Music Genre Recognition with Convolutional Neural Networks

Diego Bonato, Luca Menti, Alberto Saretto

***Abstract*—This report explores the application of advanced Convolutional Neural Network (CNN) architectures for music genre classification. The CNNs developed in this project were trained using various audio signal formats to determine which data representation is most effective for feature extraction. The Free Music Archive (FMA) dataset was used for this purpose. Baseline models included a 1D-CNN designed for processing raw audio signals and a 2D-CNN tailored for spectrogram analysis. The research then focused on proposing and implementing enhancements to improve model performance. One successful strategy involved integrating information from both raw audio data and spectrograms, with the combined contribution of these inputs leading to better performance. By leveraging advanced CNN architectures and exploring innovative methods to fuse information from diverse data formats, this study aims to enhance our understanding of music genre classification and potentially improve the accuracy and robustness of classification models in the field of audio signal processing.**

***Index Terms*—Music Genre Classification, Convolutional Neural Networks, Residual Networks.**

I. Introduction

Over the past few decades, the music industry has increased. Platforms like Spotify and iMusic have played important roles in facilitating access to music content anytime, anywhere. Given the exponential growth in content, the assistance of machine learning for music genre recognition (MGR) has become indispensable in enhancing the efficiency and effectiveness of these processes. The classification methodology for MGR falls within the domain of Music Information Retrieval (MIR) systems, which are crafted to handle different tasks such as tagging, clustering, ranking, and classification. Typically, MIR systems follow a three-step strategy: firstly, hand-crafted features are extracted from single or multiple frames of music audio; then, these short-term features are aggregated over the entire audio to create a comprehensive music representation or descriptor; finally, a machine learning model is trained using these summarized representations to perform the specific task at hand [1]. Although there have been satisfactory enhancements in the performance of MIR through the three-step process, which remains effective despite ongoing optimization efforts, insufficient attention has been directed towards modeling the interaction among music segments. Moreover, the conventional approach of aggregating short-term features by computing statistical summaries over time seems unsuitable for generating a comprehensive and discriminative music representation. As showed in [2] a Deep Learning approach allows us to overcome this challenge. By the use of Convolutional Neural Networks (CNN) fed by an audio spectrogram or Mel-spectrum (CITARE 8) the accuracy of classification has improved notably. In this project, we explore different potential architectures to improve the performance of the 2D-CNN model proposed by [3]. Moreover, we introduce a new 1D-CNN model fed by audio tracks in the time-amplitude domain. Eventually, a third neural network is presented, which extracts information from both the 1D and 2D audio representations to perform the MGR task. This report is structured as follows: in section II, a brief summary of the literature on MGR is presented and discussed. In section III, the processing pipeline followed for the development and training of the neural networks is presented. In section IV we illustrate the pre-processing of the input audio data. In section V the architectures and learning frameworks of the designed networks are described in detail. Finally, in section VI the final results are shown and discussed.

II. Related Work

In the last decade, deep learning architectures such as Convolutional Neural Networks (CNNs) have led to very satisfactory results in Music Information Retrieval (MIR) applications. In literature, it is possible to find several approaches aimed to boosting their performance. In [2] for example, the authors suggest several strategies to improve feature learning from audio data using neural networks. These include using Rectified Linear Units (ReLUs) in place of standard sigmoid units, implementing Dropout as regularization technique and a Hessian-Free (HF) optimization to improve the training of sigmoid nets. The study shows how these methods lead to an improvement both in training time and in learning, which surpass the handcrafted features shown in literature. Another approach proposed in [1], involves the introduction of a CNN with a k-max pooling layer. Furthermore, in the work described in [3], two methods are proposed to improve music genre classification using CNNs: combining max- and average-pooling to provide more statistical information to higher-level neural networks, and introducing residual connections to skip one or more layers (inspired by residual learning). In that case, the input of the CNN is the short-time Fourier transforms (STFT) of the audio signal and the CNN's output is fed into another deep neural network for classification. Preliminary experimental results indicate that both methods effectively improve classification accuracy, with the second one showing particularly promising results. These three works illustrate a sequence of successive improvements, incorporating findings from earlier studies. Our work builds on these studies and aims to introduce optimization of hy-

perparameters for the 2D-CNN proposed in [3], develop a novel 1D-CNN neural network fed by audio tracks in the time-amplitude domain, and combine information obtained from these two CNNs to assess potential enhancements in classification accuracy.

