visit our GitHub repository [12].

IV. Results

A. Experiments

To evaluate model performance across the tested architectures and hyperparameter configurations, we measured several metrics during and after training, with primary emphasis placed on training loss and validation accuracy. Training loss represents the value of the objective function observed during optimization and provides insight into how well the model fits the training data, while validation accuracy is defined as the percentage of correctly predicted samples when the trained model is evaluated on a held-out validation set. These two metrics were used jointly to assess convergence behavior and generalization performance. In addition to reporting training loss and validation accuracy independently, we introduced a composite overfitting metric designed to capture the relationship between training performance and validation error by normalizing both quantities. The overfitting metric is defined as

$$\text{Overfit Score} = \left(1 - \frac{A_{\text{val}}}{100}\right) \frac{L_{\text{train}}}{1 + L_{\text{train}}}, \tag{1}$$

where A_{val} denotes validation accuracy expressed as a percentage and L_{train} denotes the final training loss. This metric produces lower values for models that achieve low training loss while demonstrating poor validation accuracy and was used to quantify overfitting trends across architectures, hidden layer configurations, learning rates, and training durations. In this formulation, smaller overfit score values correspond to increased overfitting.

In this report, model names follow the format: # Epochs (Ep.) {GRU/LSTM} - {learning rate} LR - {number of hidden layers} HL, where each component indicates the number of training epochs, the architecture type, the learning rate, and the number of hidden layers, respectively.

B. GRU Model Performance

Figure 1 shows the comparison of GRU models with varying hidden layer sizes and hyperparameters. Validation accuracy is displayed as blue bars, while the overfitting score is represented by the red line. As summarized in Table I, models with fewer epochs and hidden layers generally achieved higher validation accuracy, and overfitting scores varied across hyperparameter settings. The top-performing GRU model attained a validation accuracy of 49% with a moderate overfit score at just 10 epochs.

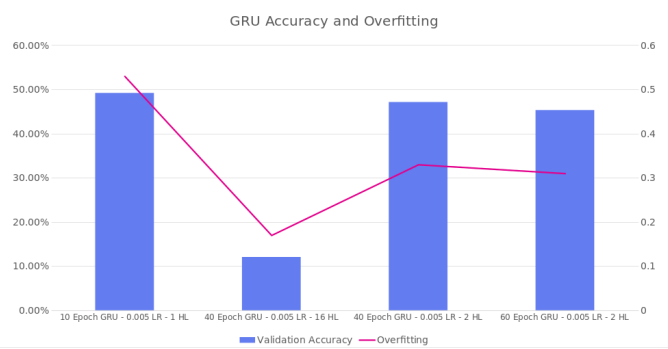

Fig. 1. Comparison of GRU models with varying hyperparameters. Validation accuracy is shown as blue bars and the overfitting score is shown as a red line.

TABLE I
PERFORMANCE METRICS FOR GRU MODELS, INCLUDING TRAINING LOSS (TRAIN. LOSS), VALIDATION ACCURACY (VALID.), AND OVERFITTING SCORE (OVERFIT).

Model	Train. Loss	Valid. (%)	Overfit
10 Ep. GRU - 0.005 LR - 1 HL	1.130	49.19	0.53
40 Ep. GRU - 0.005 LR - 16 HL	0.200	12.06	0.17
40 Ep. GRU - 0.005 LR - 2 HL	0.500	47.13	0.33
60 Ep. GRU - 0.005 LR - 2 HL	0.442	45.31	0.31
40 Ep. GRU - 0.01 LR - 2 HL	1.680	40.00	0.62

C. GRU versus LSTM Comparison

Table II compares the best GRU model against selected LSTM architectures. Across the evaluated metrics, LSTM models consistently achieved higher validation accuracy than the best GRU model while maintaining comparable or lower overfitting scores. These results guided the selection of LSTM configurations for further analysis.

TABLE II
COMPARISON OF THE BEST GRU MODEL WITH SELECTED LSTM MODELS INCLUDING VALIDATION ACCURACY, TRAINING LOSS, AND OVERFITTING SCORE.

Model	Train. Loss	Valid. (%)	Overfit
20 Ep. LSTM - 0.001 LR - 2 HL	1.280	50.06	0.56
10 Ep. GRU - 0.005 LR - 1 HL	1.130	49.19	0.53

D. LSTM Model Performance

Figure 2 presents the performance of LSTM models with varying hidden layer configurations. Validation accuracy is shown in blue, and the overfitting score in red. Table III provides the corresponding numerical values. Among the LSTM models, increasing the number of linear layers from

one to two generally led to modest improvements in validation accuracy while maintaining low overfitting scores.

