Deep Learning Assignment 2: Music Genre Classification

Emiel Miedema (p317569)
Matthijs Jongbloed (s4357833)
Hylke Westerdijk (s5423937)

February 2, 2025

Abstract

This report describes deep learning approaches for music genre classification using numerical features extracted from audio fragments of the GTZAN dataset. Two architectures, a feedforward neural network (FFNN) and a convolutional neural network (CNN), were developed, trained, and compared. The feature extraction pipeline uses methods such as mel-frequency cepstral coefficients, chroma features, and spectral descriptors to capture the characteristics of songs. The FFNN achieved a classification accuracy of 92.29%, which outperformed the CNN's 90.55% accuracy. The study shows the value of feature extraction combined with deep learning techniques for automated music genre classification.

1 Introduction

This report presents the development and training process of deep learning models for music classification. Music classification is an application that uses deep learning to analyze music and generate insights about these tracks. This specific project focuses on categorizing music in its respective genre based on the characteristics of the song. This task is widely used and plays a role in recommendation systems (Fessahaye et al., 2019; Schedl, 2019), streaming services, and music libraries. With the growing demand for personalized music experiences, automated music classification has become an increasingly popular topic of investigation. Before the availability of advanced machine learning models, music classification relied heavily on humans to label music manually, which was both labour-intensive and also prone to inconsistencies. It is important to note that categorization into music genres could be subjective and depends on the listener's knowledge and musical preferences. However, to some extent, a collective agreement exists on the definition of particular genres. These collective definitions are instrumental for the formation of one's musical preferences. Effective automation of the classification of music pieces into genres has great potential to help people discover new music they are likely to enjoy.

The complex temporal structure of music makes automatic classification into genres a non-trivial task. By plotting the frequency components of music over time, one can create spectrograms of raw audio data. In such spectrograms, one may be able to distinguish genre-specific features such as rhythm or harmonic texture. However, the complex geometry of spectrograms is challenging to interpret. The impressive capability of CNNs to capture complex visual patterns makes them a promising architecture for music genre classification.

Humans classify music into genres using several criteria. Among those are components such as rhythm, tempo, melodic progression, harmonic texture, instrumentation and dynamics. For example, genres like classical, jazz, and rock each have distinct sound colors or timbres based on their instrumentation and production styles. However, these components are not present in the raw audio itself. Audio signals are continuous waveforms that contain a mixture of sound frequencies, amplitudes, and time-varying patterns. Our ability to identify melodic or harmonic components in music is made possible by the

biological complexity of our ears and brains. Despite advances in deep learning, genre classification based on raw audio signals remains challenging. Audio signals are inherently complex and unstructured. They do not directly map to easily interpretable features like those found in text or images. For this reason for automated genre classification, feature extraction methods have to be chosen that in a sense mimic the biological mechanisms that allow us to differentiate genres. The methods used in this project are inspired by the work of Pandey (2021).

A frequently used method for feature extraction in music genre classification tasks uses the Mel scale (Stevens et al., 1937). The Mel scale is a pitch perceptual scale that is based on human auditory perception, designed to reflect how listeners subjectively judge the relative heights of different pitches. Unlike the linear frequency scale (measured in hertz), which is based on physical properties of sound waves, the Mel scale is nonlinear and approximates the human ear's response to different frequencies. It was constructed through experiments where listeners were asked to compare and fractionate pitches, resulting in a scale where a 1000 Hz tone at 60 dB is defined as 1000 mels. For a linear frequency scale, one can perform a Fast Fourier Transform (FFT) in order to analyze the signal in the frequency domain, resulting in a spectrogram. A Mel-scale spectrogram can be made by applying the Mel filter bank to the spectrogram. This captures the energy distribution across frequency bands based on the Mel scale. Figure 1 depicts a Mel spectrogram comparison between a classical and a hiphop music fragment. To reduce redundancy, features can be extracted from a Mel spectrogram called the Mel-frequency cepstral coefficient (MFCC). MFCCs provide a compact and discriminative representation of the audio. For this reason, it is one of the key feature extraction methods used in this project.