III. Processing Pipeline

In this section, we discuss the modus operandi followed during the development of this project. Besides general deep learning best practices, we followed the tips contained in Google's Deep-Learning Tuning playbook [4]. The idea is to start by getting a solid baseline, and then fine-tune it by introducing incremental changes to either the model architecture or the data processing. A modification is accepted only if it leads to better performance in the adopted metrics. As said, the process is incremental, in the sense that when a change is found to be beneficial, it is kept fixed and never tuned again, even when further changes are made on other hyperparameters. For example, if we find that a certain number of convolutional layers is optimal, we treat that number as a fixed parameter when assessing the performance of, say, BatchNorm on the net. This is done to avoid trying all possible combinations of the hyper-parameters. An exception was made for the learning rate, which has been tuned each time a new change was made to the network. We used Optuna to find the best choice of the hyperparameters [5].

To solve the music genre classification task we built three neural networks: one working with 1D audio signals (NNET1), one with 2D spectrograms (NNET2) and a third one that exploits both 1D and 2D representations (MixNet). All nets are composed of a convolutional block and a fully-connected block. In the Learning framework section (V) we explain these architectures in detail. Our approach consists in fine-tuning NNET1 and NNET2 separately, and then use the best models found to create the MixNet, of which only the final classifier is tuned.

Since we deal with a multi-class classification problem, we choose the multi-class Cross-Entropy as loss function. We use the accuracy and the F1 score as metrics (see section VI, even if the former could be enough as we are dealing with a balanced dataset. We also plot a confusion matrix to further inspect how the model deals with different music genres. During the fine-tuning process, the main metric considered has been the loss, followed by the accuracy, the convergence speed and finally the confusion matrix.

We choose the batch-size to be the maximum value possible that our computers could handle, in order to make the training faster and more stable. Then, we tested both Adam and SGD with momentum as optimizers. Moreover, we tried different processing of the data. First of all, we searched for an optimal value of sampling rate and clip length, balancing between manageability and performance. Then, we assessed the performance of two normalization strategies. Afterwards, we tried two kinds of spectrograms for the 2D net (see section IV). Finally, we tried various data augmentation techniques, tuning the probability of applying a certain transformation. Other important changes have been tested regarding the model architecture. In the order, we experimented with different number of layers of the CNN part; number of channels of the CNNs; kernel sizes; number of layers and of neurons of the fully-connected blocks; and eventually the performance of BatchNorm and Dropout. We stress the fact that these changes have been studied incrementally. Lastly, we fine-tuned weight decay and moment of the Adam optimizer.

IV. Signals and Features

The dataset used is the Free Music Archive (FMA), a free and open library directed by WFMU [6]. The FMA provides 917 GiB and 343 days of Creative Commons-licensed audio from 106,574 tracks. It contains full-length and high-quality audio, pre-computed features, together with track- and user-level metadata such as fine genre information, making it suited for music genre recognition tasks. In this study, we used the proposed small subsets of the FMA. It is composed of 8,000 30s tracks distributed in eight different root genres: Electronic, Experimental, Folk, Hip-Hop, Instrumental, International, Pop and Rock. All tracks are mp3-encoded, most of them with sampling rate of 44,100 Hz and a bit rate of 320 kbit/s (263 kbit/s on average). The dataset is already divided into 8/1/1 train, validation and test splits. The distribution of the genres in these set is balanced. FMA provides a metadata dataframe which contains information such as tracks genres, artist and path index to the correspondent mp3 file. This allows for a better manageability of data compared to working with the full dataset containing track files. The FMA dataset comes with a Pandas dataframe containing metadata, such as the genre labels for each track. During the pre-processing phase, we converted these labels to one-hot encoded arrays. To solve the music genre classification task, we exploit the two possible formats of audio, namely 1D waveforms and 2D spectrograms. The former are simply obtained by converting a mp3 file into a 1D-array, which represents amplitude as a function of time. Mp3 tracks are directly extracted from the FMA small dataset using `librosa` library [7]. To obtain lighter and more manageable input data for the CNNs, smaller clips of 6s (equivalent to 2^18 samples at a sampling rate of 44100 Hz) are extracted randomly during the train and validation steps. This is good for data-augmentation too. However, during the test phase these windows should be kept fixed to obtain a more solid comparison between different models. To this aim, windows are extracted from the test set always by considering the mid-point of the track as starting point. This is crucial for an effective comparison of the performance of different CNN models. The resulting input clips for the 1D CNN is shown in Fig. 1.

