Università degli studi di
Milano-Bicocca

Advanced Machine Learning

Final Project

Music Genre Classification

Authors:
Davide Pietrasanta - 844824 - d.pietrasanta@campus.unimib.it
Giuseppe Magazzù - 829612 - g.magazzu1@campus.unimib.it
Gaetano Magazzù - 829685 - g.magazzu2@campus.unimib.it

January 24, 2022

Abstract

The problem faced is that of the classification of music genres from audio files. Various approaches were used. Starting from approaches based on handcrafted features, the project moved on to approaches based on the mel-spectogram and CNNs. An attempt was therefore made to transfer learning methods and data augmentation to achieve better performance. The best results were achieved by using ResNet50 with **0.5012** of accuracy on the test set.

https://github.com/saiteki-kai/music-classification

1 Introduction

Nowadays, with the high traffic of multimedia data, it becomes more and more necessary to be able to organize, in an automatic way, one's music. With a better audio classification, for instance, speech recognition and recommendation algorithms can be improved. The task considered for this project is the classification of music genres from audio files. It is assumed that only one musical genre is associated with each audio track. The problem has been approached with an initial study of the theory and the state of the art. Then, it was tried to overcome the results obtained with approaches seen in class, as transfer learning.

2 Datasets

The dataset chosen is the Free Music Archive (FMA) [1]. The dataset contains 106574 high quality audio tracks lasting approximately 30 seconds. All the tracks are common creative licensed.

Each track is associated with additional information about the artist (name, location, bio, etc.), the album (title, listens, comments, etc.) and the track itself (title, creation date, duration, genres, etc.).

The genres are organized in a hierarchy of 161 unbalanced classes of different genres.

Pre-computed features are also provided such as the statistical moments of some spectral and temporal features.

The dataset propose a train/validation/test (80%/10%/10%) split and three subsets (small, medium, large). In this work only the small one was used which consists of 8000 tracks and 8 balanced genres.

2.1 Preprocessing for CNN

To use CNN, it was decided to use image spectrograms as input. Since the dataset did not provide these features, a brief exploratory data analysis was performed.

Analyzing the raw audio tracks it was found that the sampling frequency varies between 22050Hz and 48000Hz. Most of the tracks have a sampling rate of 44100Hz. It was decided to resample all tracks to 22050Hz.

Furthermore, it was found that the durations of the audio tracks are not all 30 seconds long. The range of duration values that was found was from 0 to 30.02 seconds. Then the length was reduced for all tracks to 29.70 seconds and the shorter length samples were removed due to incorrect length metadata[1].

The logarithmic Mel spectrogram was calculated on the raw audio with the following parameters:

- Sample rate: 22050Hz
- Window Size: 2048
- Hop Length: 512
- Mel bins: 128

2.2 Data Augmentation

Two new spectrograms were generated from each image of the training split using frequency masking and time masking techniques.

Masking consists of setting a range of pixels to zero in the frequency range or time range.

The pixel range is randomly generated from a uniform distribution. For the frequency (0, 27) was used, while for the time (0, 100) [2].

[1] more details: excerpts-shorter-than-30s-and-erroneous-audio-length-metadata

3 The Methodological Approach

3.1 Handcrafted Features

The first model tested was a simple neural network on handcrafted features of the small subset.

The split of the data was the same as that provided by the dataset (6400 train, 800 validation and 800 test). The number of features was 518. The features were standardized on the statistics of the training set.

The architecture of the neural network was found by testing a different number of neurons for one layer and then two layers. The best parameters found were (512, 256) [2]. Adam was used as an optimizer with a learning rate of 0.0001 and a batch size of 32.

3.2 Convolutional Neural Network

According to the state of the art[3], the approach that now seems to perform better for these tasks is the one based on spectrograms and Convolutional Neural Networks.

