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 implement and evaluate 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 describe the process I undertook to implement the network. While in the latter part, I evaluate the results my implementation achieved, and extend on their work by implementing batch normalisation.**

***Index Terms*—Music Information Retrieval, Music Genre Classification, 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: a shallow architecture and a deep architecture.

II. Related Work

Schindler et al. published their work on genre classification in 2016; more recent work has since been published in this area.

Large datasets are frequently required to train powerful deep neural networks. However, in the MIR domain, there is often a lack of large training datasets. Hung et al. [3] published some very recent work that successfully dealt with this problem. They introduced a novel method called input-dependent neural model reprogramming—a transfer learning training scheme—that leverages pre-trained models for music classification. They successfully applied this method to reprogram two published models that were pre-trained on speech and audio data.

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 0.93 second chunks that were randomly selected from each audio track.

To produce a training and test 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 test set.

IV. CNN Architecture (Schindler et al.)

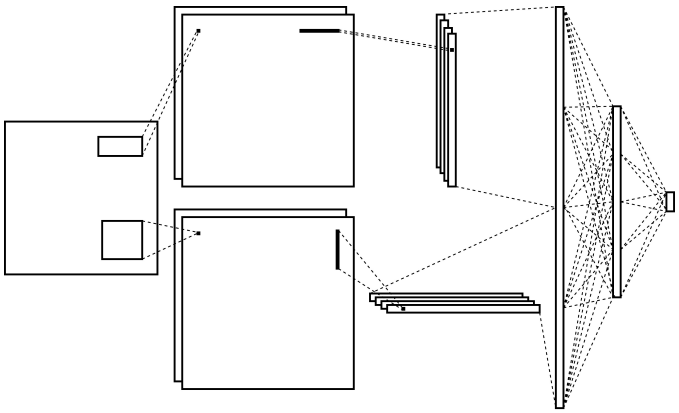


Fig. 1: Abstract representation of the shallow CNN architecture as described by Schindler et al.

I recreated the shallow CNN architecture outlined by Schindler et al.; Figure 1 displays an abstract representation of the architecture. 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. proposed a parallel architecture.

The top 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 bottom 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 too 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 is mapped to a 200 neuron fully connected layer. These final 200 neurons are then mapped to 10 output neurons—10% dropout is utilised at this layer to prevent overfitting. The softmax function is applied to these 10 output neurons to produce a pseudo-probability distribution to represent the probability that a given input belongs to each of the ten genres.

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

V. Implementation Details

I constructed and trained the shallow CNN using Python and the *PyTorch* [4] machine learning framework.

A. CNN Details

To implement the CNN I created a `ShallowCNN` class that inherits from $PyTorch$'s `nn.Modules` class. The class contains a `forward` method that conducts a forward path for an inputted batch.

B. Training Details

I implemented a `Trainer` class to train an instance of the `ShallowCNN` class. Supervised learning algorithms require a cost function to evaluate how well the model fits the training data; to maintain consistency with Schindler et al. I implemented cross-entropy loss. To prevent overfitting I used L1 regularisation, with a penalty value of 0.0001. I used the Adam optimiser [5]—an extension of stochastic gradient descent—to optimise the cross-entropy loss. I implemented Adam using $PyTorch$'s `optim.Adam` class. I used a learning rate of 5×10^{-5} with $\beta_1 = 0.9$, $\beta_2 = 0.999$, and $\epsilon = 1 \times 10^{-8}$. I trained the network on a BlueCrystal Phase 4 GPU node, which contains two NVIDIA Tesla P100 GPUs [6].

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. In this respect, I implemented He initialisation using *PyTorch*'s `nn.init.kaiming_normal` procedure due to the Leaky ReLU activation function being used throughout the network.

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.

VI. Replicating Quantitative Results

Table I displays the mean accuracy my implementation achieved on the test dataset over five runs. I have computed the accuracy at both 100 and 200 epochs. Much like Schindler et al., I also trained my network for 500 epochs, but there was negligable difference in accuracy. This was because the network had already overfit—achieving a test accuracy of near 100%—so training the network for additional epochs did not improve the network's ability to generalise. In this respect, if the network were trained for longer, it is possible the test accuracy would decrease.

