POLITECNICO
MILANO 1863

PROJECT REPORT

Music genre classification with Deep Learning

PRACTICAL PROJECT IN NUMERICAL ANALYSIS FOR MACHINE LEARNING

Authors: ANDREA BOSISIO, ACHILLE GELLENS

Academic year: 2022-2023 **Course: Numerical Analysis for Machine Learning**

1. Introduction

The following is the report of the project that is part of the Numerical Analysis for Machine Learning course. The goal of the project is to put into practice some of the concepts covered in the course with the addition of different specific machine learning techniques that were not covered in the course's curriculum. In particular, the objective is to perform *music genre classification* through Deep Learning incorporating *Squeeze & Excitation (SE) Blocks* on a *Convolutional Neural Network (CNN)* model, as already done by the authors of the paper [14]. In order to do so, we start by replicating the model described in the beforementioned paper and then try to improve the performances by introducing new techniques. With this report, we want to describe the steps we took to solve the challenge, the problems we faced, and the differences we noticed in the results compared to the ones stated in the reference paper.

The code can be found in the following `GitHub` repository: `https://github.com/achgls/music-genre-classification`.

2. About the reference paper

The main part of this work is to study the paper [14] and to reproduce the learning process that it describes.

2.1. Summary

To perform music genre classification, the authors of [14] use a deep neural network based on CNN into which the *spectrograms* of audio clips are fed. Since this has been already done by the research community, they want to improve accuracy results with the addition of *Squeeze & Excitation Block (SE-Block)* to play the role of the *attention mechanism* in CNN. They also use *Bayesian optimization* to find the optimal value of the reduction ratio r, which is an important hyper-parameter inside the SE-Block.

2.2. Theoretical background

Convolutional Neural Networks (CNNs) are specialized deep learning models designed for analyzing grid-like data, such as images, by employing convolutional layers that apply filters to extract local patterns and spatial dependencies. The parameters of those filters are learned through an iterative numerical optimization process that minimizes a *loss function*, using a large number of examples of the type $\langle \texttt{input}, \texttt{class} \rangle$.

The ability of CNNs to extract hidden and meaningful features of an image – that are lately used to perform the actual classification through *fully connected layers* – can be exploited in this case by providing as input a 2-dimensional representation of audio clips, which is the *spectrogram*. Spectrogram-like representations are often derived from the *Short-Time Fourier Transform (STFT)* which is obtained by applying FFT to overlapping frames of the audio. This results in a 2D representation which represents FFT outputs stacked along the time axis. This representation makes the relevant information in the audio more readily available for ML models to extract as opposed to the raw 1D waveform, which has an inherent noisy nature.

The addition of SE-Block in each convolutional layer of the CNN introduce the attention mechanism, which allows the network to focus on and weigh important features during the learning process. The basic idea underlying SE is to incorporate – in each convonutional layer – one learnable parameter for each *feature map*, which is the result of the application of a filter to a previous convolution block. Considering a convolutional block, in the *squeeze* operation, the SE block is simply computing a numeric value using *average pooling* – that is, compute the average of the considered feature map – for each feature map. Therefore we get a vector $\mathbf{z}$ of dimension $1 \times 1 \times C$, where C is the number of channels (in this case, feature maps). Instead, the *excitation* operation is basically the application of a *Multilayer Perceptron*

(MLP) to $\mathbf{z}$ where, in this case, we have 2 layers characterized by the learnable weights matrices W_1 and W_2. The result of the MLP can be expressed by

$$\mathbf{s} = \sigma(W_2\delta(W_1\mathbf{z}))$$

where the *sigmoid* function $\sigma(\cdot)$ and the *ReLU* function $\delta(\cdot)$ are element-wise functions (i.e., they perform the same operation to each element of the input, outputting another point of the same dimension as the one of the input). Since the dimension of W_1 is $1 \times 1 \times \frac{C}{r}$ and the one of W_2 is $C \times \frac{C}{r}$, the result $\mathbf{s}$ has the same dimension as $\mathbf{z}$, as it contains a scale (or weight) that has to be multiplied to each of the feature map. Indeed, the last step of the SE procedure is to scale each feature map in input to the SE Block (U_c) as defined with the following equation:

$$X_c = s_c \cdot U_c$$

where s_c is the scalar component of $\mathbf{s}$ related to U_c and X_c is the c-th feature map of the computed convolutional block, which is the result of the SE procedure. The hyper-parameter r, which determines the dimension of both W_1 and W_2, is called *reduction ratio* and allows to vary the capacity and the computational cost related to the SE Block [7].

2.3. Criticisms

The reference paper [14] lacks clarity and precision in explaining certain techniques and processes, and it fails to provide specific parameters (for example the learning rate, the type of loss function, the rate of the decay in the learning rate scheduling, whether they normalize the data, whether they use regularization *e.g* weight decay, etc.), thereby making it challenging to properly replicate their experiments. Moreover, the fact that the code is not provided, does not help in that matter. In general, the reference paper is lacking in terms of reproducbility..

