SAPIENZA
UNIVERSITÀ DI ROMA

Machine Learning Project
Prof: Paola Velardi

Music genre classification using Deep Neural Networks

Valerio Vinciarelli - 1193280

1. Introduction

Nowadays, companies like Spotify use music classification to be able to recommend playlists or songs to their customers so that they can reflect their tastes, or companies like Shazam use music classification to be able to recognize a certain song that the customer is listening to and whose name he wants to know.
Determining musical genres is the first step in that direction. Machine learning techniques have proven effective in extracting trends and patterns from the large data pool. The same principles are also applied in musical analysis.

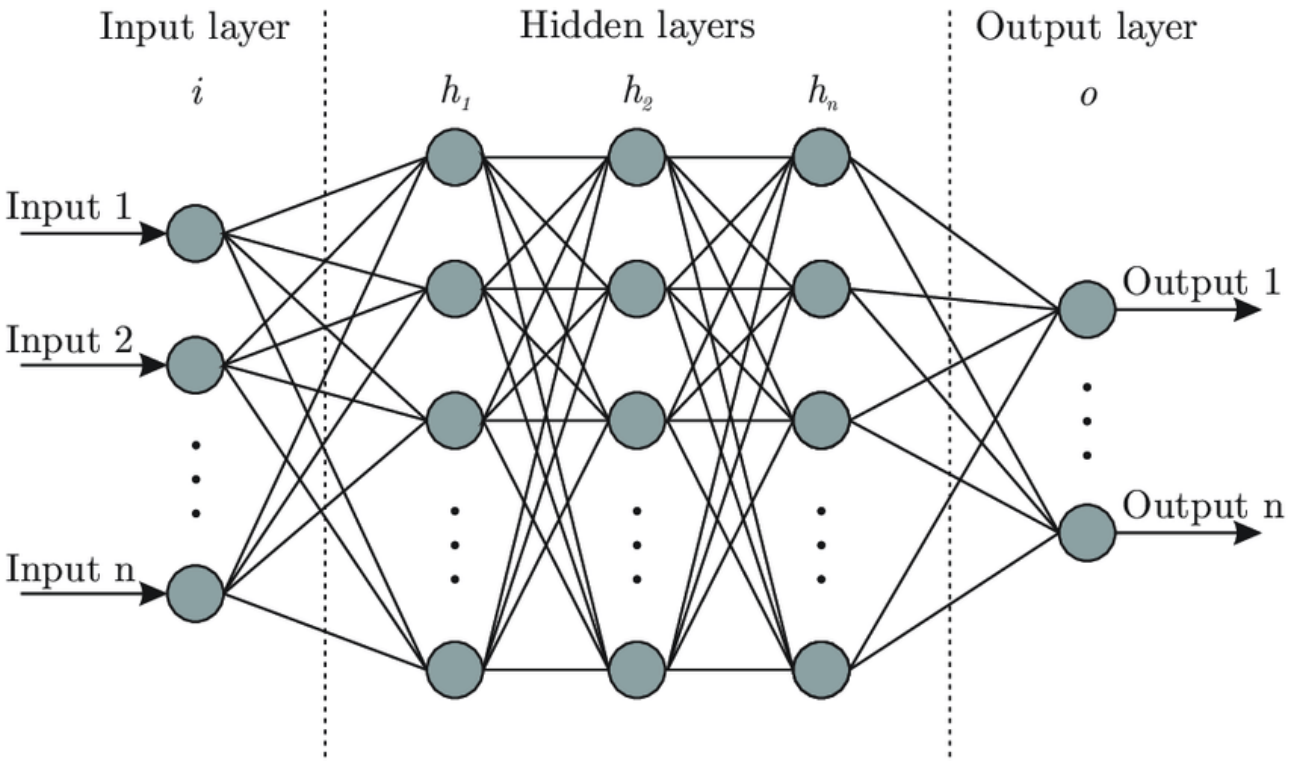
Sound is represented in the form of an audio signal having parameters such as frequency, bandwidth, decibel etc. A typical audio signal can be expressed as a function of Amplitude and Time

These sounds are available in many formats which makes it possible for the computer to read and analyze them.
Here some examples of audio formats:

- MP3 (Moving Picture Expert Group-1/2 Audio Layer 3)
- WAV (Waveform Audio File)
- FLAC (Free Codec Lossless Audio)
- OGG

