POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Numerical Analysis For Machine Learning Project

Music Genre Classification

Author(s): **Leonardo De Clara**
Samuele Delpero

Group Number: **32**

Academic Year: 2022-2023

Contents

1 | Introduction

The rise of Deep Learning (DL) techniques has clearly shown their superior strength in big data learning in recent years, leading to a growing trend in the use of DL in the field of music genre classification: among all classification models based on deep neural networks Convolutional Neural Networks (CNNs) have proven to be able to deliver remarkable results in the extraction of hierachical representations from raw data, thus making them particularly promising once applied to audio spectograms or waveforms.

The aim of this project is to implement the design by Xu and Zhou [1] of a particular type of CNN for music genre classification, making use of Squeeze&Excitement blocks (SE-Blocks), an architectural unit designed to improve the representational power of the CNN by performing dynamic channel-wise feature recalibration. In order to properly train and test our model and to follow as close as possible the authors' implementation process we used the most popular music genre classification dataset, GTZAN.

2 | Dataset and Data Augmentation

2.1. GTZAN

The GTZAN dataset, widely used in music genre classification, was adopted to train and evaluate our model. It consists of 1,000 audio clips and contains labels for 10 different music genres: Blues, Classical, Country, Disco, Hip Hop, Jazz, Metal, Pop, Reggae and Rock. Each genre contains 100 music clips for 30 s and is stored as a 22,050 Hz, 16 bit mono audio file. We proceeded to divide the training, validation and test set according to the percentages (60,10,30).

2.2. Spectrogram Creation

Convolutional networks are very powerful models once applied to image-like data, therefore the audio files of the dataset needed to be converted in spectrograms, passing from the time-amplitude representation to the time-frequency representation. This is achieved by the following process, that starts with the short time Fourier transform (STFT), that consists in splitting the original file in short overlapping windows and performing the Fourier transform on each segment, obtaining a snapshot in time of the magnitude and phase of the frequences in that window. Adding this for every window in a matrix we can see how the frequences change in time in our original audio file. After the STFT, because we are only interested in the magnitude, we extract it from the matrix by performing the element-wise absolute value, and then we represent it in decibells obtaining the spectrogram.

Following the indicatons in the original paper we used as windows size 1024 sampling points of the original file, and 512 points of overlapping between consecutive windows.

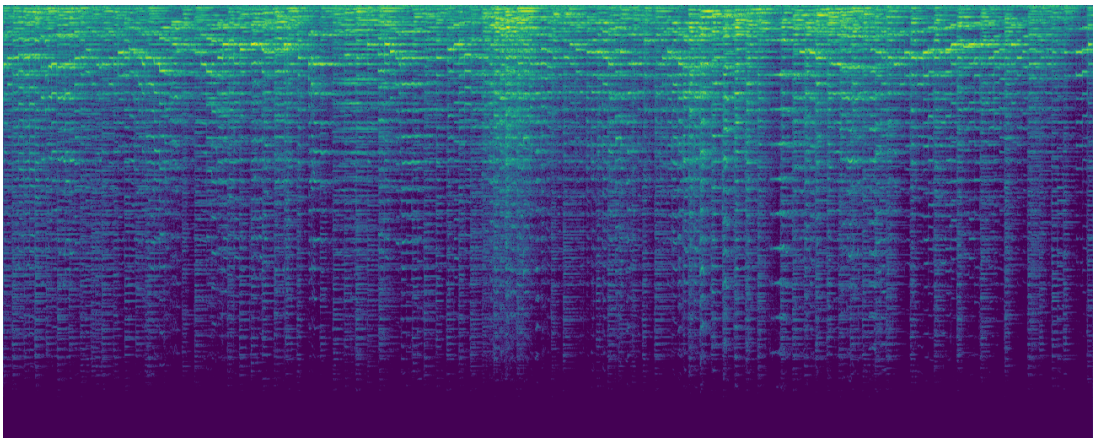

Figure 2.1: Example of a spectrogram; each column is the magnitude of the frequencies of the Fourier transform of a window in decibells

2.3. Data Augmentation

Due to the limited size of the dataset, data augmentation was needed to increase it as to avoid overfitting. Each spectrogram representing a 30 seconds audio file, obtained as described in the previous section, was divided in 19 spectrograms representing 3 seconds audio files, each one overlapping with the previous one for 1.5 seconds, allowing us to collect 19,000 spectrograms from the original 1,000 audio files.

