Major and Minor Chords Classification

Nicola Manca
Student
Department of Computer Science
University of Milan

***Abstract*—In this project I analyze the results of four machine learning algorithms, when learning to classify the type of chords (Major or Minor) played in audio files from the dataset "Musical Instrument Chord Classification". First I focus on the best audio features to extract from the files, I choose two of them (MFCC and chromagram), then I perform k-means to cluster the feature spaces, and finally I test different machine learning algorithms and analyze their performances. I find that some ML models work well with a certain feature and poorly with the other, while others don't have a 'preference'.**

I. Introduction

The problem of distinguishing between major and minor chords is very specific, but can have a wide range of applications in the music field. In the past two decades a lot of musical software has been developed, particularly for didactic purpose (e.g. smartphone applications for learning an instrument), and extracting this type of information from a raw audio recording or a finished track can be a useful part of the so-called Music Information Retrieval (MIR) process, and more specifically Automatic Chord Recognition (ACR). Some popular non-machine-learning methods are the template-matching approaches, based on the creation of chord templates in the form of binary chroma vectors, i.e. a 12-element array for the 12 pitch classes of western music with a 1 in the positions of the notes played in a chord, and 0 in the others; then a similarity measure between a sample's extracted chroma vector and the templates is used to recognize which chord is being played. There's also been some work with machine learning methods for ACR, mainly deep learning, using different types of models like MLPs [1] [2], CNNs, autoencoders, but also non-NN models like k-Nearest Neighbors [2].
Although my problem is a simpler one, in fact recognizing if a chord is major or minor turns the multi-class classification problem of ACR into a binary classification problem. I chose this task because it's more generic and can be useful on its own. Initially I discovered that the two classes were unbalanced in favor of the 'major' class, with 145 samples more than the other, so I decided to randomly remove the excess files to prevent the presence of a bias in the classification models. Next I focused on selecting the audio features to be used for classification, I chose the chromagram (or chroma vector) because it contains information about which notes are playing and is widely used for this type of task, and the MFCCs for their popularity in the MIR field. Then I used k-means to cluster each feature, after applying PCA to reduce the dimensionality, and lastly I trained and evaluated four machine learning algorithms (two traditional and two deep-learning ones) separately with both features. I analyzed the results of all combinations of features and models to find the differences.

II. Method and Experimental set-up

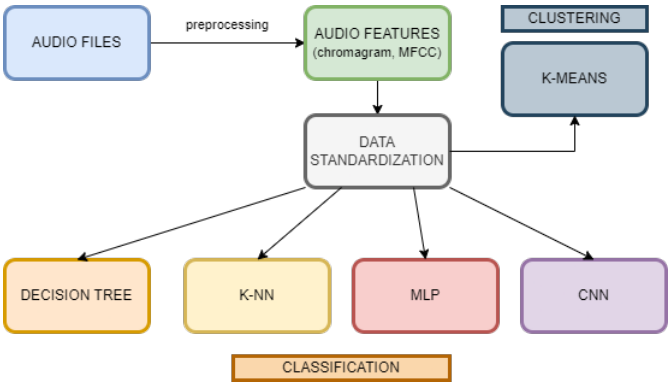

Fig. 1. Block diagram of the method.

The workflow of the project consists of four main steps, as shown in the block diagram:

- Data Preprocessing
- Feature Extraction
- Clustering with k-means
- Classification with 4 different ML architectures

A. Dataset

I found this dataset on Kaggle [3], it's called "Musical Instrument Chord Classification" and was uploaded on February 2022 by an indian user who goes by '*Deep Contractor*'. It contains a total of 859 audio files, each one is a recording of a major or minor chord played either on an acustic guitar or on a piano. They're all in the wave format, about 2 seconds long (with small differences), and divided into two classes: *Major* has 502 entries, and *Minor* has 357 entries. To achieve better performances on the ML models, I decided to balance the dataset, so I eliminate 145 excess samples from the *Major* class at random.