The architecture used was the one suggested by "Daniel Kostrzewa" [4]. "Layer 1 and 2 have both 64 kernels each, whereas layers 3 and 4 have 128 kernels. The kernel size of all layers is equal to 5. After each layer, there is 2-D max pooling applied with kernel size and stride equal 2. In every convolutional layer, ReLU is used as an activation function. Batch normalization is performed afterward. The convolutional layers are followed by one fully connected linear layer with linear activation function and the final output of 8 nodes" [4]. Dropout probability was 0.20. The number of trainable parameters was 735 944.

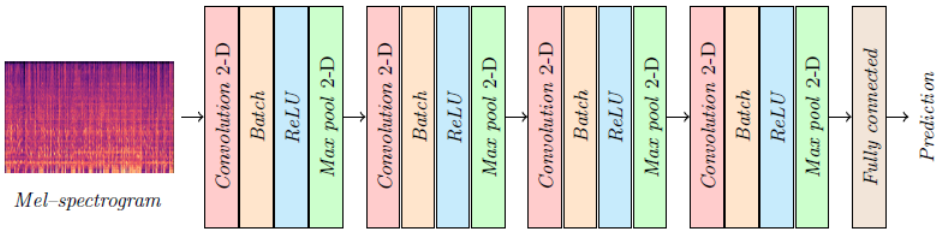


Figure 1: The architecture suggested by "Daniel Kostrzewa" [4].

An hyper-parameter optimization was performed on this architecture. Not being able to do an optimization from start to finish and on all the hyper-parameters space, it was decided to use Hyperband [5] and to optimize only some hyper-parameters.

The hyper-parameters considered for the optimization were:

- The size of the kernel (fixed on each layer) in [3,6].
- The number of kernels for each layer in [32, 256].
- The probability of dropout in {0.2, 0.25, 0.5}.
- The initial learning rate in {0.01, 0.001, 0.0001}.

The optimization of the network led to have, in the case without date augmentation, the first layer with 32 kernels, the second one with 128, the third with 192 and the fourth with 98. The kernel size of all layers was equal to 6. Dropout probability was 0.25 and initial learning rate was 0.001.

While in the case with data augmentation the optimization led to have, the first layer, the second and the third all with 64 kernels and the fourth with 96. The kernel size of all layers was equal to 4. Dropout probability was 0.25 and initial learning rate was 0.0001.

3.3 Feature extraction with CNN

Later the approach was changed to using transfer learning as the amount of data is not large.

A features extraction process was applied. Two Top-5 CNN architectures trained on ImageNet were chosen: VGG16 and ResNet50. These architectures are similar in accuracy, but different in the number of parameters.

Since the considered domain differs from ImageNet's natural images, it is expected that lower levels should perform better.

The characteristics were extracted in different cut levels and each vector was standardized and then a PCA (Principal Component Analysis) was applied in order to reduce the high dimensionality due to the lower layers.
Different values of explained variance percentage were tested on the fc2 layer of VGG16: 99%, 95%, 90%.

After the results were obtained it was found out that PCA 90% was the best. Then several cut levels were tried for VGG16 (block5_pool, fc1, fc2) and for ResNet50 (conv5_block1_2_relu, avg_pool)
For computational reasons no tests were performed without PCA and at much lower levels.

For this phase, the training and validation data were combined to then carry out a k-fold cross-validation with the grid search to fine-tune the machine learning models.
Different standard classifiers were tried, linear SVM, SVM with kernel RBF and a MLP(Multi Layer Perceptron).
The hyper-parameters considered was:

- C: [0.01, 0.1, 1, 10] (Linear SVM).
- C: [0.01, 0.1, 1, 10], gamma: [0.01, 0.001, 0.0001] (SVM RBF).

For the Multi Layer Perceptron the Adam optimizer was used with a batch size of 32 and a learning rate of 1e-4. The number of epochs was set to a max of 200 and early stopping was also used. All layers use a relu activation function, while the last layer was a Softmax and a Cross Entropy Loss was used.

- network architectures: [(512, 256), (512, 32), (512,)],
- l2 regularization alpha: [0.01, 0.03, 0.05]

4 Results and Evaluation

The following results were obtained with the following hardware.