2D based CNN models take as input data audio spectrograms which are obtained from 1D waveforms. In this study we considered two different types of spectrograms: *Mel* and *MFCC* spectrograms. Mel spectrograms show the distribution of energy among different frequencies over time, using the Mel scale. It is widely used in audio processing and machine learning applications, especially in speech and music

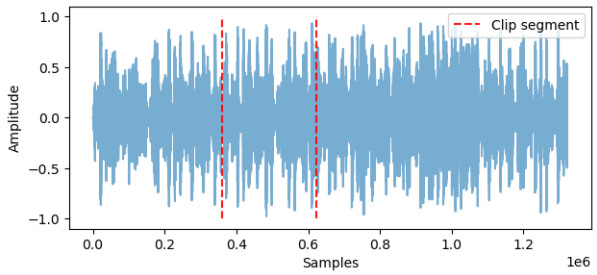


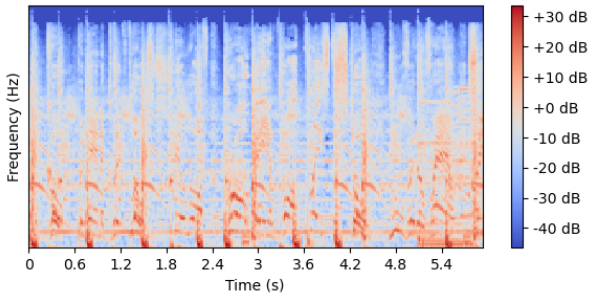
Fig. 1: **1D audio signal.** Audio signal of a FMA dataset track. Signals are composed by a one-dimensional array that represents the amplitude of the audio signal over time. The red dashes represent the 6s time-window given to the neural network.

recognition as it provides a more perceptually relevant representation of the sound signal. To obtain a Mel spectrogram one needs to compute first the short-time Fourier transform (STFT) on the audio signal. Each STFT frame spans 46 ms (2048 samples) of audio signals with 50% overlapping between adjacent frames. Then, frequencies are mapped into the Mel scale by applying 128 overlapping triangular filters that calculate the energy of the spectrum in each band. By doing this, a 256×128 spectrogram is obtained, which is then converted into log scale that compresses the dynamic range of the signal, making it easier to see the relative strengths of the different frequency components. Instead, Mel-Frequency Cepstral Coefficients (MFCC) spectrograms are obtained by compacting the information contained into Mel spectrograms into a smaller set of coefficients. In this project, MFCC spectrograms are obtained considering the first 20 Mel-frequency spectral coefficients, thus obtaining a 256×20 input data vector. Moreover, inspecting the raw audio clips we see that they are already normalized in a range between -1 and +1. On the other hand, both the 2D spectrograms can be normalized using the `preprocessing.StandardScaler()` function from python sci-kit learn. This subtracts the mean to each input data and divides it by its standard deviation. The resulting spectrograms are reported in Fig.2

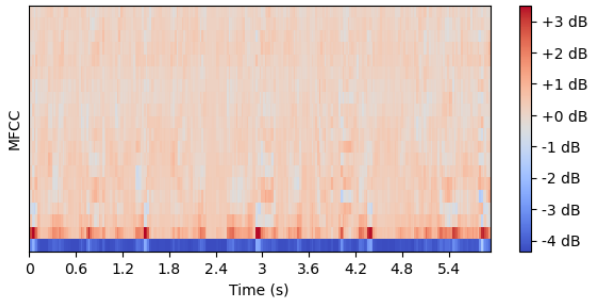
V. Learning Framework

A. NNET1: 1D Convolutional Neural Networks

We call NNET1 the neural network designed for processing 1D audio clips. It is comprised of a convolutional block that extracts relevant features, followed by a fully-connected block and a Softmax layer that performs the final classification. The two blocks are connected to each other via concatenation of a Max-Pool and Average-Pool layer. The convolutional block is formed by four 1D-convolutional layers, each but the last one followed by a Max-Pooling layer. Batchnorm is applied to each convolutional layer. The activation function is the ReLU. Weights are initialised using the Xavier prescription and bias is set to 0. We find dropout to worsen the performance of the net. As for data augmentation, we use the *audiomentations* library [8]. We apply gaussian noise with a probability of



(a) Mel Spectrogram



(b) MFCC Spectrogram

Fig. 2: **2D spectrograms.** Mel spectrogram (a) and MFCC spectrogram (b) of an audio clip. Log scale is applied to the Mel spectrogram. MFCC spectrograms are normalised through the `preprocessing.StandardScaler()` from sci-kit learn

50%, minimum amplitude of 0.001, maximum amplitude of 0.015. A time-mask is applied too, with probability equal to 1, ranging from 0.1 and 0.15 of the total length of the clip.

