Université Paris-Dauphine
Informatique et mathématiques pour la décision et les données

Convolutional Neural Networks for Music Classification Tasks

Elise Chin

July 25, 2020

Bachelor Internship
Supervised by Tristan Cazenave

Abstract

Categorizing music files according to their genre or mood is a challenging task in the area of music information retrieval (MIR). In this work, we tackle these two classification tasks. For the first task, we evaluate the accuracy scores of CNN-based models with different input types using the GTZAN dataset, where a 3-layer architecture shows incredible performance with mel spectrogram input. Then, we varied the number of layers on a larger dataset, and found that deeper models outperformed the 3-layer architecture with an accuracy score of 89.2%. Furthermore, we trained a feedforward neural network and five other traditional machine learning classifiers with hand-crafted time domain and frequency domain features to compare their performance with a CNN-based model. CNNs eventually outperforms them. For the second task focusing on mood classification, a subset of the MTG-Jamendo Mood Theme dataset was used. Resampling techniques were carried out as the dataset was fairly imbalanced, and we measure the AUC-ROC scores of the CNN architectures. Unfortunately, the results were not conclusive.

Contents

List of Figures

List of Tables

1 Introduction

Music tags are a set of descriptive keywords which provide meta-information about a music clip, such as emotion (sad, anger, happy) or genre (jazz, classical). These two types of high-level information are very useful when we want to describe a music. With the rapid growth of online music databases and streaming services, we have now easy access to dozens of millions digital music[1] that building an automatic tagging system have become fundamental to structure these huge databases. This is exactly what automatic content-based music classification systems do by providing proper labels to song files, and therefore helping manage growing song database. Listeners as well as audio streaming services would both benefit from those systems. Large music databases will be tagged and categorized by computers. Such solution is not only inexpensive and scalable, but also allows to create interesting content for the user through music discovery and recommandation, enhancing the listener's experience.

Since 2000s, music information retrieval (MIR) has been widely studied for important applications and one of the challenging and valuable task for MIR systems is automatic music genre/mood classification. Building this system requires extracting acoustic features that are relevant for the type of genre/mood we are interested in, followed by a single or multi-label classification using various machine learning algorithms. However the first step demands esoteric knowledge on signal processing and which feature are the most relevant to perform each task. Humans has to manually engineer good representations of the music data. The recent approaches using Deep Neural Networks (DNNs) completely automates this step: with deep learning, the relevant features to each task are learned at the same time as the system is learning to classify them. Convolutional Neural Networks (CNNs) in particular have received much success in image recognition and been tried in the MIR field.

In this paper, we study an automatic tagging algorithm based on Convolution Neural Networks (CNNs) for music genre and mood classification separately. For the genre task, a first CNN-based architecture is trained end-to-end on several audio representations (mel spectrogram, spectrogram, and MFCC), then using the best one as input of five other architectures with various number of layers to find the best model. A feedforward neural network is also trained to access the performance improvement that can be achieved by CNNs. Finally, we provide a comparative study between deep learning models which only requires a spectrogram as input and conventional machine learning classifiers (Logistic Regression, Random Forest, Gradient Boosting, Support Vector Machine, and K-Nearest Neighbors) which need to be trained with hand-crafted features. For the mood task, we focus on handling imbalanced datasets by using resampling methods and we trained several CNNs to determine which dataset leads to the best performance[2].

We introduce CNNs in detail in Section 2, then we firstly focus on music genre classification task in Section 3, followed by music mood classification in Section 4. In both sections, architectures are explained, the different experimentations are presented, and evaluations are reported. We conclude in Section 5 by mentionning encountered difficulties, and possible future work.

2 CNNs for Music Classification

2.1 Motivation for using CNNs in Music Classification Tasks

In this section, we review the properties of CNNs with respect to music signals. CNNs are designed to provide a way to learn local patterns that are translation invariant and increasingly complex in their input [5]:

- Translation invariant because a pattern learnt in a specific region of the input can be recognized in another location. The question focuses on whether the feature is present rather than where it was present. If the filter is designed to detect a particular type of feature in the input, then the application of that filter systematically across the entire input image allows the filter to discover that feature anywhere in the image.

- Increasingly complex patterns because a first convolution layer will learn small local patterns such as edges, a second convolution layer will learn larger patterns made of the features of the first layers, and so on. This allows CNNs to efficiently learn hierarchical features and abstract visual concepts.

[1]For example, Spotify has over 50 million tracks according to https://newsroom.spotify.com/company-info/
[2]The source code can be found here: https://github.com/Elxse/deep-learning-for-music-classification-tasks

Although the 2D representation of an audio signal is not exactly the same as a visual image, CNNs' properties work well with audio signals too and can be useful for automatic tagging for several reasons. First, musical events that are relevant to tags can occur at any time or frequency range so CNNs' local invariance property allow them to learn musical features specific to a tag more efficiently. Second, music tags are often considered as high-level features representing song-level above intermediate level features such as chords, beats, tonality. Since CNNs assume features are in different levels of hierarchy, such high-level features can be extracted by convolutional kernels [3].