CPU Specifications:

- Intel(R) Xeon(R)
- CPU Freq. of 2.30GHz
- 4 CPU cores
- 16 Gigabytes of RAM

GPU Specifications:

- Nvidia P100
- GPU Memory Clock of 1.32GHz
- 2 CPU cores
- 12 Gigabytes of RAM

4.1 Handcrafted Features

Units	**Parameters**	**Train Loss**	**Val Loss**	**Test Loss**
(512, 512)	532,488	1.482	1.723	1.898
(512, 256)	399,112	1.470	1.715	1.900
(256, 256)	200,712	1.503	1.733	1.946
(256, 64)	149,832	1.475	1.733	1.943
(128, 64)	75,208	1.476	1.729	1.905
(64)	33,736	1.332	1.598	1.790
(128)	67,464	1.321	1.577	1.800
(256)	134,920	1.338	1.592	1.806
(512)	269,832	1.334	1.579	1.803

Table 1: Values of loss for the different number of neurons.

Units	**Parameters**	**Train Acc**	**Val Acc**	**Test Acc**
(512, 512)	532,488	0.677	0.563	0.487
(512, 256)	399,112	0.680	0.567	**0.495**
(256, 256)	200,712	0.683	0.577	0.482
(256, 64)	149,832	0.690	0.577	0.484
(128, 64)	75,208	0.683	0.564	0.489
(64)	33,736	0.680	0.560	0.465
(128)	67,464	0.686	0.566	0.475
(256)	134,920	0.685	0.566	0.479
(512)	269,832	0.683	0.571	0.472

Table 2: Values of accuracy for the different number of neurons.

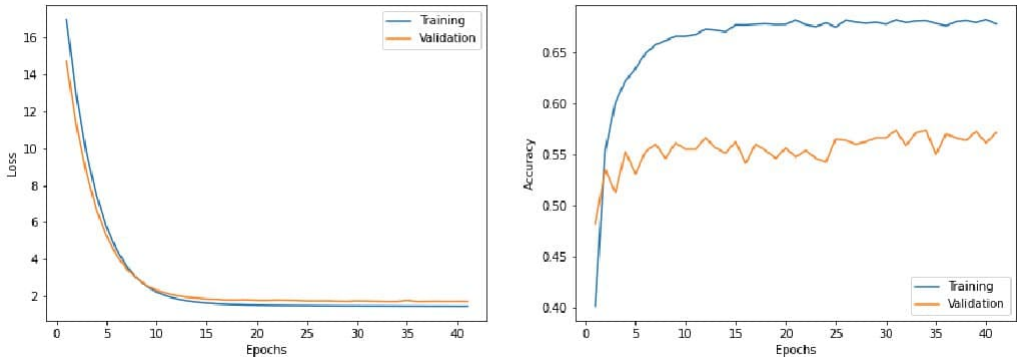


Figure 2: Loss and Accuracy curves for the best model.

4.2 CNN

The following tables compare CNN with and without augmentation and with and without tuning in terms of prediction time, accuracy and loss.

	Prediction time (CPU)	Prediction time (GPU)
CNN	0.064s	0.042s
Tuned CNN	0.060s	0.043s
Tuned CNN augmented	0.061s	0.044s

Table 3: Inference time over 10 runs of the same random value for CNN models.

	Test accuracy	Test Loss
CNN	**0.3975**	3.7596
CNN augmented	0.3225	8.0137
Tuned CNN	0.3900	2.3860
Tuned CNN augmented	0.3787	2.0166

Table 4: Test accuracy and loss of the CNN models.

	Train accuracy	Train Loss
CNN	0.9998	0.0011
CNN augmented	0.9968	0.0101
Tuned CNN	0.9998	0.0065
Tuned CNN augmented	0.8073	0.6131

Table 5: Train accuracy and loss of the CNN models.

	Validation accuracy	Validation Loss
CNN	0.4663	2.9336
CNN augmented	0.3938	5.6310
Tuned CNN	0.4175	2.0767
Tuned CNN augmented	0.4450	1.7493

Table 6: Validation accuracy and loss of the CNN models.