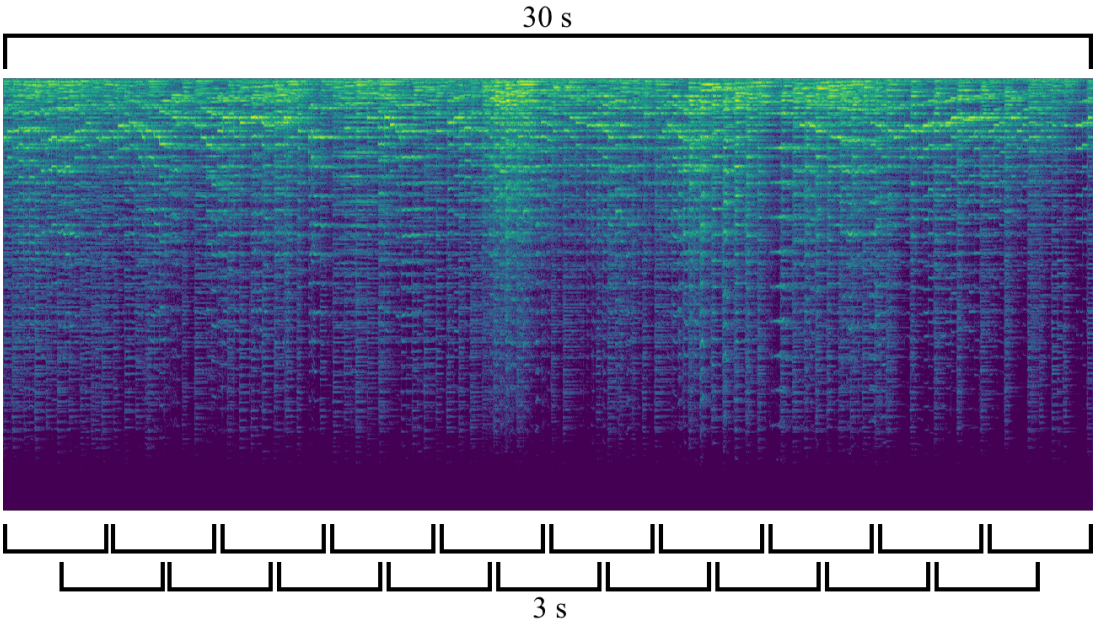

Figure 2.2: Splitting of the spectrogram

3 | Squeeze and Excitation Block

Convolutional layers of CNNs return output data in three-dimensional form H x W x C that can be intepreted as a sequence of two-dimensional matrices placed one above the other, each of which is called a feature map. In particular, in these layers a dot product operation is carried out between two matrices: one is referred to as a kernel, the feature detector, which will move across the fields of the image, checking if the feature is present, the other is the input image. The ouput data will have a depth determined by the number C of filters, also called channels. More in details, the main hyperparameters of each convolutional layer are:

- Kernel Size: The kernel size defines the field of view of the convolution. In this project we mainly used 3x3 kernels.
- Stride: The stride defines the step size of the kernel when traversing the image. Its default value is usually 1.
- Input and output channels: A convolutional layer takes a certain number of input channels and calculates a specific number of output channels.