B. NNET2: 2D Convolutional Neural Networks

The neural network architecture denoted as NNET2 is designed for processing 2D spectrograms. The NNET2 neural network architecture is inspired by the architecture proposed in [3] and is made of a residual block which acts as an encoder followed by four fully connected layers to perform the classification. The residual block is made of three 2D-convolutional layers with the same number of output channels for the extraction of feature maps. Each convolutional layer is followed by a ReLU activation function. A residual connection is created by adding the input to the output of the third convolutional layer before the ReLU activation function. Once the output from the residual block is obtained, a max pooling and average pooling are applied to perform subsampling and enhance translation invariance. The output is then passed to the classifier block made of four fully connected layers. After each linear layer a ReLU activation function is applied with the exception of the last one. Finally, the network's output is passed through a SoftMax activation function, which gives the probabilities associated to the multi-class classification. The weights of the convolutional and linear layers are initialized using Xavier (Glorot) uniform initialization, promoting a stable training. The bias terms are initialized

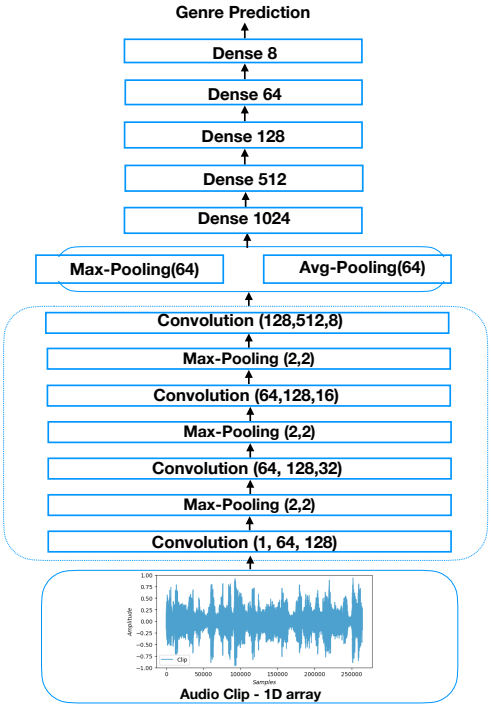


Fig. 3: **NNET1 diagram.** Architecture diagram of the NNET1 - 1D convolutional neural network.

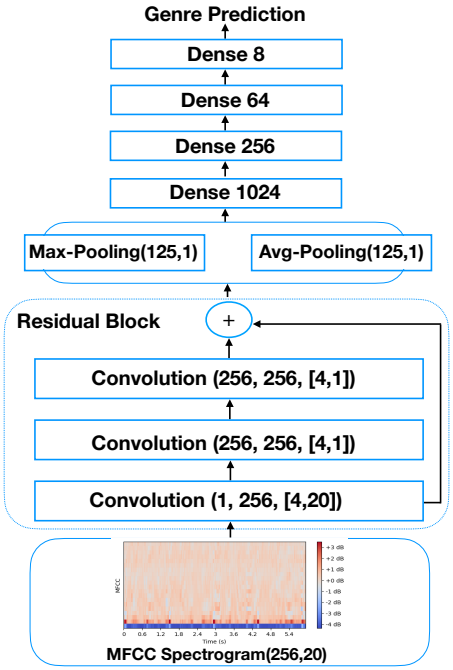


Fig. 4: **NNET2 diagram.** Architecture diagram of the NNET2 convolutional neural network.

to zero. Using the approach described in section III we add batch normalization after each convolutional layer to increase performance . To reduce overfitting, weight decay was also applied, which optimal weight decay coefficient were found to be $\lambda = 0.005$ after a search in the parameter space with Optuna, as it will further discussed in next sections. NNET2 was fed and trained with both Mel and MFCC spectrograms, the latter leading to significantly better performance. Thus, the network was fine-tuned by taking MFCC spectrograms as reference input data. We also applied different transformations to the input spectrograms in order to augment the input samples at our disposal during the training phase. Specifically, time and MFCC coefficients masking were applied to the spectrograms by applying random masks of maximum 30 and 2 bins respectively. Moreover gaussian noise with $\sigma = 0.015$ was applied to MFCC values. All these transformations were applied randomly to the train samples with a 50% probability. These data augmentation procedures led to an increase in the classification performances.