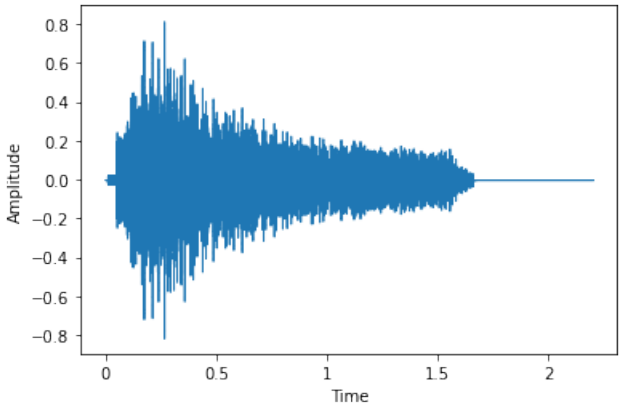

Fig. 2. Example of waveplot of an audio sample from the dataset.

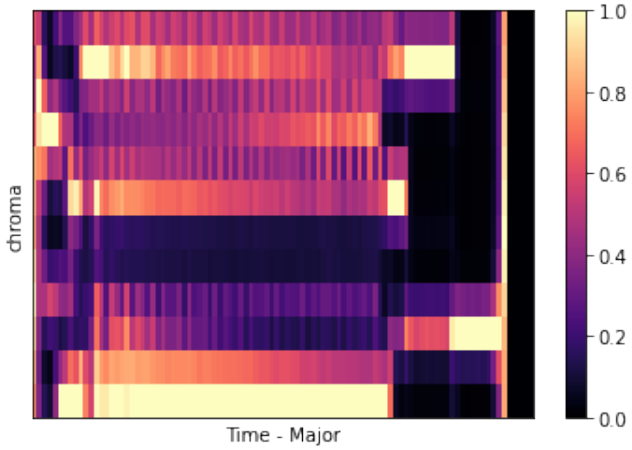

Fig. 3. Example of chromagram of a 'Major' audio sample.

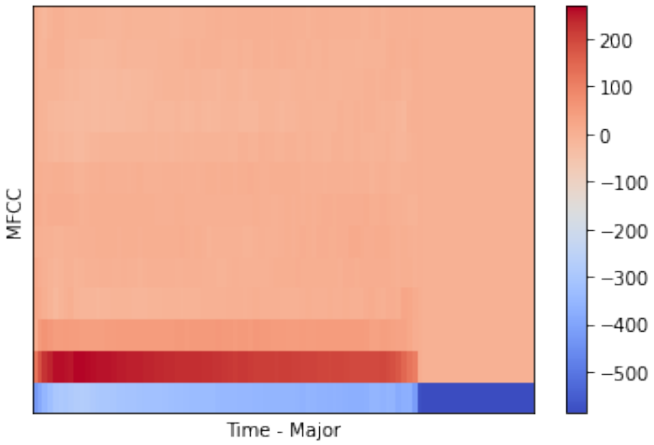

Fig. 4. Example of MFCCs of a 'Major' audio sample.

B. Feature extraction

Audio files on their own are not sufficient to train a predictor, without previously extracting some features, because of the volume of samples and their noisiness. For this reason it's common to choose one or more features to be extracted from the audio files, depending on the specific task.

I select two features, chromagram and MFCCs, they're both computed in the frequency domain which means that a short-time Fourier Transform is computed first, dividing the audio files into frames. I decided to compare the different classifiers' performances when using each of these features.

For chord recognition, the chromagram is the most popular feature, some works even aimed at optimizing the extraction quality of chromagrams using neural networks (Korzeniowski & Widmer [4]), however I decided to use a method from the famous library for music and audio analysis *librosa* [5] to compute a chromagram from a waveform. A chromagram comprises a time-series of chroma vectors, which represent harmonic content at a specific time in the audio as $c \in \mathbb{R}^{12}$, in which each c_i stands for a pitch class, and its value indicates the current saliency of the corresponding pitch class.

The Mel-frequency Cepstral Coefficients (MFCCs) are one of the most popular features in the MIR field, typically only the first 13 coefficients are used, as they are considered to carry enough discriminative information in various classification tasks. An important fact about the MFCCs is that they're computed in the Mel scale, which is a logarithmic scale that simulates the human's perception of the pitch scale. I use a method from *librosa* [5] also to compute this feature.