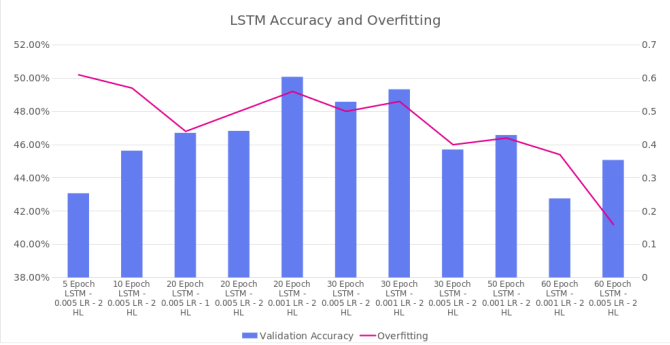

Fig. 2. Comparison of LSTM models with varying hyperparameters. Validation accuracy is shown in blue bars and the overfitting score is shown as a red line.

TABLE III
PERFORMANCE METRICS FOR LSTM MODELS WITH DIFFERENT HYPERPARAMETERS, TRAINING LOSS (TRAIN. LOSS), VALIDATION ACCURACY (VALID.), AND OVERFITTING SCORE (OVERFIT).

Model	Train. Loss	Valid. (%)	Overfit
5 Ep. LSTM - 0.005 LR - 2 HL	1.590	43.06	0.61
10 Ep. LSTM - 0.005 LR - 2 HL	1.360	45.62	0.57
20 Ep. LSTM - 0.005 LR - 1 HL	0.779	46.69	0.44
20 Ep. LSTM - 0.005 LR - 2 HL	1.000	46.81	0.50
20 Ep. LSTM - 0.001 LR - 2 HL	1.280	50.06	0.56
30 Ep. LSTM - 0.005 LR - 2 HL	1.020	48.56	0.50
30 Ep. LSTM - 0.001 LR - 2 HL	1.140	49.31	0.53
50 Ep. LSTM - 0.001 LR - 2 HL	0.723	46.56	0.42
60 Ep. LSTM - 0.001 LR - 2 HL	0.593	42.75	0.37
60 Ep. LSTM - 0.005 LR - 2 HL	0.198	45.06	0.16

E. Small Dataset Comparisons

Table IV summarizes performance metrics for models trained on smaller subsets of the dataset. Both GRU and LSTM models show reduced validation accuracy compared to training on the full dataset, and overfitting scores indicate increased sensitivity to dataset size. These results highlight the impact of dataset size on generalization without drawing interpretive conclusions.

TABLE IV
PERFORMANCE METRICS FOR MODELS TRAINED ON SMALLER DATASETS, INCLUDING TRAINING LOSS (TRAIN. LOSS), VALIDATION ACCURACY (VALID.), AND OVERFITTING SCORE (OVERFIT).

Model	Train. Loss	Valid. (%)	Overfit
20 Ep. LSTM - 0.005 LR - 2 HL	0.674	45.69	0.40
5 Ep. GRU - 0.005 LR - 1 HL	1.100	41.88	0.52
20 Ep. GRU - 0.005 LR - 2 HL	0.639	40.94	0.39
40 Ep. GRU - 0.005 LR - 1 HL	0.148	35.75	0.13

F. Confusion Matrix

Figure 3 displays the confusion matrix for the best-performing model. Each cell represents the number of samples predicted for each class versus the actual class, providing insight into class-specific performance.

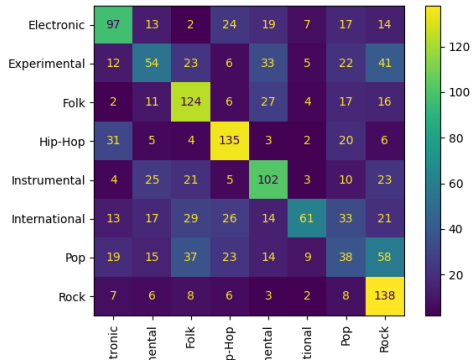

Fig. 3. Confusion matrix for the best-performing model, showing the distribution of predicted versus actual classes.

V. CONCLUSION AND FUTURE IMPROVEMENTS

In this study, we evaluated the performance of GRU and LSTM models for music genre classification using a publicly available dataset. The analysis focused on training loss, validation accuracy, and an overfitting metric designed to quantify the extent of overfitting across different architectures, hyperparameters, and dataset sizes.

A. GRU Model Observations

The GRU models demonstrated significant limitations. As summarized in Table I and Figure 1, the best-performing GRU model achieved only 49% validation accuracy at 10 epochs. Regardless of the number of epochs, learning rate, or hidden layer configuration, GRU models overfit almost immediately, as indicated by the overfitting scores. This rapid overfitting limited their ability to generalize, making GRUs a less suitable architecture for this task under the tested configurations.

B. LSTM Model Observations

LSTM models with one linear layer demonstrated slower overfitting compared to GRUs, though validation accuracy remained similar, reaching only 49% (Table III). Increasing the number of linear layers to two yielded a modest improvement, achieving 50% validation accuracy with 20 epochs. The overfitting scores indicate that LSTMs tolerate longer training durations better than GRUs. It is plausible that using a smaller learning rate and additional epochs could further improve performance beyond 50%, suggesting that LSTMs are more robust for this classification task.