TABLE I: Shallow CNN Accuracy on the Test Set

Model	Epoch	Accuracy
My CNN	100	62.96
	200	64.01
Schindler et al.	100	66.56
	200	67.49

Notably, the accuracy achieved is approximately 3% lower than that of Schindler et al. Though I cannot determine the exact reason for this discrepency, I can speculate on several potential ones. Firstly, Schindler et al. may not have specified the full design details of the network necessary for reproducibility. Alternatively, it may due to the fact that the $Keras$ library they used has different default parameters than the $PyTorch$ library I produced my implementation in.

The confusion matrix in Figure 2 displays the performance of my implementation after 200 epochs. The model achieved a high accuracy on specific genres, but a low accuracy on others, as evidenced by the confusion matrix in Figure 2. There is a large discrepency in the per-class accuracy of different genres. The network achieved a per class accuracy of over 75% on the blues, classical and metal genres. While conversely, it achieved less than a 40% per-class accuracy on the rock and reggae genres—notably it classified 21% of rock songs as blues.

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 3 and Figure 4 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 test set, I logged the loss and accuracy of the entire test set at the end of each epoch.

After 200 epochs, there was a discrepency of approximately 35% in the accuracy of the model on the training and test sets—this strongly indicates that the shallow CNN architecture overfit. Despite using L1 regularisation and dropout, the model

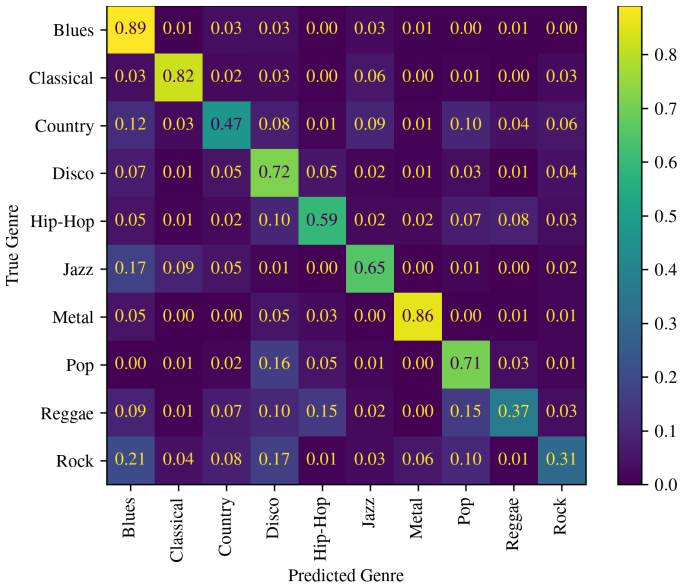


Fig. 2: Confusion matrix displaying the performance of my network on the test set after 200 epochs for a single training run. The value in a given cell represents the proportion of samples from the true genre categorised as the predicted genre.

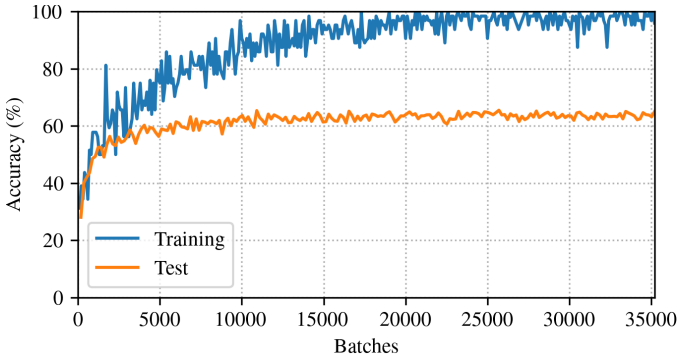


Fig. 3: Plot of accuracy data from the same training run as Figure 2. The line labelled 'training' is the accuracy achieved on the training set, calculated every 100 batches; the line labelled 'test' is the accuracy achieved on the test set, calculated every epoch.

had memorised almost every sample in the training set, and therefore struggled to generalise to unseen data.

VIII. Qualitative Results

Deep neural networks are frequently described as black-box models, because it can be exceptionally difficult to reason about the features they extract to produce their outputs. Figure 5 displays three log-mel spectrograms produced from samples from the audio tracks in the GTZAN dataset. An analysis of these spectrograms, as well as listening to the audio files from which they were derived, yields some clues as to the discrepencies in per-class accuracy that the network achieved.

