Music Genre Classification

Harel Yerushalmi (ID. 203456603) - Ron Petrov (ID. 308286228)

Submitted as final project report for Deep Learning, IDC, 2019

1 Introduction

The project goal is to predict the genre of a given song.
Our initial goal was to predict what human **feelings** a given song may arise, but this kind of mission required us to collect our own data and to conduct a psychological research prior to the work - a non-feasible mission given the circumstances.
The motivation to predict a song's **genre** came from our will to learn from music. We wanted to see how we can extract sufficient features of an audio piece, regardless of the specific goal in mind. If we can successfully predict the genre of a song, we might be able to use this information to predict more elaborate things in the future. Plus - predicting a song's genre is a pretty cool thing to do!

2 Solution

2.1 General approach

We want to solve this problem using a Convolutional Neural Network.
The motivation for this is that a song (or an audio file) can be observed as a time series, that can be divided into consecutive time segments. Each time segment has, naturally, many different features (frequencies, amplitude, pitch etc.) and using a CNN to extract different features out of each segment (using different filters) seemed like the natural choice.
Because of the properties of a time series kind of data, the relevant "scan" of the filters on each instance should be made in one dimension - the Time dimension (i.e. the X axis. As oppose to images classification where scanning both the X and Y axes may be more useful). This is why we believe in using a **1-Dimesional CNN** as our main CNN.

The alternatives we plan to try are using a 2-Dimesional CNN, mostly used for image recognition, and using an image input that represents the audio (a spectrogram). In addition, we plan to try different inner layers. Since we want to preserve the initial features that the network is trying to learn, we plan to

use **ReLU** as the only activation in all networks (such that only positive values will percolate). This approach can be beneficial when using data that is "wave-like", since negative wave values (like negative amplitudes) can be "ignored" - making work easier while keeping relevant information given by the positive values. In a similiar manner, we want to preserve the most relevant values of the audio features and so we plan to use only **Max-Pooling** layers, as oppose to averaging the values.

2.2 Design

The code consists of the data's fetching, manipulation and the needed adjustments; followed by three CNNs.
The data set is comprised of 1000 audio tracks, each 30 seconds long. It contains 10 genres, with 100 tracks for each genre. In order to enlarge the data set, we took each audio track and divided it into 10 consecutive segments of 3 seconds each - treating each segment as an instance with the appropriate class (genre). That way we get 10000 instances - each is a 3-seconds long audio piece that contains 44100 samples (or "raw" features). The class of each instance is represented as a 1-hot vector (1 for the correct class and 0 for the others). The final layer of each network has the same length of the 1-hot vector - this is the vector we're trying to achieve.

The first CNN uses a 1-dimensional convolution and takes the data as is ("wave data" in a given sample rate). This CNN was very hard to train and never gave us good results. Changing the optimizer was helpful sometimes but changes to the network itself didn't seem to improve the results much (leading very often to over fitting). We decided to give the CNN what it wants - an image.

The second CNN uses a 1-dimensional convolution on a Melspectrogram input. A Melspectrogram is an image representation of sound (a detailed explanation of how it's created is commented in the code). This CNN was easier to train and gave good enough results.

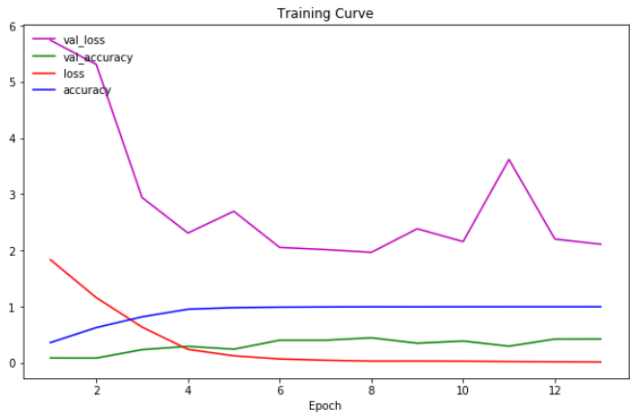
The third CNN uses a 2-dimensional convolution on a Melspectrogram input. Since a spectrogram is an image we wanted to try a 2D CNN for the image classification. The results were good but usually less promising than the the 1D CNN.

3 Experimental results

We experimented with different hyper-parameters. Some parameters' values gave us better results and other values were chosen more arbitrarily, since the change in results wasn't significant. The results shown are for the following values:
Train set size to test set size proportion is 0.75 to 0.25. Those values were chosen quite arbitrarily, but usually gave us better results than other divisions. The validation split was chosen in the same manner and has a proportion of 0.1 of the training set. Batch size is 32 - chosen after alternatives didn't show a significant change, while training time is quite fast (in the spectrogram based CNNs).
Each training process consists of a maximum of 15 epochs. We chose this number because results didn't seem to improve much beyond that number. Since the results might sometimes stabilize before 15 epochs, we use a "validation loss patience" parameter that stops the learning process if the validation loss doesn't improve several times in a row.