The convolutional filters are responsible for constructing the feature maps based on the learned weights. These feature maps also have a different magnitude of importance, meaning that some feature maps are more important than others. Thus, we would want to provide the "more important" feature maps with a higher degree of importance compared to the others, focusing our "attention" on the most important channels by scaling them by a higher value. In order to do so, [1] introduces in the CNN the Squeeze and Excitation Block [2], which consists of three components:

- Squeeze module: it is based on the Global Average Pool operation, which essentially reduces the whole feature map corresponding to a channel to a singular value by taking the average of all pixels in that feature map. Thus it constructs a smoother average of all the pixels in that window. In terms of dimensionality, if the input tensor is H x W x C, the output tensor obtained will be of shape 1 x 1 x C.
- Excitation module: it is composed by two fully connected layers to map the scaling

weights: the first has ReLu activation and the dimension of layer's weights is C/r × C , where r is a reduction ratio. The purpose of this parameter is to reduce the number of channels in order to reduce the amount of calculation. Since the dimension of the input is 1 x 1 x C , the result will be 1 x 1 x C/r. The output is given as input to a layer with sigmoid activation: the dimension of layer's weights is C x C/r: the output will be then a 1 x 1 x C tensor. The role of these two fully connected layers is to merge the feature map information of each channel.

- Scale module: the ouput of the excitation module is applied directly to the input by a simple element-wise multiplication, which scales each channel in the input tensor with its corresponding learned weight from previous module.

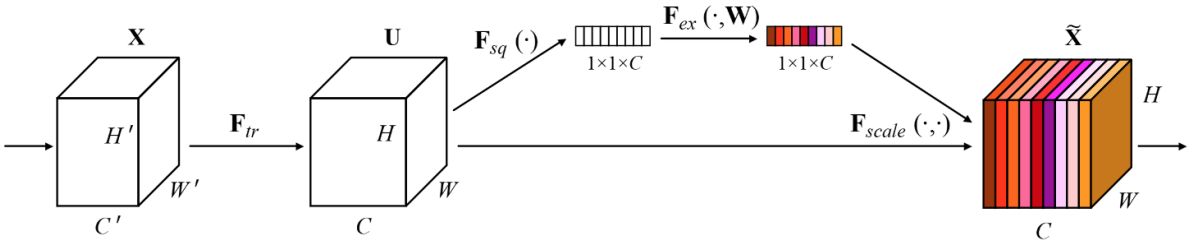

Figure 3.1: Computational process of SE-Block

4 | Model

The final architecture of the CNN consists of:

- Five Convolutional layers, whose optimal channel parameters are set to (32, 64, 128, 256, 128);
- Five MaxPooling layers that select the maximum element from the 2 x 2 region of the feature map covered by the filter. Thus, the output of each Max-pooling layer is a feature map containing the most prominent features of the previous feature map;
- Five Dropout modules setting weights to 0 with a rate of 25% at each step during training time, which helps prevent overfitting;
- Five Squeeze and Excitation Blocks, each one after a Dropout module, with ratio r set to 16;
- A Flatten layer, converting data into a 1-dimensional vector;
- A fully connected layer with Softmax activation returning the classification result.

Each layer's hyperparameters have been selected following the modelization defined in [1], apart from the ratio and channels' values whose optimal values have been found through cross validation.