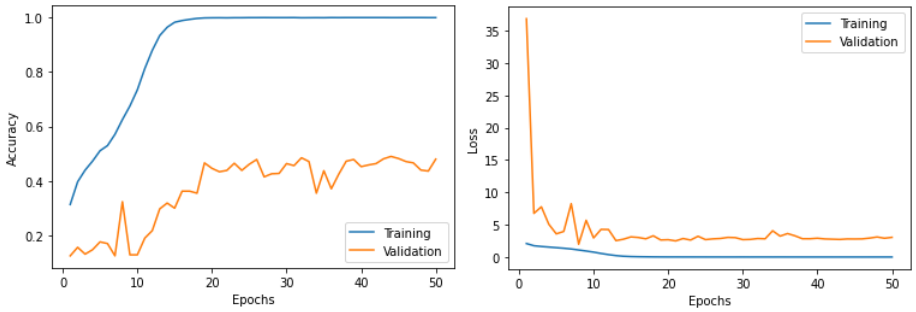


Figure 3: CNN's Accuracy and Loss function of the training and validation set.

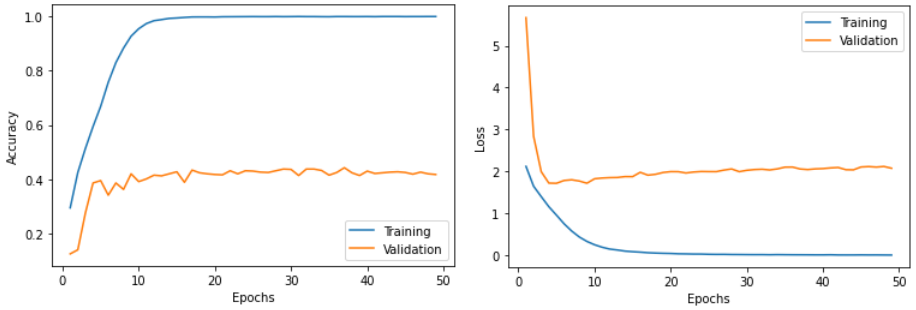


Figure 4: Tuned CNN's Accuracy and Loss function of the training and validation set.

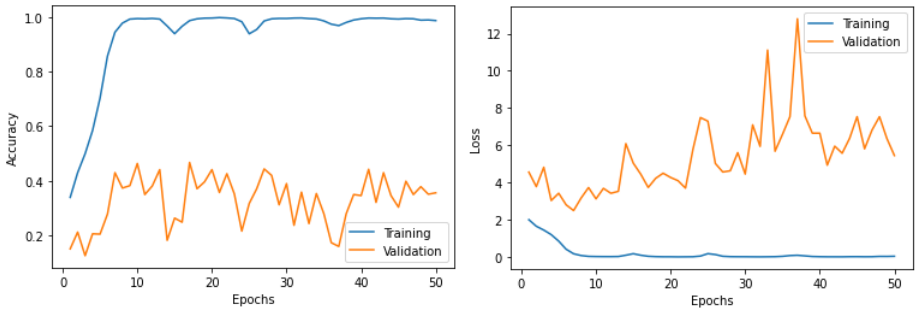


Figure 5: CNN's Accuracy and Loss function of the training and validation set with data augmentation.

