# Music Genre Classification

DEEP LEARNING

Laura Horsfall Folmer, 201905054 & Tony Lung Ca Hin, 201905921

December 17, 2021

Music genre classification is a popular task in music information retrieval (MIR). One could imagine applications for recommending songs to add to a Spotify playlist or automatically generate playlists based on a given genre. A music genre is a conventional label made to help characterise and categorise music. A genre is typically classified by, for example, instrumentation, rhythm, tempo, flow and geographical origin. With this project we explore, how well we can classify different bites of music from 10 different predefined genres. We want to explore how we can visualise music such that the computer can separate and distinguish each snippet of music using Deep Learning.

## Data set presentation

The data used in this project is the *GTZAN Genre Collection* [2]. The data set consists of 1000 audio tracks where each track is 30 seconds. It contains 10 genres - blues, classical, country, disco, hiphop, jazz, metal, pop, reggae and rock - each represented by 100 tracks. The tracks are all 22050Hz Mono 16-bit audio files in .wav format.

This data set is commonly used for the problem of music genre classification. The tracks have been collected from many different sources to ensure diversity.

Specifically for this task, we have used a 80-10-10 train, validation, test, with a seed, to ensure we had the same split. The .wav files are loaded into waveform using torchaudio and librosa package, which can be seen in figure 1.

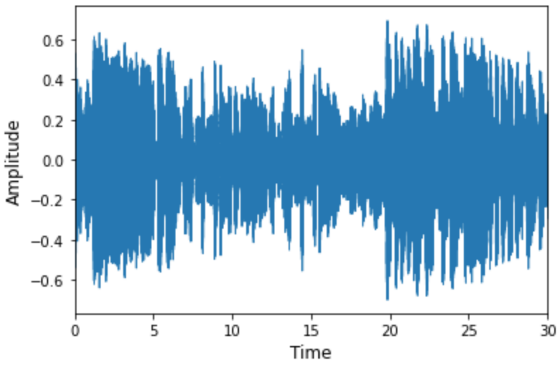



Figure 1: Waveform

### Feature extraction

For solving the task of music genre classification, we could have used a many-to-one sequential model on the waveform. A more common approach is however, to do feature extraction by computing melspectrograms. This is done by mapping the audio signal to the frequency domain using Fast Fourier Transform. Melspectrograms are very much like spectrograms but the frequencies are scaled by the mel scale. It provides a visual representation of the audio, in a time-frequency domain, and it is widely used to capture the characteristics of a given audio signals. Different music genres have different sounds with different frequencies and rhythms, which the melspectrograms can visualize. Figure 2 shows a computed melspectrogram of an audio sample from GTZAN.

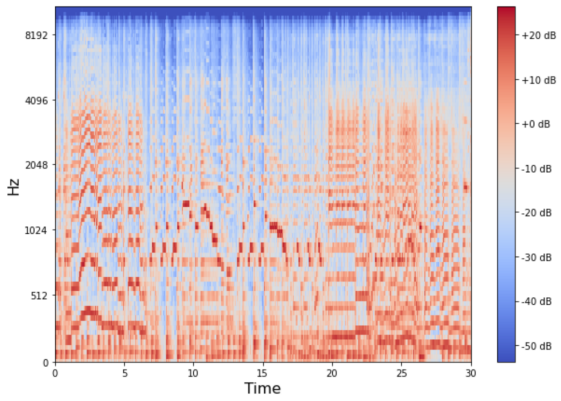



Figure 2: Melspectrogram

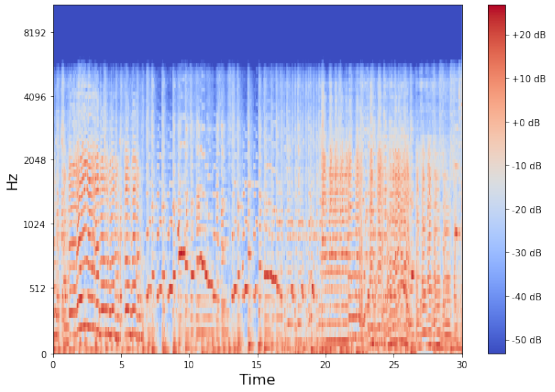



Figure 3: Pitch-shifted with n_step = -9

To transform the waveforms into melspectrograms we have used 2048 fast-fourier transforms. This is a typical value for this. We have used a window size of the same length and a hop length of 1024. This means that the window of which the FFT is applied is shifted with 1024. Furthermore, we have used 90 mel filter banks.

The melspectrograms have also been converted from power (amplitude) to decibels to both reflect how humans hear music and to make the visualisation more clear.

## Data augmentation

The data set is relatively small with only 1000 samples. The purpose of data augmentation is to use the available data to create more data. For data augmentation for audio it is possible to change the speed, do time shifting and inject background noise. For avoiding that the network over-fits genres to specific frequencies we have applied pitch shift to the training data. Specifically we have applied a shift with librosa.effects.pitch_shift with bins_per_octave = 12 with a random n_step of -9,-8,...,-5, 5, ..., 9, for each sample in the training data set. After that, we again compute the melspectrograms.