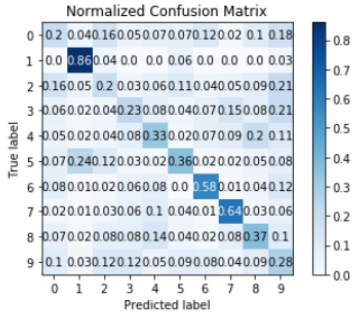
The results of the first CNN (1-dimensional convolution on raw data) were always "disappointing" and gave an accuracy score of less than 0.5 every time, even when we changed the first layer in order to treat the input differently. We checked different inner layers and changed the optimizer to "adam" (which slightly improved the results). We decided to keep this network because it gives a nice insight on how important it is to make the proper changes to the data as it enters the network as input (The second CNN is almost identical, only the data is first converted to a spectrogram; what improved the results significantly).

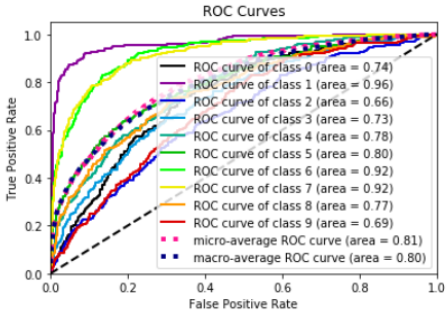
TRAINING PROCESS:



RESULTS ON TEST SET:

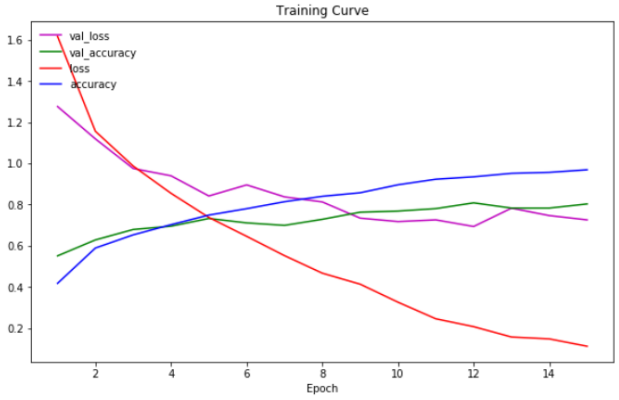
Accuracy score: 0.42





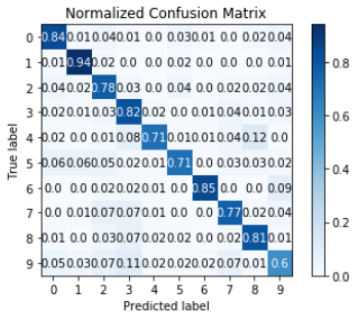
The results of the second CNN (1-dimensional convolution on Melspectrogram) were the best. We wanted this network to resemble the first one with only the input changed, so we added as first inner layers a Melspectrogram, followed by a normalization layer and shaping of the output to 1-dimension. For the network to consider the Melspectrogram as its "real" input, we set the 'trainable' parameter of this layer to False. Otherwise different parts of the raw data itself will get different weights when converted to a spectrogram (while all parts should be converted equally). Trying to train those weights as well gave a validation accuracy of lower than 0.6 in all tests.

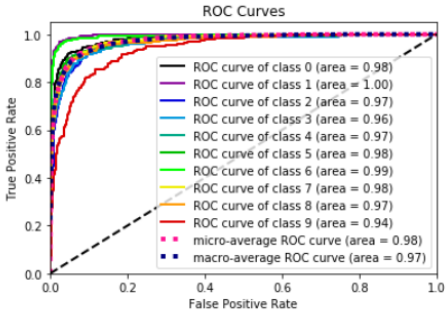
PROCESS:



RESULTS ON TEST SET:

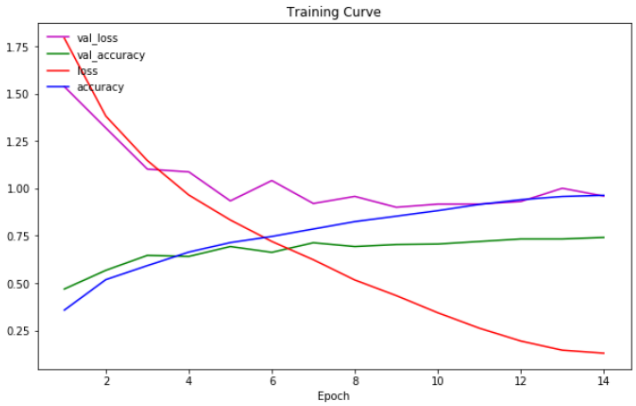
Accuracy score: 0.8





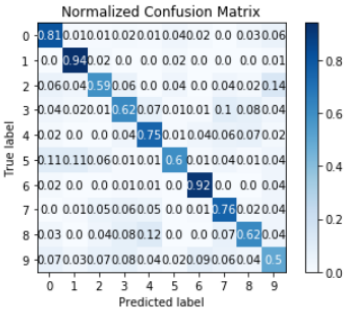
The last CNN (2-dimensional convolution on Melspectrogram) gave good results but not like the 1-dimensional one. As mentioned before, the spectrogram represents a times-series kind of data, and looking for features in both dimensions might be redundant. Indeed, most of the tests (using filters of about the same size as the 1-dimensional filter's width) confirmed that.

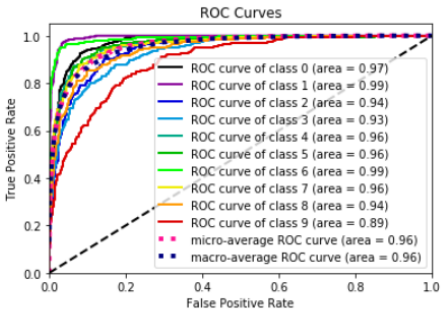
PROCESS:



RESULTS ON TEST SET:

Accuracy score: 0.73





4 Discussion

The experiments we've made shed a lot of light on the importance of each aspect of the neural network along its design. We learned the importance of data handling and splitting, and how to make more data out of whatever exists.

The other important message is how crucial it is to adjust the network to the data at hand. It might sound trivial and obvious but it led us to understand that sometimes the network itself might make sense (as much as it can) but the **data itself** may need to be adjusted to the network.
Treating audio as an image made it very easy to learn from it and let us use the power of CNNs in an efficient way. The thing about sound is that it naturally has a lot of well defined features (as oppose to photos for example). Those features - such as frequencies and amplitude - could easily be used for learning in a supervised way. So, the use of a neural network in order to predict a genre could have worked well if the network knew all of those features and how to treat them; and since CNNs can detect different features from the same samples, it seemed like a good fit for our goal. Even though the network looked, the data, as mentioned, wasn't given to it in an adequate format, so the only thing left to do was to adjust it the best we can, using a Melspectrogram. A good lesson from this would probably be that if you have something that should be working but doesn't, don't rush to change it. The problem might be with yourself and with what you're giving (or how you're treating) to that something in the first place... Lesson for life.

When we started the project we were wandering how would the network learn the features that we know of when it comes to music; features like the Hip Hop's beats or the Rock's electric guitar and drums. When we thought about it that

way we were wandering if the number of features we should tell it to learn should resemble the number of features we can think of as humans. We thought It could be interesting to see how many features (or filters) a CNN prefers (treating this number as a hyper-parameter) and to check if it can correlate somehow to the number, as well as the characteristics, of our human features. We understood quite quickly that this kind of mindset is wrong when it comes to neural networks, and the results gave us a nice insight about it:
The genre that was always predicted most successfully is classical music. Even the first network, that usually couldn't predict much of anything, has always predicted classical music well. On the one hand it's quite easy to recognize classical music, based on the instruments' sounds themselves (violin for example is not common in the other genres tested), which might give us a hint to what the network easily "pays attention" to. On the other hand, classical music, like metal in a sense (which is the second most predicted genre by the first network, using enough convolution layers), is quite abstract. It's beat is much less intuitive to most people, there are no drums, no coherent structural repetitions and no notable general features. Blues music on the other hand is very structural, and different songs use a lot of the same relative chords' and style. Yet, the raw data CNN always predicted this genre the worst. It's not such a big surprise when we think about it - the network "scans" the audio, as time passes. It doesn't look at the big picture in order to find structures throughout the song, nor does it look for general characteristics that are very obvious to us as humans when specifying a genre; it simply gets waves samples in a given time and learns from it. This observation might also sound a bit obvious, but it emphasizes that the learning process of neural networks (the ones mentioned here at least), or in other words - the things that the networks find important, are very different from what we perceive to be important when specifying a song. For this reason, any attempt to build a network around how we perceive reality and features, might be a false approach to begin with. So it's quite safe to say that even though the network can predict genres well, most of the features that it learned (if not all) are different from the features we attribute to different genres. The former question regarding the number of filters to be used is still relevant of course, but the means to figuring that out are very different from what our human consciousness is used to.

Last thing to mention is that the 2-dimesional CNN gave most of the time slightly worst results than the 1-dimesional one. Having our input as an image should have made sure that a 2D CNN could classify the different images well, but the fact that those images represent a time-series based data seemed to be important after all. A good lesson would be that if you decide to make changes to the data, just don't forget where you came from, even if seems like an all-new kind of data now.

5 Code

Code in colab notebook.

Download data.