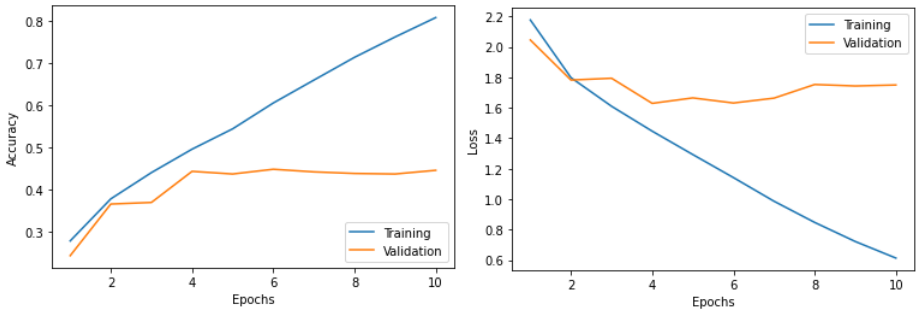
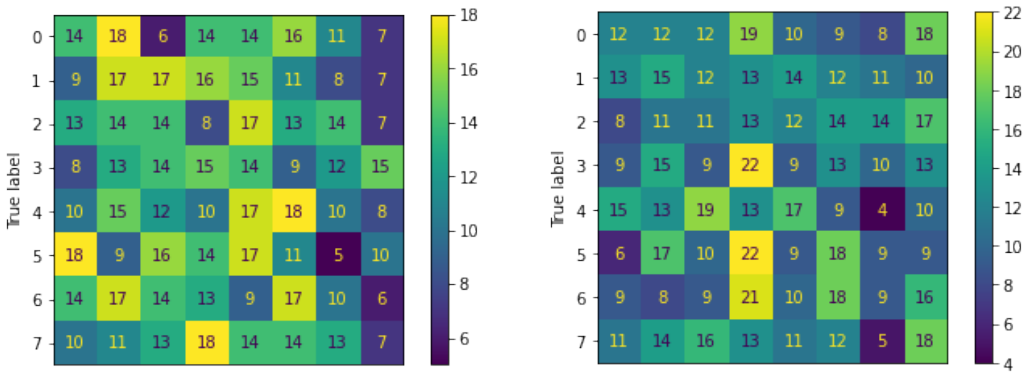
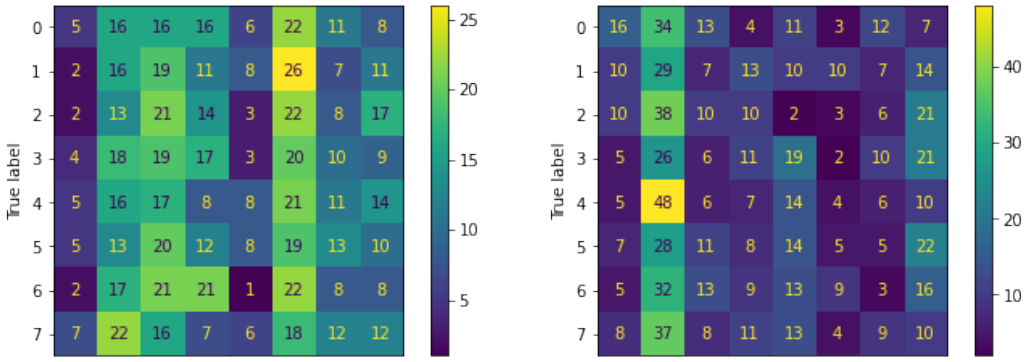


Figure 6: Tuned CNN's Accuracy and Loss function of the training and validation set with data augmentation.



(a) Simple CNN (b) Tuned CNN

Figure 7: CNN's confusion matrix.



(a) Simple CNN (b) Tuned CNN

Figure 8: CNN's confusion matrix with data augmentation.

4.3 Feature Extracted with CNN

Cut Level	Size	Size after PCA	PCA value	Model	Accuracy		
					svm linear	svm rbf	mlp
Fc2	4096	1480	0.99	VGG16	0.4440/0.4003	0.5168/0.4454	0.4795/0.4191
Fc2	4096	429	0.95	VGG16	0.4965/0.4530	0.5206/0.4542	0.4890/0.4361
Fc2	4096	153	0.9	VGG16	0.5099/0.4586	0.5195/0.4617	0.4920/0.4398

Table 7: Results of the FC2 cut level of VGG16 with different classifier and different values of PCA.

N	svm linear	svm rbf	mlp
FC2, PCA 0.99	{'C': 0.01}	{'C': 10, 'gamma': 0.0001}	{'alpha': 0.03, 'hidden_layer_sizes': (512, 32)}
FC2, PCA 0.95	{'C': 0.01}	{'C': 10, 'gamma': 0.0001}	{'alpha': 0.05, 'hidden_layer_sizes': (512, 32)}
FC2, PCA 0.90	{'C': 0.01}	{'C': 10, 'gamma': 0.0001}	{'alpha': 0.05, 'hidden_layer_sizes': (512, 256)}

Table 8: Hyperparameters results for the VGG16 FC2 with different values of pca.