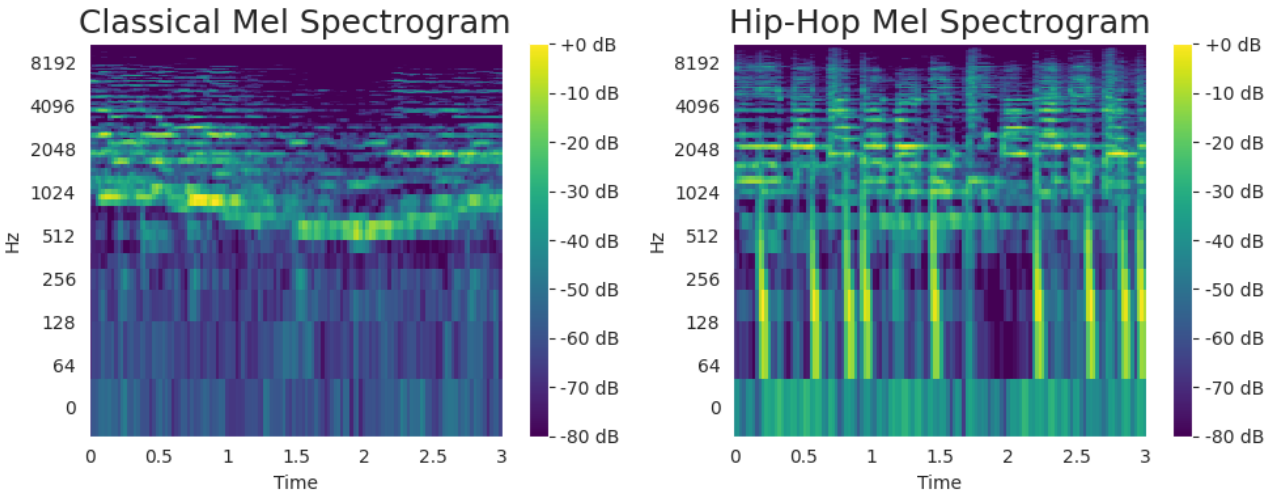

Figure 1: Comparison of of Mel Spectrograms of Classical and Hip-Hop Fragments

The *Librosa* library lists several other feature extraction methods that are frequently used when analyzing audio data. McFee et al. notes that while the Mel scale is a good representation of the timbre of music, it does not offer a good resolution of pitch classes (chroma). Using short-time Fourier transform (STFT) chroma are constructed that represent the 12 pitch classes (corresponding to the notes of the musical scale) over time. This representation particularly excels in capturing harmony, while it is less sensitive to variation in octave height, timbre and loudness.

We use several other methods of characterizing the frequency spectrum in encode components like the aforementioned octave height, timbre and loudness. The root-mean-square 'energy' or amplitude effectely captures the signal's loudness. The spectral centroid reflects the perceived brightness or "center" of the frequency spectrum. Spectral Bandwidth measures the spread of frequencies, indicating how "sharp" or "smooth" the sound is. Spectral rolloff identifies the frequency below which a specified

percentage of the total energy lies. This can be used to distinguish harmonic (periodic) parts of the signal from noise. We also use a representation of the harmonics present in the track. Harmonics are multiples of a fundamental frequency (higher octaves) that contribute to the tonal quality of a sound, giving it its timbre. Lastly we use several non-harmonic features: The zero crossing rate quantifies how frequently the signal changes polarity, providing information about the noisiness of the sound. Spectral contrast is a metric that is able to capture rhythmic or percussive elements of audio: Percussive sounds lead to intense energy peaks in spectrograms, resulting in high contrast values, while sustained sounds (like vocals or strings) tend to have smoother spectra with lower contrast. Closely related these, is the tempo of music. All listed features provide information about the harmonic and rhythmic components of music that can be used for effective music genre classification.

This article aims to investigate the design principles for a successful music genre classification pipeline. In this project, we use two models: convolutional neural networks (CNNs), feedforward neural networks (FFNNs). CNNs famously excel at machine vision tasks and extracting spatial features, whereas FFNNs excel at identifying generalized patterns. In this project, the GTZAN dataset (Tzanetakis & Cook, 2002), a widely used benchmark in music classification research, is used. It consists of 1000 audio tracks evenly distributed across ten genres, each 30 seconds long. Using this dataset, we aim to explore different methods of how feature extraction techniques and two types of neural network architectures can be combined to address the task of music classification.

