Classifying Music Genre Using Deep Learning Methods

Ryan Bushman, Hyrum Hansen, and Landon McNamara

December 2021

Task 1

Introduction

The purpose of this task is to classify music genre using different types of data. We will be working with numerical data and image data. There are 10 genres that we are trying to classify: blues, classical, country, disco, hip hop, jazz, metal, pop, reggae, and rock.

Data

This task included work with four datasets:

- genres_original: 1000 audio files, 100 per genre, that are 30 seconds in length. Note that one audio file in jazz genre was corrupted and had to be removed.
- features_30_sec.csv: 999 observations of features extracted from 30 second audio files from the genres_original dataset.
- features_3_sec.csv: 9990 observations of features extracted from 3 second audio files contained in the genres_original dataset. This dataset takes the 30 second audio files and splits them into 3 second audio files in order to have more data to work with.
- images_original: Mel spectrogram images for the 10 different genres. There were 100 mel spectrogram images for each genre class with the exception of the Jazz class which had 99 images.

Methods and Results

1. Neural Nets

A feedforward neural network was trained on the features_30_sec dataset using the **pytorch** python framework. Design choices include the following:

- L1 and L2 regularization
- 5 fully connected layers with 500 neurons per hidden layer
- Dropout rate of 0.05
- The Adam optimizer algorithm with a learning rate of 0.001.
- ReLU activation function
- Used the negative log likelihood loss function (NLLLoss)

A grid search was performed to tune the hyperparameters including: dropout rate, learning rate, etc. After 100 epochs, the network's test accuracy was **75**%. After 300 epochs, the network's test accuracy improved to **76**%. A dataset that contains more than 999 observations would likely lead to increased classification accuracy. sklearn.metrics, sklearn.model_selection, sklearn.preprocessing, torchvision, torch, and the skorch neural net classifier were used for this problem.

A feedforward nerual network was trained on the features_3_sec dataset using the **tensorflow** python framework. A grid search was performed to select optimal hyperparameters. Design choices include the following:

- L2 regularization
- 3 fully connected layers with 750 neurons per layer
- The Adam optimization algorithm with a learning rate of 0.0001
- ReLU activation

This network outperformed the others. After 600 epochs, the network's training accuracy was **99.91**% and its validation accuracy was **92.06**%. The model had a total of 1,178,260 trainable parameters.

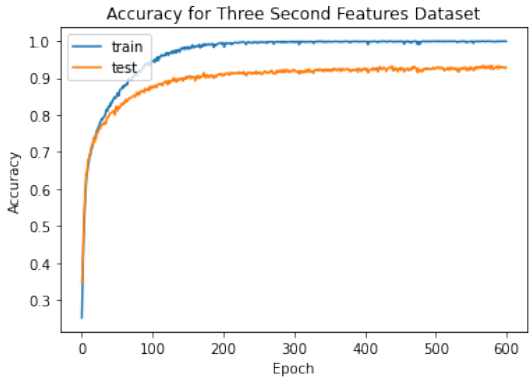


Figure 1: Validation accuracy flat-lines around 250 epochs, but further training gave marginal accuracy improvements.

A diagram of the feedforward network is given in the following figure.

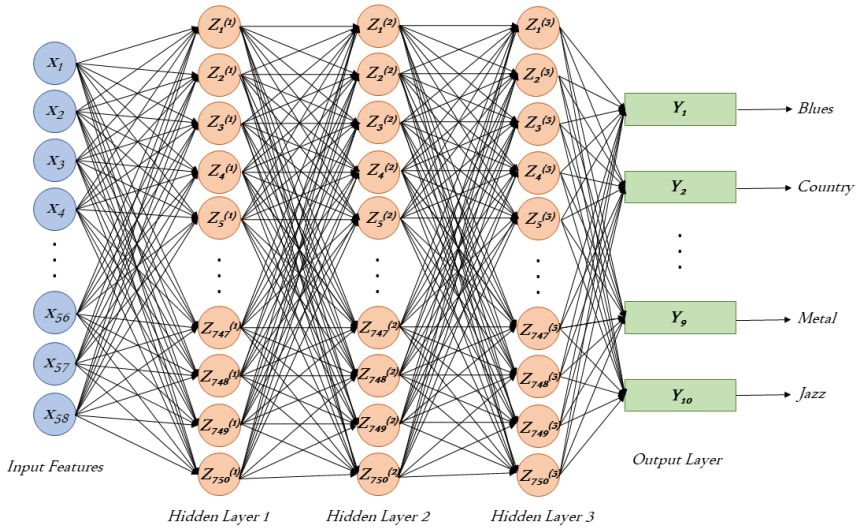


Figure 2: Three hidden layers with 750 neurons each resulted in fairly good validation accuracy