2. Neural Networks

A neural network is based on a collection of connected units or nodes called artificial neurons, which freely shape neurons in a biological brain. Each connection can transmit a signal to other neurons. An artificial neuron that receives a signal, processes it and can signal the neurons connected to it. The "signal" in a connection is a real number and the output of each neuron is calculated from a non-linear function of the sum of its inputs. The connections are called edges. Neurons and edges typically have a weight that adjusts as learning progresses. Weight increases or decreases the strength of the signal to a connection. Neurons can have a threshold such that a signal is only sent if the aggregate signal exceeds that threshold. Typically, neurons are aggregated in layers. Different levels can perform different transformations on their inputs. The signals travel from the first level (the input level) to the last level (the output level), possibly after crossing the levels several times.



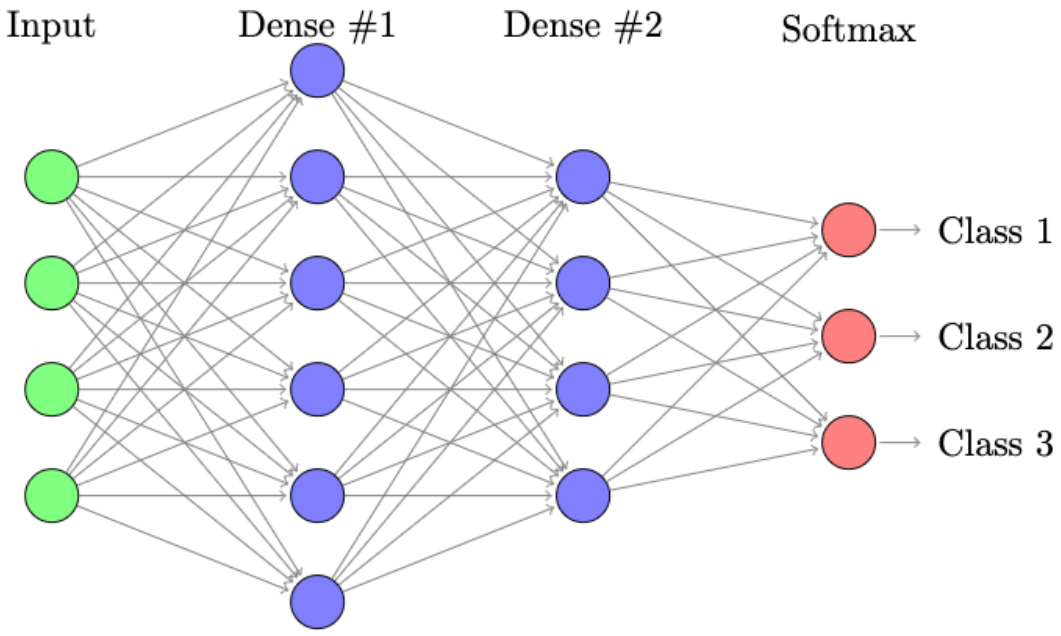
2.1. Deep Neural Networks

A deep neural network (DNN) is a neural network with multiple layers between the input and output layers. The DNN finds the correct mathematical manipulation to turn the input into the output, whether it be a linear relationship or a non-linear relationship. The network moves through the layers calculating the probability of each output. For example, a DNN that is trained to recognize dog breeds will go over the given image and calculate the probability that the dog in the image is a certain breed. The user can review the results and select which probabilities the network should display (above a certain threshold, etc.) and return the proposed label. Each mathematical manipulation as such is considered a layer, and complex DNN have many layers, hence the name "deep" networks.

DNNs can model complex non-linear relationships. DNN architectures generate compositional models where the object is expressed as a layered composition of primitives. The extra layers enable composition of features from lower layers, potentially modeling complex data with fewer units than a similarly performing shallow network.

Deep architectures include many variants of a few basic approaches. Each architecture has found success in specific domains. It is not always possible to compare the performance of multiple architectures, unless they have been evaluated on the same data sets.

DNNs are typically feedforward networks in which data flows from the input layer to the output layer without looping back. At first, the DNN creates a map of virtual neurons and assigns random numerical values, or "weights", to connections between them. The weights and inputs are multiplied and return an output between 0 and 1. If the network did not accurately recognize a particular pattern, an algorithm would adjust the weights.
That way the algorithm can make certain parameters more influential, until it determines the correct mathematical manipulation to fully process the data.