The full implementation is available online on our GitHub Repository

2 Methodology

2.1 Data

The dataset used in this study is the GTZAN Music Genre Dataset, which is commonly used for research in music genre classification. The dataset consists of 1,000 audio tracks divided into 10 genre categories, with 100 tracks per genre. The genres included are: Blues, Classical, Country, Disco, Hip-hop, Jazz, Metal, Pop, Reggae, and Rock.

Each track is in 16-bit raw WAV format and has a duration of 30 seconds, with a sampling rate of 22,050 Hz and mono channel (single audio channel). The complete dataset has a size of 1.2 gigabytes.

The features extracted from the tracks are mostly invariable with respect to the duration of the audio fragment. This is because the previously discussed metrics are averaged over each track. This processing of averaging leads to information loss. For this reason, we split each track into 10 non-overlapping sub-fragments of 3 seconds long that are treated as independent samples. This means both that the classifier will be trained on a wider variety of acoustic patterns that are present within subsections of the tracks and that the amount of training data is effectively increased 10-fold. One of the audio fragments is plotted in Figure 2.

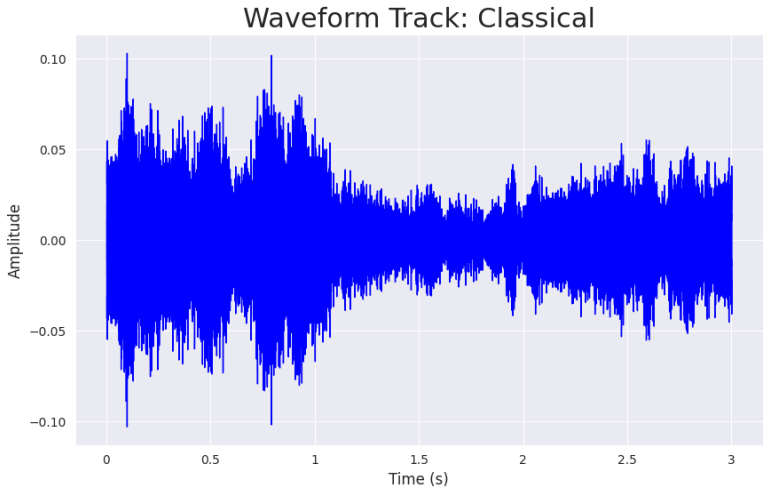

Figure 2: Waveform of the 3 second audio fragment

2.2 Feature extraction

Extracting the relevant features from the tracks was done using the *Librosa* python library (McFee et al., 2015). *Librosa* is a widely used toolset enabling the extraction of acoustic components in order represent audio signals in a meaningful way.

The extraction process of the features listed in the previous section is depicted in Figure 3. Note that 20 MFCCs are computed, each representing a specific frequency range's contribution to the signal. The MFCCs, Chroma STFT, RMS energy, zero crossing rate, spectral centroid, spectral bandwidth, spectral rolloff, harmonics, percussives and tempo components, in total make up 29 features for each sample of the audio signal. The mean and the variance of each of these features are taken which results in the final feature vector $x \in \mathbb{R}^{58}$.

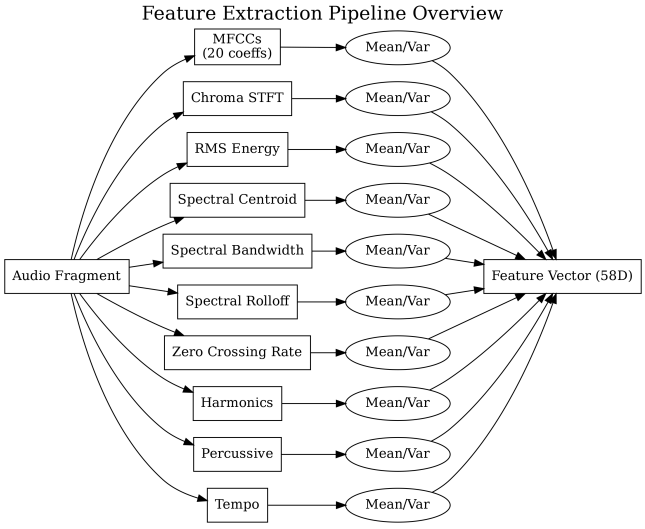

Figure 3: Feature extraction pipeline for audio classification. The pipeline begins with a raw 3-second audio fragment, and progresses through various feature extraction. The final output is a 58-dimensional feature vector.

