Music Genre Classification: Using Deep Learning Methods on GTZAN Dataset

Ruinan Ma
UC San Diego
r7ma@ucsd.edu

Jialin He
UC San Diego
jih088@ucsd.edu

Shang Zhou
UC San Diego
shz060@ucsd.edu

Abstract

The report focuses on using neural networks for music genre classification on the GTZAN dataset.[1] The GTZAN dataset is a well-known collection of 1,000 thirty-seconds audio tracks evenly distributed across 10 genres. By applying feature extraction and neural networks to dataset, we aims to explore the potential of this machine learning technique for accurately classifying music by genre. We finally achieve an accuracy of 95.85%, which could beat all models we have seen on Kaggle.

1 Introduction

The purpose of this report is to present the results of a study on music genre classification using the GTZAN dataset. Even though some online GTZAN dataset contains In this study, we performed feature extraction on the dataset and implemented two different neural networks for classification: one without using any deep learning frameworks and another using Keras. We also improved the performance of the latter network using model blending techniques.

This link to our code: Click Here

2 Dataset

The GTZAN dataset, also known as the Genre Collection, is a popular dataset used for music genre classification. It was created by George Tzanetakis in 2001, and includes 1,000 audio tracks uniformly distributed across 10 music genres: blues, classical, country, disco, hiphop, jazz, metal, pop, reggae, and rock. Each track is 30 seconds long and is labeled with the corresponding genre. The GTZAN dataset has been widely used in research and has become a standard benchmark for evaluating the performance of music genre classification algorithms.

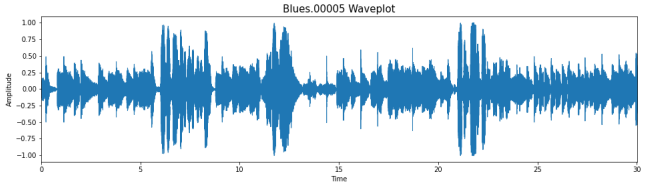

Figure 1: Overview plot of an example audio file

To extract features from the audio tracks in the GTZAN dataset, we can use various techniques from the field of signal processing. One common approach is to convert the raw audio data into a spectrogram, which is a visual representation of the frequency content of the audio over time.

From the spectrogram, we can then extract various features that capture different aspects of the audio, such as the overall spectral content, the rhythm, and the timbre. These features can then be fed into a machine learning algorithm, such as a neural network, to train a model for music genre classification.

To reduce the noise in audio information and help our models better understand the input, we preprocessed audio signals by extracting useful features. Features extracted and were used as input include audio length, chroma STFT, rms, spectral centroid, spectral bandwidth, rolloff, zero crossing rate, harmony, tempo, MFCC, and plp. Means and variances were calculated for most of these features and for MFCC broken down to 20 windows [2][3][4].

Length describes the exact duration of the audio. Chroma stft refers to chroma features calculated using short-time Fourier transform, which is a robust tool to capture the similarity between parts of the audio [5].

Function for feature extraction is defined in `extra_feature.py`.

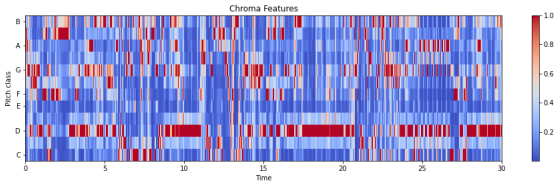

Figure 2: Chroma feature plotted for the sample audio

RMS basically describes how loud the audio is on average. Spectral centroid refers to the weighted mean of frequency in the audio signal, using the magnitude as the weight. Then, based on spectral centroid, spectral bandwidth is derived from calculating the variance of it. It reflects the resolution of the audio. The rolloff value sets a cutoff value to distinguish harmonics and noise and implies the sharpness of the audio. Zero-crossing rate indicates how often the audio signal crosses the horizonal axis, which represents when the speaker rests, changing from positive to negative or vice versa.