3. Implementation

In order to develop this Machine Learning project, I used the following libraries:

1. **Librosa**: It is a Python module to analyze audio signals in general but geared more towards music. It includes the nuts and bolts to build a MIR (Music information retrieval) system.
2. **Keras**: is an open-source neural-network library written in Python. It is capable of running on top of TensorFlow, Microsoft Cognitive Toolkit, R, Theano, or PlaidML. Designed to enable fast experimentation with deep neural networks, it focuses on being user-friendly, modular, and extensible. It was developed as part of the research effort of project ONEIROS (Open-ended Neuro-Electronic Intelligent Robot Operating System), and its primary author and maintainer is François Chollet, a Google engineer. Chollet also is the author of the XCeption deep neural network model.

3. **Sklearn**: is a free software machine learning library for the Python programming language. It features various classification, regression and clustering algorithms including support vector machines, random forests, gradient boosting, k-means and DBSCAN, and is designed to interoperate with the Python numerical and scientific libraries NumPy and SciPy.

3.1. Dataset

GTZAN+ is a music database composed of 15 music genres. It is an extension of the GTZAN database composed of 10 music genres augmented by the 5 following Afro music genres: Bikutsi, Makossa, Bamileké, Salsa and Zouk. Each genre in GTZAN+ is represented by 100 ".wav" files of 30 seconds each (22050 Hz, 32-bit floating point, mono). Bikutsi, Makossa and Bamileke are Cameroonian traditional music genres. Zouk is French-Carabbean music genre and Salsa is a music genre having Cuban origins, but many salsa songs played by Cameroonian singers have been included in the database.

The url to download the GTZAN+ dataset is the follow:
https://perso-etis.ensea.fr//sylvain.iloga/GTZAN+/download/gtzan+.rar

I did a little change to the folder hierarchy; the updated zip is available here:
https://drive.google.com/file/d/1QpavfbzTmUbtIONXNhyawkhlhIPWJR3i/view?usp=sharing

4. Features extraction

Every audio signal consists of many features. However, we must extract the characteristics that are relevant to the problem we are trying to solve. The process of extracting features to use them for analysis is called feature extraction.
All the audio files get converted into their respective spectrograms. We can now easily extract features from them.
The features to be analyzed and exported are the following:

- Zero Crossing Rate
- Spectral Centroid
- Spectral Rolloff
- MFCC
- Chroma

All the features are then appended into a .csv file so that classification algorithms can be used.

4.1. Zero Crossing Rate

The zero-crossing rate is the rate of sign-changes along a signal, i.e., the rate at which the signal changes from positive to negative or back. This feature has been used heavily in both speech recognition and music information retrieval. It usually has higher values for highly percussive sounds like those in metal and rock.

The zero-crossing rate is the rate of sign-changes along a signal, i.e., the rate at which the signal changes from positive to zero to negative or from negative to zero to positive. This

feature has been used heavily in both speech recognition and music information retrieval, being a key feature to classify percussive sounds.

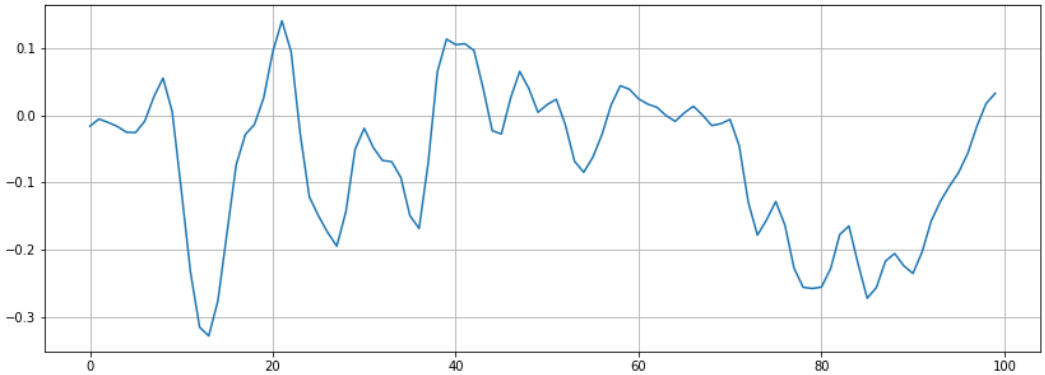
ZCR is defined formally as

$$zcr = \frac{1}{T-1}\sum_{t=1}^{T-1} 1_{\mathbb{R}_{<0}}(s_t s_{t-1})$$

where is a signal of length T and $1_{R<0}$ is an indicator function.