C. Small Dataset Training

Models trained on smaller subsets of the dataset (Table IV) performed worse overall, with both GRU and LSTM architectures achieving lower validation accuracy and higher sensitivity to overfitting. This highlights the importance of dataset size in generalization performance and confirms that reducing the training data negatively impacts model outcomes.

D. Dataset and Labeling Considerations

Analysis of the confusion matrix for the best-performing model (Figure 3) indicates that the 50% validation accuracy, though modest, is reasonable given the dataset characteristics. Some genres are highly subjective or overlapping, such as experimental and international categories, which could reasonably belong to multiple of the other genres in our dataset (folk, hip-hop, instrumental, pop, rock, and electronic). Improving the labeling strategy, including assigning multiple genre labels per audio track, may yield more realistic performance metrics. Prior work using the same dataset achieved a maximum accuracy of 54.24% using a CNN architecture, and only a staggering 40.75% using their AST [13]. Given these constraints, our results demonstrate that LSTM models are competitive within the context of this dataset and labeling complexity.

E. Summary

Overall, our experiments indicate that GRUs are prone to immediate overfitting and limited in accuracy, while LSTMs offer greater stability and slightly improved validation performance. In addition, with 30 second audio clips, more linear layers are required to capture all of the information. Dataset size and labeling quality significantly affect outcomes, and future work should consider multi-label classification and more objective labeling standards to better capture the nuanced structure of music genres.

REFERENCES

[1] J. Irvin, E. Chartock, and N. Hollander, "Recurrent neural networks with attention for genre classification," Stanford University, Tech. Rep., 2016, cS229 Course Project. [Online]. Available: https://cs229.stanford.edu/proj2016/report/IrvinChartockHollander-RecurrentNeuralNetworkswithAttentionforGenreClassification-report.pdf

[2] A. Pun and K. Nazirkhanova, "Music genre classification with mel spectrograms and convolutional neural networks," Stanford University, Tech. Rep., 2021, cS229 Course Project. [Online]. Available: https://cs229.stanford.edu/proj2021spr/report2/81973885.pdf

[3] M. Defferrard, K. Benzi, P. Vandergheynst, and X. Bresson, "Fma: A dataset for music analysis," in *18th International Society for Music Information Retrieval Conference*, 2017.

[4] Q. Lhoest, A. Villanova del Moral, Y. Jernite, A. Thakur, P. von Platen, S. Patil, J. Chaumond, M. Drame, J. Plu, L. Tunstall, J. Davison, M. Šaško, G. Chhablani, B. Malik, S. Brandeis, T. Le Scao, V. Sanh, C. Xu, N. Patry, A. McMillan-Major, P. Schmid, S. Gugger, C. Delangue, T. Matussière, L. Debut, S. Bekman, P. Cistac, T. Goehringer, V. Mustar, F. Lagunas, A. Rush, and T. Wolf, "Datasets: A community library for natural language processing," in *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*. Online and Punta Cana, Dominican Republic: Association for Computational Linguistics, Nov. 2021, pp. 175–184. [Online]. Available: https://aclanthology.org/2021.emnlp-demo.21

[5] kkroening, "ffmpeg-python: Python bindings for ffmpeg – with complex filtering support," GitHub repository, 2025, accessed: 2025-12-16. [Online]. Available: https://github.com/kkroening/ffmpeg-python

[6] Z. Yu *et al.*, "A survey of deep learning for music information retrieval," *arXiv preprint arXiv:2104.01778*, 2021. [Online]. Available: https://huggingface.co/papers/2104.01778

[7] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz, J. Davison, S. Shleifer, P. von Platen, C. Ma, Y. Jernite, J. Plu, C. Xu, T. L. Scao, S. Gugger, M. Drame, Q. Lhoest, and A. M. Rush, "Transformers: State-of-the-art natural language processing," in *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*. Online: Association for Computational Linguistics, Oct. 2020, pp. 38–45. [Online]. Available: https://www.aclweb.org/anthology/2020.emnlp-demos.6

[8] PyTorch. (2023) torch.nn.rnn. PyTorch API documentation. [Online]. Available: https://docs.pytorch.org/docs/stable/generated/torch.nn.RNN.html

[9] ——. (2023) torch.nn.lstm. PyTorch API documentation. [Online]. Available: https://docs.pytorch.org/docs/stable/generated/torch.nn.LSTM.html

[10] ——. (2023) torch.nn.gru. PyTorch API documentation. [Online]. Available: https://docs.pytorch.org/docs/stable/generated/torch.nn.GRU.html

[11] Lightly.ai. (2023) Pytorch loss functions: A practical guide. Technical blog. [Online]. Available: https://www.lightly.ai/blog/pytorch-loss-functions

[12] R. Chang, J. Horne, and A. Hughes, "PSU Comp-520 Music Genre Classification with RNNs," Dec. 2025. [Online]. Available: https://github.com/alexanderjhughes/comp-520-final-project

[13] J. Vos. (2025) Genre classification with neural networks. HackMD notes. [Online]. Available: https://hackmd.io/@jimvos/rJltGnVvh