2. CNN

We designed and trained two convolutional neural networks (CNN) to classify music genres based on Mel spectrogram images. The first CNN we designed had the following architecture:

- Two convolutional layers each followed by a max pooling layer; followed by five feed-forward layers
- The first hidden layer had 2048 neurons, the second hidden layer had 512 neurons, and the third hidden layer had 128 neurons
- ReLU activation functions and L2 regularization
- Dropout rate of 0.4
- The Adam optimizer algorithm with a learning rate of 0.0001
- Used the negative log likelihood loss function (NLLLoss)

A gird search was performed to tune a few of the hyperparameters (learning rate, dropout rate, etc.). After determining the best parameters and training for 50 epochs the CNN's test accuracy was **40.70**%. The intent for this first CNN was to create a baseline CNN to compare our second CNN to.

The second CNN we designed used transfer learning from AlexNet. We made two architectural changes to AlexNet. The first was to reduce the number of neurons in the last hidden layer from 4096 to 1024. The second was to reduce the number of neurons/classes of the last layer from 1000 to 10. The design choices include the following:

- Unfreezing and retraining of all of AlexNet's weights and biases.
- L2 regularization
- The Adam optimizer algorithm with a learning rate of 0.00001.
- The CrossEntropyLoss loss function was used as required by AlexNet.

The CNN performed best when all the layers were unfrozen compared to just having the feedforward layers unfrozen. A grid search was performed to tune the learning rate and L2 regularization hyperparameters. After training for 25 epochs the CNN's test accuracy was **77.89**%. While tuning the network we noticed that the performance of the CNN depended heavily on the random initialization. The test accuracy could change as much as 10% depending on the random initialization. Although the model depended heavily on the initialization we do know that it was learning because it performed better than the baseline CNN we designed. Using transfer learning helped us to improve our test accuracy from 40% to 77%. Since we know that the model was learning we can infer that it is challenging to classify music genres based on Mel spectrogram images.

The CNN with transfer learning had similar accuracy to the neural net in the 'Neural Nets' section above that used the features_30_sec dataset. Since the accuracy of the CNN with transfer learning was not significantly higher than the neural net from the previous section we do not see any advantage to using one architecture over the other.

CNN Classification Confusion Matrix

	Predicted →										
Labeled ↓	Blues	Classical	Country	Disco	Hip Hop	Jazz	Metal	Pop	Reggae	Rock	**Correct**
Blues	16	0	1	1	0	1	1	0	0	1	**76.19%**
Classical	0	12	0	0	0	0	0	0	0	0	**100%**
Country	1	1	14	1	0	1	0	1	1	4	**58.33%**
Disco	0	0	0	18	1	0	0	2	0	1	**81.82%**
Hip Hop	0	0	0	0	13	0	0	1	0	0	**92.86%**
Jazz	0	1	1	0	0	25	0	0	0	0	**92.59%**
Metal	0	0	0	0	0	0	18	0	0	0	**100%**
Pop	0	2	1	0	1	0	0	12	2	1	**63.16%**
Reggae	1	0	0	1	1	2	0	2	15	0	**68.18%**
Rock	1	0	1	4	0	0	1	0	1	12	**63.16%**

From the confusion matrix above we can see that the CNN excelled in classifying 'Classical' and 'Metal'. We also see that the CNN struggled to classify 'Country', 'Pop', 'Reggae', and 'Rock'. We suppose this may be due to some level of overlap within these genres. For example it would be possible for one of the songs selected from this dataset to be considered country/pop or reggae/rock making it more difficult to label. This kind of crossover could make the Mel spectrogram images less distinct between genres and thus make classification more challenging.