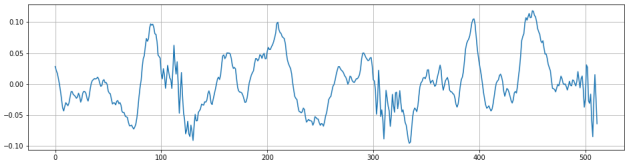

Figure 3: Zero-crossing plot for the sample audio

Harmony refers to the point where two or more notes played at the same time, or they follow each other very closely. Tempo measures how fast the audio is being played by calculating the number of beats per minute. Both MFCC and PLP processes audio to eliminate information that is normally not processed by human. At the same time, MFCC is broken up to multiple windows, better capturing the characteristics of different moments in the time-series audio.

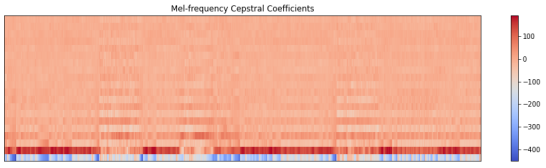

Figure 4: MFCC features ploted for the sample audio

To increase the size of our training dataset, we implemented data augmentation, separating each 30-second audio file to ten 3-second pieces, and generated 10,000 data points. Each training point

consists of 58 features as described above. Then we did train/test dataset split with the ratio of train: test as 4:1.

3 Model

3.1 Neural Network by Numpy

This neural network is implemented from stretch using numpy library. Code for this model is in `Code/Custom_NN (Model 1)`

3.1.1 Network Topology and Activation Function

In our best model, we used the rectified linear unit (ReLU) activation function for the hidden layers and the softmax activation function for the output layer. The number of units per layer is [58, 256, 256, 10], respectively. The corresponding best accuracy we obtained is 87.74%.We also tested other model topologies and other activation functions, as the chart showed.

Network Topology	Activation Function	Accuracy	Loss
[58,258,258,10]	ReLU	0.8774643615407947	0.4681550509349491
[58,128,128,10]	ReLU	0.8717015468607825	0.49907889887458795
[58,128,256,10]	ReLU	0.861995753715499	0.5054690367424867
[58,128,10]	ReLU	0.8571428571428571	0.5248580932669833
[58,128,128,10]	Sigmoid	0.7919320594479831	0.6174177036465014
[58,128,10]	Sigmoid	0.7895056111616621	0.632712939148163
[58,128,128,10]	tanh	0.8255990294206855	0.6306833448744908
[58,128,10]	tanh	0.8331816803154383	0.5378751535750848

Table 1: Accuracy for different network topology

3.1.2 Forward Pass

For the forward pass, we initialize:

$$z_1 \leftarrow W_1 x + b_1$$
$$a_1 \leftarrow ReLU(z_1)$$

For hidden layers:

$$z_i = W_i x + b_i$$
$$a_i = ReLU(z_i)$$

For the output layer:

$$z_L = W_L x + b_L$$
$$a_L = softmax(z_L)$$

3.1.3 Backward Pass

Define

$$\delta_j^n = -\frac{\partial E^n}{\partial a_j^n}$$

where

$$a_j^n$$

is the weighted sum of the inputs to unit j for pattern n.

For the output layer,

$$\delta_k^n = t_k^n - y_k^n$$

For the hidden layers,

$$\delta_j^n = (f'(a_j^n)) \sum_k (t_k^n - y_k^n)\, w_{jk}$$

where f' is the derivative of the activation function.

Weights are updated by:

$$w_{ij} \leftarrow w_{ij} + \delta_j z_i$$

Where α is the learning rate, w_{ij} is the weight from unit i to unit j.

3.1.4 L2 Regularization

Adding a L2 regularization term to the loss function by:

$$Loss = Error(Y - \widehat{Y}) + \lambda \sum_{1}^{n} w_i^2$$

Theoretically, adding regularization term to our model could reduce overfitting by preventing some weights in the model to inflate, which usually occurs in an overfitting model. We find out that the difference between test accuracy and train accuracy indeed decrease slightly after we add the regularization term. The decrease is not as obvious as mentioned in the theory. This can due to the fact that our model only has two or three hidden layers and thus the overfitting issue is not that obvious. We also discovered that the best hyperparameter for $lambda$ is 0.3.