2.2 CNNs Architectures

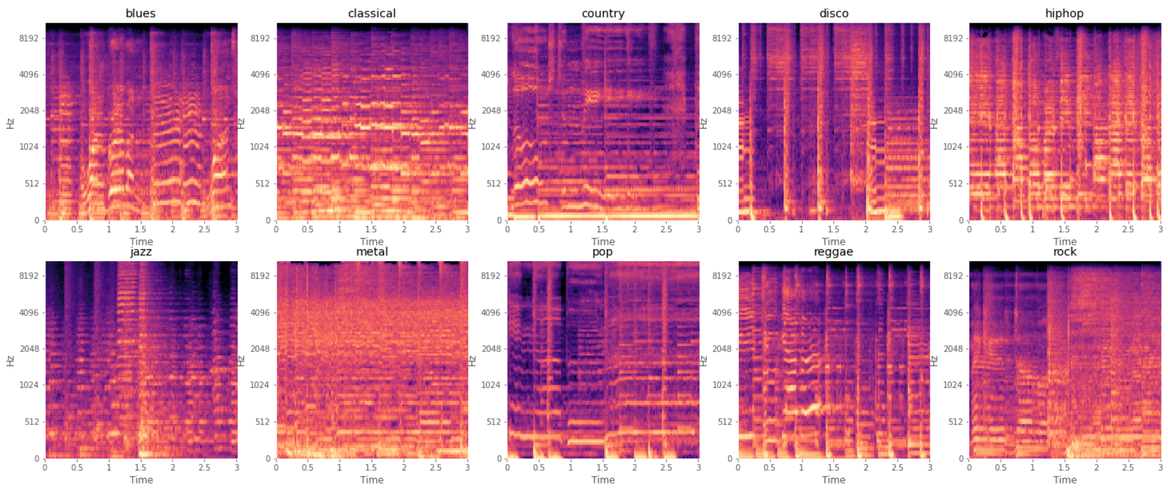


Figure 1: Mel spectrograms for one audio signal segment from each music genre

In Figure 1, one can observe mel spectrograms of audio signals belonging to different classes shows different characteristic patterns. Hence, it is interesting to consider these mel spectrograms as 'images' and feed them into a CNN which will extract relevant features for the classification task.

A convolutional neural network takes as an input an image, a 3-dimensional tensor. The first two dimensions correspond to the image width and height, and the last one represents the number of channels (1 for a black-and-white image and 3 for a colored one). Contrary to a dense neural network where layers are fully connected and act as a classifier, a CNN-based architecture has in addition a convolution part and thus is composed of two distinctive parts: a convolution part and a classification part.

- The convolution part extracts the most revealing features specific to each image by compressing and reducing their initial size. More precisely, each image goes through a succession of filters, are transformed and combined to form new images called *feature maps*. It involves convolution and pooling layers.
- Feature maps obtained in the convolution part are then flattened to create a vector and fed to the second part of the neural network composed of fully-connected layers. The objective is to classify the image.

2.2.1 Convolution[3]

The convolution layer is the first layer to extract features from an input data. It applies a mathematical operation called "convolution" to create a feature map as output. This step involves sliding a feature detector (e.g. of size 3x3), often referred as a "kernel" or "filter", over the input image. It moves progressively from left to right of a number of predefined steps (strides) until it reaches the end of the image. At each portion of the image matrix, an element-wise multiplication is computed between the kernel and the overlapping portion of the input data, followed by a summation to obtain a single value, a cell of the feature map. The operation is repeated as many times as the number of filters and all the resulting feature map are combined in a single feature map of depth $n\_filters$. A CNN does not require us to guess what filter we should use. The training of a CNN is the training of these filters.

[3]https://www.superdatascience.com/blogs/convolutional-neural-networks-cnn-step-1-convolution-operation

2.2.2 Max-Pooling

The pooling layer is a way to reduce the dimension of the feature map obtained in the convolution step. This process of *downsampling* reduces the computational cost by reducing the number of learned parameters while retaining important information in a region. It also provides translation invariances, meaning that a pattern learnt in a specific location can be recognized in another location.

The type of pooling used in this work is the Max-Pooling. For instance, the max-pooling operation with a 2x2 window size creates an output feature map where each value corresponds to the maximums of each 4-element region of the feature map that are covered in this window. In the same way as the kernel in the convolution operation, the filter moves across the feature map with a predefined *stride*.

3 Music Genre Classification

Music genre classification is a single-label classification problem where we assign only one tag to each data point. Following prior experiments by Choi et al. in [3], I firstly evaluate mel spectrogram, spectrogram (STFT), and Mel Frequency Cepstral Coefficients (MFCC) as input of a 3-layer CNN and secondly, I evaluate five architectures (CNN-{3,4,5,6,7}) with mel spectrograms input. A feedforward neural network is also trained to access the performance improvement that can be achieved by CNNs. Finally, we provide a comparative study between deep learning models which only requires a spectrogram as input and conventional machine learning classifiers (Logistic Regression, Random Forest, Gradient Boosting, Support Vector Machine, and K-Nearest Neighbors) which need to be trained with hand-crafted features.

3.1 Dataset Acquisition

3.1.1 GTZAN Dataset

GTZAN dataset created by Tzanetakis et al. [7] is compounded of 1000 audio tracks of 30 seconds long, with 100 examples in each of 10 different music genres: Blues, Classical, Country, Disco, Hip Hop, Jazz, Metal, Popular, Reggae, and Rock. The files were collected in 2000-2001 from a variety of sources including personal CDs, radio, microphone recordings, in order to represent a variety of recording conditions. All tracks are 22,050 Hz, Mono 16-bit audio files in .wav format.

3.1.2 Input Types and Length

Training deep neural networks is computationally intensive, therefore when applying deep learning methods to MIR problems, it is particularly important to understand audio data representations. The objective is to effectively and efficiently represent the information contained in an audio signal - effectively so that the network can easily use it and efficiently so that the memory usage and/or the computation is not too heavy [4]. The majority of deep learning approches in MIR take advantage of 2-dimensional representations of the audio signal instead of its 1-dimensional representation which is the raw audio itself. This is because 2D representations provide audio data in an effective form. 2-dimensional representations of the audio data such as spectrogram and mel spectrogram have been the most popular choices in MIR works because training a network from the audio signals requires even a larger dataset. In this work, MFCC are extracted in addition to spectrogram and mel spectrograms.

Spectrogram. Spectrograms are a way to visually represent a signal's loudness, or amplitude, as it varies over time at different frequencies. The horizontal axis is time, the vertical axis is frequency, and the color is amplitude. It is calculated using the fast Fourier transform on short time windows (Short-time Fourier transform) of the signal and transforming the frequency axis to log scale and the amplitude axis to decibals. In this study, each audio signal was converted into a spectrogram of 513 frequency bins and 130 frames (513×130). The parameters used to generate spectrogram using STFT are listed below:

- Sampling rate (sr) = 22050
- Frame/Window size (n_fft) = 1024
- Time advance between frames (hop_length) = 512
- Window Function: Hann Window
- Highest Frequency ($fmax$) = $sr/2$

Mel Spectrogram. Mel spectrogram is a 2D representation that is optimised for human auditory perception. Humans are in fact better at detecting differences in lower frequencies than higher frequencies.