In some cases, only the "positive-going" or "negative-going" crossings are counted, rather than all the crossings - since, logically, between a pair of adjacent positive zero-crossings there must be one and only one negative zero-crossing.

For monophonic tonal signals, the zero-crossing rate can be used as a primitive pitch detection algorithm.



1 Zero-Crossing rate for file rock00001

4.2. Spectral Centroid

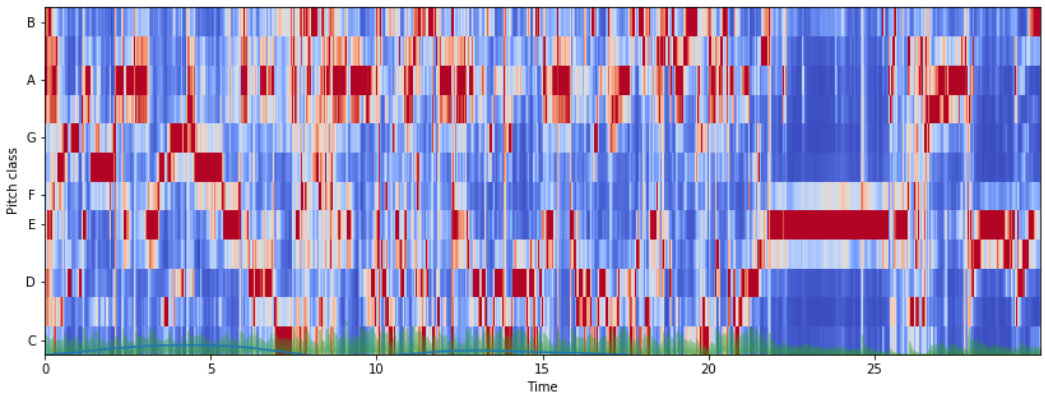
It indicates where the “center of mass” for a sound is located and is calculated as the weighted mean of the frequencies present in the sound. Consider two songs, one from a

blues genre and the other belonging to metal. Now as compared to the blues genre song which is the same throughout its length, the metal song has more frequencies towards the end. So spectral centroid for blues song will lie somewhere near the middle of its spectrum while that for a metal song would be towards its end.

It is calculated as the weighted mean of the frequencies present in the signal, determined using a Fourier transform, with their magnitudes as the weights:

$$\text{Centroid} = \frac{\sum_{n=0}^{N-1} f(n)x(n)}{\sum_{n=0}^{N-1} x(n)}$$

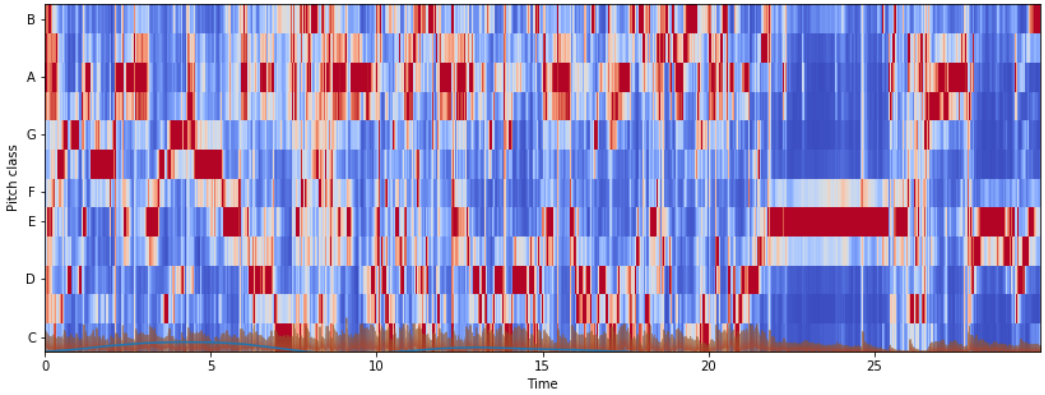
where x(n) represents the weighted frequency value, or magnitude, of bin number n, and f(n) represents the center frequency of that bin.



2 spectral centroid for file rock00001

4.3. Spectral Rolloff

It is a measure of the shape of the signal. It represents the frequency below which a specified percentage of the total spectral energy, e.g. 85%, lies.



