# 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

The explosive growth of digital music platforms has sparked significant advancements in the field of music information retrieval (MIR), a rapidly evolving discipline that focuses on developing computational techniques to extract valuable insights from music and audio signals. As MIR technologies continue to advance, they are becoming increasingly crucial for the music industry, enabling the creation of more effective tools such as recommender systems, which can provide a competitive edge in a crowded market.

One of the key challenges in MIR is genre classification, which involves identifying the musical genre of a given audio signal. Accurate genre classification can help music providers organise and categorise their catalogs, and enable users to search and discover new music in a more efficient and effective way. Early efforts to solve this problem, such as that proposed by Tzanetakis and Cook [2], employed statistical classifiers that were trained on vector summaries of features such as timbral texture, rhythmic content and pitch content. However, these summaries fail to capture the temporal structure of the underlying audio. In recent years, researchers have turned to audio spectrograms, which represent frequency data over time, to train state-of-the-art deep-learning models that can effectively classify audio signals based on genre.

Convolutional neural networks (CNNs) are one type of network that have been widely employed, following their successes in the field of computer vision. In this paper, we explore the use of CNNs for genre classification, and specifically investigate the shallow CNN architecture proposed by Schindler et al. [1], which was shown to achieve modest performance on this task.

## II. Related Work

Liu et al. [3] recently proposed a novel architecture named a Bottom-up Broadcast Neural Network (BBNN) to deal with some of the problems traditionally associated with genre classification. They identified that many previously developed architectures had focused on abstracting high-level semantic features layer-by-layer; as a result, these architectures suffer from a huge loss of lower-level features which are critical to the task of genre classification. Thus, the BBNN architecture was specifically designed to simultaneously abstract high-level information while preserving the lower-level features.

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. [4] 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 GTZAN dataset, compiled by Tzanetakis and Cook [2], to train and evaluate my models. The dataset contains a total of 1000 WAV audio tracks, each 30 seconds in length. GTZAN is a balanced dataset, containing 100 tracks for each of the 10 genres in the dataset: blues, classical, country, disco, hip-hop, jazz, metal, pop, reggae and rock.

I utilised a stratified-by-genre split to produce a training and test set: the training set contained 75% of the audio tracks, 75 tracks from each genre; while the test set contained the other 25% of audio tracks.

## IV. CNN Architecture (Schindler et al.)

I recreated the shallow CNN architecture outlined by Schindler et al. [1]; Figure 1 displays an abstract representation of the architecture. Log-mel spectrograms of shape $80 \times 80$ are provided as input to the network. Since the dimensions of the spectrograms correspond to time and frequency, the network utilises a parallel architecture.

The top pipeline aims to capture frequency relations. It first contains a convolutional layer (with padding) with 16 kernels of shape $10 \times 23$ to produce 16 square feature maps of shape $80 \times 80$. These are then downsampled using a $1 \times 20$ max pooling layer to produce 16 vertical rectangular feature maps of shape $80 \times 4$. The kernel and max pooling shapes

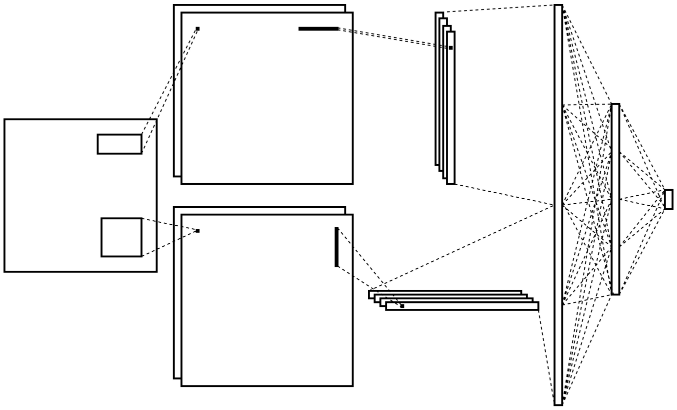


Fig. 1: Abstract representation of the shallow CNN architecture as described by Schindler et al.

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 $21 \times 20$ to produce 16 square feature maps of shape $80 \times 80$. These too are then downsampled using a $20 \times 1$ max pooling layer to produce 16 horizontal rectangular feature maps of shape $4 \times 80$, specifically to capture temporal changes in intensity.