C. MixNet: 1D CNN + 2D CNN

The MixNet neural network architecture is designed to process 2D spectrograms through the combination of the convolutional blocks from both NNET1 and NNET2, described in the previous section. Specifically, the encoder block combines these two convolutional blocks in parallel, as shown in Fig. 5.

The pipeline goes as following. First, the 1D and 2D audio signals are independently processed by their respective residual blocks, formed by the convolutional layers plus a concatenation of a max and average pooling. Finally, we feed to the classifier block a concatenation of the outputs of the 1D and 2D nets. The classifier module is structured as a sequential neural network made of four fully connected layers. ReLU activation function is applied with the exception of the last one. The final output is passed to the SoftMax activation function for the final classification. Weights of the convolutional blocks and linear layers are again initialized using Xavier (Glorot) uniform initialization and the bias terms are initialized to zero. Dropout of neurons with probability $p = 0.1$ is applied at the end of each linear layer as it had been noticed to lead to better performances in the evaluation phases reducing the overfitting. Dropout operation is turned-off during validation and test phases.

Data augmentation is applied to audio signals and MFCC spectrograms input data respectively by considering the same transformations and probabilities described in the previous sections for NNET1 and NNET2.

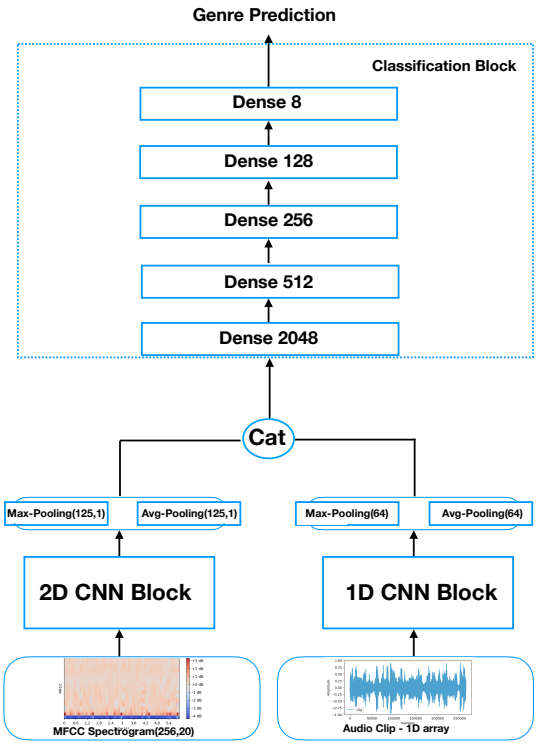


Fig. 5: **MixNet schema.** Schema of the 1D CNN + 2D CNN.

For all of the described models we used the multi-class Cross Entropy loss function performing the training over batches of size 64. Moreover, both SGD and Adam classifiers were tested, the latter leading to better convergence during the training phase. A learning rate of 10^{-4} was found to lead to optimal convergence for all of the models considered. A validation step is performed at the end of each training epoch. Since overfitting issues were faced during models training, early stopping was implemented by setting a condition on the improvement of the validation loss. Weight decay was also applied. This regularization technique adds a penalty to the loss function proportional to the magnitude of the weights, helping to prevent overfitting by discouraging large weights. Search for hyperparameters such as the weight decay coefficient λ was conducted utilizing Optuna. Optuna is an automated hyperparameter optimization framework tailored for machine learning tasks which allows users to dynamically construct search spaces for hyperparameters. We implemented the presented models using PyTorch Lightning. PyTorch Lightning is often applied to CNNs implementation due to its ability to simplify and accelerate the development of deep learning models. It offers a standardized training loop, which streamlines the process of training CNNs. With the use of PyTorch Lightning, a common learning framework was thus implemented for all of the considered models.