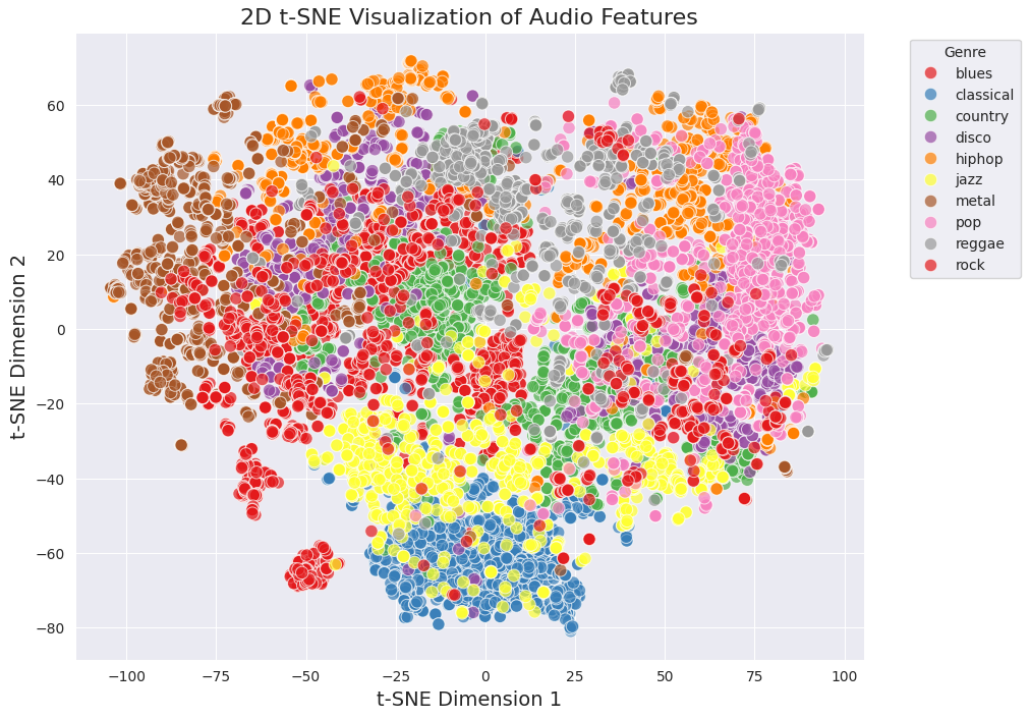

Figure 4: t-SNE Visualization of Audio Features. The plot represents the high-dimensional audio features reduced to two dimensions, with each point colored by its corresponding class label.

In order to visualize the separation between different music genres, we applied a dimensionality reduction process using t-SNE (t-distributed Stochastic Neighbor Embedding). The resulting 2D scatter plot, shown in Figure 4, reveals moderately well defined clusters of music genres, with each point representing an audio sample and colored according to its genre label.

We applied the K-Means clustering algorithm with $K = 10$, corresponding to the number of genres in the dataset. With a Silhouette score of 0.35, the reduced-dimensionality data shows clusters of moderate quality.

The original dataset provided a CSV file with precomputed feature extractions, which was initially used for hyperparameter tuning of our models. After determining the optimal hyperparameters, we implemented the feature extraction process ourselves and retrained our models from scratch using the extracted dataset.

2.3 Preprocessing

Feature and Target Separation
The dataset was divided into features (X) and target labels (y). The features comprised all columns except the column with the music genres, which are the target labels.

Label Encoding
The target labels, being categorical, were encoded into numerical values using `LabelEncoder` to facilitate model training.

Dataset Splitting
The dataset was split into training (70%) and testing (30%) sets using a split with stratified sampling to preserve class distribution. A test size of 30% ensured adequate data for evaluation while leaving sufficient data for training.

