Shallow Convolutional Neural Network Architectures for Music Genre Classification

George Herbert
Department of Computer Science
University of Bristol
Bristol, United Kingdom
cj19328@bristol.ac.uk

***Abstract*—In this paper I investigate the shallow convolutional neural network architecture proposed by Schindler et al. [1] for the task of music genre categorisation. In the first part of this paper, I attempt to replicate the results achieved by Schindler et al., while in the latter part, I extend on their work by**

***Index Terms*—music information retrieval, convolutional neural networks**

I. Introduction

Music genre classification—categorising a music sample into one or more genres—is a fundamental problem in music information retrieval. Early approaches to genre classification, such as that by Tzanetakis and Cook [2], focused on training statistical classifiers using features such as timbral texture, rhythmic content and pitch content. More recently, convolutional neural networks (CNNs) have been widely investigated in the context of genre classification, following their successes in the field of computer vision.

In one such paper, Schindler et al. [1] investigated the performance of two different CNN architectures; in conjunction, they examined how data augmentation applied to four different datasets impacted the performance of both architectures.

II. Related Work

III. Dataset

I used the the GTZAN dataset compiled by Tzanetakis and Cook [2] to train and validate my models; it contains 1000 WAV audio tracks, each 30 seconds in length. There are 100 tracks for each of the 10 genres in the dataset: blues, classical, country, disco, hip-hop, jazz, metal, pop, reggae and rock. The CNNs were trained on chunks of approximately 0.93 seconds that were randomly selected from each audio track.

To produce a training and validation set, a stratified split was deemed suitable to prevent imbalance. Chunks from 75% of the WAV audio tracks for each genre were randomly selected to make up the training set, with chunks from the other 25% of audio tracks for each genre making up the validation set.

IV. CNN Architecture (Schindler et al.)

I recreated the shallow CNN architecture outlined by Schindler et al. Log-mel spectrograms of shape 80×80 are provided as input to the network.

Since the dimensions of the spectrograms correspond to time and frequency, Schindler et al. implemented a parallel architecture. The left pipeline aims to capture frequency relations. It first contains a convolutional layer (with padding) with 16 kernels of shape 10×23 to produce 16 square feature maps of shape 80×80. These are then downsampled using a 1×20 max pooling layer to produce 16 vertical rectangular feature maps of shape 80×4. The kernel and max pooling shapes were specifically selected to capture spectral characteristics. Conversely, the right pipeline aims to capture temporal relations. It too initially contains a convolutional layer (with padding) with 16 kernels, but of approximately square shape 21×20 to produce 16 square feature maps of shape 80×80. These are then downsampled using a 20×1 max pooling layer to produce 16 horizontal rectangular feature maps of shape 4×80, specifically to capture temporal changes in intensity.

The 16 feature maps from each piepline are flattened and concatenated to a shape of 1×10240, which serves as input to a 200 neuron fully connected layer—10% dropout is utilised at this layer to prevent overfitting. These final 200 neurons are then mapped to 10 output neurons, which represent the probabilities of each of the ten genres for a given input.

With the exception of the final layer, each convolutional and fully connected layer is passed through the Leaky ReLU activation function. The final layer uses the softmax activation function.

V. Implementation Details

I used the *PyTorch* [3] machine learning framework to implement and train my models.

A. Optimiser

I optimised my network using the Adam optimisation algorithm [4], as implemented by the `torch.optim.Adam` class, with $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 1 \times 10^{-8}$ and a learning rate of 5×10^{-5}.

B. Loss function

I implemented the cross-entropy loss function using *PyTorch*'s `torch.nn.CrossEntropyLoss` class to measure the error between the output of the network and the label encoded in a one-hot representation.

Regularisation.

C. Weight initialisation

Weight initialisation is an important design choice, since it determines the starting point of the optimisation procedure. Schindler et al. did not specify the procedure they used. However, modern weight initialisation heuristics exist that depend on the activation function. The shallow CNN architecture uses LeakyReLU, so I implemented He initialisation using *PyTorch*'s `torch.nn.init.kaiming_normal` procedure.

D. Batch size

Batch size is a fundamental hyperparameter to consider when training deep networks. Smaller batches give rise to longer epochs and introduce extra noise to the weight updates; however, this noise can prove beneficial if the error manifold has many deep local optima. Conversely, larger batches give rise to shorter epochs, but networks trained with large batches often struggle to generalise. Schindler et al. did not identify the batch size they used for training. Thus, I experimented with multiple batch sizes, and found that my results were most similar with a batch size of 64.

E. BlueCrystal Phase 4

To implement the CNN I created a `ShallowCNN` subclass that inherit's from *PyTorch*'s `torch.nn.Modules` class. The class contains a `forward` method to conduct the forward pass. Additionally, I created a `Trainer` class

I trained my network on BlueCrystal Phase 4.