VI. Results

In order to estimate and compare the performances of the trained models two metrics were used: Accuracy and F1 score. Also, confusion matrices were used to further assess the behaviour of the models. The accuracy is defined as

$$Acc = \frac{1}{k}\sum_{i=1}^{k}\frac{TP_i + TN_i}{TP_i + TN_i + FP_i + FN_i}$$

where TP is the number of true positives, TN true negatives, FP false positives and FN is the number of false negatives. The variable k indicates the number of classes (8 in our case). On the other hand the F1 metrics is defined as

$$F1 = 2\frac{Precision \cdot Recall}{Precision + Recall}$$

$$Precision = \frac{1}{k}\sum_{i=1}^{k}\frac{TP_i}{TP_i + FP_i},$$

$$Recall = \frac{1}{k}\sum_{i=1}^{k}\frac{TP_i}{TP_i + FN_i}$$

. To fully exploit the information contained in the confusion matrices and to avoid redundant computations, the accuracy was deduced as the sum of the elements of its diagonal divided by the total number of samples. For simplicity, the F1 score was calculated using the `torchmetrics` library. Figures 6, 7 and 8 show the plots of the loss and accuracy for the training and validation sets as a function of the training epochs for the NNET1, NNET2 and MixNet, respectively.

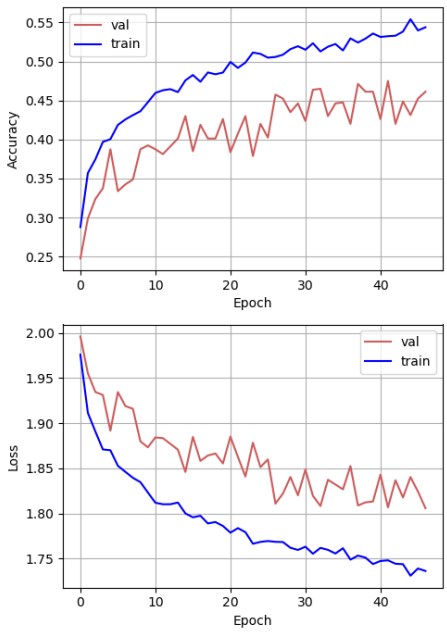


Fig. 6: **NNET1 Results.** Accuracy and Loss for training and validation set. Training is early-stopped before convergence due to validation loss not improving.

Looking at the plots, in all three networks the validation accuracy converges to a lower value compared to the training accuracy, indicating the presence of overfitting. However, its magnitude differs: it is highest in NNET1 while it is reduced

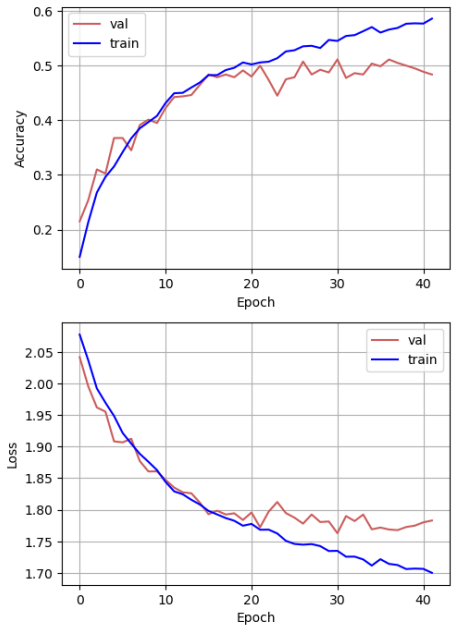


Fig. 7: **NNET2 Results.** Accuracy and Loss for training and validation set. The two curves mostly overlap, showing that NNET2 is able to learn effectively. Training is early-stopped before convergence due to validation loss not improving.

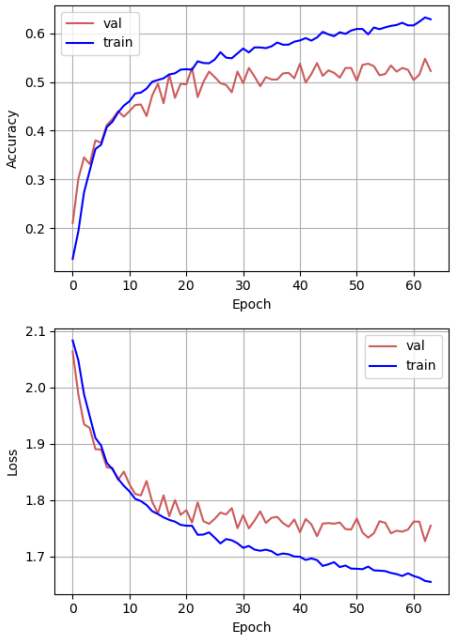


Fig. 8: **MixNet Results.** Accuracy and Loss for training and validation set. Training is early-stopped before convergence due to validation loss not improving.

for NNET2 and MixNet. This shows that learning from raw audio only is possible, even if it is easier if relevant features are extracted in the preprocessing phase, using spectrograms.