The mel scale transforms the frequency scale such that sounds at equal distances from each other also sound equal in distance. For example, let's take the deviation between pure tones of frequencies 300 Hz and 600 Hz as a reference. In order to reproduce the same deviation from a pure tone of frequency 1000 Hz, a second pure tone will have to be presented, not at frequency 2,000 Hz but at about 2,363 Hz. 300 Hz corresponds to 402 mels and 600 Hz corresponds to 698 mels. The ratio between the two is therefore 1,735. 1,000 Hz corresponds to 1,000 mels by definition. The difference perceived as idential must be with a sound at 1,735 mels, which corresponds to 2,563 Hz [4]. In this study, each audio signal was converted into a mel spectrogram of 128 frequency bins and 130 frames (128×130). The parameters used to generate mel spectrogram are identical as the ones for spectrogram. The additional parameter *n_mels* corresponding to the number of mel bins is fixed to 128.

Mel Frequency Cepstral Coefficients (MFCC). The mel frequency cepstral coefficients of a signal are a small set of features (usually about 10-20) which concisely describe the overall shape of a spectral envelope. In MIR, it is often used to describe timbre [5]. The parameters used to compute MFCCs across an audio signal are identical as the ones for spectrogram. The additional parameter n_mfcc corresponding to the number of coefficients was set to 20. So we have computed 20 MFCCs over 130 frames.

3.1.3 Data Augmentation

	Number of Samples	Data type	Data Size	Dataset Size
Experiment I	10000	Mel spectrogram	128×130	666.9 MB
	10000	Spectrogram	513×130	2.7 GB
	10000	MFFC	20×130	105.3 MB
Experiment II Baseline	19000	Mel Spectrogram	128×130	1.3 GB

Table 1: Different datasets generated from GTZAN

Data Augmentation is a technique to avoid overfitting by increasing the number of samples in the dataset considered. For this task, I adopted a multiframe strategy that allows us to extract more than one input data per song. More precisely, it consists of dividing each song into equal time-length slices and extracting the desired type of input at each frame. Thanks to this approach, I was able to generate more data than the number of songs, which is very useful to increase the performance of the network. In Experiment I, each song is divided into a set of 10 three-second excerpt with no overlapping ($hop\_length = 0$). Spectrogram, mel spectrogram, and MFCCs are then extracted at each frame using the parameters listed above. In Experiment II, the *hop_length* is set to 512 which corresponds to a 50% overlap and which results in 19 frames per song. Only mel spectrograms are extracted. Not only a larger dataset of mel spectrograms is obtained, but time overlapping also permits to retrieve possible loss data due to the Hann window when STFT is applied. In fact, the Hann window is decreasing to zero at boundaries, thus some data can be lost. In order to retrieve this data, a 50% overlap is often applied.

Also, the reason why no overlap was set in the first experiment is due to memory limitations and the size of spectrograms. Extracting spectrograms with an overlap means more data samples and thus more memory needed to store these spectrograms. Since I was limited by the storage capacity provided by Google Drive, I could not proceed the same way as in the second experiment. In order to have the same number of data samples for each type of input for the sake of dataset size equality, I aligned the number of mel spectrograms and MFCCs extracted to the number of spectrograms.

Four different datasets, whose specifications can be found in Table 1, were generated in total.

3.1.4 Manually Extracted Features[6]

In order to compare the performance of CNNs and shallow machine learning techniques and see if CNNs outperform them indeed, I prepared hand-crafted features to be fed in classifiers.

[4]https://en.wikipedia.org/wiki/Mel_scale
[5]https://musicinformationretrieval.com/mfcc.html
[6]https://musicinformationretrieval.com/

Time Domain Features.

1. Central moments
2. Zero Crossing Rate (ZCR)
3. Root Mean Squared Energy (RMSE)
4. Tempo

Frequency Domain Features.

1. Mel Frequency Cepstral Coefficients
2. Chroma Features
3. Spectral Centroid
4. Spectral Band-width
5. Spectral Contrast
6. Spectral Roll-off

3.2 Models

3.2.1 Convolutional Neural Networks

Table 2: The configuration of CNN-3 with mel spectrogram input

CNN-3
Mel spectrogram (input: $128 \times 130 \times 1$)
Conv $3 \times 3 \times 32$
MP(2,4) (output: $63 \times 64 \times 32$)
Conv $3 \times 3 \times 32$
MP(4,4) (output: $31 \times 31 \times 32$)
Conv $3 \times 3 \times 32$
MP(2,5) (output: $15 \times 15 \times 32$)
Output 10×1 (softmax)

Table 2 shows one of the proposed architectures for the genre classification task. It is a 3-layer CNN *(CNN-3)* composed of 3 convolution layers and 3 max-pooling layers, and takes a 128 x 130 log-amplitude mel spectrogram as input. The difference between the architecture of the CNN for spectrograms and mel spectrograms lies in the pooling sizes, which are adapted to the number of frequency bins. For the third type of input (MFCC), time-axis convolution and pooling layers are used instead of 2D convolutions and poolings.

The architecture is then extended to deeper ones with 4, 5, 6 and 7 layers *(CNN-{4, 5, 6, 7})*. The number of feature maps of CNN-4 and CNN-5 are adjusted based on CNN-3 to make the hierarchy of learned features deeper. Similar to [3], CNN-6 and CNN-7 have additional 1×1 convolution layer(s) on top of CNN-5 (Table 3)

Rectified Linear Unit (ReLU) is used as an activation function in every convolutional layer except the dense output layer, which uses Softmax. The Softmax activation function converts the inputs into a discrete probability distribution over the target classes whereas the rectified linear activation function returns the value provided as input directly, or the value 0.0 if the input is 0.0 or less. Using ReLU improves the convergence of the algorithm because it reduces the likelihood of vanishing gradient, which is when the value being backpropagated decreases dramatically with each additional layer, hence making it difficult to know in which direction the parameters should be update to minimize the loss function[7].