Regarding the data, it's not clear whether they used the raw magnitude STFT or the "power spectrogram" *i.e.* the squared magnitude of the STFT. According to our early experiments, we concluded that they were using the squared STFT.

The main problem, however, is that it is not specified how they split the dataset into train and test set: it is a crucial point for the evaluation of the model to know whether the split of the dataset has been done before or after the already mentioned audio framing process. This is because by evaluating the model on some audio clips of a song of which some other audio clips has been used for training the model (i.e., by performing audio framing before doing the test-train dataset split), it could introduce a bias on the accuracy results, since the samples are not independent, as audio framing is performed with overlaps and as it is likely that some audio clips of the same songs are highly similar. Considering the first results we had with training the model described in the guideline paper, we came to the hypothesis that they were using contaminated data. To verify this hypothesis, we, made a contaminated dataset class and conducted experiments with both clean and contaminated data. Our analysis of training and evaluating the model described in the paper in both of these two cases shows that by using *contaminated dataset*, we get similar performance metrics as the ones stated in the paper while training the model with a clean uncontaminated split results in much lower testing accuracy. The curves showed in 1 speak for themselves and support our hypothesis.

Another criticism concerning the results is that they mention performances of other models without providing descriptions of them, also leaving it unclear whether these results were achieved by the paper's author or by another research group. Ideally their results table should provide a reference to their bibliography for each model.

3. About the Dataset

The dataset used is the GTZAN dataset [13], which consists of 1000 audio tracks, each 30 seconds long. It contains 10 different music genres (Blues, Classical, Country, Disco, Hip Hop, Jazz, Metal, Pop, Reggae and Rock), each represented by 100 tracks[1] (i.e., balanced dataset).

For what concern the audio data, we've firstly downloaded the dataset from the dataset link originally provided but after having realized that some of the files were faulty, we've decided to use a version of that dataset that we found did not contain those faulty files [3].

For easier usage, we converted the `.au` files present in this version of the dataset into `.wav` format and precomputed spectrograms for conveniency. We made our own version of the dataset [5] which aims to make it moe readily usable as it is one of these datasets that are hard to find a proper souce for (the original dead is link). Following that idea, we also managed to retrieve song information using a Shazam-like python API, and we will soon add them as metadata

[1] The tracks are all 22050Hz Mono 16-bit audio files in `.wav` format.