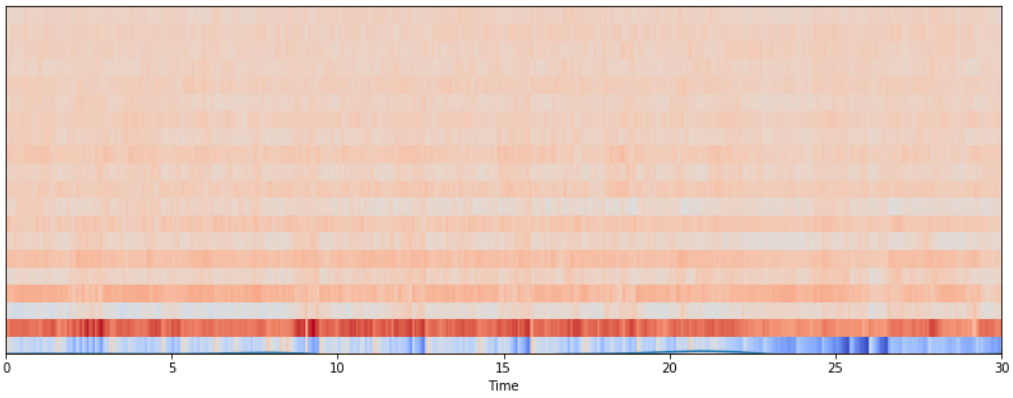
3 spectral rolloff for file rock00001

4.4. MFCC

The Mel frequency cepstral coefficients (MFCCs) of a signal are a small set of features (usually about 10–20) which concisely describe the overall shape of a spectral envelope. It models the characteristics of the human voice.

Mel-frequency cepstral coefficients (MFCCs) are coefficients that collectively make up an MFC. They are derived from a type of cepstral representation of the audio clip (a nonlinear "spectrum-of-a-spectrum"). The difference between the cepstrum and the mel-frequency cepstrum is that in the MFC, the frequency bands are equally spaced on the mel scale, which approximates the human auditory system's response more closely than the linearly-spaced

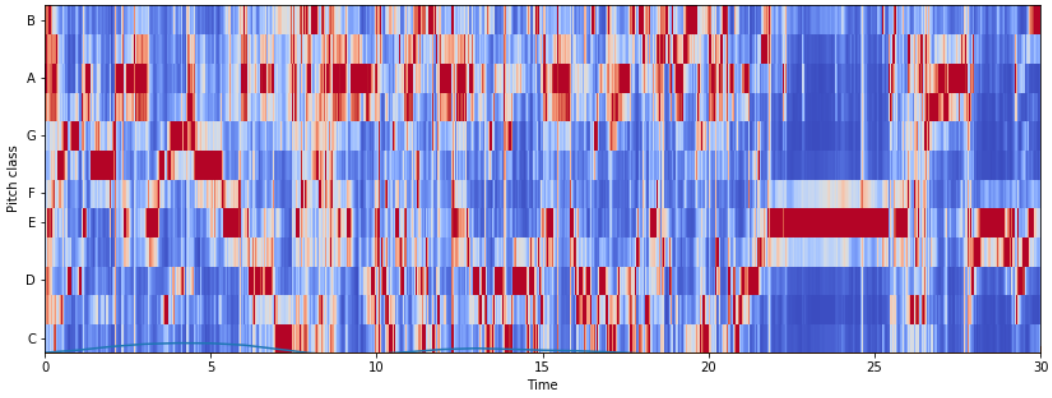
frequency bands used in the normal cepstrum. This frequency warping can allow for better representation of sound, for example, in audio compression.



4 mfcc for file rock00001

4.5. Chroma

Chroma features are an interesting and powerful representation for music audio in which the entire spectrum is projected onto 12 bins representing the 12 distinct semitones (or chroma) of the musical octave. Since, in music, notes exactly one octave apart are perceived as particularly similar, knowing the distribution of chroma even without the absolute frequency (i.e. the original octave) can give useful musical information about the audio -- and may even reveal perceived musical similarity that is not apparent in the original spectra.