# Network structure

Since the audio signals are transformed to melspectrogram the problem is like-wise transformed to an image classification problem with 10 classes. We have tried to approach the problem using an MLP, which gave reasonable results, but trying to expand the MLP gave us out-of-memory errors. Working with fully connected layers and images gave an explosion in parameters. The more suitable choice was to use CNNs.

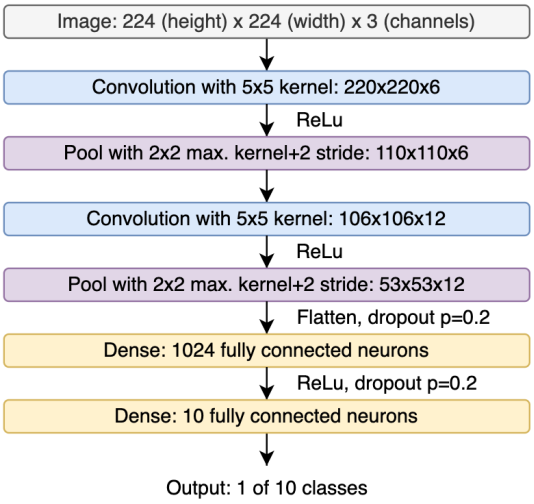
## CNN

We have implemented a rather simple CNN. Our CNN consists of two convolutional layers and two fully connected layers. We have use convolution layers to extract the features of the melspectrogram. After each convolution and each fully connected layer, we have applied ReLu activation. The two convolutinal layers are followed by a batch normalisation step before doing pooling. Batch normalization is applied to make the network converge faster and be more stable, and pooling is applied to reduce the spatial dimensions of the feature space.

The network architecture for our CNN model is visualised in figure 4.

Figure 4: CNN architecture

*Made with diagrams.net*




### DenseNet

Given the results of our own simple CNN network, and that we have a relatively small data set, we found it appropriate to use transfer learning and fine-tune the DenseNet which claims to reduce overfitting in small data sets. DenseNet uses dense blocks. Dense blocks reuses features by concatenation, which then reduces the amount of parameters and also alleviates the vanishing gradient problem. For this problem we use DenseNet-121 from Pytorch.

## Performance

For hyperparameter tuning, we sought to choose parameters that would increase validation accuracy. Due to limited time, this was mostly done through trial-and-error and tracked using Neptune.

The loss of the training and validation run was saved and plotted to evaluate and adjust the learning rate parameter.

We would have liked to systematically and automatised the process of hyperparameter search. Furthermore, better estimates of hyperparameter performance could have been achieved by the use of cross-validation, instead of using a fixed seed on a random split. The number of epochs in training for every model has been set to 30. The value is small because of limited time and because our models tend to converge and also overfit quite fast.

We have used a dropout rate of $p = 0.2$. A higher drop out rate made the training accuracy fluctuate a lot. Additionally, it did not improve the validation loss a lot either.

The image resolution is 224x224 to be able to fit and compare with with the pretrained DenseNet model we have used from Pytorch.

### Larger dataset

The dataset is relatively small and humans might not need to listen for 30 seconds to classify a song to a genre. In an attempt to increase the amount of samples we tried to segment the audio files into 6 seconds segments. We then compute the melspectrograms, using the same parameters and resize them to (3,224,224) images. Each audio segment is now regarded as a song belonging to a class. We think that this might help alleviate the problem of overfitting. To ensure that we did not shuffle audio segments of the same song into the validation set or test set, we first split the dataset into train, validation and test sets, and only after the split, we further divide each song into 6 second segments. Using a 80-10-10 split we now have 4000 samples in our training set and 1000 samples for validation and test.

## Results

To try and understand the effect and impact of data augmentation and creating a larger data set, we train the CNN on three different data sets. The original 800 samples, 1600 samples with augmented data as described earlier, and with the 4000, 6 second samples. We then do the same with a pre-trained DenseNet-121. For ease of training the networks, we saved all 3 data sets as (3,224,224) tensors. Training was done on Kaggle, since it has a free 16GB GPU. We train the models for 30 epochs and save the models with the lowest validation loss. All models use the same split in training,

| Model | Accuracy % |
|---|---|
| CNN, 1000 samples | 65 |
| CNN, 1800 samples (with augmentation) | 57 |
| CNN, 5000 samples | 65.8 |
| DenseNet-121, 1000 samples | 68 |
| DenseNet-121, 1800 samples (with augmentation) | 74 |
| DenseNet-121, 5000 samples | 77.8 |

Table 1: Test accuracy on each model

validation and test set. Test accuracies are shown in table 1, using the models with the lowest validation loss.