The 16 feature maps from each piepline are flattened and concatenated to a shape of $1 \times 10240$, 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 mitigate 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 10 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

### A. Preprocessing

The network was trained and evaluated using a total of 15000 log-mel spectrograms. Each audio track in the GTZAN dataset was split into chunks of approximately 0.93 seconds, using a step size of 50%. Log-mel spectrograms of shape $80 \times 80$ were produced from 15 randomly selected chunks from each track using a fast Fourier transform with a window size of 1024, and a step size of 50%. It was imperative to split the tracks into the training and test sets prior to producing the spectrograms to avoid data leakage.

### B. Training Details

I constructed and trained the CNN using Python and the PyTorch [5] machine learning framework. I reimplemented the training process as described by Schindler et al. [1]. To this respect, I used cross-entropy loss to evaluate the performance of the network, with L1 regularisation to mitigate overfitting using a penalty value of 0.0001. I used the Adam optimiser [6]—an extension of stochastic gradient descent—to optimise the network, using a learning rate of $5 \times 10^{-5}$, $\beta_1 = 0.9$, $\beta_2 = 0.999$, and $\epsilon = 1 \times 10^{-8}$ for numerical stability. I trained the network on a BlueCrystal Phase 4 GPU node, which contains two NVIDIA Tesla P100 GPUs [7].

### 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 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 set over five runs; the accuracy achieved by Schindler et al. [1] is also displayed for comparison. I trained my network for a total of 500 epochs, but there was negligable difference in accuracy after 200 epochs 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 likely the test accuracy would decrease.

TABLE I
Accuracy achieved 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. This discrepency can potentially be attributed to Schindler et al. not specifying the full design details necessary for reproducibility. Importantly, Schindler et al. produced their implementation in Keras, which often has different default parameters than Pytorch.

Figure 2 is a confusion matrix displaying the the performance of my implementation after 200 epochs. Notably, there is a large inconsistency in the per-class accuracy of different genres. For example, the network achieved a high per class accuracy of over 80% on the blues, classical and metal genres. While conversely, it achieved less than a 40% per-class accuracy on the reggae and rock genres; in particular, it misclassified 21% of rock songs as blues.

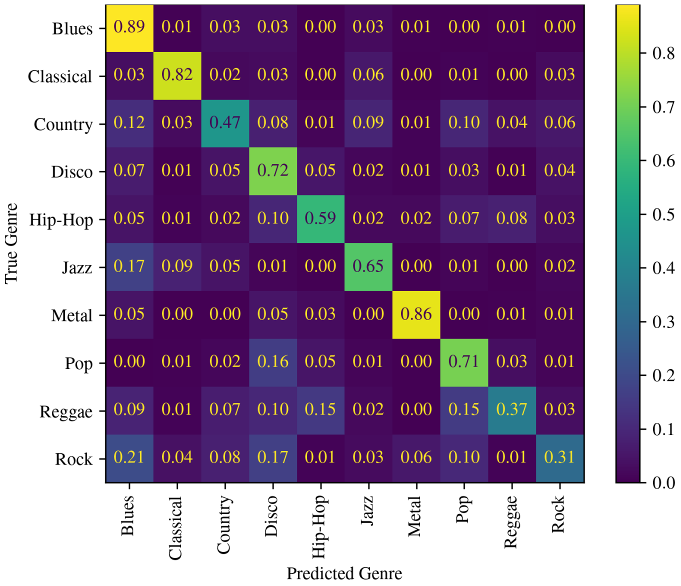


Fig. 2: Confusion matrix displaying the performance of my network on the test set after 200 epochs for a single training run. The value in a given cell represents the proportion of samples from the true genre categorised as the predicted genre.

## 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, I monitored the loss and accuracy my network achieved on both the training and test set throughout the training process. Figure 3 and Figure 4 display the accuracy and loss curves for my network, respectively.

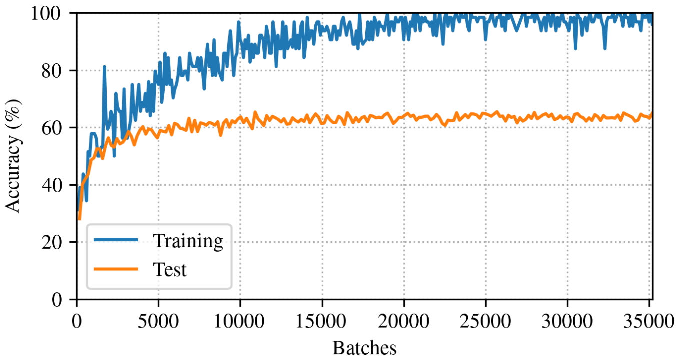


Fig. 3: Plot of accuracy data from the same training run as Figure 2. The line labelled 'training' is the accuracy achieved on the training set, calculated every 100 batches; the line labelled 'test' is the accuracy achieved on the test set, calculated every epoch.