3.1.5 Momentum

Momentum is a technique that is commonly used to accelerate the training of a neural network. In our Neuralnetwork class, start with initial momentum $v_0 = 0$. In the backpropagation step, we use the momentum update rule:

$$v_i \leftarrow \gamma v_{i-1} - \epsilon \frac{\partial E}{\partial w}$$

And we use the momentum term to update weights(α is the learning rate):

$$w_i \leftarrow w_{i-1} + \alpha v_i$$

The idea behand momentum is that, if we stuck at local minimum or plateau during training, incorporating the previous update of the weights into the current update can help the network to escape from this suboptimal state and continue to make progress. This is because the previous update provides some information about the direction of the error gradient, which can help the network to make more informed updates to the weights. By using momentum, the network can avoid getting stuck in suboptimal states and converge more quickly to a global minimum of the loss function.

3.1.6 Hyperparameters

Below is the set of hyperparameters we use.

activation function	ReLU
batchsize	16
early stop epoch	5
L2 penalty	0.05
momentum	True
momentum gamma	0.3
weight type	random initialization
learning rate	0.2

Table 2: config file for 256-256-relu.yaml

3.2 Neural Network by Keras

We implemented a multi-layer dense neural network in Keras, which used the Adam optimizer and uses the sparse categorical crossentropy loss function. Dropout is also applied to avoid overfitting. Even though the model itself performs excellent, our primary purpose to implemented this neural network is to explore model blending and stacking.

3.2.1 Network Topology

The topological structure of this network is a multi-layer dense neural network. It has an input layer with 2048 nodes, followed by five hidden layers with 1024, 512, 256, 128, and 64 hidden units respectively. Finally, it has an output layer with 10 nodes. The hidden layers use the ReLU activation function and the output layer uses the softmax activation function. The network also uses dropout regularization with a dropout rate of 0.5 for each hidden layer.

3.2.2 Dropout

Dropout is a regularization technique for reducing overfitting in neural networks. It works by randomly dropping out, or setting to zero, a specified number of output units in a layer during the training process. This forces the network to learn multiple independent representations of the same data by randomly dropping different units on each forward pass. This makes the network less reliant on any individual unit and reduces overfitting by preventing the network from becoming too complex and by promoting the development of a more diverse set of features.

Dropout is hard to implement from scratch. We tried in our first model, failed to get the correct implementation.

3.3 Model Blending

Model blending and stacking are ensemble learning methods used to combine the predictions of multiple models [6].

In model blending, the predictions of multiple base models are combined using a weighted average or a weighted sum. The weights are typically determined using cross-validation on the training data. It can improve the performance of the ensemble by leveraging the strengths of multiple models and by reducing the variance of the predictions.

3.3.1 Model Selection

In particular, we choose models to blend in the following way: for our best Keras model, we trained it three times. It's obvious that we won't get three models with identical parameters. Weights of the network are typically initialized randomly, and so the starting point for each training run will be different. This can cause the network to learn different patterns in the data and can lead to different final weights.

Another reason is that the training process involves stochastic gradient descent, which is a randomized optimization algorithm. This means that the update steps for the weights are not deterministic and will be different on each training run. Little interupt in the deep net could lead to a significant difference in the final parameters.

We also selected other three best non-deep learning classifiers that is shown to perform good on this dataset: XGBoost (Extreme Gradient Boosting), K-NN (K Nearest Neighbors), and Random forest [7].

3.3.2 Parameter Searching

Given y_{pred} and y_{label}, the predicted catagory and the actual catagory(both are a number from 1 to 10), we want to find the six scalar weights $w_1 ... w_6$ such that:

$$y_{pred} = argmax(w_1 y_1 + w_2 y_2 + w_3 y_3 + w_4 y_4 + w_5 y_5 + w_6 y_6)$$

is closest to the y_{label}. Here, $y_1 ... y_6$ are the individual 10 by 1 one-hot encoded prediction from the six models. This is like a "voting" between the six models.