In Fig. 9, 10 and 11 the confusion matrices are presented for all cases. The values displayed in the matrices are absolute but in this specific case they can be interpreted as percentage, since each genre is populated by 100 tracks. In NNET1 the genres that display the most correct classification are Folk and Hip Hop while the worst one is the Pop genre. In general, NNET1 does not show any strong preference for classifying specific genres, but instead its error are well distributed among all classes.

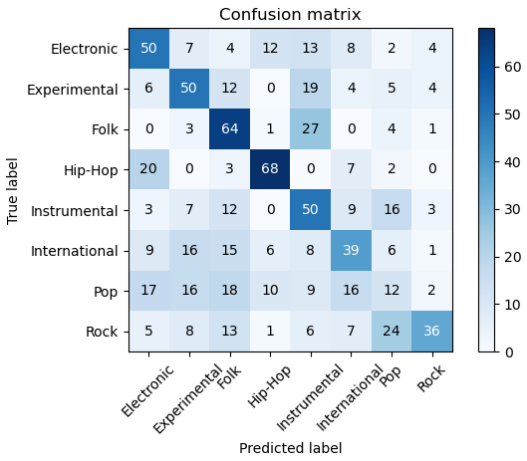


Fig. 9: **NNET1 Results.** Confusion Matrix for the test set.

As concerns NNET2, we can see that overall its performance is better than in NNET1. Again, the best predicted genres are Folk and Hip-Hop, while Pop is completely missed from the net. Another aspect to underline is the performance on the Instrumental genre, which is mostly confused with Experimental. This is different from NNET1, which has an overall lower accuracy, but has fewer extreme errors.

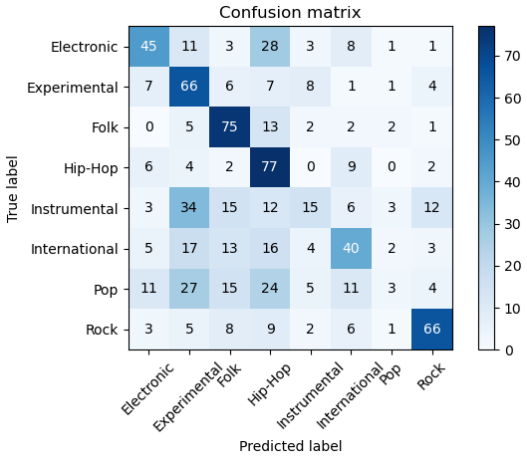


Fig. 10: **NNET2 Results.** Confusion Matrix for the test set.

Finally in Fig. 11 the results for the MixNet are showed. Even in this case, the Folk genre is the best classified one and Pop is the worst one. We can appreciate that MixNet is able to join the best aspects from both NNET1 and NNET2. In fact, it has the best overall performance in terms of accuracy, with the plus of classifying correctly almost all classes. In fact,

as shown in Tab. 1 and in Fig. 12, NNET2 has a slightly lower accuracy than MixNet, but the former misclassifies some genres completely, whilst the latter does not present any particular preference with respect to specific genres.

Furthermore, the fact that Pop is completely missed in all three nets makes us suspect that this is an ill-defined class. Indeed, even humans can find this as a too generic classification for songs that in reality are very different from one another, and that could be labelled more specifically, just like our nets did.

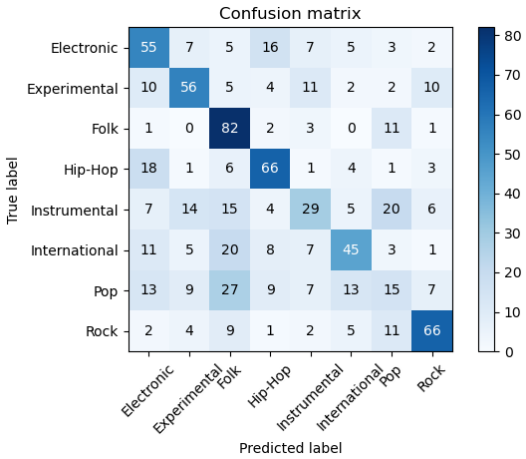


Fig. 11: **MixNet Results.** Confusion Matrix for the test set.