To accelerate the convergence even more and to prevent the network from overfitting, Batch Normalization and Dropout are added. Contrary to Choi et al. who added Batch Normalization before every activation, I added it after every max-pooling layer which are placed after activation. They also added a Dropout of 0.5 after every max-pooling layer but in my case, I added only one Dropout of 0.3 before the final dense layer. Strides of all max-pooling layers are fixed to 2 to reduce the number of learned parameters. These choices are based on my experimentations.

3.2.2 Baseline Feedforward Neural Network

To evalutate the performance improvement that can be achieved by CNNs, we also train a baseline feedforward neural network that takes as input the same mel spectrogram image of size 128×130. The 2-dimensional image is first flattened into a 1-dimensional vector and then fed into a simple neural network to predict the genre of a song. The neural network is compounded of 3 hidden layers: the first hidden layer consists of 512 units, the second layer has 256 units, and the last one 64 units. The activation function used is ReLU for the hidden layers and Softmax is used for the output layer. Finally, in order to prevent overfitting, L2-Regularization with $\lambda = 0.001$ and Dropout of 0.3 are adopted.

[7]https://machinelearningmastery.com/rectified-linear-activation-function-for-deep-learning-neural-networks/

Table 3: The configuration of CNN-4, 5, 6, 7 with mel spectrogram input

CNN-4	**CNN-5**	**CNN-6**	**CNN-7**
Mel spectrogram (input: $128 \times 130 \times 1$)			
Conv $3 \times 3 \times 32$			
MP(2,4) (output: $63 \times 64 \times 32$)			
Conv $3 \times 3 \times 32$			
MP(4,4) (output: $31 \times 31 \times 32$)			
Conv $3 \times 3 \times 32$			
MP(2,5) (output: $15 \times 15 \times 32$)			
Conv $3 \times 3 \times 64$			
MP(2,4) (output: $7 \times 7 \times 64$)			
[redacted]	Conv $3 \times 3 \times 64$		
[redacted]	MP(4,4) (output: $3 \times 3 \times 64$)		
[redacted]	[redacted]	Conv $1 \times 1 \times 128$	
[redacted]	[redacted]	[redacted]	Conv $1 \times 1 \times 128$
Output 10×1 (softmax)			

3.2.3 Machine Learning Classifiers

In this section, we describe the five machine learning classifiers trained to access the performance of neural networks over these traditional classifiers. The hand-crafted features listed in Section 3.1.4 are fed into the classifiers.

Logistic Regression (LR). Logistic Regression is one of the most simple and commony used machine learning algorithms for binary classification. For this multi-class classification task, we used a one-vs-rest (OvR) scheme, that means 10 separate binary classifiers are trained, one for each label.

Random Forest (RF). Random Forest is an ensemble classifier that combines the prediction of a large number of individual decision trees. Each decision tree are trained using a random subset of the training samples and can only pick from a random subset of features when splitting a node. These two methods allow variation amongst the trees in the model which results in more accurate ensemble predictions. In this work, 500 trees are trained (*n_estimators*) and the minimum number of samples required to split an internal node *min_samples_split* is set to 5.

Gradient Boosting (XGB). Gradient Boosting is another machine-learning technique which consists of weak predicitions models, generally decisions trees, that operate as an ensemble. However, unlike RFs, boosting algorithms are trained in a sequential manner in which new models are specialized in adressing weak points of the previous models. Hence applied to decision trees, this technique results in models that outperform random forests most of the time. The parameters used to initialize $XGBClassifier$ are listed below:

- Maximum Tree Depth (*max_depth*) = 5
- Learning Rate (*learning_rate*) = 0.1
- Number of Trees (*n_estimators*) = 500
- Subsample Ratio of Training Instance (*subsample*) = 0.8

Support Vector Machines (SVM). SVMs are a set of binary classification algorithms which transform the original input data into a new high-dimensional space. More precisely, they aim at finding a good decision boundary that separates the training data into two spaces corresponding to two categories. In the high-dimensional space, the decision boundary can be expressed as a hyperplane and the optimal hyperplane is the one that maximizes the margin or the distance between the hyperplane and the closest data points from each class. To find good decision boundary, we compute the distance between pairs of point in the target representation space instead of explicitly computing all coordinates of our data points in the new space. This operation is done by a kernel function, in our case a radial basis function (RGF) kernel is used to train the SVM to address the non-linear problem. Similar to the logistic regression setting discussed above, the SVM is also implemented as a one-vs-rest classification task.

K-Nearest Neighbors (KNN). K-Nearest Neighbor is a non-parametric supervised classification algorithm which essentially finds the k closest training examples given a new observation. This "unseen" data point is then assigned to the most common class among its k nearest neighbors. Similarity is defined according to a distance metric between two data points, here we choose the Euclidean distance. The number of neighbors to find $n\_neighbors$ is fixed to 7.

3.3 Experiments and discussions

3.3.1 Implementation details

For practical implementation, I used Google Colab which allows us to write and execute Python in our browser with free access to GPUs.

Dealing with large datasets. Google Colab has a RAM capacity of 12.72 GB but some datasets were still too large to fit into memory. Instead of loading the data in one go, the solution adopted was to break it into chunks with a Python generator. The main advantage of using a generator lies in the fact that each batch won't occupy the memory until it is necessary, which is very useful in this example where memory is scarce. Concretely, after extracting mel spectrograms, spectrograms, and MFCCs from the music data, I prepared for each dataset a train, a validation, and a test sets of the filenames. The samples as well as the splits were saved in a *.npy* format and imported in Google Drive. Then in the notebooks where models are trained, the generator reads a split file and creates mini batches of 128 samples based on the filename contained in the split. Not only it allows us to handle large datasets, but it also permits to train the different models using the same training and validation sets for comparison.

Training details. In all experiments, model training is implemented with Adam, starting with a learning rate of 0.001. Two callbacks are used: the first one is *ReduceLROnPlateau* to reduce the learning rate by a factor of 0.1 if the validation accuracy does not improve and the second one is *ModelCheckpoint* to save all models at each epoch. Models are trained for 20 epochs with a mini batch of 128 samples. The best models are then picked out across the different epochs to carry out evaluation.