Feature Scaling
To normalize the feature space, the standard scaler was applied. The scaler was fitted on the training and test sets separately.

2.4 Models

Both of the models used for this project are implemented using the Tensorflow/Keras API (Chollet et al., 2015).

2.4.1 Feed-Forward Neural network

The FFNN was designed with a dense architecture consisting of six fully connected layers and dropout regularization layers to mitigate overfitting. The architecture is as follows:

Input Layer

The input layer takes a feature vector of shape $\mathbf{x} \in \mathbb{R}^{58}$, where 58 is the number of features in the dataset.

Hidden Layers

The network comprises five hidden layers with decreasing numbers of units. Each hidden layer uses the Rectified Linear Unit (ReLU) activation function, defined as:

$$\text{ReLU}(z) = \max(0, z),$$

where z is the weighted sum of inputs. The layers are configured as follows:

- Fully connected dense layers: 512, 256, 128, 64, and 32 units, each followed by a dropout layer with a rate of 0.4.

Dropout regularization is applied to mitigate overfitting by randomly deactivating 40% of the neurons during training.

Output Layer

The output layer is a fully connected layer with 10 units, corresponding to the 10 classes in the dataset. The softmax activation function is used to produce a probability distribution over the classes:

$$\text{softmax}(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{10} e^{z_j}},$$

where z_i is the logit for the i-th class.

2.4.2 Convolutional Neural Network

The model consists of an input layer, multiple 1D convolutional layers with batch normalization and max-pooling, dropout regularization, and a fully connected output layer. The architecture is described as follows:

Input Layer

As is the case for the FFNN, the input layer takes a feature vector of shape $\mathbf{x} \in \mathbb{R}^{58}$. The input is reshaped into a 2D tensor of shape $(58, 1)$ to facilitate 1D convolutional operations.

Convolutional Layers

The network comprises two 1D convolutional layers, each followed by batch normalization, max-pooling, and dropout regularization. The layers are configured as follows:

- A 1D convolutional layer with 32 filters, a kernel size of 3, and ReLU activation. This is followed by batch normalization, max-pooling with a pool size of 2, and dropout with a rate of 0.3.
- A second 1D convolutional layer with 64 filters, a kernel size of 3, and ReLU activation. This is also followed by batch normalization, max-pooling with a pool size of 2, and dropout with a rate of 0.3.

Output Layers

The output of the final convolutional layer is flattened into a 1D vector to prepare it for the fully connected layers. The network includes a fully connected layer with 128 units and ReLU activation, followed by dropout regularization with a rate of 0.2. The output layer is a fully connected layer with 10 units, corresponding to the 10 classes in the dataset. The softmax activation function is used to produce a probability distribution over the classes.

2.5 Training and Hyperparameter Tuning

For hyperparameter tuning, we initially used the feature-extracted CSV provided in the original dataset (*features_30_sec.csv*). This was because we encountered challenges in implementing the feature extraction process and only completed it later. However, the features in both files are identical, and their values are very close. Once the optimal hyperparameters were identified, we applied our own feature extraction process and used the resulting dataset (*features_3_sec_extracted.csv*) to retrain the final models.

2.5.1 FFNN

Hyperparameter tuning
Hyperparameter optimization was conducted using K-fold cross-validation (5-folds) using the `RandomizedSearchCV` class with 30 iterations due to memory constraints. Models were evaluated with respect to their accuracy on the validation set. The following hyperparameters were tuned:

- Batch size: {64, 128, 256, 512}
- Epochs: {100, 200, 300}
- Dropout rate: {0.2, 0.3, 0.4}
- Optimizer: {'adam', 'sgd', 'rmsprop'}

Tuning results
Randomized search showed that the optimal model uses the Adam optimizer for gradient-based optimization. Sparse categorical cross-entropy was used as a loss function. Training was conducted over 300 epochs with a batch size of 256.

Evaluation
The FFNN, optimized through hyperparameter search, was retrained on our own feature-extracted dataset using the entire training set and evaluated on the test set.