Both CNNs were designed using the **pytorch** python framework and were trained using the skorch library. The numpy, torchvision, and sklearn libraries were also used.

3. Feature Extraction

Features were extracted from the original audio data using methods found in the **librosa** module. The following features were extracted:

- **Total Zero Crossings:** A count of the number of times voltage of a signal changes sign. This value can help determine if human speech is present in an audio clip.

Figure 3: An arbitrary blues song segment for visualization. In 100 time steps voltage switches signs 11 times. The extracted feature is the sum of all such sign changes.

- **Spectral Centroid:** A weighted mean of the frequencies indicating the spectrum's center of mass. The value can be interpreted as "brightness," providing a notion of musical tambre.
- **Spectral Rolloff:** 85% of the total spectral energy is equal to or less than this frequency value. Used to distinguish between harmonic and dissonant sounds.
- **Mel Frequency Cepstral Coefficients:** We extracted 20 mfcc's. These were derived using the following algorithm:
  1. Compute the Fourier transform of a segment of the signal.
  2. Map the powers of the resultant signal onto the mel scale, which can be thought of as a mapping of frequencies (Hz) to a scale defined by equally spaced pitches via a logarithmic transformation (measured in mels).
  3. Log the powers at relevent mel frequencies.
  4. Take the discrete cosine transform and the resultant values comprise the mfcc's.

  Note that the algorithm was **not** implemented by hand. The librosa implementation of the algorithm was used.

The model trained on extracted features did not perform well. After 600 epochs the testing accuracy was only 63%. This was due in part to the limited amount of data – there were only 999 observations. The following figure visualizes the training and testing accuracies over epochs.

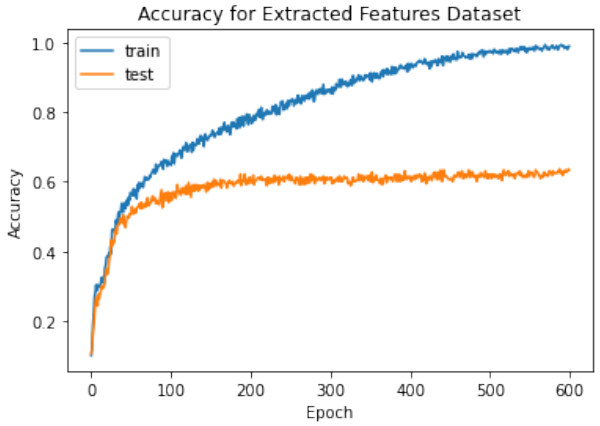


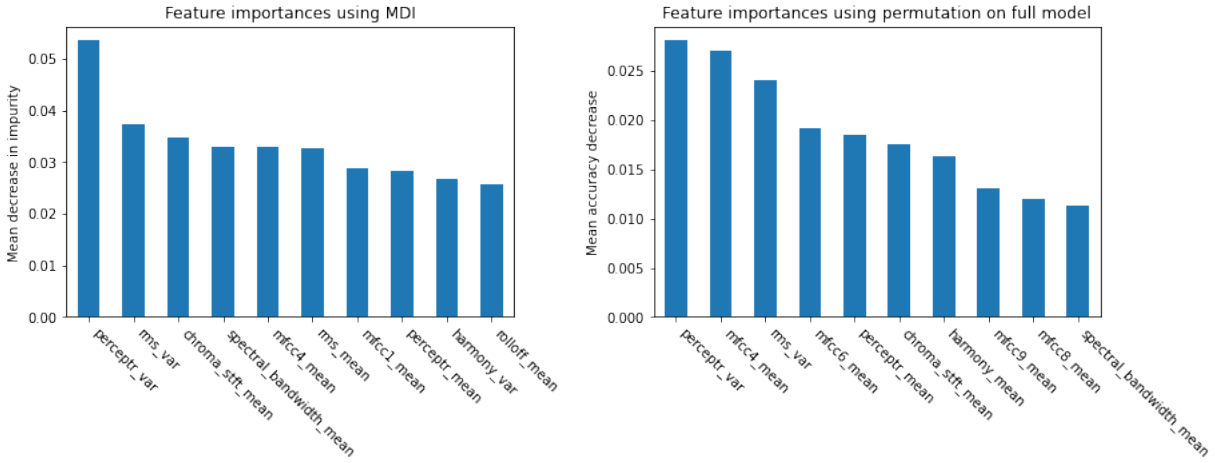
Figure 4: The validation accuracy flat-lines after around 300 epochs.