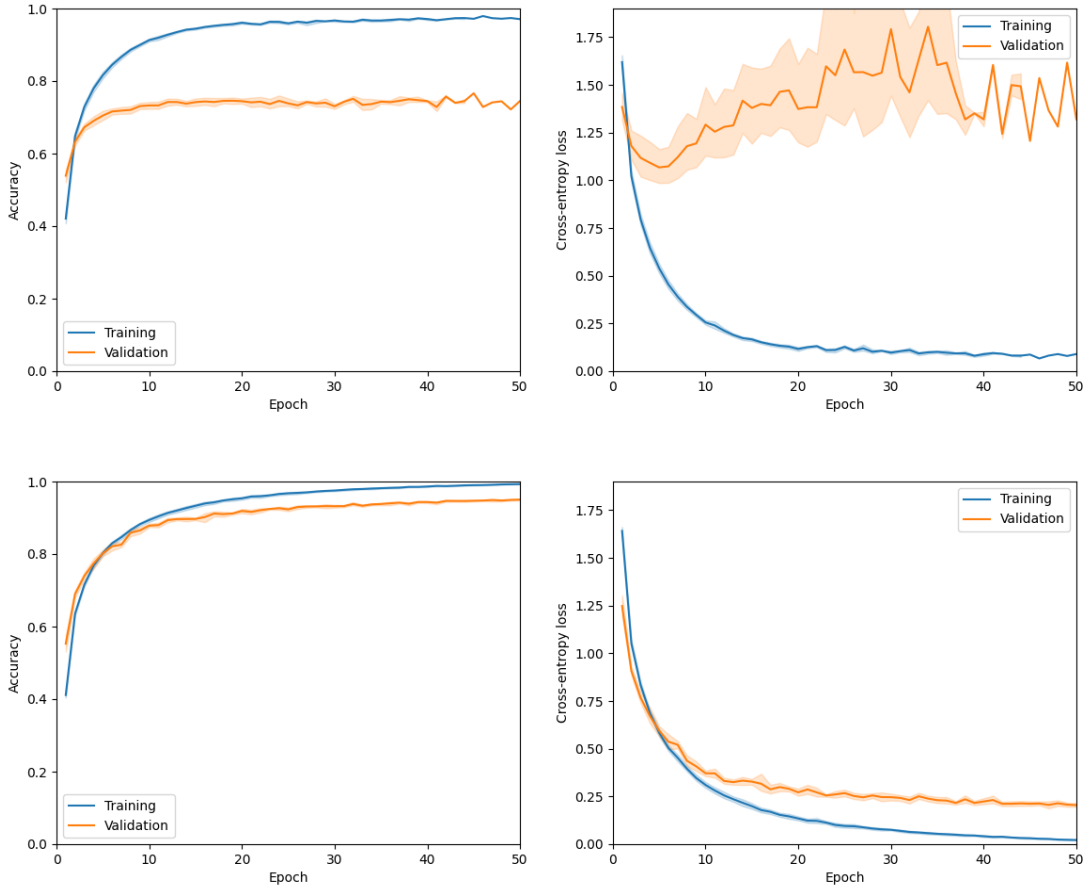

Figure 1: Training / validation accuracy and loss over training for the reference SE-CNN model described in [14]. Top: clean data split. Bottom: contaminated data split. Note that the bottom curves look exactly similar to the training metrics curves provided in the guideline paper, thus supporting our hypothesis that they made their claims based on contaminated training.

for our dataset. Under the `Code` section of our Kaggle dataset, you will find some additional notebooks not present in the repo, *e.g* the notebook used to precompute and store the spectrogram images.

Yet, we retained the CSV files from the first linked dataset to both train and validate our baseline model, which is a simple MLP that takes as input the features contained in two the CSV files. Those are mean and variance of some handcrafted features computed on each audio clip (of 3-seconds or 30-seconds depending on the file). The handcrafted features considered are: 20 *Mel-Frequency Cepstral Coefficients*, the *Zero Crossing Rate*, the *Harmonics and Perceptrual*, the *Tempo BMP (beats per minute)*, the *Spectral Centroid*, the *Spectral Rolloff* and the *Chroma Frequencies*. Each of these meaningful audio features are explained by the author of the two CSV files through this notebook.

Furthermore, we should mention that results may be affected by the quality of the dataset. In fact, it is known that the GTZAN dataset contain some defects such as replications, mislabelings, and distortions. Please refer to Bob L. Sturm's analysis of the data for more details [12].

3.1. Data exploration

An analysis on the data present in the 2 CSV files is carried out in the first place, in order to both better understand the meaning of the handcrafted features used and to measure their potential. This is beneficial as those 2 dataset are used to train and evaluate our baseline model. The performed analysis is mainly related to *Principal Component Analysis (PCA)* which is implicitly carried out with *Singular Value Decomposition (SVD)* using the `sklearn.decomposition.PCA` class. In order to perform correctly PCA, the data is scaled using `sklearn.preprocessing.StandardScaler`, which standardize features by removing the mean and scaling to unit variance. By projecting the data of the