2.5.2 CNN

Hyperparameter tuning
Hyperparameter optimization was conducted using 5-fold cross-validation with the `RandomizedSearchCV` class, performing 30 iterations due to memory constraints. Models were evaluated based on their accuracy on the validation set. The search space included the following hyperparameters:

- Batch size: {64, 128, 256, 512}
- Epochs: {100, 200, 300}
- Dropout rate: {0.1, 0.3, 0.5}
- Optimizer: {'adam', 'sgd', 'rmsprop'}

Early stopping, monitored on validation loss with a patience of 5 epochs, was employed to avoid overfitting and reduce training time.

Tuning results
Randomized search showed that the optimal CCN-based model is identical to the FFNN with respect to the optimizer and loss function. Training was performed using batches of size 64 over 300 epochs.

Evaluation
The CNN, optimized through hyperparameter search, was retrained on our own feature-extracted dataset using the entire training set and evaluated on the test set.

3 Results

This section presents the performance evaluation of the implemented Feedforward Neural Network (FFNN) and Convolutional Neural Network (CNN) models for music genre classification.

The FFNN achieved an accuracy of 92.29%, whereas the CNN achieved an accuracy of 90.55%. Both models significantly outperformed the state-of-the-art result of 83.30% accuracy reported in the 2020 study "Convolutional Neural Networks Approach for Music Genre Classification". While the 2020 study used Mel-spectrograms as input features, our work uses a 3-second numerical dataset derived from audio and transformed into multiple extracted features. (Cheng et al., 2020)

3.1 Training Loss and Accuracy

The training loss and accuracy curves for both models are presented in Figure 5.

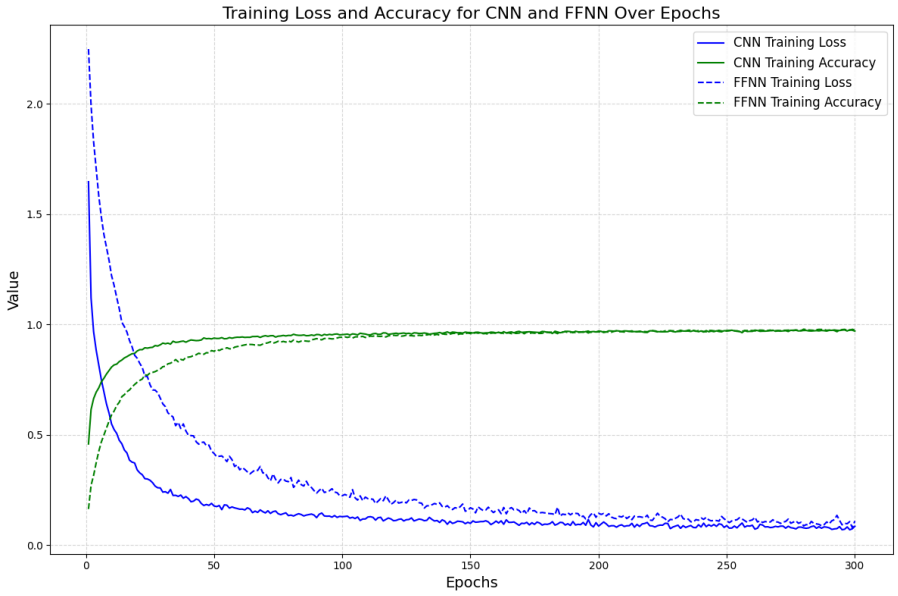

Figure 5: Training Loss and Accuracy of FFNN and CNN Models Over Epochs

The CNN showed faster convergence and reached a stable accuracy plateau earlier in training compared to the FFNN. While both models achieved similar training accuracy, the CNN displayed a lower overall training loss. Considering the rate at which the CNN converges and its overall training loss, it may be potentially slightly overfitting due to the large number of epochs (although, EarlyStopping should prevent this). In contrast, the FFNN required more time to converge, but the additional epochs were beneficial, ultimately resulting in a slightly better performing model compared to the CNN.

3.2 Misclassification Analysis

The confusion matrices (Figure 6) and misclassification count graphs (Figure 7) provide detailed insights into the classification performance for each genre.