4. Baseline

As baseline models for comparison's sake we fit a Random Forests Model and a Support Vector Machine (SVM) model to the features_3_sec dataset.

Random Forest: For this model we tuned the following parameters: n_estimators, max_depth, bootstrap, max_features, min_samples_leaf, and min_samples_split. After extensive tuning using a grid search we achieved about 87% accuracy with the following parameter settings: n_estimators = 800, max_depth = 100, bootstrap = False, max_features = auto, min_samples_leaf = 1, and min_samples_split = 2. To achieve these results we did 5-fold cross validation. Using the test dataset we achieved about 89% accuracy.

Below are two variable importance plots using different methods and built on the Random Forest model. They have been set to show only the top 10 most important features. One is based on mean decrease in impurity while the other is based on feature permutation. Using mean decrease in impurity can tend overlook binary or categorical features. None of the features in our dataset are binary or categorical so that should not be an issue. However, we included the permutation importance plot for comparison's sake. Both methods agreed that perceptr_var is the most important feature. The two methods also agreed that rms_var, chroma_stft_mean, spectral_bandwidth_mean, mfcc4_mean, and perceptr_mean are within the top 10 most important features. Though they did not assign them in the same order of importance.



SVM: For this model we tuned the following parameters: C, gamma, kernel, degree, max_iter, and scale. After tuning the model using a grid search, the best results we were able to achieve was about 41% accuracy with the following parameter settings: C = 10000 and kernel = rbf with the other parameters excluded or set to default. We used 5-fold cross validation to achieve these results. We also achieved 41% accuracy on using the test dataset. Despite using gridsearch to tune many values for many parameters, we were not able to get a very high accuracy

with the SVM. After meeting with Dr. Moon and discussing the issues we were having with the SVM model, he suggested the SVM algorithm may just perform poorly on this data.

These models were built primarily using functions from the sklearn.ensemble, sklearn.svm, and sklearn.model_selection packages.

Conclusion

The greatest challenges we experienced while working on this involved insufficient data and preprocessing hurdles. Specifically, our lack of experience creating a labeled image dataset for the convolutional neural network made preprocessing difficult; however, using online resources and the assistance we received in office hours we were able to resolve this issue. The problems with insufficient data (namely the features_3_sec dataset) were less easily overcome. 999 observations is a very small dataset for deep learning, especially when part of the dataset must be separated into a testing set.

If we had more time to work on this project, we would have liked to experiment with data augmentation for the mel spectrogram images. It may be beneficial to slightly alter the images generating noisy replicates, resulting in a dataset with a couple thousand images. Given the structure of a mel spectrogram image, adding noise may make the image indistinguishable as its intended music genre; however, it would have been a fun experiment to try. For feature extraction, it would likely have been beneficial to split the 30 second clips into 10 3-second partitions. Further research would include the extraction of additional features as well.

After creating our models, we found that a feedforward network built on the dataset with 9990 observations of extracted features performed the best with a validation accuracy of **92.06**%. This model performed better than our baseline models. The random forest model had validation accuracy of around 89% (which is surprisingly good for this type of data) and the SVM had a validation accuracy of about 41%.

Contributions

- Problem 1.1: Landon McNamara
- Problem 1.2: Hyrum Hansen
- Problem 1.3: Landon McNamara & Ryan Bushman
- Problem 1.4: Hyrum Hansen
- Problem 1.5: Ryan Bushman

Task 2

Introduction

The purpose of this task is to forecast PM2.5 air pollution in Beijing, China based on multiple weather and atmospheric measurements.

Data

This task involved working with the Beijing PM2.5 dataset. This dataset contained 43,824 observations and 13 features of which four were date and time related. There were also eight features that were pollution, weather, and atmospheric measurements. The feature we were interested in forecasting was the PM2.5 measurement.