The spectrogram in Figure 5a was classified correctly, and as were the other 14 log-mel spectrograms extracted from 'classical.00070.wav'. The audio within the file is very characteristic of classical music, with long notes played from

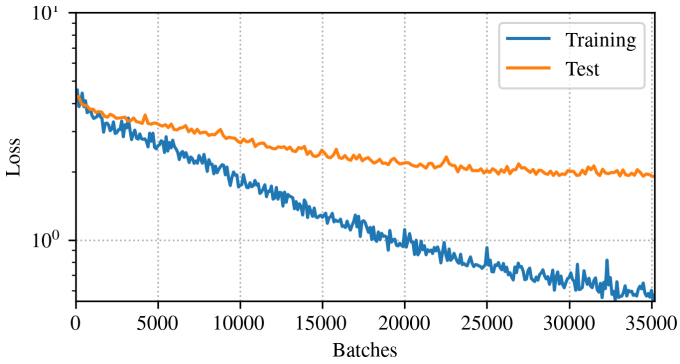
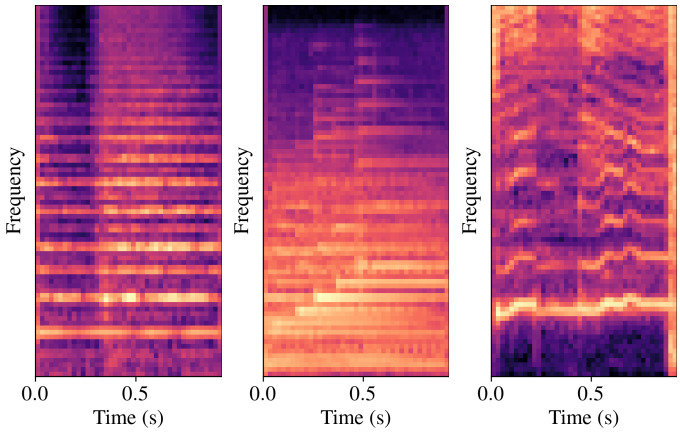


Fig. 4: Plot of loss data from the same training run as Figure 2. The line labelled 'training' is the loss achieved on the training set, calculated every 100 batches; the line labelled 'test' is the loss achieved on the test set, calculated every epoch.



(a) classical.00070 (b) jazz.00003 (c) Correct

Fig. 5: Log-mel spectrograms produced from three of the samples in the test dataset.

a violin. Harmonics produced by the violin are represented by the long parallel lines that stretch across the entirety of the file. An analysis of other spectrograms produced from classical music appear very similar. One can see how a neural network could relatively easily learn that extended parallel lines such as these indicate classical music.

Conversely, the network incorrectly predicetd the class for all spectrograms produced from 'jazz.00003.wav'. A spectrogram produced from a segment of the audio file is shown in Figure 5b, in which there are long parallel lines characteristic of classical music. In fact, nine of the spectrograms derived from the file were classified as being classical. This lends credence to the idea that the network classifies spectrograms with long parallel lines as being classical music.

The spectrogram in Figure 'pop.00064.wav' was also incorrectly classifed, as were 13 of the other spectrograms derived from the same file. In fact, 11 of the spectrograms were incorrectly classified as being disco music. This arguably highlights a fundamental problem with using genre as a way of classifying music: its subjective nature. Genres are often not defined objectively. It is highly unlikely even a human would

agree with all of the genre classiciations in the dataset, which makes it understandable that a machine too would incorrectly classify so many samples.

IX. Improvements

A. Maximum Probability and Majority Voting

Thus far, I have computed the 'raw' accuracy of my model, by individually predicting the genre for each sample in my dataset. However, this is not the optimal method, since it ignores all file dependencies; each WAV audio track within the dataset is represented by fifteen 0.93 second samples. Schindler et al. recognised this, and achieved a substantial improvement in accuracy by introducing two new methods of classification: maximum probability and majority vote—both of which take file dependencies into account. To classify an inputted audio file by maximum probability, the probabilities output by the final softmax layer for each segment are first summed; then, the predicted class is determined by the largest value amongst the summed probabilities. To classify an inputted audio file by majority vote, a class is determined for each segment by the largest value output by the final layer; then, a majority vote is conducted over the predicted classes for each of the segments.