C. Clustering

Before clustering the data, I flatten it to one dimension, normalize it, using the function *StandardScaler* from *sklearn* [6], and then I apply PCA (Principal Component Analysis), a dimensionality reduction method. At this point I use k-means to cluster the features and visualize the results in a two dimensional graph. K-means is an unsupervised machine learning algorithm which groups similar data points into a predetermined number of clusters, it works by computing each cluster's centroid and assigning every data point to the nearest one, until the centroids stabilize. The number of centroids can be computed using various methods (e.g. the elbow method), but in this case I simply set it to two, because there are two classes in the data.

D. Classification

Finally, I train and evaluate eight ML models (from four different architectures, each one having a MFCC version and a chromagram version), to compare every combination of architectures and features. I choose to experiment with Decision Trees and k-Nearest Neighbors (using the implementation provided by *sklearn* [6]), which are 'traditional' (non deep learning) supervised machine learning models, along with Multi-layer Perceptrons and Convolutional Neural Networks (using the implementation provided by *tensorflow* [7] and *keras* [8]), that are deep supervised models and for which I also perform model selection (i.e. hyperparameter tuning).

To evaluate the performance of the first two I split the dataset into a train set and a test set, then use 10-fold cross validation on the train set and obtain the mean and standard

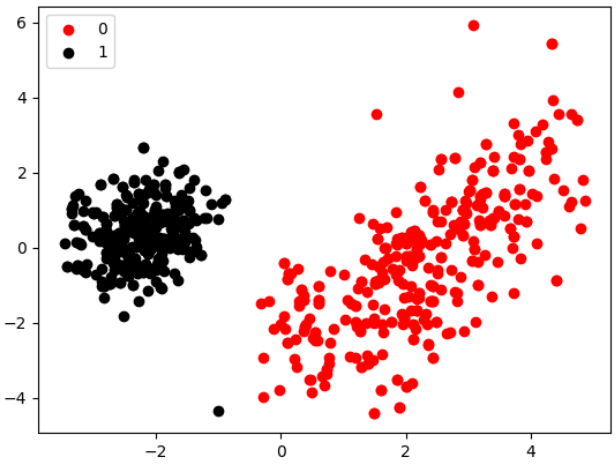

Fig. 5. K-means clustering on MFCCs feature data.

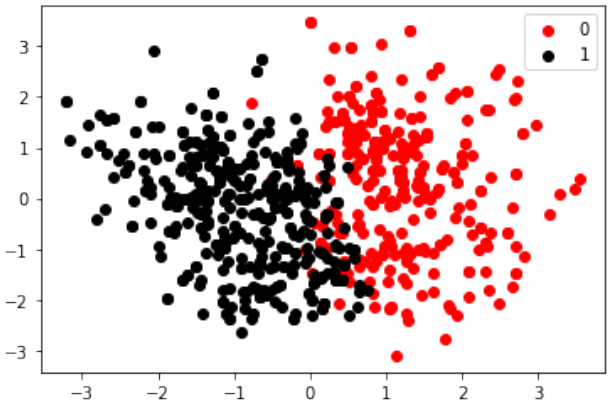

Fig. 6. K-means clustering on chromagram feature data.

deviation of the accuracy of the 10 models, ultimately I train a model and test it using the test set, to show its generalization capability.

While for the latter two (MLP and CNN) I divide the dataset into a train set, a validation set and a test set. I select the best set of parameters by performing hyperparameter tuning (with the *GridSearchCV* function from *sklearn* [6]) using the train set, then I train a model accordingly (also using the validation set) and evaluate it on the test set.

III. Results

In this section I present the empirical results of the experiments with the four classes of models. For each one, I first talk about the version trained on MFCCs, then the chromagram version. I explain the results of the 10-fold cross validation for the decision tree and k-NN models, and I present the results of hyperparameter tuning for the MLP and CNN models. Then I show, for every experiment, the resulting confusion matrix for the test set, and the ROC curve, with relative AUC value. In the case of decision tree models, I do not present the ROC curve graphs because it would only contain one point, as these models only output a categorical prediction.