Evaluation metrics. In order to evaluate the performance of the models described in Section 3.2, the following metrics will be used:

- Accuracy (during training and test): the percentage of correctly classified test samples.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

- F1-score (during test): the harmonic mean of the Precision and Recall. The Precision refers to the proportion of Actual Positives out of those Predicted Positives and Recall computes the proportion of Actual Positives labelled as Positive by the model.

$$Precision = \frac{TP}{TP + FP}$$

$$Recall = \frac{TP}{TP + FN}$$

$$F1 - score = 2 \times \frac{Precision \times Recall}{Precision + Recall}$$

- AUC[8](during test): the area under the receiver operator characteristics (ROC) curve, which a graph between the true positive rate (TPR) and the false positive rate (FPR) at various thresholds settings. It tells how much a model is capable of distinguishing between classes. A system is expected to have

[8]https://towardsdatascience.com/understanding-auc-roc-curve-68b2303cc9c5?gi=ab65ea04e407

a AUC higher than 0.5 because a 0.5 value means that the model has no discrimination capacity to distinguish between positive class and negative class.

$$TPR = Recall = \frac{TP}{TP + FN}$$

$$FPR = \frac{FP}{TN + FP}$$

3.3.2 Results

In this section, the different modelling approaches discussed in Section 3.2 are evaluated based on the metrics described in Section 3.3.1.

Scores

Table 4: Performance comparison of NN and ML models

	Accuracy	**F1-score**	**AUC**
Experiment I			
CNN-3, Mel spectrogram	**0.866**	**0.866**	**0.987**
CNN-3, Spectrogram	0.700	0.686	0.951
CNN-3, MFCC	0.611	0.608	0.920
Experiment II **Spectrogram-based models**			
CNN-3, Mel spectrogram	0.856	0.855	0.985
CNN-4, —	0.890	0.889	0.990
CNN-5, —	0.878	0.878	0.990
CNN-6, —	0.888	0.887	**0.993**
CNN-7, —	**0.892**	**0.891**	0.991
Feedforward NN Baseline	0.200	0.080	0.681
Feature Engineering based models			
Logistic Regression (LR)	0.743	0.745	0.857
Random Forest (RF)	0.730	0.730	0.850
Gradient Boosting (XGB)	**0.776**	**0.775**	**0.875**
Support Vector Machine (SVM)	0.746	0.748	0.859
K-Nearest Neighbors (KNN)	0.720	0.723	0.844

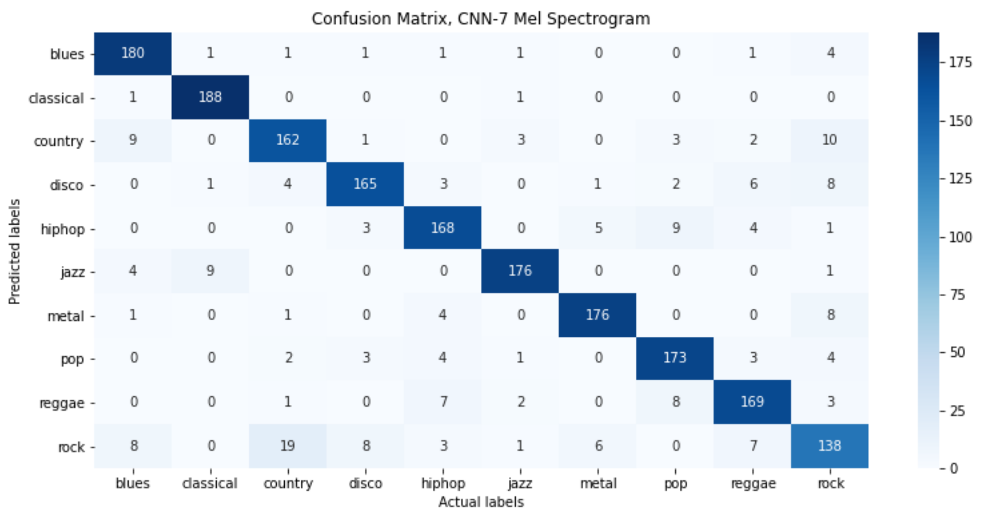
The first part of Table 4 summarizes the results of the first experiment where mel spectrograms, spectrograms, and MFCC representations are used as input of CNN-3, which is trained on a datasets of 10,000 samples (see Table 1). Mel spectrograms shows better performance out of the other types of input signals - spectrograms (STFT) and MFCC. On the second part of Table 4, the results of the second experiment of the proposed architectures with different number of layers and training on the dataset of 19,000 samples are reported. There are multiple things to notice.

1. The baseline feedforward neural network that uses flatten mel spectrograms as input performs poorly on the test set. With an accuracy score of 20%, it performs slightly better than a 'dumb' classifier which assigns all predictions to the same class (accuracy of 10%), but it is still a poor performance where there are 10 genres to distinguish. On the contrary, the best genre prediction using CNN has been made with a total accuracy of 89.2%. This shows that CNNs can significantly improve the scores on such image classification task.
2. Regarding the traditional machine learning models that use manually crafted features, SVMs outperform logistic regression models and random forests and the one with the best performance is the XGB version of the gradient boosting algorithm with an accuracy of 77.6%. This is expected as they are specialized in adressing weak points of previous trained models. Then again, the results shows that CNNs does improve the scores (here by 0.116 for the accuracy).

3. Finally, we can notice that CNN-3 showed a slightly worse performance than the same model trained on a smaller dataset in Experiment I (accuracy difference of 0.01) but CNN-{4,5,6,7} resulted in significant improvement compared to CNN-3. It seems that increasing the number of observations in the dataset indeed permits to improve performance, at least for the architecture with 4 layers and more. Second, these results also show that deeper structures benefit more from sufficient data. However, if we look at the results more precisely, performance varies a lot: based on the accuracy, CNN-4 performs better than CNN-{3,5,6}, but then CNN-7 outperforms CNN-4 by a little 0.02. In theory, more complex structures can perform at least equal to simples ones and our results support this because the performance of CNN-5 and CNN-6 were still making small improvements at the end of the training, implying it may perform equal to or even outperform CNN-4 if we trained for more epochs. But in practice, adding more layers to deep model leads to higher training error and thus accuracy gets saturated [6].