Cut Level	Size	Size after PCA	Model	Accuracy		
				svm linear	svm rbf	mlp
Fc2	4096	153	VGG16	0.5099/0.4586	0.5195/0.4617	0.4920/0.4398
Fc1	4096	311	VGG16	0.5157/0.4573	0.5315/0.4837	0.5076/0.4718
block5_pool	7x7x512	1174	VGG16	0.4316/0.4185	**0.5317/ 0.4968**	0.4726/0.4429

Table 9: Results of different cut levels with PCA 90% on the VGG16 with classical classifiers.

Cut Level	svm linear	svm rbf	mlp
Fc2	{'C': 0.01}	{'C': 10, 'gamma': 0.0001}	{'alpha': 0.05, 'hidden_layer_sizes': (512, 256)}
Fc1	{'C': 0.01}	{'C': 10, 'gamma': 0.0001}	{'alpha': 0.05, 'hidden_layer_sizes': (512, 32)}
block5_pool	{'C': 0.01}	{'C': 1, 'gamma': 0.0001}	{'alpha': 0.03, 'hidden_layer_sizes': (512, 256)}

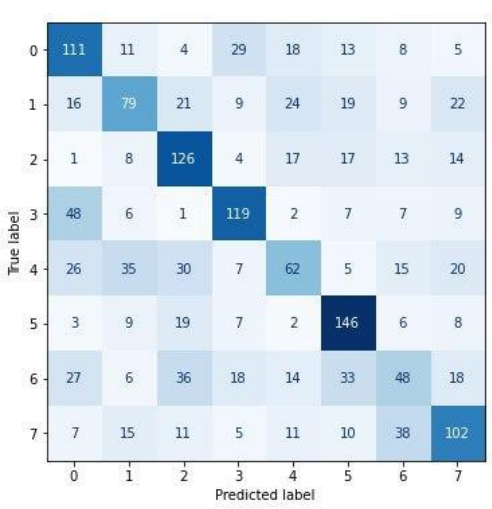
Table 10: Hyperparameters result for different cut level of VGG16 with PCA 90%.

Cut Level	Size	Size after PCA	Model	Accuracy		
				svm linear	svm rbf	mlp
avg_pool	1000	87	ResNet50	0.5268/0.4649	0.5381/0.4874	0.5246/0.4592
conv5_block1_2_relu	7x7x512	1321	ResNet50	0.4193/0.4104	**0.5484/0.5012**	0.5221/0.4755

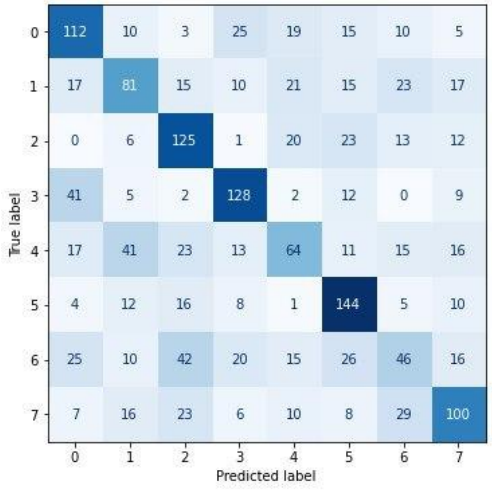
Table 11: Results of different cut levels with PCA 90% on the ResNet50 with classical classifiers.

Cut Level	svm linear	svm rbf	mlp
avg_pool	{'C': 0.01}	{'C': 10, 'gamma': 0.0001}	{'alpha': 0.05, 'hidden_layer_sizes': (512, 32)}
conv5_block1_2_relu	{'C': 0.01}	{'C': 10, 'gamma': 0.0001}	{'alpha': 0.01, 'hidden_layer_sizes': (512,)}

Table 12: Hyperparameters result for different cut of ResNet50 with PCA 90%.