A. Decision Tree

The decision tree models reach somewhat satisfying results. Both models (MFCCs and chromagram) have similar performances. For the MFCCs one, the 10-fold cross validation averages *71%* accuracy with *0.05* standard deviation, having some runs that achieve *80%*. The presented confusion matrix is taken from a run with *78%* accuracy, and from that we can see that the model fails the most when classifying Major samples. While for the chromagram model, the cross validation averages *71%* accuracy with *0.07* standard deviation, and some runs that achieve over *80%* accuracy. The presented confusion matrix is also taken from a run with *78%* accuracy, and in this case the model seems to misclassify more the Minor samples.

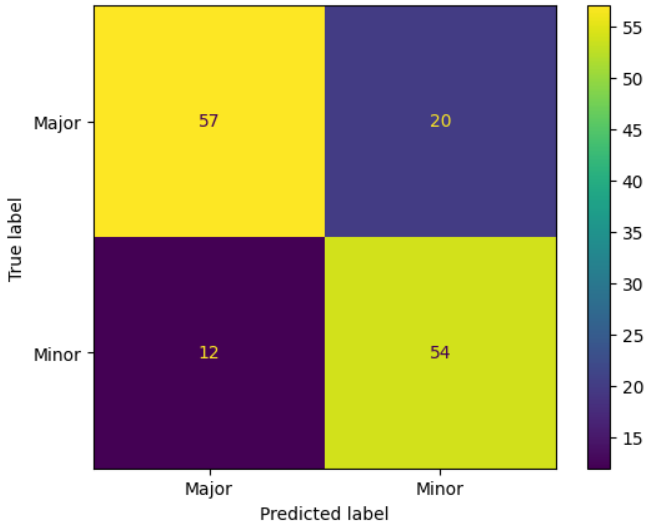

Fig. 7. Confusion matrix for Decision Tree model trained on MFCCs.

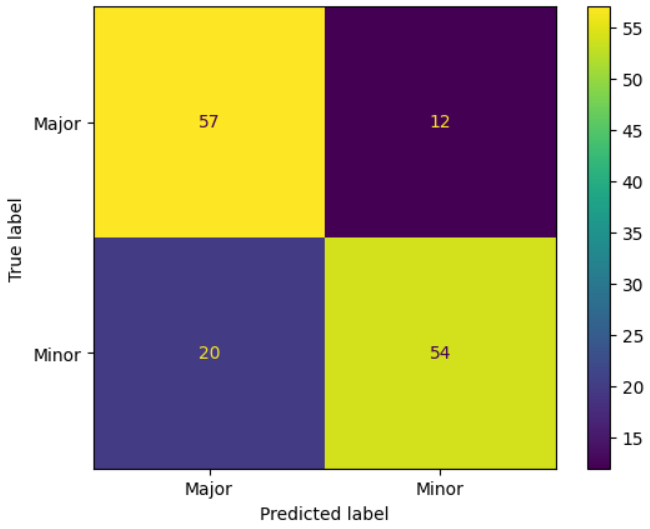

Fig. 8. Confusion matrix for Decision Tree model trained on chromagrams.

B. k-Nearest Neighbor

The k-NN models have poorer performances, with respect to the decision trees, in fact both models struggle to reach *70%* accuracy. For the MFCCs one, the 10-fold cross validation averages *63%* accuracy with *0.05* standard deviation. While for the chromagram model, the cross validation averages *63%* accuracy with *0.06* standard deviation. Here the confusion matrices and ROC curves are related to a MFCCs model with *70%* and a chromgram model with *64%* accuracy. One relevant difference is the AUC value, which is of *0.76* for the MFCCs model and *0.69* for the chromgram model. Also, from the two confusion matrices we can see that both models have more difficulty when classifying the Minor samples.

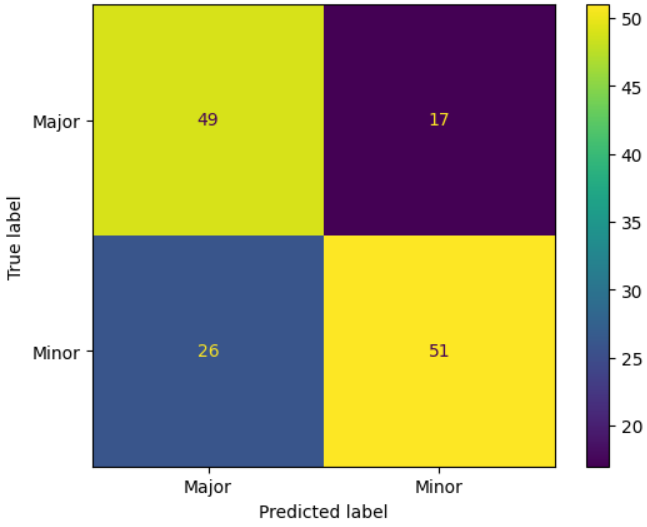

Fig. 9. Confusion matrix for k-Nearest Neighbor model trained on MFCCs.