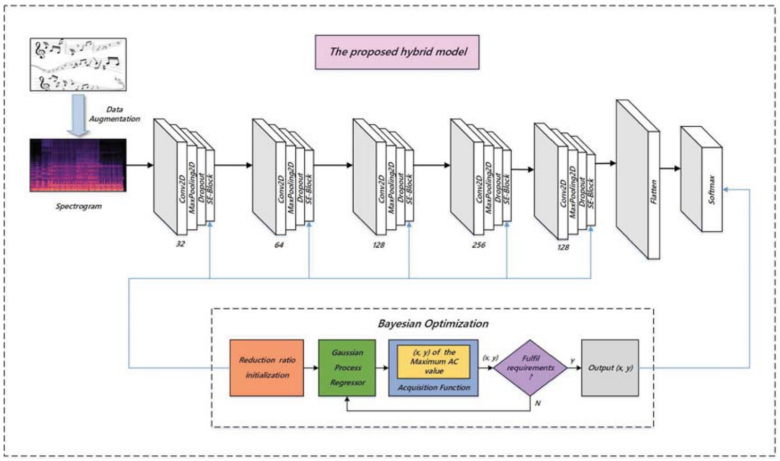

Figure 4.1: Network's model

Model architecture

	Filter shape
Conv	3 x 3 x 1 x 32
MaxPooling	pool: 2 x 2
Dropout	rate: 0.25
SE-Block	r: 16
Conv	3 x 3 x 32 x 64
MaxPooling	pool: 2 x 2
Dropout	rate: 0.25
SE-Block	r: 16
Conv	3 x 3 x 64 x 128
MaxPooling	pool: 2 x 2
Dropout	rate: 0.25
SE-Block	r: 16
Conv	3 x 3 x 128 x 256
MaxPooling	pool: 2 x 2
Dropout	rate: 0.25
SE-Block	r: 16
Conv	3 x 3 x 256 x 128
MaxPooling	pool: 2 x 2
Dropout	rate: 0.25
SE-Block	r: 16
Flatten	-
Dense	Softmax output: 1x10

Table 4.1: Tables of model's parameters

5 | Training and Results

5.1. Implementation and training

The implementation of the network and its training was performed using Google's Colaboratory environment. In particular the audio files from the GTZAN dataset and the resulting spectograms have been handled using Librosa's Python library. The implementation of the SE-Block and in general of the CNN's model is based on the use of Keras' library, which allowed us build the network using its basic blocks.

In order to overcome Colab's strict limitations to RAM and GPU usage we implemented custom data generator systems, which make it possible to generate batches of tensor data from files stored in the hard disk. Specifically, they take as inputs the files where spectograms have been stored and load into memory the required data during the training and testing phases. Following these schema we implemented one generator for training data, one for validation data and one for testing data using the *.npy* files containing the audio spectograms.

The model was trained using Adam optimizer, a stochastic gradient descent method based on adaptive estimation of first-order and second-order moments, with a learning rate equal to 0.001. The samples have been shuffled after each epoch. The loss function used is Sparse Categorical Cross Entropy and the performance metric is accuracy.

5.2. Results

The training and testing phases have been performed multiple times in order to properly evaluate different configurations of parameters such as the values of channels in the convolution layers and the ratio r in the SE-Block. Moreover, to show the difference in accuracy between implementations with and without SE-Blocks we evaluated different models which presented from zero up to five blocks. The accuracies shown in the tables below are obtained by averaging multiple runs of the same configuration. Notably the highest performance results, corresponding to an accuracy of 93.6%, have been found with

a configuration defined by use of (32, 64, 128, 256, 128) as channels' parameters and r set to 16.

5.2.1. SE-Block's effect

To accurately evaluate the influence of SE-Block embedded in each layer, we set the number of channels of the five convolutional layers to [32,64,128,256,128] and the value of r to 16. As the table shows the performances are positively affected by the presence of the SE-Block: the CNN without any SE-Block is able to provide only 87.5% accuracy. Morevoer, by adding a block one at a time starting from the fith and final layer, we can see that results are significantly more accurate.

Effect of SE-Blocks

SE-Blocks	Accuracy (%)
0	87.5
5	90.2
4,5	90.8
3,4,5	92.1
2,3,4,5	92.7
1,2,3,4,5	**93.6**

Table 5.1: Accuracy obtained with different number of SE-Blocks

5.2.2. Ratio r's effect

The effect of the ratio r on the model's performance has been evaluated by the manually setting its value and performing the training-testing procedure. We also implemented a Bayesian optimization method to find its optimal value: however the technique executed in [1] requires to complete the training-testing operation 50 times, which we could not achieve due to Colab's limitations. However, we have been able to compare the results given the ratio: our model achieves higher quality outcomes with a smaller value of r; in particolar the highest accuracy has been obtained with r=16, while in [1] the optimal value of r is 31.43.

Ratio results

Ratio r	Accuracy (%)
8	92.6
16	**93.6**
32	93.0

Table 5.2: Accuracy obtained with different values of r

5.2.3. Channels' values' effect

The values for the convolutional layers' channels have also been tuned by running different configuration, since they provide important relationship with the feature maps extracted by the convolution operations. After comparison the configuration (32, 64, 128, 256, 128) leads to the best outcomes, being able to learn enough features without overfitting.