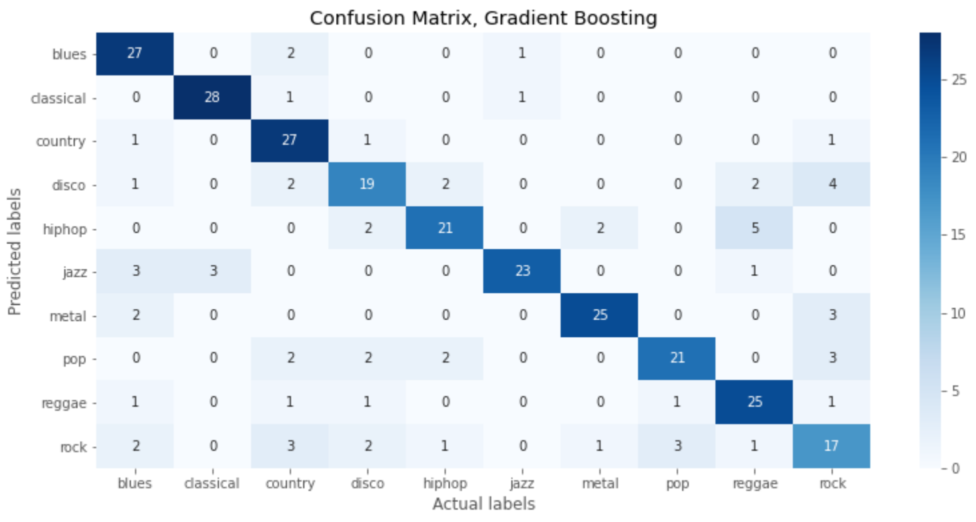
Confusion Matrices

Figure 2: Confusion matrices of the best performing models



Confusion Matrix, CNN-7 Mel Spectrogram

Predicted labels \ Actual labels	blues	classical	country	disco	hiphop	jazz	metal	pop	reggae	rock
blues	180	1	1	1	1	1	0	0	1	4
classical	1	188	0	0	0	1	0	0	0	0
country	9	0	162	1	0	3	0	3	2	10
disco	0	1	4	165	3	0	1	2	6	8
hiphop	0	0	0	3	168	0	5	9	4	1
jazz	4	9	0	0	0	176	0	0	0	1
metal	1	0	1	0	4	0	176	0	0	8
pop	0	0	2	3	4	1	0	173	3	4
reggae	0	0	1	0	7	2	0	8	169	3
rock	8	0	19	8	3	1	6	0	7	138



Confusion Matrix, Gradient Boosting

Predicted labels \ Actual labels	blues	classical	country	disco	hiphop	jazz	metal	pop	reggae	rock
blues	27	0	2	0	0	1	0	0	0	0
classical	0	28	1	0	0	1	0	0	0	0
country	1	0	27	1	0	0	0	0	0	1
disco	1	0	2	19	2	0	0	0	2	4
hiphop	0	0	0	2	21	0	2	0	5	0
jazz	3	3	0	0	0	23	0	0	1	0
metal	2	0	0	0	0	0	25	0	0	3
pop	0	0	2	2	2	0	0	21	0	3
reggae	1	0	1	1	0	0	0	1	25	1
rock	2	0	3	2	1	0	1	3	1	17

Figure 2 compares the confusion matrices of the best performing CNN model (CNN-7) and XGB, the best model among the feature-engineered classifiers. Both models seem to be good at classifying all genres except 'Rock' music where many instances are often confused with classes 'Country' and 'Blues'. This can be explained as Rock music has its roots in 1940s and 1950s rock and roll, a style which drew heavily from the genres of blues, rhythm and blues, and from country music according to Wikipédia[9]. Overall, some songs may fall into multiple genres.

[9]https://en.wikipedia.org/wiki/Rock_music

4 Music Mood Classification

Whereas music genre was a single-label task, music mood classification is a multi-label classification task, meaning that each data point can have one or more tags. Moreover, contrary to the dataset used in the first task, the one employed here has a disproportionate ratio of samples per class. Therefore, to handle this imbalanced dataset, I experiment the performance of CNNs on different datasets resampled from the original to evaluate two resampling techniques.

4.1 Dataset Acquisition

4.1.1 MTG-Jamendo Dataset

MTG-Jamendo Dataset created by Bogdanov et al. [2] is built using music available at Jamendo under Creative Commons licenses and tags provided by content uploaders. The dataset contains over 55,000 full audio tracks with 195 tags from genre, instrument, and mood/theme categories. A subset of this dataset focusing on mood/theme tags is also provided which includes 18,486 songs with mood and theme annotations. There are 59 tags in total, and tracks can possibly have more than one tag. Due to memory limitations, I downloaded 992 tracks (8.4 GB) out of 18,486. All audios are distributed in 320kbps MP3 format.

4.1.2 Input Type and Length

In the original paper [2], log-amplitude mel spectrograms are used as input of the neural networks. I followed the same parameters setting except for the *hop_length* which I increased to reduce the time-axis of mel spectrograms, due to memory limitations again. Each audio slice was converted into a mel spectrogram of 96 frequency bins and 341 frames (96×341) by setting the following parameters:

- Sampling rate (sr) = 12000
- Frame/Window size (n_fft) = 2048
- Time advance between frames (hop_length) = 1024
- Number of Mels Bins (n_mels) = 96
- Window Function: Hann Window
- Highest Frequency ($fmax$) = $sr/2$

4.1.3 Data Preparation

In this section, we describe the process used to prepare the dataset for the music mood classification task.

Tags Processing. Since the subset collected from the MTG-Jamendo Mood Theme Dataset contained both mood and theme annotations, I firstly discarded theme tags based on my own subjectivity, then merged synonyms (fun and funny are considered the same meaning), and finally removed less frequent tags. We obtained at the end 35 tags out of 59 tags.