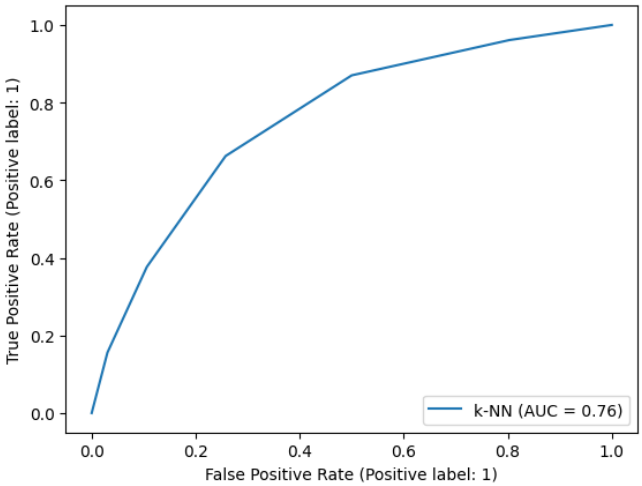

Fig. 10. ROC curve for k-Nearest Neighbor model trained on MFCCs.

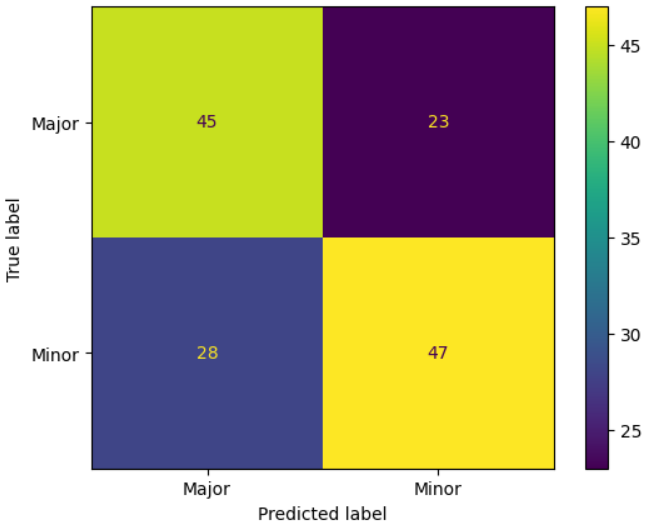

Fig. 11. Confusion matrix for k-Nearest Neighbor model trained on chromagrams.

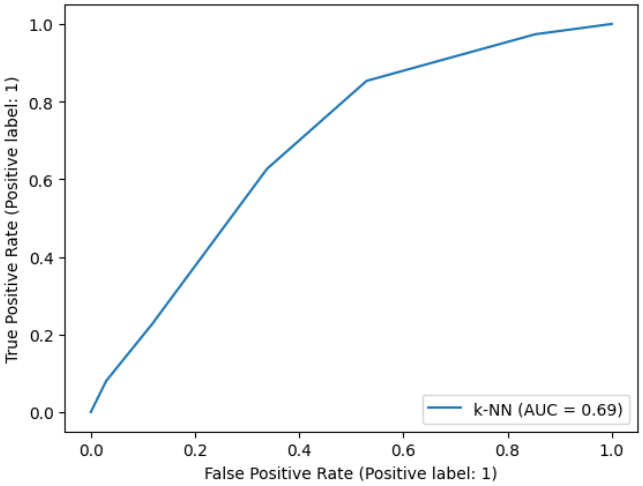

Fig. 12. ROC curve for k-Nearest Neighbor model trained on chromagrams.