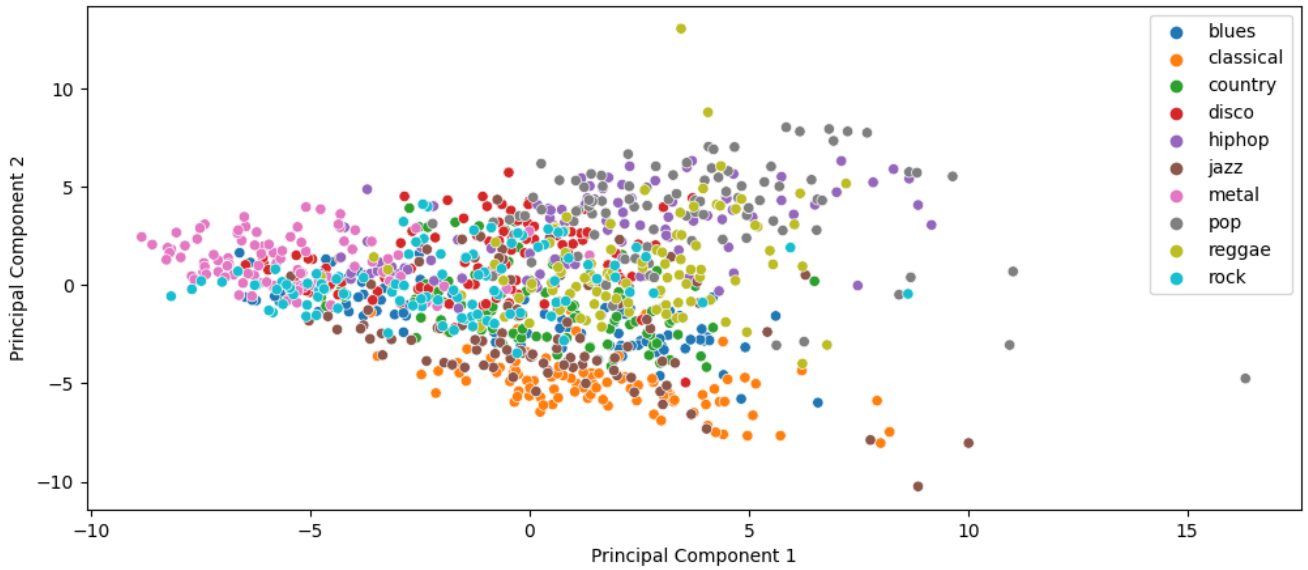

Figure 2: 2D Principal Component Analysis of the data `features_30_seconds.csv`

full audio clips on the first two *principal components*, some clusters and pattern can been seen (Figure 2). However, this is not sufficient to correctly separate the dataset in this combined-feature space. This is also explained by looking at the graphs in Figure 3, in which functions of the *singular values* obtained with PCA on the considered dataset are plotted. The first two singular values (related to the first two principal components), are able to explain only the 52% of the variance of the dataset. In order to capture the majority of the information of the dataset (for example, to achieve the 95% of explained variance), more than the first 30 principal component should be used. Therefore, it may not be particularly justified to use PCA for dimensionality reduction since the original number of feature is 57, and by discarding some of the components we could neglect important information of the dataset.

3.2. Data pre-processing and splitting

Following what's stated in the paper of reference, we augment the size of the dataset by performing *audio framing*: each track is split through window functions in audio clips of the same size with 50% of overlap. After these operations, each 30-second music track is divided into 19 3-seconds audio clips, so the total number of data points in the dataset becomes 19000. During training or evaluation, the spectrogram is generated on-the-fly for each audio excerpt using the `Spectrogram` class of the `torchaudio` library [15]. The parameters used to do so are: `n_fft=1024, win_length=1024, hop_length=512, power=2.0` where `power` is set to 2 because, being the exponent of the magnitude spectrogram, we are interested in computing the power spectral density of the signal.

This process is performed to avoid *overfitting*. In fact, by increasing the number of training sample we are decreasing the variance of the CNN model, which is an *unstable learner* due to its high number of learnable parameters (i.e., large hypothesis space).

As done in the reference work, the dataset is split into a train (80%) and a validation/test set (20%). In our case, the validation and testing set are identical since we haven't performed any kind model selection, as our main goal is to replicate the model described in the paper. However, it is important to mention that this split is done *before* performing the augmentation procedure mentioned before, as this would introduce a bias on the evaluation of the model performances. As already mentioned in 2.3, we also have a *contaminated* version of the dataset in which the dataset split is done *after* having performed audio framing. This is used to compute the results shown in Figure 1.

3.3. Online data augmentation

In order to try to improve the learning process, we experiment *online data augmentation* – differently to what is described in Section 3.2 – in order to avoid overfitting by increasing the number of samples. In this case, data augmentation is used to mainly introduce noise to the samples, both the audio and the spectrogram ones and it is done *epoch-wise* during training. For what concern the audio files, a combination of two kind of transformations is applied:

- `RandomWhiteNoise`: add noise to the audio sig-

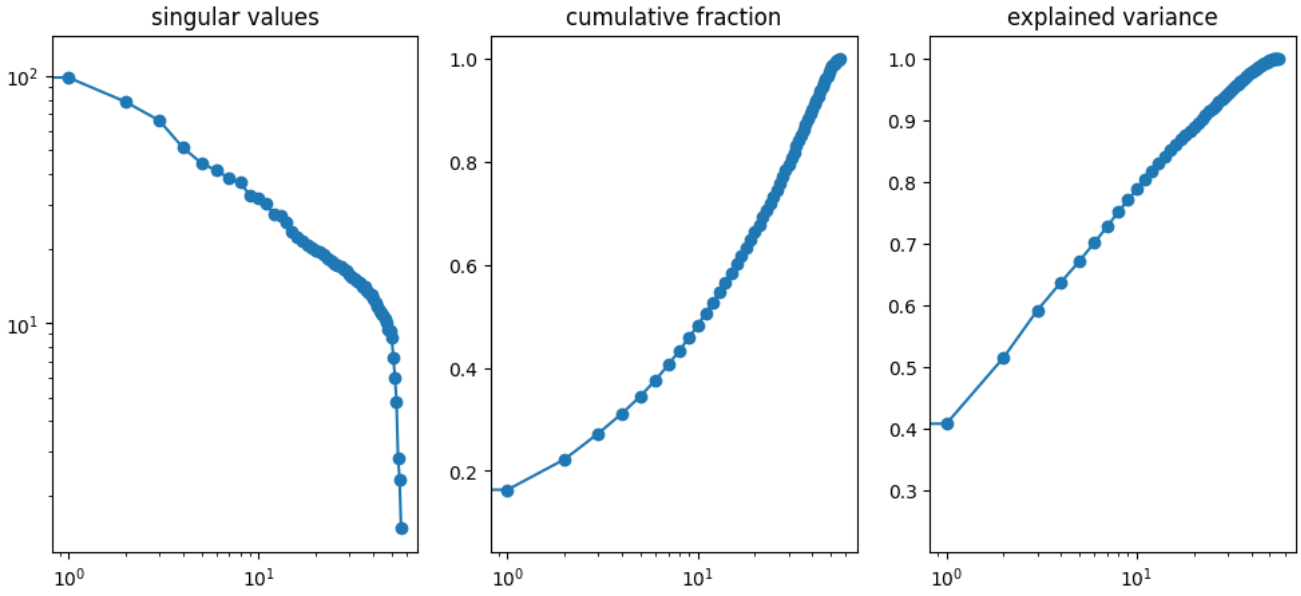

Figure 3: Singular Values plots of the data `features_30_seconds.csv`

nal

- `RandomGain`: randomly modifies the amplitude of the audio clip

Instead, for what concern the spectrogram images generated from the audio files, the transformations considered as a combination are:

- `FrequencyMasking`: which mask the spectrogram content of the specified frequencies, for each time frame
- `TimeMasking`: which mask the spectrogram content for the specified frame segment, for each frequency

and they are both provided as `torchaudio` classes.

4. Models

As already stated, we are presenting a baseline model, the model that we want to reproduce from the reference paper [14] and the changes that we used when trying to improve the performances of the model.

4.1. Baseline

The baseline model is a MLP trained using the CSV files described in Section 3.

The structure and the parameters used for the baseline model is shown with Listing 4.1.

Listing 1: Baseline model pythorch definition

```
model = nn.Sequential(
          nn.Linear(57, 256),
          nn.ReLU(),
          nn.Dropout(0.4),
          nn.Linear(256, 64),
          nn.ReLU(),
          nn.Dropout(0.2),
          nn.Linear(64, 10)
        )

loss_fn = nn.CrossEntropyLoss()
scheduler=None
optimizer=torch.optim.Adam(model.parameters(),
    weight_decay=0.0001)
```

The *learning rate* is set to 0.001 by default in this case. *Dropout* is used to force the network to learn different and independent learning path by shutting down a random set of neurons. This helps with respect to the overfitting problem [11]. *Adam optimizer* is preferred over more classical methods such as *Stochastic Gradient Descend (SGD)* because of its known better ability to find good local minima of the loss function [8], which in this case is the *CrossEntropy Loss* (since the problem is classification). Another technique used to overcome the overfitting problem is *Early Stopping*, which is a stopping condition of the training phase that is triggered when the validation accuracy is not improving for more than `early_stopping` epochs. In this case this value is set to 30 and the maximum number of epochs is 300.

Layer	Filter shape	Stride and padding
Conv	$3 \times 3 \times 1 \times 32$	$1 \times 1, 1 \times 1$
MaxPooling	pool: 2×2	$2 \times 2, -$
Dropout	rate: 0.25	-
SE-Block	r : 32	-
Conv	$3 \times 3 \times 32 \times 64$	$1 \times 1, 1 \times 1$
MaxPooling	pool: 2×2	$2 \times 2, -$
Dropout	rate: 0.25	-
SE-Block	r : 32	-
Conv	$3 \times 3 \times 64 \times 128$	$1 \times 1, 1 \times 1$
MaxPooling	pool: 2×2	$2 \times 2, -$
Dropout	rate: 0.25	-
SE-Block	r : 32	-
Conv	$3 \times 3 \times 128 \times 256$	$1 \times 1, 1 \times 1$
MaxPooling	pool: 2×2	$2 \times 2, -$
Dropout	rate: 0.25	-
SE-Block	r : 32	-
Conv	$3 \times 3 \times 256 \times 128$	$1 \times 1, 1 \times 1$
MaxPooling	pool: 2×2	-
Dropout	rate: 0.25	-
SE-Block	r : 32	-
Flatten	-	-
Dense	Softmax Output: 1×10	-

Table 1: Structure of the reference CNN using SE Blocks

4.2. Reference CNN+SE

Even though the paper lacks of the definition of some parameter, we try to reproduce their model with the one described with Table 1. This model is a CNN with the introduction of SE-Block on each of the convolutional layers. *MaxPooling* is used in order to reduce the spatial dimensionality of the feature maps at each convolutional block. This filter does it by taking the maximum value in a grid of dimension defined by the `pool` parameter. The input shape is 512×130 since the input of this model are the spectrograms of 3-seconds audio slices – as explained in Section 3.2.

For what concern the training hyper-parameters, we use the following for each configuration of the model:

- learning rate: 0.001;
- batch size: 64;
- weight decay: 0.0001 (L2-norm regularization);
- optimizer: Adam;
- LR scheduling: the learning rate is linearly decreased to a $\frac{1}{10}$ of its value over 50 epochs
- early stopping: 10 epochs
- maximum number of epochs: 50

4.3. ResNet

The architecture proposed in the reference paper [14] is a very naive, basic CNN architecture, in spite of the added squeeze-excitation mechanism. When it comes to CNN, Residual Networks, *a.k.a* ResNets, have been a gold standard ever since they have been introduced by He, et. al. in 2015 [6]. The identity-forwarding mechanism introduced by these models is used in most novel architectures.