(a) VGG16 - block5_pool - SVM RBF



(b) ResNet50 - conv5_block1_2_relu - SVM RBF

Figure 9: Confusion matrix of the best model for the two CNN architectures.

Model	Test Time(CPU)	Test Time(GPU)	svm rbf(CPU)
vgg16_best	73.2190 ms	23.9014 ms	2.6992 ms
resnet50_best	51.3436 ms	23.7502 ms	3.2712 ms

Table 13: inference time per image with 10 runs, for the best cut level for vgg16 and resnet50 with the rbf svm.

5 Discussion

With the handcrafted features, accuracy results on the test set are equal to **0.495**. The prediction time is fast as the number of parameters is low compared to CNNs.

Regarding the approach based on spectrograms and Convolutional Neural Networks it is noted how the models predict more or less with the same speed. It is also noticeable that they are all overfitting. The optimization of the hyper-parameters also brings greater stability to the validation curve. It should also be noted that the data augmentation, in this case, leads the task to be more difficult, thus leading to worse results. The best performing CNN model is the one described in the paper by "Daniel Kostrzewa" [4], with an accuracy of **0.3975**. It is also important to say that the performances

obtained are lower than those stated in the paper, which reaches **0.5163** of accuracy. This could be due to a different pre-processing or a number higher than 50 epochs.

With the approach based on Transfer Learning we have obtained the best results. Using ResNet50 leads to **0.5012** of accuracy.

From the table [12] it was noted how decreasing the percentage of explained variance allows to obtain better results with a greater reduction.

From table [9] it possible to see how the linear SVM gets worse going to lower layers. Probably because the data becomes less linearly separable. The radial SVM instead gets improvements at lower levels, while for the MLP its trend is not clear, probably due to the scarcity of data.

From table [11] it is also possible to see here the same effect on the linear SVM as on the radial SVM, while the MLP also improves as you go down in depth.

In the end, the two best models obtained are VGG16 at the `block5_pool` level with radial SVM and with an accuracy of 0.4968, while ResNet50 at the `conv5_block1_2_relu` level with radial SVM with an accuracy of 0.5012. Comparing the models we can see that the two confusion matrices are very similar as expected from their very similar accuracy. What differentiates the two models are the number of parameters 138,357,544 and 25,636,712 and the prediction times of 75.913ms/26.601ms and 54.615ms/27.021ms for respectivly CPU/GPU.

Possible ways to improve could be:

- Better data augmentation (for example by acting directly on the temporal domain)
- Other subset of the dataset (medium, large)
- Optimization of hyper-parameters from start to finish
- Consider a different loss function and consider a greater number of epochs
- Other features as MFCCs and its derivatives
- Other models such as LSTM or RCNN could also be tried, even if the state of the art shows that they are not as performing as CNNs [4].

6 Conclusions

The task that was analyzed is the music genre classification. Various methods were tried. First an approach based on handcrafted features was tried, obtaining accuracy 0.495. Then, considering the state of the art, CNNs with mel-spectogram as input were chosen. Initially the CNNs were approached from scratch using an architecture suggested by [4] and the accuracy was 0.3975. Then hyperparameters optimization and data augmentation were made. This path was found to be the worst, obtaining 0.3900 without data augmentation and 0.3787 with it. Finally, considering the quantity of data available and the results previously obtained, transfer learning were tried. Two Top-5 accuracy CNN on imagenet were used to do feature extractions. The extracted features were reduced in size with PCA, which has been shown to improve the results. In the end the best model obtained was the one using ResNet50 at cut `conv5_block1_2_relu` with radial SVM, which has fewer parameters than VGG16 and is faster, with accuracy of 0.5012, reaching results close to 0.5163, that are achieved by the paper previously cited . The use of spectograms with CNNs was proved to be the best method in the literature to deal with this task. In order to obtain better results, however, a greater amount of data is required.