C. Multi-layer Perceptron

The MLPs have significant differences in performances between their MFCCs and chromagram versions. Starting with the MFCCs model, the grid search optimization selected the following best performing hyperparameters:

- dropout rate: 0.3
- number of neurons for 1st layer: 32
- number of neurons for 2nd layer: 64
- number of neurons for 3rd layer: 64
- regularization: L1 and L2

This model reaches *58%* accuracy, and a AUC value of *0.59*. It seems to struggle a lot with samples from the Minor class, in fact its accuracy for the class is less than *50%*.

The chromagram model performs way better, the best hyperparameters selected by the grid search were:

- dropout rate: 0.1
- number of neurons for 1st layer: 256
- number of neurons for 2nd layer: 128
- number of neurons for 3rd layer: 32
- regularization: none

This model reaches *70%* accuracy, and a AUC value of *0.76*. It misclassifies the Major samples more than the Minor ones, but doesn't come close to its MFCCs counterpart's rate of errors.

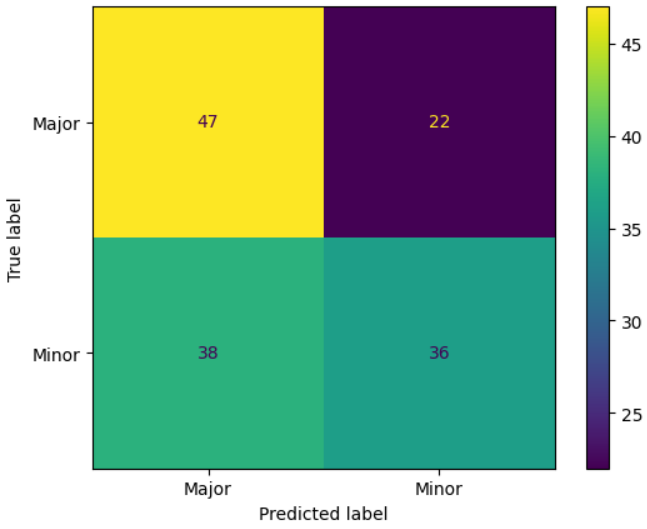

Fig. 13. Confusion matrix for MLP model trained on MFCCs.

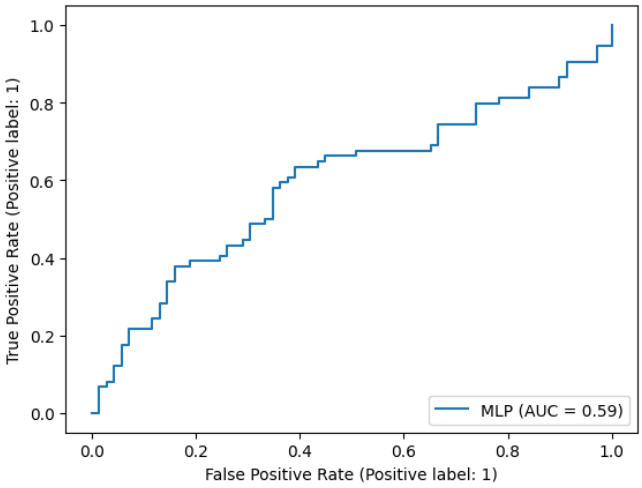

Fig. 14. ROC curve for MLP model trained on MFCCs.

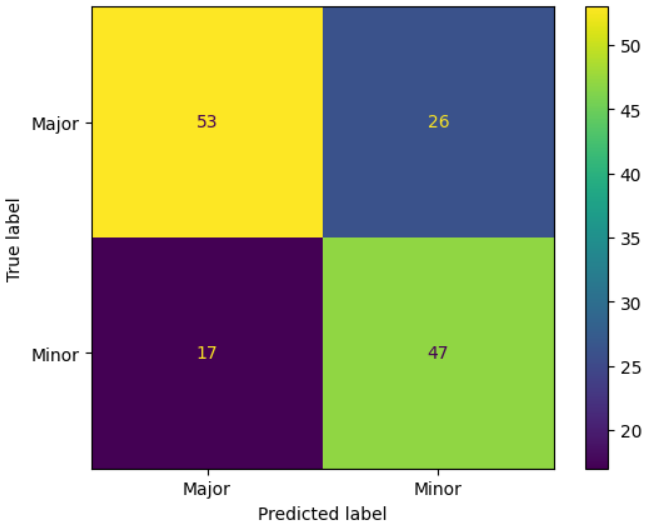

Fig. 15. Confusion matrix for MLP model trained on chromagrams.