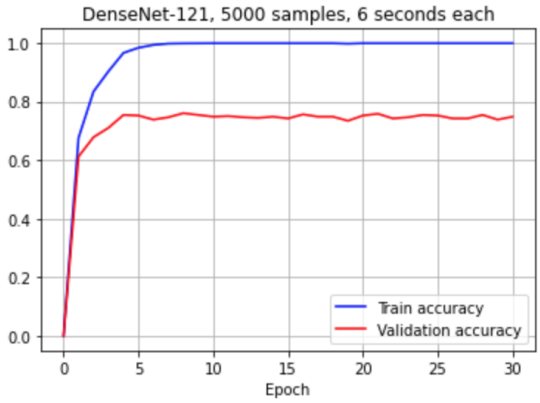



Figure 5: DenseNet, accuracy

Convergence is generally fast for every network, but there is a big gap between the validation accuracy and the test accuracy. Training for more epochs does increase the validation loss significantly. For the DenseNet in figure 5, validation accuracy reached 74.8%.

## Confusion Matrix

The confusion matrix summarises the performance of our model. Because we have split the data randomly we are not guaranteed to have equal amount of data within each of the ten classes (genres). The confusion matrix helps to give a better overview of how our model performs individually on each genre, and potentially which genres the model is most likely to mix up. Each row of the confusion matrix corresponds to a true genre, while each column corresponds to a predicted genre. The confusion matrix for DenseNet-121 with 5000 samples (6 seconds per track) is showed in figure 6.

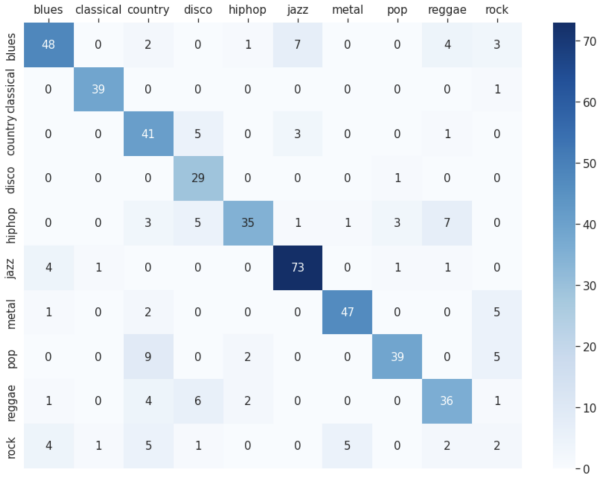



Figure 6: Confusion matrix, DenseNet 5000 samples, test

This shows how our data has been split randomly. We have for example 80 samples in the genre "jazz" and only 20 samples in the genre "rock". How well the model performs on the individual genres also depends on the samples we have seen in the training data. Perhaps the training data has had many samples within the disco-genre and fewer samples within the rock-genre.

# Discussion

Our results show that melspectrograms combined with a CNN can be used to classify genres with relatively high accuracy. The networks with 5000 samples used, i.e. 6 second segments show similar accuracies to the networks with 30 second samples. This

shows that we are able to classify the genres with much less than the 30 seconds. For our CNN, the artificial extra amount of samples did not further improve the validation accuracy nor the test accuracy significantly. A wider gap is seen when using the DenseNet-121, with an increase of almost 10% in test accuracy. The CNN trained with data augmentation showed worse validation accuracy and test accuracy, which suggests we have over-fitted even more, even though we save the model with the lowest validation loss. Given these results we would like to have tried to implement the dense blocks into our own CNN. If we had more time it would have made a lot of sense to create an ensemble classifier from the DenseNet trained on 6 second segments. It could be used to classify any song by splitting it into segments of 6 seconds. Furthermore, we would have liked to test the network on songs that did not belong to any of the predefined genres. Everything was done on an initial random split, and the performances could be dependant on that. The validation accuracy was lower than the test accuracy, which we did not expect. From the confusion matrix, we see that the training data has an uneven distribution. There were a lot of jazz samples and very few rock samples. If we had more time the networks should be tested on various different splits in the data set or done the same on a bigger data set. Music genre classification is a tough task, and the class labels can be ambiguous.

# Ressources


[1] Kumar, Pankaj. (2019). *Music Classification Using nn*.<https://www.kaggle.com/pk15467/music-classification-using-nn?scriptVersionId=35952717>

[2] Marsyas (Music Analysis, Retrieval and Synthesis for Audio Signals). (2015). *GTZAN Genre Collection*. [Data set]. <http://marsyas.info/community/people.html>

[3] Pytorch. (2021). *Audio manipulation with torchaudio*. <https://pytorch.org/tutorials/beginner/audio_preprocessing_tutorial.html>

[4] Pytorch. (2021). *DENSENET*. <https://pytorch.org/hub/pytorch_vision_densenet/>

[5] Pytorch. (2021). *Finetuning torchvision models*.<https://pytorch.org/tutorials/beginner/finetuning_torchvision_models_tutorial.html>

[6] Roberts, Leland. (Mar 6, 2020). *Understanding the mel spectrogram*.<https://medium.com/analytics-vidhya/understanding-the-mel-spectrogram>