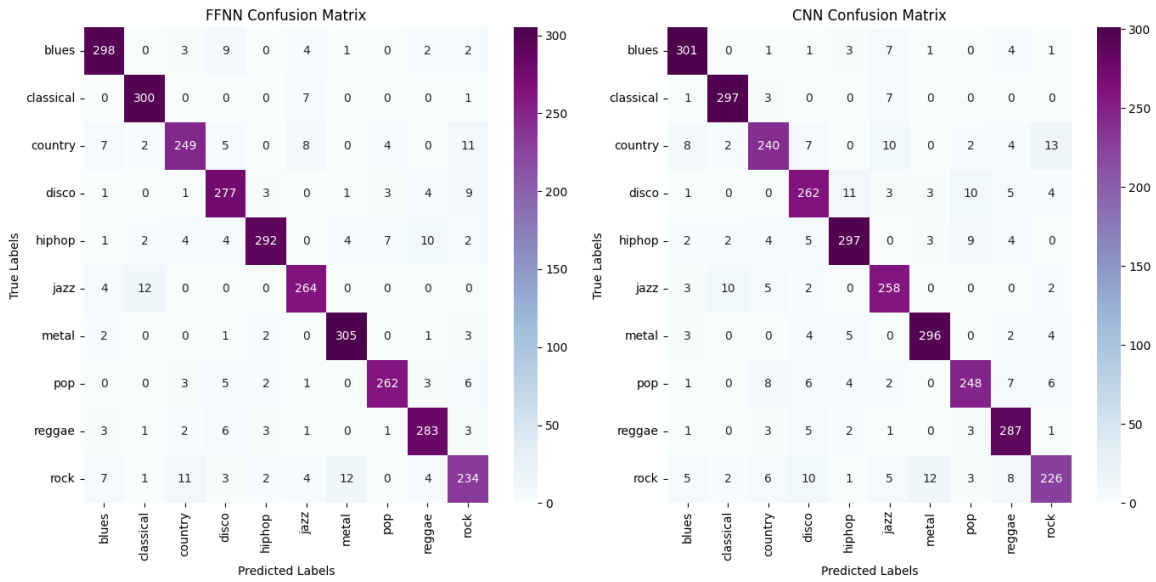

Figure 6: Confusion Matrices Comparing FFNN and CNN Models

The FFNN excelled in classifying "classical" and "metal" genres, achieving near-perfect results with very few misclassifications. However, it also showed a lack of understanding for "country" and "rock". Similarly, the CNN performed well in identifying "classical" and "reggae", but struggled more significantly with "country" and "rock". The CNN also struggled significantly more with the "disco" and "pop" genres. Overall, the FFNN generally had fewer misclassifications across most genres. Exceptions were observed in "blues," "hiphop," and "reggae," where the CNN slightly outperformed the FFNN. However, these differences were minor compared to the larger differences in favor of the FFNN for other genres.