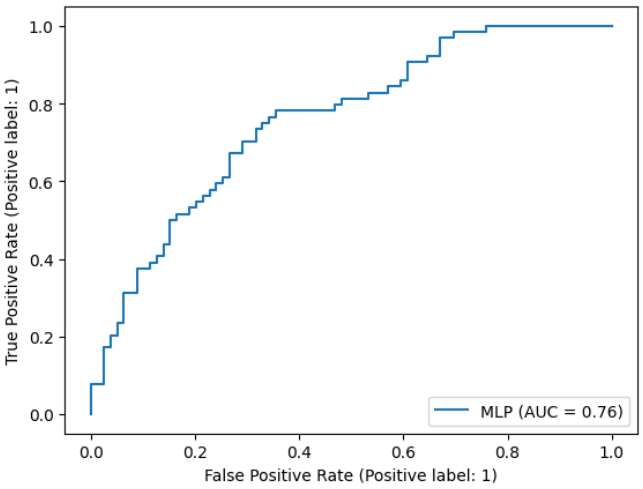

Fig. 16. ROC curve for MLP model trained on chromagrams.

D. Convolutional Neural Network

The CNNs have similar performances for the MFCCs and chromagram versions. Starting with the MFCCs model, the grid search optimization selected the following best performing hyperparameters:

- number of filters: 32
- size of filters: 3×3
- number of neurons in dense layer: 16

This model reaches *69%* accuracy, and a AUC value of *0.78*. It seems to miscalssify samples from the Major class more than the others, although the difference is small.

The chromagram model performs slightly better, the best hyperparameters selected by the grid search were:

- number of filters: 128
- size of filters: 4×4
- number of neurons in dense layer: 16

This model reaches *73%* accuracy, and a AUC value of *0.80*. It's performance on the two classes seems to be balanced. Also, looking at the AUC value, this is the best performing model, however the decision trees reached a higher accuracy.

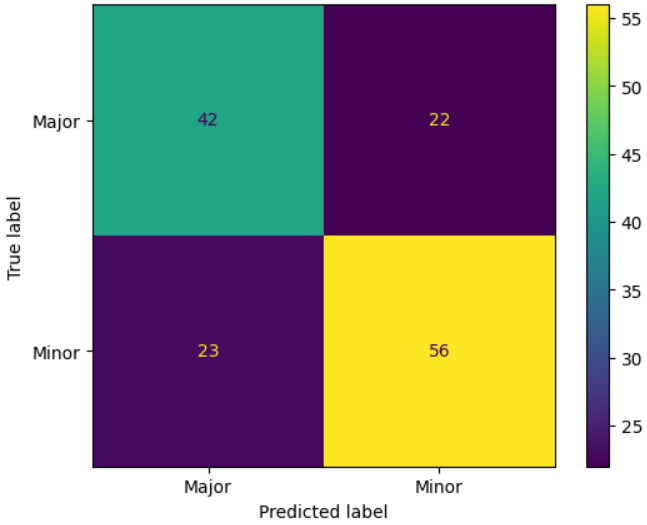

Fig. 17. Confusion matrix for CNN model trained on MFCCs.

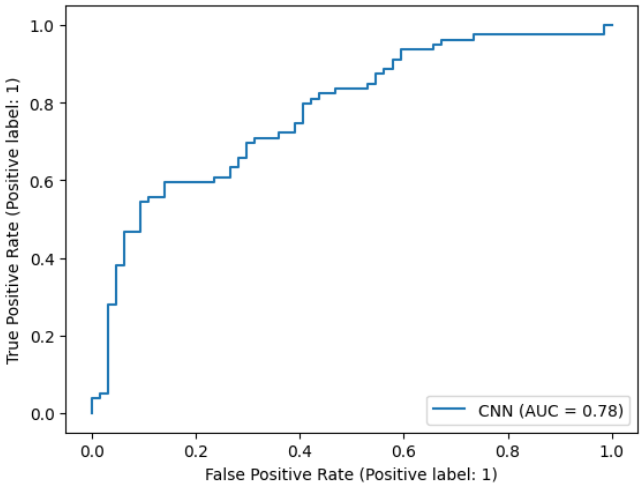

Fig. 18. ROC curve for CNN model trained on MFCCs.