A Hyperband

Hyperband is an hyper-parameter optimization method invented by Li, Lisha, and Kevin Jamieson [5]. It was formulated as a pure-exploration nonstochastic infinite-armed bandit problem where a predefined resource like iterations, data samples, or features is allocated to randomly sampled configurations.

Hyperband extends the Successive Halving algorithm proposed for hyper-parameter optimization by Jamieson and Talwalkar [6].

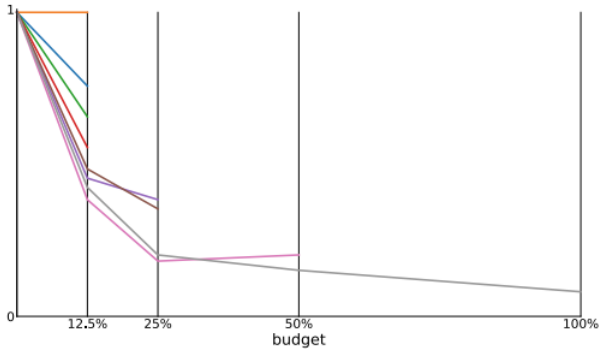


Figure 10: Successive Halving example.

The idea behind the original Successive Halving algorithm its about uniformly allocate a budget to a set of hyper-parameter configurations, evaluate the performance of all configurations, throw out the worst half, and repeat until one configuration remains. The algorithm allocates exponentially more resources to more promising configurations [5][6]. Unfortunately, Successive Halving requires the number of configurations n and some finite budget B as inputs to the algorithm. Than B/n resources are allocated on average across the configurations.

However, in many cases it's not known if it's better to consider many configurations (large n) with a small average training time or consider a small number of configurations (small n) with longer average training times.

Hyperband requires two inputs: R, the maximum amount of resource that can be allocated to a single configuration, and η,that controls the proportion of configurations discarded in each round of Successive Halving [5].

It tries to solve the problem by running several Successive Halving runs with different budgets and number of configurations, to find the best set.

Hyperband set $s_{max} = \lfloor \log_\eta B \rfloor$. It than begins with the most aggressive bracket $s = s_{max}$, which sets n to maximize exploration. Each subsequent bracket reduces n by a factor of η until the final bracket, s = 0, in which every configuration is allocated R resources (this bracket simply performs classical random search). Hence, Hyperband performs a geometric search in the average budget per configuration and removes the need to select n for a fixed budget at the cost of approximately $s_max + 1$ times more work than running Successive Halving for a single value of n. By doing so, Hyperband is able to exploit situations in which adaptive allocation works well, while protecting itself in situations where more conservative allocations are required [5].

References

[1] M. Defferrard, K. Benzi, P. Vandergheynst, and X. Bresson, "FMA: A dataset for music analysis," in *18th International Society for Music Information Retrieval Conference (ISMIR)*, 2017. [Online]. Available: https://arxiv.org/abs/1612.01840

[2] D. S. Park, W. Chan, Y. Zhang, C.-C. Chiu, B. Zoph, E. D. Cubuk, and Q. V. Le, "Specaugment: A simple data augmentation method for automatic speech recognition," *arXiv preprint arXiv:1904.08779*, 2019.

[3] Y. Zeng, H. Mao, D. Peng, and Z. Yi, "Spectrogram based multi-task audio classification," *Multimedia Tools and Applications*, vol. 78, no. 3, pp. 3705–3722, 2019.

[4] D. Kostrzewa, P. Kaminski, and R. Brzeski, "Music genre classification: Looking for the perfect network," in *International Conference on Computational Science*. Springer, 2021, pp. 55–67.

[5] L. Li, K. Jamieson, G. De Salvo, R. A. Talwalkar, and A. Hyperband, "A novel bandit-based approach to hyperparameter optimization," *Computer Vision and Pattern Recognition, arXiv: 1603.0656*, 2016.

[6] K. Jamieson and A. Talwalkar, "Non-stochastic best arm identification and hyperparameter optimization," in *Artificial Intelligence and Statistics*. PMLR, 2016, pp. 240–248.