5 chroma for file rock00001

5. Model, Training and Validation

The model is a fully connected Neural Network, composed by 5 Dense layers:

```
Model: "sequential_8"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_35 (Dense)             (None, 512)               13824
_________________________________________________________________
dense_36 (Dense)             (None, 256)               131328
_________________________________________________________________
dense_37 (Dense)             (None, 128)               32896
_________________________________________________________________
dense_38 (Dense)             (None, 64)                8256
_________________________________________________________________
dense_39 (Dense)             (None, 15)                975
=================================================================
Total params: 187,279
Trainable params: 187,279
Non-trainable params: 0
_________________________________________________________________
```

The training and the validation follow the K-fold approach, each iteration trains the model by using 8 epochs and a batch size of 15 or 125:

```
# fix random seed for reproducibility
seed = 7
np.random.seed(seed)
acc_per_fold = []
loss_per_fold = []
model = create_model()

fold_no = 1

# define 10-fold cross validation test
kfold = StratifiedKFold(n_splits=10, shuffle=True, random_state=seed)
cvscores = []
for train, test in kfold.split(X, y):

    model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])

    # Training
    print('------------------------------------------------------------------------')
    print(f'Training for fold {fold_no} ...')

    history = model.fit(X[train], y[train], epochs=8,batch_size=15)
    # evaluate
    scores = model.evaluate(X[test], y[test], verbose=0)
    print(f'Score for fold {fold_no}: {model.metrics_names[0]} of {scores[0]}; {model.metrics_names[1]} of {scores[1]*1
    acc_per_fold.append(scores[1] * 100)
    loss_per_fold.append(scores[0])

    fold_no = fold_no + 1
```

Cross-validation is a resampling procedure used to evaluate machine learning models on a limited data sample.

The procedure has a single parameter called k that refers to the number of groups that a given data sample is to be split into. As such, the procedure is often called k-fold cross-validation. When a specific value for k is chosen, it may be used in place of k in the reference to the model, such as k=10 becoming 10-fold cross-validation.

Cross-validation is primarily used in applied machine learning to estimate the skill of a machine learning model on unseen data. That is, to use a limited sample in order to estimate how the model is expected to perform in general when used to make predictions on data not used during the training of the model.

It is a popular method because it is simple to understand and because it generally results in a less biased or less optimistic estimate of the model skill than other methods, such as a simple train/test split.

The general procedure is as follows:

- Shuffle the dataset randomly.
- Split the dataset into k groups
- For each unique group:
- Take the group as a hold out or test data set
- Take the remaining groups as a training data set
- Fit a model on the training set and evaluate it on the test set
- Retain the evaluation score and discard the model
- Summarize the skill of the model using the sample of model evaluation scores

Importantly, each observation in the data sample is assigned to an individual group and stays in that group for the duration of the procedure. This means that each sample is given the opportunity to be used in the hold out set 1 time and used to train the model k-1 times.

There are a number of variations on the k-fold cross validation procedure.

Three commonly used variations are as follows:

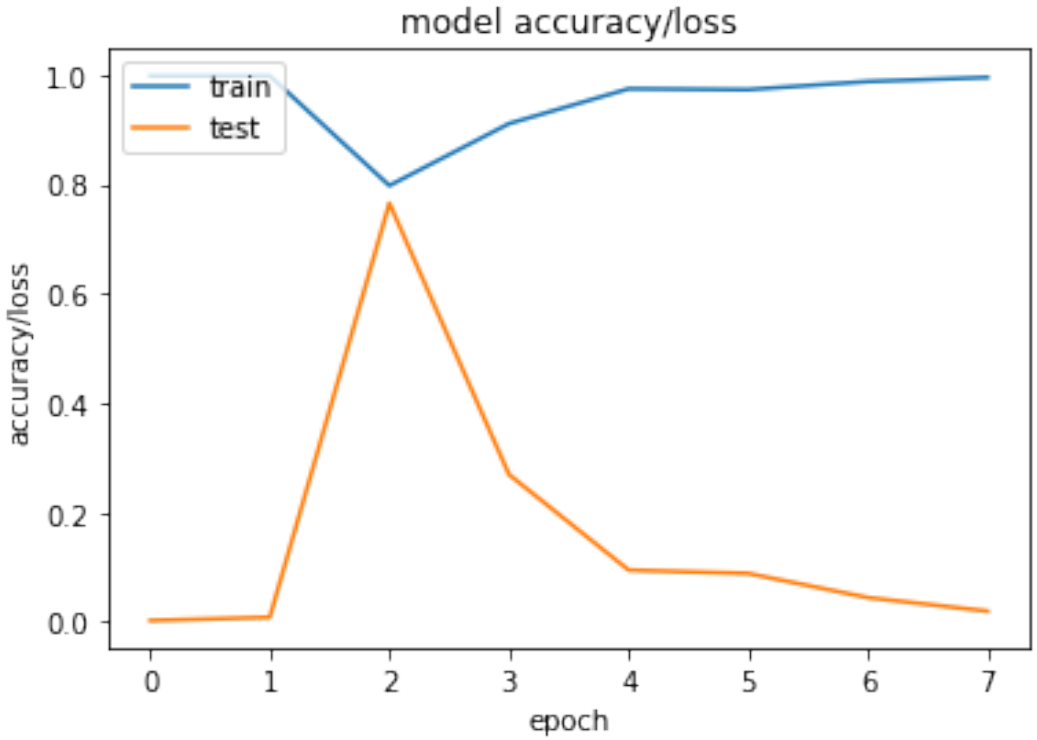
- Train/Test Split: Taken to one extreme, k may be set to 2 (not 1) such that a single train/test split is created to evaluate the model.
- LOOCV: Taken to another extreme, k may be set to the total number of observations in the dataset such that each observation is given a chance to be the held out of the dataset. This is called leave-one-out cross-validation, or LOOCV for short.
- Stratified: The splitting of data into folds may be governed by criteria such as ensuring that each fold has the same proportion of observations with a given categorical value, such as the class outcome value. This is called stratified cross-validation.
- Repeated: This is where the k-fold cross-validation procedure is repeated n times, where importantly, the data sample is shuffled prior to each repetition, which results in a different split of the sample.
- Nested: This is where k-fold cross-validation is performed within each fold of cross-validation, often to perform hyperparameter tuning during model evaluation. This is called nested cross-validation or double cross-validation.

The results after the training shows the following score:

```
------------------------------------------------------------------------
Score per fold
------------------------------------------------------------------------
> Fold 1 - Loss: 0.926558453241984 - Accuracy: 68.66666674613953%
------------------------------------------------------------------------
> Fold 2 - Loss: 0.4729801821708679 - Accuracy: 83.33333134651184%
------------------------------------------------------------------------
> Fold 3 - Loss: 0.2503801793853442 - Accuracy: 92.00000166893005%
------------------------------------------------------------------------
> Fold 4 - Loss: 0.2500366201003393 - Accuracy: 91.33333563804626%
------------------------------------------------------------------------
> Fold 5 - Loss: 0.08818703080217044 - Accuracy: 97.33333587646484%
------------------------------------------------------------------------
> Fold 6 - Loss: 0.04618426411102215 - Accuracy: 98.66666793823242%
------------------------------------------------------------------------
> Fold 7 - Loss: 0.003332723496326556 - Accuracy: 100.0%
------------------------------------------------------------------------
> Fold 8 - Loss: 0.00039379083279830713 - Accuracy: 100.0%
------------------------------------------------------------------------
> Fold 9 - Loss: 0.00023360115475952624 - Accuracy: 100.0%
------------------------------------------------------------------------
> Fold 10 - Loss: 0.12194080412387848 - Accuracy: 93.99999976158142%
------------------------------------------------------------------------
```

```
Average scores for all folds:
> Accuracy: 92.53333389759064 (+- 9.412993727871829)
> Loss: 0.21602276494194905
------------------------------------------------------------------------
```

The model history for the last iteration show the following results for the accuracy/loss scores:



6. Predictions

I tested the model against different audio files.
Unfortunately, there weren't good results, the training generates an overfitting and in few cases the model predict correctly the label of the song
All those features extracted were a good starting point to classify music genres, but using a CNN on the spectrograms is more efficient.

7. Conclusions

Using a completely connected neural network to be able to classify musical genres starting from some characteristics in the form of time series has been very interesting and has led to good results.
In any case, it is possible to do a lot of experimentation in terms of models. Using a CNN model (on the spectrogram images) offers greater accuracy and I would like to try it out in the future.

The classification of the musical genre is one of the many branches of the retrieval of musical information. From here you can perform other music data tasks such as rhythm detection, music generation, recommendation systems, track separation and instrument recognition, etc. Music analysis is a varied and interesting field. A music session somehow represents a moment for the user. Finding these moments and describing them is an interesting challenge in the field of Data Science.