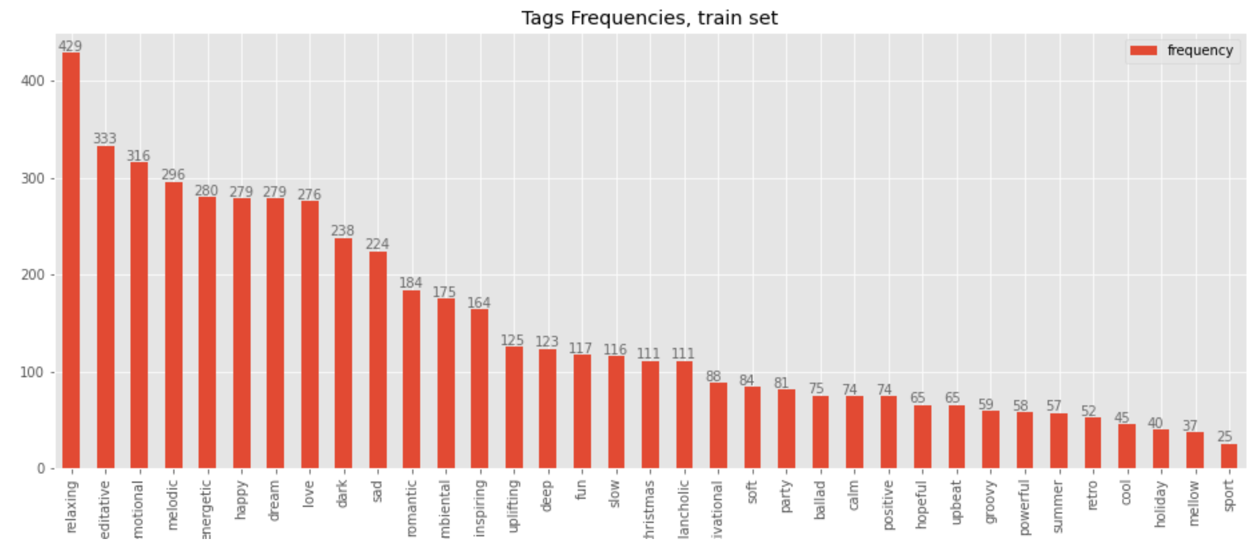
Multiframe Approach for Data Augmentation. In [2], Bogdanov et al. used a centered 29.1s audio segment for each track but since I only had a few samples, I adopted a multiframe strategy half similar to the one used in Section 3.1.3. Contrary to the GTZAN dataset, audio tracks from the MTG-Jamendo dataset don't have the same duration and have different sample rate. To address this problem, all tracks are firstly downsampled to 12kHz as suggested in [3]. Then for each song, I discarded the first and last 20 seconds which are empirically non representative of the song content most of the time, divided the rest into non-overlapping frames of equal time-length 29.1s. Since the songs are of different duration, the last frame is removed if its duration is lower than 29.1s. The number of frames per song are inevitably variable. Lastly, a log-amplitude mel spectrogram using 96 mel bins is computed for each frame song.

Resampling Techniques for Imbalanced Dataset. Figure 3 shows a disproportionate ratio of audios in each class which is problematic because most machine learning algorithms work best when the number of samples are balanced. To handle this imbalance problem, two resampling methods[10] using the resamping module from Scikit-Learn were adopted on the training set.

1. **Oversample minority classes**: Oversampling consists in adding more copies of the minority class, which can be a good choice when we don't have a lot of data to work with.

[10]https://towardsdatascience.com/methods-for-dealing-with-imbalanced-data-5b761be45a18

Figure 3: Mood labels repartition in the training set



2. **Undersample majority classes**: Undersampling can be defined as removing some observations of the minority class. One disadvantage is the possible removal of valuable information, leading to underfitting and poor generalization to the test set.

In addition to the base split separating the dataset in train, validation, and test sets (80/10/10), I generated eight supplemental splits, four based on the upsampling method and four based on the downsampling method. The difference between each split lies in the number of samples to resample for the minority/majority classes. For example in the oversampling case, the parameter that controls this specific number is *upsampling_size* which is the percentage of the maximum frequency tag for the minority classes to match. For instance, if the number of samples of the maximum frequency max_freq_tag is 100 and we fix *upsampling_size* to 0.4, then each class that has a number of observations less than $max\_freq\_tag \times upsampling\_size = 40$ will be resampled to match 40. In the downsampling case, the parameter *downsampling_size* indicates the proportion of the difference between a chosen class and the minority class to remove. For example, if $downsampling\_size = 0.5$, the number of samples in the minority class is 25, the number of samples in the chosen class is 100, then we remove $difference \times downsamling\_size = (100 - 25) \times 0.5 = 37$ observations from the chosen class. All datasets information can be found in Table 5.

Table 5: Different datasets generated from MTG-Jamendo Mood Theme dataset

Training set	Size parameter	Number of samples	Frequency of most frequent tag	Frequency of less frequent tag
Base		3635	429	25
Upsample	0.25	4931	497	111
	0.5	7887	567	218
	0.75	11279	660	321
	1	15015	875	429
Downsample	0.25	3293	319	25
	0.5	2270	218	25
	0.75	1341	117	25
	1	875	50	25

We can note that even though the *downsampling_size* was set to 1 meaning that the number of samples in each class should match the number of songs labelled with the least frequent tag, it is not the fact in reality. This is because songs can have multiple labels, hence when resampling for a specific class, instances from other class can also be resampled. I minimized this phenomena by resamling from samples with a unique label when it is possible. Otherwise the subset representing a label can have instances from other labels, still resulting in imbalanced datasets.

4.2 Models

4.2.1 Convolutional Neural Networks

For this task, I took as a starting point the architecture proposed by Choi et al. [3]. It is a fully convolutional neural network composed of 5 convolutional layers of 3×3 kernels and max-pooling layers as illustrated in Table 6. Since the size of the mel spectrogram input (96×341) differs from [3], I adjusted poolings kernels. ReLU is used as an activation function in every convolutional layer except the output layer, which uses Sigmoid. Batch Normalization is added after every convolution and after activation to accelerate the convergence. Dropout of 0.5 is added after every max-pooling layer to prevent the network from overfitting.