For example, if current weights are $w_1 = 1, w_2 = 1, w_3 = 0, w_4 = 0, w_5 = 3, w_6 = 2$, and

$$y_1 = \begin{bmatrix}0\\0\\1\\0\\0\\0\\0\\0\\0\\0\end{bmatrix}, y_2 = \begin{bmatrix}0\\0\\0\\0\\0\\1\\0\\0\\0\\0\end{bmatrix}, y_3 = \begin{bmatrix}0\\0\\0\\1\\0\\0\\0\\0\\0\\0\end{bmatrix}, y_4 = \begin{bmatrix}0\\0\\1\\0\\0\\0\\0\\0\\0\\0\end{bmatrix}, y_5 = \begin{bmatrix}0\\0\\1\\0\\0\\0\\0\\0\\0\\0\end{bmatrix}, y_6 = \begin{bmatrix}0\\0\\0\\1\\0\\0\\0\\0\\0\\0\end{bmatrix}$$

Then $y_{pred} = \sum_{i=1}^{6} w_i y_i = 3$.

Our procedure to do model blending is:

- Load the six trained models we selected, output their prediction for test data.
- Assign initial weight range to be $[0, 5]$. Here we restrict weights to be an integer between 0 to 5.
- Using a external C++ program (see `Model_Blending/G.cpp`)to brute-force enumerate all possible combination of weights. For each weight combination, we could calculate a y_{pred} and measuring how close it is to y_{label}. Update global optimal weight parameters while searching.
- Expand our searching range by adding 1 to the upper bound. (i.e.$[0, 5]$ to $[0, 10]$). Go back to previous step. If the parameters won't further update after we increase the weight range, end the program and output the global optimal weights.

4 Result

4.1 Customed NN

For the first model, we obtained the best accuracy when using a neural network with four hidden layers and hidden unit sizes of [58,258,258,10]. This network used the ReLU activation function and achieved an accuracy of 87.74%.

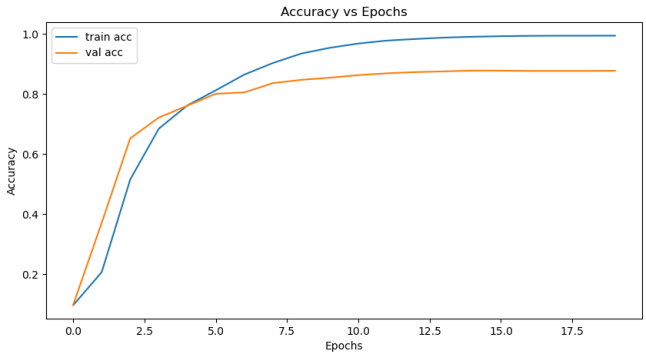

Figure 5: Accuracy for the first model

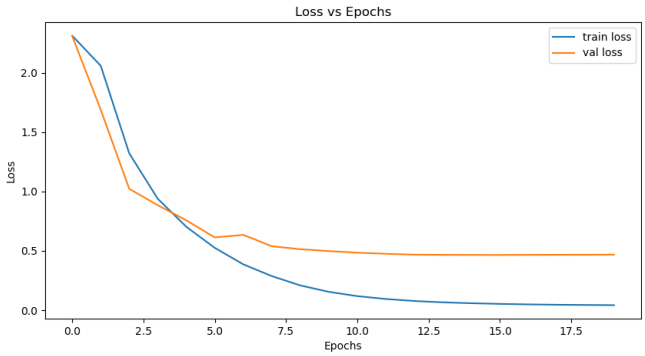

Figure 6: Loss for the first model

4.2 Keras NN

For the second model, we obtained the best accuracy when using a neural network with four hidden layers and hidden unit sizes of [58,2048,1024,512,256,128,64,10]. This network used the ReLU activation function , dropout rate 0.5. The achieved accuracy is 94.34%, corresponding loss is 0.5134.

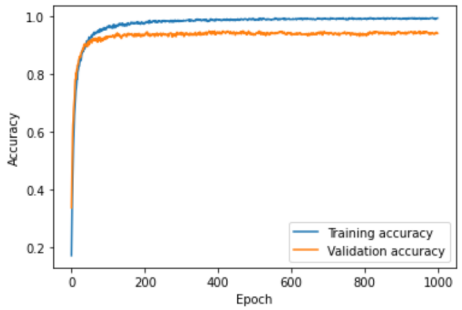

Figure 7: Accuracy for the second model

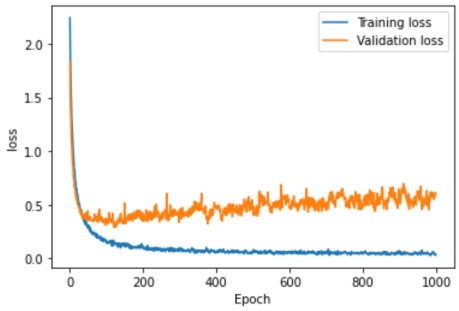

Figure 8: Loss for the second model

4.3 Blending Model

For the blending model, the best parameters we obtained are:

Three NN Models:

$$w_1 = 1, w_2 = 3, w_3 = 5$$

XGBoost:

$$w_4 = 1$$

Random Forest:

$$w_5 = 3$$

K-NN:

$$w_6 = 2$$

The accuracy for this model is 95.85%, loss 0.3255.

We alse collected other model's from Kaggle, as shown in the picture:

Model	Accuracy
Our	0.9585
NN	0.9284
XGB	0.8972
Random Forest	0.8222
KNN	0.8145
LSTM	0.8060
Support Vector Machine	0.7541
ResNet	0.7000
Logistic Regression	0.6743
Stochastic Gradient Descent	0.6537
Decision trees	0.6497
Inceptionv3	0.5455
Naive Bayes	0.5379

Figure 9: Accuracy of our model and other models from Kaggle

5 Conclusion

In this project, we applied neural networks to the task of music genre classification on the GTZAN dataset. We found that feature extraction was effective in dealing with audio data. The backpropagation algorithm was successfully implemented in the first model. However, we discovered that as the number of parameters increased, the model became more difficult to train. By using techniques such as batch norm and dropout, and choosing a better weight initialization method, we were able to improve the model's accuracy in the future. We also used the model blending method to further improve the performance of the classifier. These results demonstrate the effectiveness of using neural networks and model blending for music genre classification.

6 Team Contribution

In this study, Ruinan Ma was responsible for implementing the first neural network and performing the feature extraction on the dataset. Shang Zhou implemented the second neural network using Keras and model blending, and Jialin He was responsible for visualizing the dataset and compared our models to existing models on Kaggle. Ruinan Ma and Jialin He contributed to writing the report. Overall, the team worked well together to successfully complete the study and obtain promising results.

7 Reference

References

[1] Tzanetakis, G. and Cook, P. "Musical Genre Classification of Audio Signals." *IEEE Transactions on Speech and Audio Processing*, vol. 10, no. 5, pp. 293-302, 2002.

[2] Padmavathi, D. "Let's Tune the Music with CNN XGBoost", 2022. *Kaggle*. Available: https://www.kaggle.com/code/aishwarya2210/let-s-tune-the-music-with-cnn-xgboost. [Accessed: 02- Dec- 2022].

[3] Olteanu, A. "Work w/ Audio Data: Visualise, Classify, Recommend", 2019. *Kaggle*. Available: https://www.kaggle.com/code/andradaolteanu/work-w-audio-data-visualise-classify-recommend. [Accessed: 02- Dec- 2022].

[4] Olteanu, A. ”GTZAN Dataset - Music Genre Classification”, 2019. *Kaggle*. Available: https://www.kaggle.com/datasets/andradaolteanu/gtzan-dataset-music-genre-classification?resource=download-directory. [Accessed: 12- Nov- 2022].

[5] Lewis, C. ”How to Create & Understand Mel-Spectrograms”, 2021. *Medium*. Available: https://importchris.medium.com/how-to-create-understand-mel-spectrograms-ff7634991056. [Accessed: 15- Nov- 2022].

[6] Sharma, S. ”Music Genre Classification using CNN”, 2021. *Kaggle*. Available: https://www.kaggle.com/code/soumyasharma20/music-genre-classification-using-cnn. [Accessed: 29- Nov- 2022].

[7] Senkal B. ”Audio Data: EDA, Processing, Modeling & Recommend”, 2022. *Kaggle*. Available: https://www.kaggle.com/code/psycon/audio-data-eda-processing-modeling-recommend. [Accessed: 30- Nov- 2022].

[8] Assistant. ”Reply to question”, 2022. *OpenAi*. Available: https://openai.com/blog/language-models/. [Accessed: 09- Dec- 2022].