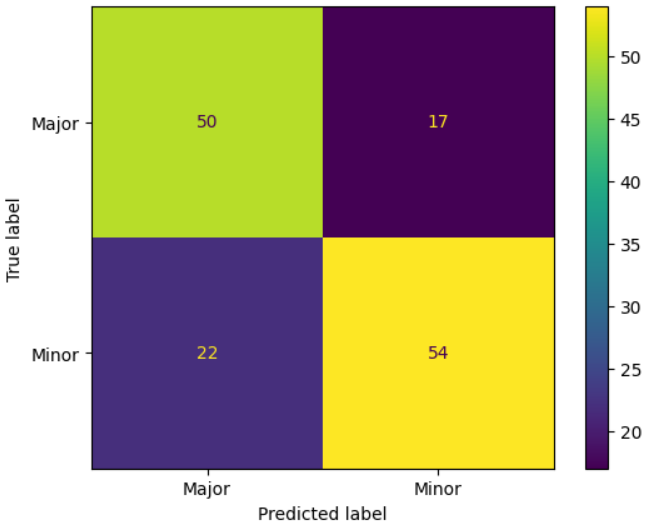

Fig. 19. Confusion matrix for CNN model trained on chromagrams.

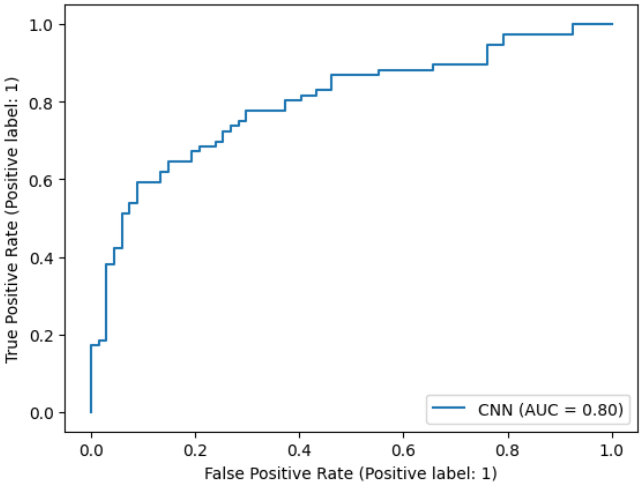

Fig. 20. ROC curve for CNN model trained on chromagrams.

IV. Conclusions

In this work, I have shown how features like MFCC and chromagram can be extracted from audio files containing chords played with a guitar or a piano, and used to train machine learning models to classify these chords as major or minor. I experimented with both 'traditional' and deep learning architectures, and presented their performances, from which I can tell that deep learning approaches are not necessarily better performing than others, despite their increased complexity and capability (shown in different other tasks). Also, the results show that both features (MFCC and chromagram) can lead to the same performances, especially with non-deep learning approaches, even though the MFCC seemed to better divide the datapoints in the clustering analysis. While this experiments aimed at exploring the possibility of exploiting complex features and ML approaches for the major and minor

chords classification task, the performances reached by these methods showed that they're not superior to more specific methods, like the use of features extracted from harmonics (and the intervals between them), presented on kaggle by the user *ahmetcelik158* [9], which obtained far better results.

REFERENCES

[1] Alessandro Bonvini, "Automatic chord recognition using Deep Learning techniques" Politecnico di Milano, 2014.
[2] Osmalskyj, Embrechts, Droogenbroeck, Piérard, "Neural networks for musical chords recognition", 2012.
[3] Kaggle, https://kaggle.com
[4] Korzeniowski and Widmer, "Feature Learning for Chord Recognition: the Deep Chroma Extractor", 2016
[5] Librosa, https://librosa.org
[6] scikit-learn, https://scikit-learn.org
[7] tensorflow, https://tensorflow.org
[8] keras, https://keras.io
[9] Ahmet Celik's approach, https://www.kaggle.com/code/ahmetcelik158/mathematics-of-music-chord-classification