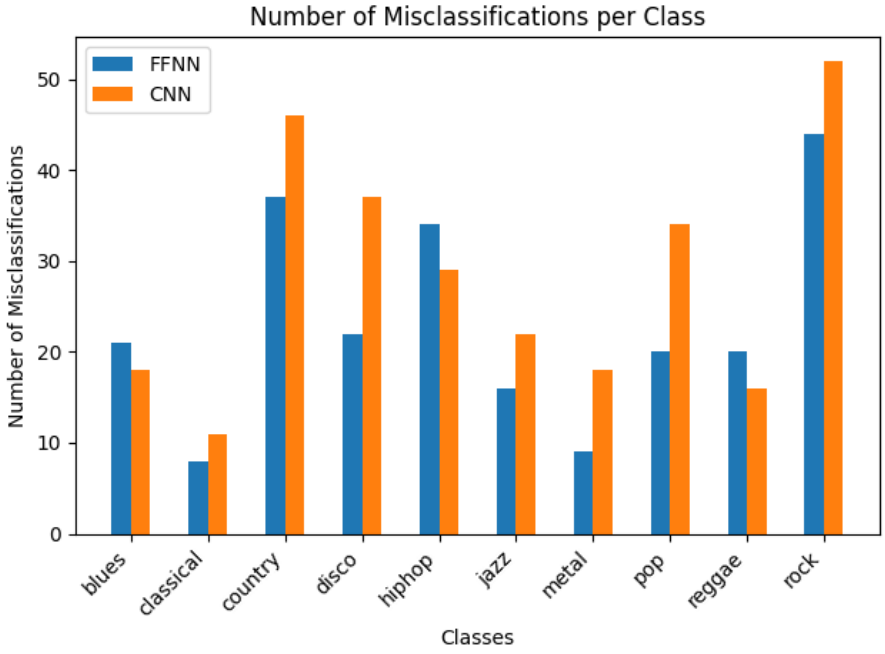

Figure 7: Comparison of Misclassifications per Genre for FFNN and CNN Models

4 Discussion and Conclusion

Overall, the results show that both model architectures are well-suited for music genre classification. When looking at the number of misclassifications per genre, the FFNN outperforms the CNN in terms of performance. This study highlights several strengths, such as using a feature extraction pipeline—including MFCCs, chroma features, and various spectral descriptors effectively summarised the audio content. Furthermore, comparing FFNN and CNN architectures provided insights into how different neural network designs can impact performance on the same dataset.

4.1 Limitations

Despite these results, the reported approach has limitations:

1. **Dataset:** The models were trained only on the GTZAN dataset. This possibly limits the generalizability of the findings to broader music collections or to genres that are not represented in the dataset.

2. **Feature Averaging:** The decision to average features over 3-second fragments while increasing the training sample size could lead to a loss of temporal dynamics that are important for some genres.

3. **Recurrent models:** Given the time constraints, the study did not investigate recurrent models, such as LSTMs, which might better capture the sequence of the songs. We would have liked to create a third model, an LSTM, considering the time series nature of the data. Sadly, we couldn't implement this due to a lack of time. The original plan was to create an ensemble model using majority voting to determine if combining the 3 models would result in even better performance.

4.2 Directions For Future Projects

Future projects could address the current limitations of the project presented in this report by:

- **Expanding the Dataset:** Incorporating additional datasets or augmenting the GTZAN collection with more diverse music samples could improve model robustness and generalizability.

- **Temporal Dynamics:** Explore models that capture temporal dependencies (e.g. LSTM) may enhance overall performance.

- **Ensemble Techniques:** Developing an ensemble model that combines the FFNN, CNN, and potentially a recurrent architecture.

4.3 Conclusion

This study demonstrates that the feature extraction pipeline combined with neural network architectures can perform well for the task of music genre classification. While the FFNN outperformed the CNN in this setup in terms of accuracy, using both models enabled us to learn from the strengths and weaknesses. In our view, addressing the limitations and exploring different architectures in future projects could lead to improved automated music classification systems.

References

Cheng, Y.-H., Chang, P.-C., & Kuo, C.-N. (2020). Convolutional neural networks approach for music genre classification. *2020 International Symposium on Computer, Consumer and Control (IS3C)*, 399–403. https://doi.org/10.1109/IS3C50286.2020.00109.

Chollet, F., et al. (2015). *Keras*. https://github.com/fchollet/keras.

Fessahaye, F., Perez, L., Zhan, T., Zhang, R., Fossier, C., Markarian, R., Chiu, C., Zhan, J., Gewali, L., & Oh, P. (2019). T-recsys: A novel music recommendation system using deep learning. *IEEE International Conference on Consumer Electronics (ICCE)*.

McFee, B., Raffel, C., Liang, D., Ellis, D. P. W., McVicar, M., Battenberg, E., & Nieto, O. (2015). Librosa: Audio and music signal analysis in python. *SciPy*. https://api.semanticscholar.org/CorpusID:33504.

Pandey, P. (2021, May). Music genre classification with python. https://towardsdatascience.com/music-genre-classification-with-python-c714d032f0d8.

Schedl, M. (2019, August). Deep learning in music recommendation systems. https://doi.org/10.3389/fams.2019.00044.

Stevens, S. S., Volkmann, J., & Newman, E. B. (1937). A scale for the measurement of the psychological magnitude pitch. *The Journal of the Acoustical Society of America*, *8*(3), 185–190. https://doi.org/10.1121/1.1915893.

Tzanetakis, G., & Cook, P. (2002). Musical genre classification of audio signals. *IEEE Transactions on Speech and Audio Processing*, *10*, 293–302. https://doi.org/10.1109/TSA.2002.800560.