We thought that we could obtain better results with more refined CNN architectures, but it extremely overfitted. This could be due to the lack of good quality and representative data with respect to the complexity and expressiveness of an architecture like ResNet. This is why we avoid listing the results we have obtained with this model in the Section 5.

5. Results

The following results showcase our experiments with the baseline MLP model, as well as the CNN model as described in the reference paper [14] and in Table 1. For the MLP model, our experiments were designed to provide a baseline for music genre classification, with a perspective on "traditional ML", or "feature engineering" method, as opposed to more *end-to-end* Deep Learning methods based on raw representations. For the CNN model, we experimented with two improvements upon the base model:

1. using *Global Average Pooling (GAP)* [9] layer as a pooling layer for the final full-connected layer of the network, in place of the usual "flattening" layer
2. applying *online data augmentation* during training as explained in Section 3.3

Using GAP as a pooling layer shows several advantages. First, you don't need to specify an input size anymore, your model becomes adaptable to varying shape input. Second, since you pool the final vector into a fixed-size vector which has no positional information, it becomes easier to consider the CNN output as latent embeddings, and we can easily project our audio samples into this CNN-derived feature space (see Figure 5). Another advantage is that it is able to smooth out some positional bias that your data might have.

K-Fold Cross Validation is used for evaluating the different models. It is a technique that divides the dataset into K subsets, or folds, for training and evaluation, iteratively using each fold as a validation set while the remaining folds serve as the training set. This helps assess the model's performance and reduce the impact of data variability [1]. In particular, `sklearn.model_selection.StratifiedKFold` is used in order to keep the fold balanced (with respect to the number of samples for each genre). We used the exact same 5-fold split across all experiments be-

Fold	Vanilla	w/ Data augmentation
1	76.29 ± 0.26	78.37
2	75.13 ± 1.08	80.42
3	78.07 ± 0.53	78.13
4	77.13 ± 0.56	79.79
5	75.09 ± 0.58	77.26
Avg.	76.34 ± 1.32	$\mathbf{78.79 \pm 1.15}$

Table 2: Extract-wise test accuracy with simple flattening, *i.e.* the model described in Table 1

Fold	Vanilla	w/ Data augmentation
1	74.34	73.82
2	75.82	73.92
3	79.58	64.97
4	79.16	78.42
5	76.79	77.61
Avg.	$\mathbf{77.14 \pm 1.99}$	73.75 ± 4.77

Table 3: Extract-wise test accuracy with Global Average Pooling (GAP) in place of the flattening layer

low, *i.e.* the fold identifier represents the same data across all following results. This consistent split was obtained by 5-folding on all sorted song filenames with a set seed `123456789`, which is an attribute of the implemented `GTZANDataset` class.

For evaluating model's performance, we can refer at *extract-wise* or *file-wise* accuracy. The extract accuracy is the one computed during training and validation rounds, and refers to the 3.0 seconds slices extracted by our `GTZANDataset` object. The file-wise accuracy refers to the model's decision based on averaging the predictions across the 19 extracts a file is composed from. This score fusion can be seen as ensembling, and will surely always show better accuracy.

Figure 4 show training curves for the different training configurations studied in this section. From these figures, it seems that the change that contributes most to preventing overfitting is the data augmentation which drastically changes the looks of the curves

Fold	Vanilla	w/ Data augmentation
1	81.5	83.0
2	85.0	79.5
3	86.0	74.5
4	**96.5**	**97.0**
5	95.5	92.0
Avg.	$\mathbf{88.9 \pm 5.99}$	85.20 ± 8.21

Table 4: File-wise test accuracy with Global Average Pooling (GAP) in place of the flattening layer

Fold	Vanilla	w/ Data augmentation
1	$\mathbf{84.5 \pm 0.41}$	85.0
2	78.83 ± 3.40	**87.5**
3	81.17 ± 1.65	84.5
4	80.50 ± 2.68	85.0
5	80.17 ± 1.25	82.0
Avg.	81.03 ± 2.87	$\mathbf{84.80 \pm 1.75}$

Table 5: File-wise test accuracy with simple flattening, *i.e.* the model described in Table 1

when present. Regarding the GAP layer, it seems to be slightly reducing the "plateau" in accuracy which is notably visible in the experiments where simple flattening was used. Looking at the curves, the configuration GAP + data aug. seems to be the most promising and it seems like the model could have kept on converging for a few epochs, and that the training was stopped a bit too early.

For what concern the baseline model, we didn't have time to train and test properly with KFold cross validation, as we have done for the CNN model. However, by using a non-contaminated split of the dataset `features_3_seconds.csv` we get 73% of accuracy on the test set, while we get 91% of accuracy on a contaminated split. Instead, by using the dataset `features_30_seconds.csv` and performing a single split for train and test (non-contaminated by definition, as there is only one audio clip for each song), we get 80% of accuracy. We can't derive any conclusion with this kind of setting, since it is only 1 fold. However, it is again confirmed that using a contaminated split of the dataset boosts the performance while carrying a bias.