We cleaned the data by removing observations that contained missing values. Since this is a time series dataset the data was not randomly split into training and testing datasets. The first 70% of the data were assigned to the training dataset and the final 30% of the data were assigned to the testing dataset.

Methods and Results

Due to the time series nature of the data we chose to use a recurrent neural network (RNN) model. The model design choices included the following:

- Input size is 11 features
- Hidden size is 2
- One LSTM layer
- Predicting to 1 feature
- The Adam optimizer algorithm with a learning rate of 0.01
- The loss function selected is MSELoss

After training for 301 epochs the model's training loss was **0.00012** and its test loss was **0.0054**.

A RNN with only 6 features was also trained to see if a simpler model would perform as well. In Figure 5 (left) below, we can see that the model will 11 features predicted PM2.5 pollution levels very closely to the actual values for both the training and testing sets. Also from Figure 5 (right) we can see that the model with 6 features does not predict PM2.5 very well. It fails to capture the overall trends of the actual PM2.5 pollution levels.

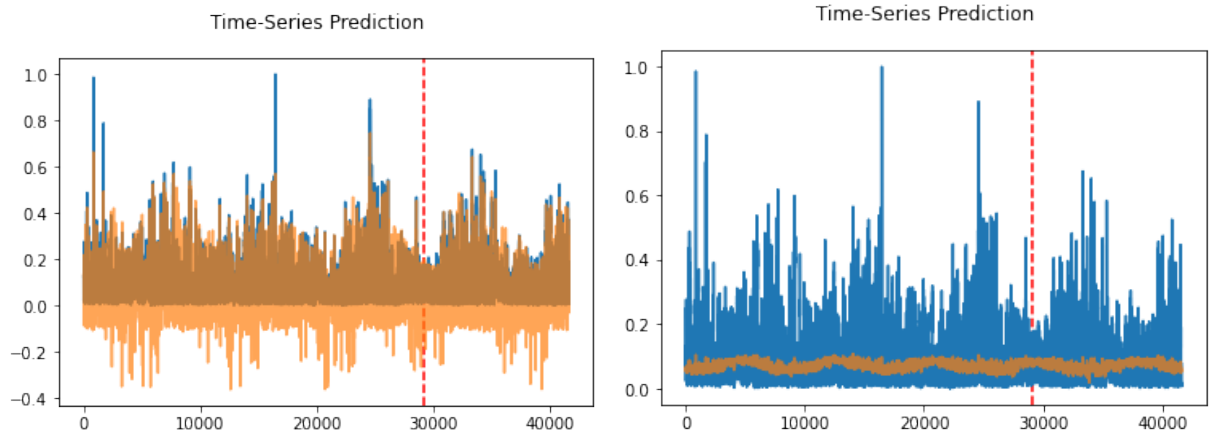


Figure 5: Predictions of PM2.5 in orange with actual in blue. The dotted red line is the transition from training data to test data. Left: Model with 11 features. Right: Model with 6 features.

The **pytorch** python framework was used to design the RNN. Packages and libraries used: numpy, pandas, matplotlib, datetime, sklearn preprocessing, and torch.

Conclusion

The greatest challenges we experienced while working on this problem were due to issues with data preprocessing and understanding the RNN architecture. The problem we encountered during data preprocessing was creating training and test datasets that had the correct dimensions for the RNN. As this was our first time working with RNN models our understanding of RNN design was lacking, which limited us to simpler RNN designs. However, using online resources and assistance we received in office hours we were able to overcome these problems. If we had more time to work on this project, we would have liked to spend more time on understanding RNNs to see if we could alter the architecture to improve the model's predictive capabilities. We also would have liked to have found a way to set a lower limit for the predicted PM2.5. The RNN as it is now predicts negative amounts of PM2.5 pollution but in reality the prediction should not be able to go lower than zero. If we were able to implement this lower limit we might have seen improved predictions. Based on the fact that our test dataset loss was 0.0054 and that visually (Figure 5 Left) the RNN's predicted PM2.5 pollution levels overall followed the actual PM2.5 pollution levels closely, we can say that the model performed very well.

Contributions

Task 2: Landon McNamara & Ryan Bushman