The final performances of the three networks evaluated on the test set are shown in Tab. 1.

Model	*Test Accuracy*	*Test Loss*	*F1 Score*
NNET1	0.47	1.73	0.33
NNET2	0.51	1.70	0.38
MixNet	0.55	1.65	0.35

TABLE 1: Test Accuracy, Test Loss and F1 Score results for the three models.

We can see that MixNet achieves the best scores both in *Test Accuracy* and *Test Loss*, followed by NNET2 and NNET1. This is compatible with what has been said so far, and in fact we see that extracting information from both 1D and 2D audio representations turns out to be beneficial for deep-learning models, while it is more difficult to work with 1D audio only. It is worth mentioning the fact that the F1 score is higher in NNET2 even if it seems in contrast with what we see in the confusion matrices, where it is clear that this net should get the smallest value using a multi-class weighted score as the F1. In 12 a visual comparison between the three models performances based on the validation accuracy as a function of the training epochs is finally shown.

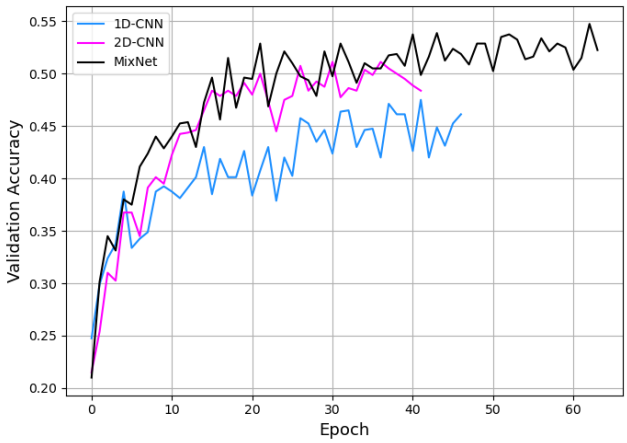


Fig. 12: **Final Results.** Validation accuracy for the three models.

VII. Concluding Remarks

In this study we examined music genre recognition using 1D and 2D Convolutional Neural Networks, also combining the two archtecture in the MixNet model. The 2D CNN, fed by MFCC spectrograms, outperformed the 1D CNN, demonstrating the advantage of spectral features for the extraction of features. MixNet, which combines 1D and 2D features, showed slight improvements, suggesting that both types of inputs add value. Fine-tuning the model parameters and data augmentation proved to be crucial for achieving optimal performance. Future research could explore better methods for combining features and use larger datasets. This could enhance accuracy and reliability, allowing for the development of deeper models without incurring overfitting issues.

Members contributions:

- Alberto Saretto: load of the dataset; signal preprocessing; implementation, fine-tuning and training of NNET2 and MixNet; processing pipeline definition; MGR python package improvement.
- Luca Menti: implementation and training of MixNet; NNET1, NNET2 and MixNet training visualization, final performances computation, comparison and related visualization.
- Diego Bonato: implementation, training and fine-tuning of NNET1; Mix-Net deployment; data-augmentation; Optuna; Lightning Pytorch; metrics implementation; processing pipeline definition; MGR Python package management.

REFERENCES

[1] P. Zhang and X. Zheng., "A deep neural network for modeling music," Apr. 2015.

[2] S. Sigtia and S. Dixon, "Improved music feature learning deep neural networks," (Mile End Road, London E1 4NS, UK).

[3] X. X. W. Zhang, W. Lei and X. Xing, "Improved Music Genre Classification with Convolutional Neural Networks," *INTERSPEECH 2016*, Sept. 2016.

[4] V. Godbole, G. E. Dahl, J. Gilmer, C. J. Shallue, and Z. Nado, "Deep learning tuning playbook," 2023. Version 1.0.

[5] T. Akiba et al., "Optuna: A next-generation hyperparameter optimization framework," 2019.

[6] M. Defferrard, K. Benzi, P. Vandergheynst, and X. Bresson, "FMA: a Dataset for Music Analysis," *arXiv*, Sept. 2017.

[7] B. McFee, C. Raffel, D. Liang, D. Ellis, M. Mcvicar, E. Battenberg, and O. Nieto, "Librosa: Audio and music signal analysis in python," *Python in Science Conference*, Jan.

[8] I. J. et al., "Audiomentations," 2024.