5.1. Analysis of extracted features from the CNN

To compare the learned features by the CNN with respect to the handcrafted ones, we have also performed PCA on those extracted features. They have been defined by performing *Global Average Pooling* on the last convolutional layer. In particular, we used the model trained in the 4th fold of Table 3, trained without data aug. The result of projecting the new feature space onto the first 2 principal components is shown in Figure 5. As can be seen, the separability of genres in the CNN-derived feature space can arguably be said to have increased compared to the hand-crafted feature space (see Figure 2). In any case, it is to be remembered that this feature space was derived from a very low-level, raw representation of the data, as op-

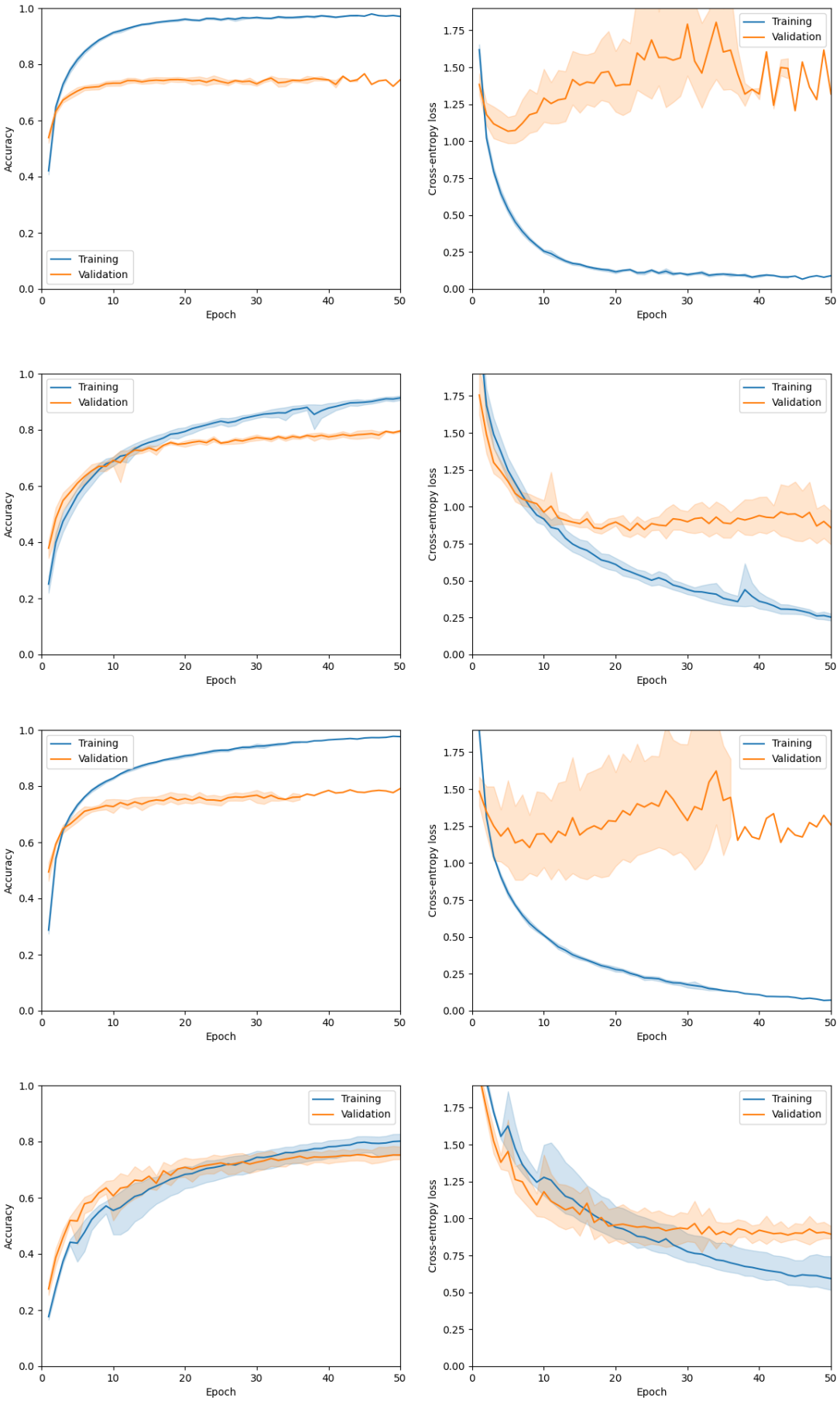

Figure 4: Training metrics for different configurations across folds and experiments. From top to bottom: 1. reference CNN, 2. w/ data aug., 3. ref. CNN w/ GAP pooling, 4. GAP and data aug.