Table II displays the accuracy achieved using each of the raw, maximum probability and majority vote approaches. The relative improvement to accuracy provided by maximum probability classification and majority vote classification are consistent with the findings of Schindler et al.

TABLE II: Improved Accuracy on the Test Set

		Accuracy		
Model	**Epoch**	**Raw**	**Max**	**Maj**
My CNN	100	62.96	75.80	73.53
	200	64.01	77.60	76.60
My CNN + Batch Norm	100	66.30	77.30	77.56
	200	**66.55**	**78.68**	**77.64**

B. Batch Normalisation

Batch normalisation [7] is a technique frequently employed to increase the speed and stability of the training process. This is achieved via a normalisation step that fixes the means and variances of each layer's input. When applied to the output from a convolutional layer, this is implemented as follows:

$$\hat{x}_{i,c,x,y} = \gamma_c \frac{x_{i,c,x,y} - \mu_c}{\sqrt{\sigma_c^2 + \epsilon}} + \beta_c$$

where x is the output from the previous layer; $\hat{x}$ is the input to the subsequent layer; i, c, x and y are the batch, channel, width and height indices; γ_c and β_c are parameters learned during the optimisation process to restore the representative power of the network; and γ is a small constant for numerical stability.

I implemented a two-dimensional batch normalisation layer following each of the two convolutional layers in my network using *PyTorch*'s `nn.BatchNorm2d` class. Table II displays the accuracy of my network with the inclusion of the batch normalisation layers. Not only did batch normalisation provide a reasonabe improvement in accuracy after 200 epochs—especially in the case of raw accuracy—but, it also reduced the number of epochs required for divergence to a small degree. Unlike in the case of the CNN trained without batch normalisation, the difference in accuracy between 100 and 200 epochs is very minimal. The precise reasons for the improved performance are unclear: Batch normalisation has strong empirical performance, but there is some disagreement as to the theory behind its effectiveness. Santukar et al. [8] proposed that batch normalisation significantly smoothes the optimisation landscape, which induces a more predictive and stable behaviour of the gradients, thus facilitating faster training.

Figure 6 displays the raw accuracy the network achieved on the training and test set during the training process. It is evident that the degree to which the network is overfitting has been reduced. This is because batch normalisation also works as a regulariser, due to the fact that the mean and standard deviation is computed from each mini-batch, rather than the entire dataset.

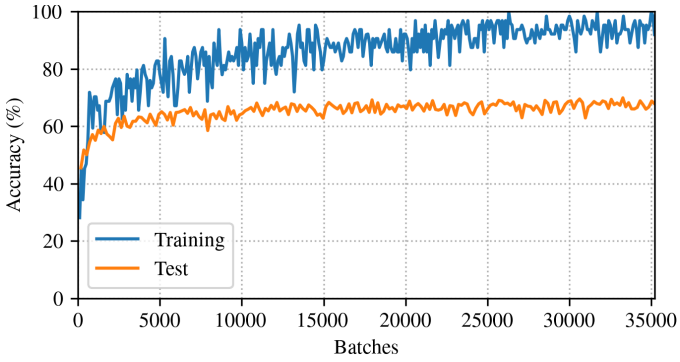


Fig. 6: Plot of accuracy data for my improved network from a single training run. Format is the same as for Figure 3.

X. Conclusion and Future Work

In this paper I have successfully reimplemented the shallow CNN architecture as described by Schindler et al. I also proposed an extension to the architecture, with the inclusion of two batch normalisation layers.

Problems with the dataset. Is the dataset representative?

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] Y.-N. Hung, C.-H. H. Yang, P.-Y. Chen, and A. Lerch, "Low-resource music genre classification with advanced neural model reprogramming," *arXiv preprint arXiv:2211.01317*, 2022.

[4] 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."

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

[6] Bluecrystal phase 4. [Online]. Available: https://www.acrc.bris.ac.uk/acrc/phase4.htm/

[7] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *CoRR*, vol. abs/1502.03167, 2015. [Online]. Available: http://arxiv.org/abs/1502.03167

[8] S. Santurkar, D. Tsipras, A. Ilyas, and A. Madry, "How does batch normalization help optimization?" 2018. [Online]. Available: https://arxiv.org/abs/1805.11604