VI. Replicating Quantitative Results

Table I displays the accuracy my implementation achieved on the validation dataset at both 100 and 200 epochs.

TABLE I
Shallow CNN Accuracy on the Validation Set

Epoch	Accuracy (%)
100	63.71
200	65.09

VII. Training Curves

Overfitting occurs when a network learns the noise in the data as if it represents the structure of the underlying model. To detect overfitting, loss and accuracy are important metrics to monitor during training; Figure 1 and Figure 3 display the accuracy and loss curves for my network, respectively. For the training set, I logged the loss and accuracy of the batch at the end of each step; for the validation set, I logged the loss and accuracy of the entire validation set at the end of each epoch.

After 200 epochs, there is a discrepency of approximately 35% in the accuracy of the model on the training and validation sets—this strongly indicates that the shallow CNN architecture has overfit. Despite using L1 regularisation and dropout, the model has memorised almost every sample in the training set, and therefore struggles to generalise to unseen data.

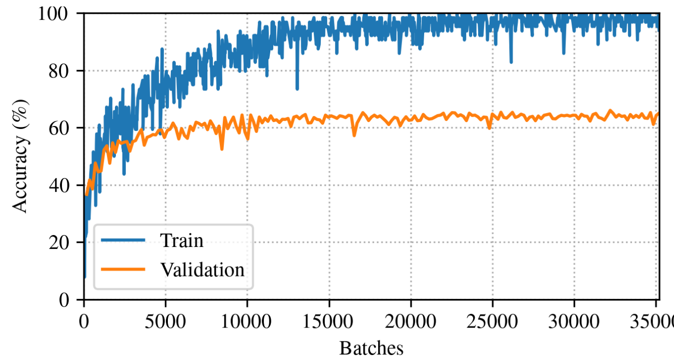

Fig. 1. Accuracy curves

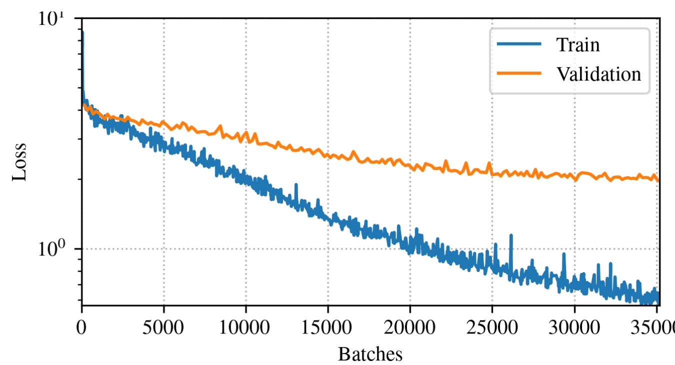

Fig. 2. Loss curves

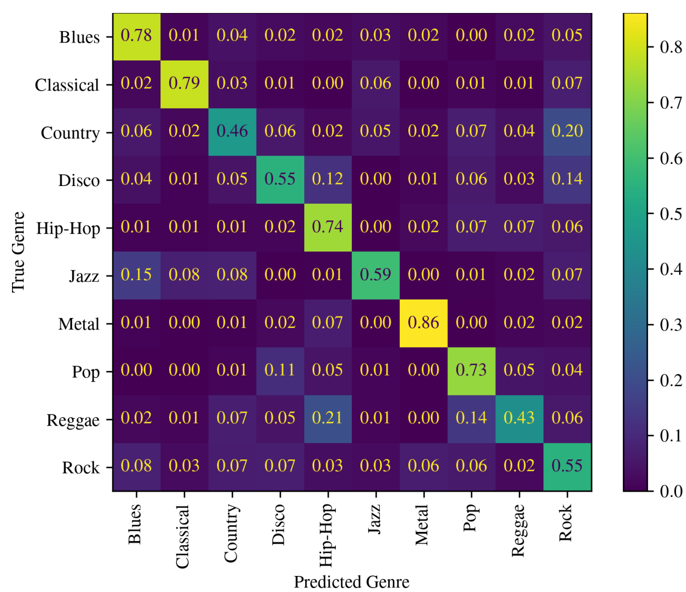

Fig. 3. Confusion matrix

VIII. Qualitative Results

IX. Improvements

From the training curves, it was evident that the network was overfitting.

X. Conclusion and Future Work

References

[1] A. Schindler, T. Lidy, and A. Rauber, "Comparing shallow versus deep neural network architectures for automatic music genre classification," Nov. 2016.

[2] G. Tzanetakis and P. Cook, "Musical genre classification of audio signals," *IEEE Transactions on Speech and Audio Processing*, vol. 10, no. 5, pp. 293–302, 2002.

[3] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library."

[4] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2014. [Online]. Available: https://arxiv.org/abs/1412.6980