Figure 5: 2D Principal Component Analysis of the feature extracted by the CNN with GAP

posed to the higher-level, feature-engineered, tailor-made feature space generated by the hand-crafted features. This is promising in itself.

6. Future work

Unfortunately, there are a lot of things that we could not explore in the time frame of the project.

Namely, we would have hoped to try out several other architectures.

Firstly, in order to capture structural informations from the song, we would have liked to add a LSTM (or RNN-based) component on top of the CNN feature extractor.

Alongn that same idea, we would have liked to experiment with Audio Transformers, an architecture that proved to excel in natural language processing tasks, and could also help capture long-range sequential structural patterns in audio. Unlike CNN-LSTM, which processing can not be parallelized, the transformers provide scability and efficiency by being easily parallelizable.

Lastly, in pure Deep Learning fashion, we would have liked to experiment with models that learn directly from the raw waveform, as opposed to spectro-temporal representations of sound. This task was thought to be hard but recently, some models have succesfully been able to learn from the raw waveform, at a low computational cost. This is the case of SincNet [10] which uses parameterized sinc functions to learn its own spectral embedding.

On another note, we also wanted to perform cross-dataset evaluation as it can provide insights into the actual generalization capabilities of the models and uncover potential biases. We had identified a couple of datasets for this task. Namely, FMA [4], and MTG-Jamendo [2] should be datasets to look into for this.

7. Conclusion

In conclusion, our project on Music Genre Classification with Deep Learning CNN highlighted the challenges associated with audio deep learning. Our work demonstrated that Deep Learning models have the capability to learn significant features from low-level spectro-temporal representations, paving the way for the utilization of raw representations in various audio tasks. This breakthrough signifies a significant advancement in the field, opening up new possibilities for more efficient and accurate audio analysis and classification techniques.

Furthermore, our project provided valuable insights into the significance of the evaluation phase in a learning process. We discovered that even minor alterations in the data pre-processing stage could yield performance outcomes not representing the truth. Surprisingly, we observed that even papers accepted at conferences may contain errors when calculating the performance of proposed models. It is essential to

always verify the validation process employed before trusting the results. This is why a paper should always be supplemented with its corresponding source code.

References

[1] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag, Berlin, Heidelberg, 2006.

[2] Dmitry Bogdanov, Minz Won, Philip Tovstogan, Alastair Porter, and Xavier Serra. The mtg-jamendo dataset for automatic music tagging. In *Machine Learning for Music Discovery Workshop, International Conference on Machine Learning (ICML 2019)*, Long Beach, CA, United States, 2019.

[3] Thomé Carl. Kaggle dataset, `GTZAN Genre Collection`. https://www.kaggle.com/datasets/carlthome/gtzan-genre-collection, 2019.

[4] Michaël Defferrard, Kirell Benzi, Pierre Vandergheynst, and Xavier Bresson. Fma: A dataset for music analysis, 2017.

[5] Achille Gellens and Andrea Bosisio. Kaggle dataset, `GTZAN Music Genre Classification Dataset`. https://www.kaggle.com/datasets/achgls/gtzan-music-genre, 2023.

[6] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015.

[7] Jie Hu, Li Shen, Samuel Albanie, Gang Sun, and Enhua Wu. Squeeze-and-excitation networks, 2019.

[8] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.

[9] Min Lin, Qiang Chen, and Shuicheng Yan. Network in network, 2014.

[10] Mirco Ravanelli and Yoshua Bengio. Speaker recognition from raw waveform with sincnet, 2019.

[11] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958, 2014.

[12] Bob L. Sturm. An analysis of the gtzan music genre dataset. In *Proceedings of the Second International ACM Workshop on Music Information Retrieval with User-Centered and Multimodal Strategies*, MIRUM '12, page 7–12, New York, NY, USA, 2012. Association for Computing Machinery.

[13] G. Tzanetakis and P. Cook. Musical genre classification of audio signals. *IEEE Transactions on Speech and Audio Processing*, 10(5):293–302, 2002.

[14] Yijie Xu and Wuneng Zhou. A deep music genres classification model based on cnn with squeeze & excitation block. In *2020 Asia-Pacific Signal and Information Processing Association Annual Summit and Conference (APSIPA ASC)*, pages 332–338, 2020.

[15] Yao-Yuan Yang, Moto Hira, Zhaoheng Ni, Anjali Chourdia, Artyom Astafurov, Caroline Chen, Ching-Feng Yeh, Christian Puhrsch, David Pollack, Dmitriy Genzel, Donny Greenberg, Edward Z. Yang, Jason Lian, Jay Mahadeokar, Jeff Hwang, Ji Chen, Peter Goldsborough, Prabhat Roy, Sean Narenthiran, Shinji Watanabe, Soumith Chintala, Vincent Quenneville-Bélair, and Yangyang Shi. Torchaudio: Building blocks for audio and speech processing, 2022.