Channels results

Channel parameters	Accuracy (%)
(16, 32, 64, 128, 64)	90.1
(32, 64, 128, 256, 128)	**93.6**
(64, 128, 256, 512, 256)	93.2
(128, 256, 512, 1024, 512)	90.8

Table 5.3: Accuracy obtained with different values for the channels' parameters

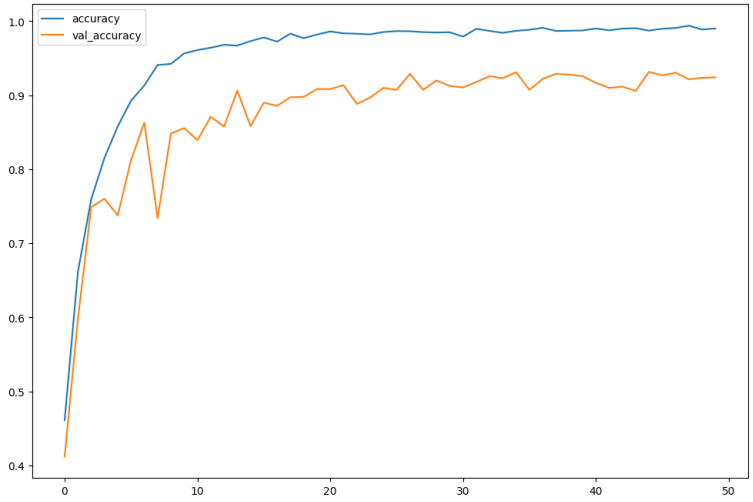

Figure 5.1: Accuracy on training/validation set

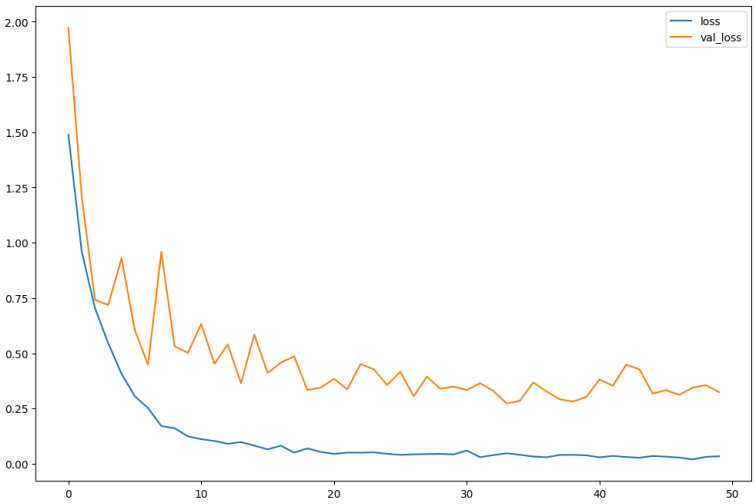

Figure 5.2: Loss on training/validation set

6 | Conclusions

Our implementation of the work conducted in [1] have shown that the use of a channel attention mechanism such as SE-Block is able to deliver impressive results in the field of music genre classification: SE-Blocks can significantly and efficiently enhance the network's understanding of feature maps, making it possible to grasp in a more precise manner the latent information hidden in the the spectogram of an audio file. We firmly believe that the excellent results accomplished by these classes of models are only a small example of what channel mechanism designs can be able to deliver in the field of music genre classification.

As compared to the experimental results obtained in [1], the final accuracies obtained by our models are slightly higher: this effect could be due to a different Python implementation of the SE-Block and to different setting of the hyperparameters for the learning phase such as learning rate and weight decay. However both implementations follow a very similar trend in results with regard to the use of different values of r and channels' parameters.

Bibliography

[1] Yijie Xu and Wuneng Zhou, Deep music genres classification model based on CNN with Squeeze Excitation Block,*APSIPA Annual Summit and Conference 2020*.

[2] Jie Hu, Li Shen, Samuel Albanie, Gang Sun,Enhua Wu, Squeeze-and-Excitation Networks, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2019.