Table 6: CNN architecture for music mood classification

Mood-CNN-5
Mel spectrogram (input: 96×341)
Conv $3 \times 3 \times 32$
MP(2,4) (output: $48 \times 85 \times 32$)
Conv $3 \times 3 \times 64$
MP(2,4) (output: $24 \times 21 \times 128$)
Conv $3 \times 3 \times 128$
MP(2,4) (output: $12 \times 7 \times 128$)
Conv $3 \times 3 \times 256$
MP(2,2) (output: $4 \times 1 \times 256$)
Conv $3 \times 3 \times 512$
MP(3,3) (output: $1 \times 1 \times 512$
Output 35×1 (sigmoid)

4.3 Experiments and discussions

4.3.1 Implementation details

Dealing with large datasets. I followed the same process described in Section 3.3.1 with Python generators.

Training details. In all experiments, model training is implemented with Adam, starting with a learning rate of $1 \exp -3$. Two callbacks are used: the first one is *ReduceLROnPlateau* to reduce the learning rate by a factor of 0.1 if the validation AUC does not improve and the second one is $ModelCheckpoint$ to save all models at each epoch. Models are trained for 15 epochs with a mini batch of 128 samples. The best models are then choose across the different epochs to finally carry out evaluation. In order to evaluate the performance of the models with the different types of datasets, AUC score is used during training as it is robust to imbalanced datasets. A random guess scores an AUC of 0.5 while a perfect classification 1.0, so we expect our system to score an AUC higher than 0.5.

Test details The predictions are done using the *predict* method of Keras $Model$ class which outputs the probabilities for the input samples to belong to each class. Then the predictions array of shape ($n\_samples, n\_classes$) is one-hot encoded: if the probability is greater than 0.9 then the value is fixed to 1, otherwise it is fixed to 0. The threshold greatly influence the evaluation metrics. Finally, the ROC AUC score is computed with parameter $average$ = "*samples*" which is only meaningful for multilabel classification according to Scikit-learn.

Table 7: Performance comparison using datasets with different resampling methods

	AUC
Base	**0.619**
Upsample Training Set	
Upsample, 0.25	0.565
Upsample, 0.5	0.505
Upsample, 0.75	0.522
Upsample, 1	0.500
Downsample Training Set	
Downsample, 0.25	0.506
Downsample, 0.5	0.499
Downsample, 0.75	0.512
Downsample, 1	0.500

4.3.2 Results

Table 7 summarizes the results of the experiments of different resampling methods applied to the training set with various sampling size as described in Section 4.1.3. Unexpectedly, the model trained on the base training set shows the best performance with an ROC AUC score of 0.619. In the end, the resampling methods were useless as they lead to worse performance, and the majority of them performs no better than a popularity baseline that always predict the most frequent tag among tracks in the training set. Unfortunately, I cannot explain these results. There might be a problem in one of the steps that I carried out, either during the data prepartion, the CNN architecture, or the threshold value and the chosen metrics.

5 Conclusion

We presented two tasks of music classification using deep convolutional neural networks. For the first task focusing on classifying music per genre, the experiments were carried with the GTZAN dataset. In Experiment I, it was shown that using mel spectrograms resulted in better performance compared to spectrograms and MFCCs. In Experiment II, different number of layers were evaluated on a bigger dataset which contains approximately twice the number of mel spectrograms. The optimal number of layers were found to be different than in the first experiment. Furthermore, CNN-based models were shown to outperform feedforward neural networks and feature-engineered classifiers that were trained based on time domain and frequency domain features from the audio signals. For the second task on music mood classification, a subset of the MTG-Jamendo Mood Theme dataset was used. Since the dataset was imbalanced, two resampling methods were applied on the training set to rebalance the tags repartition. Ultimately, the best result was obtained with the unmodified training set but overall, the results were not conclusive.

This internship was an opportunity for me to learn about deep learning, as this field was unfamiliar to me before I worked on these music classification tasks. I learned a lot of things about neural networks and especially about convolutional neural networks and how they could be applied for music information retrieval [4]. I also learned how to prepare a dataset and deal with large datasets, choosing the 'right' metrics for models evaluation, but most importantly, the importance of understanding your data. The latter is indeed a crucial step that affects the way in which the algorithms are designed and the interpretation of the results afterwards. For instance, I was unable to interpret the results I obtained for the classification of music by mood, mainly because I lacked understanding of the initial data. I learned a bit about signal processing in order to find the best representation of the audio data for the two tasks but it is not enough. Finally, the choices that I made in every step such as the methods used for each dataset acquisition, the CNNs architectures, the evaluation metrics, and so on, clearly affect the final performance. However, knowing in which way these choices influence the final results is something that remains mysterious for me. I certainly made some mistakes which I am not aware of. This is the reason why I hope to learn more about signal processing, music, and deep learning in the future to get a better and in depth understanding of the classification process, and eventually improve my experiments.

References

[1] Wenhao Bian, Jie Wang, Bojin Zhuang, Jiankui Yang, Shaojun Wang, and Jing Xiao. “Audio-Based Music Classification with DenseNet and Data Augmentation”, PRICAI 2019: Trends in Artificial Intelligence, Lecture Notes in Computer Science, vol. 11672, Springer, Cham, 2019.

[2] Bogdanov, D., Won M., Tovstogan P., Porter A., and Serra X. “The MTG-Jamendo Dataset for Automatic Music Tagging”, Machine Learning for Music Discovery Workshop, International Conference on Machine Learning, ICML, 2019.

[3] Keunwoo Choi, George Fazekas, and Mark Sandle. “Automatic Tagging Using Deep Convolutional Neural Networks”, *arXiv:1606.00298*, 2016.

[4] Keunwoo Choi, George Fazekas, and Kyunghyun Cho. “A Tutorial on Deep Learning for Music Information Retrieval”, *arXiv:1709.04396*, 2018.

[5] François Chollet. *Deep Learning with Python*, Manning, p. 174, 2017.

[6] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. “Deep Residual Learning for Image Recognition”, *arXiv:1512.03385*, 2015.

[7] George Tzanetakis, and Perry Cook. “Musical genre classification of audio signals”, IEEE Transactions on Audio and Speech Processing, vol. 10, no. 5, pp. 293-302, 2002.