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

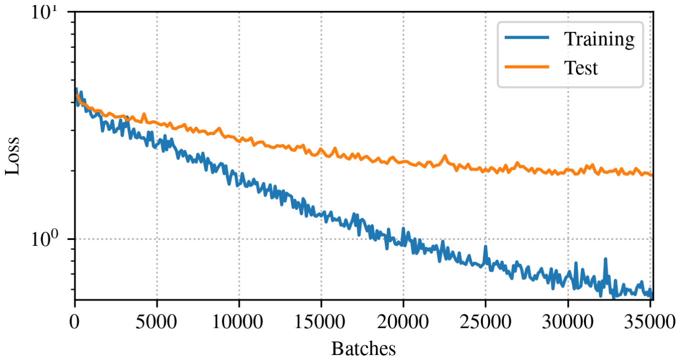
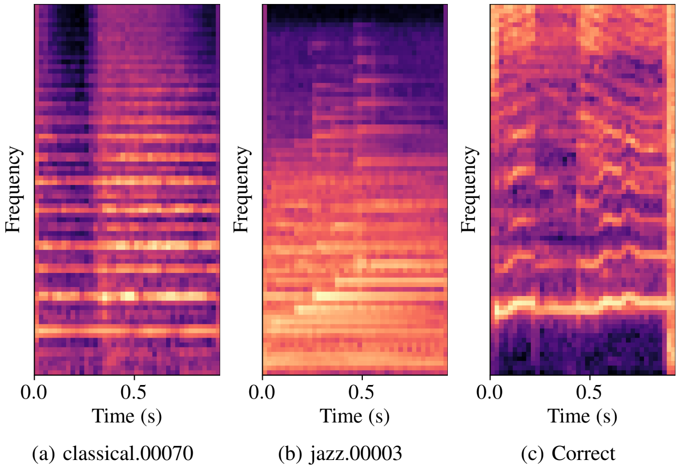


Fig. 4: Plot of loss data from the same training run as Figure 2. The line labelled 'training' is the loss achieved on the training set, calculated every 100 batches; the line labelled 'test' is the loss achieved on the test set, calculated every epoch.

## 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.



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

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

The spectrogram in Figure 5a was correctly classified as belonging to the classical genre, 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 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 predicting the genre for each sample in my dataset independently. However, this is not the optimal method since it ignores all file dependencies—there are 15 spectrograms for each WAV audio track in the GTZAN dataset. 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 in accuracy provided by the two new approaches are consistent with the findings of Schindler et al. The increase in accuracy is due to to there being some level of independence in the predictions of samples from a given file. This independence can be attributed to the 0.93 second samples often being too short to be representative, as well as the model overfitting the training data. The majority vote and maximum probability classification methods essentially cancel out many of the individual errors to produce more accurate predictions for entire files.

TABLE II
Improved accuracy achieved 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 [8] 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; $\gamma_c$ and $\beta_c$ are parameters learned during the optimisation process to restore the representative power of the network; and $\gamma$ 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 convergence 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. [9] 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.

## X. Conclusion and Future Work

In this paper I have reimplemented the shallow CNN architecture as originally described by Schindler et al. [1]. While I was unable to reproduce the results published in the original paper, I have exposed a significant discrepency in

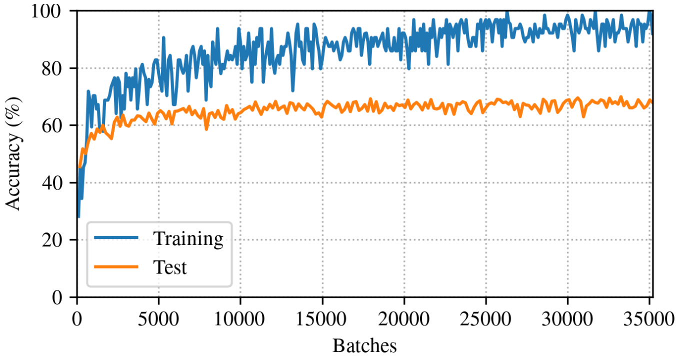


Fig. 6: Plot of accuracy data for my improved network from a single training run. Format is the same as for Figure 3.

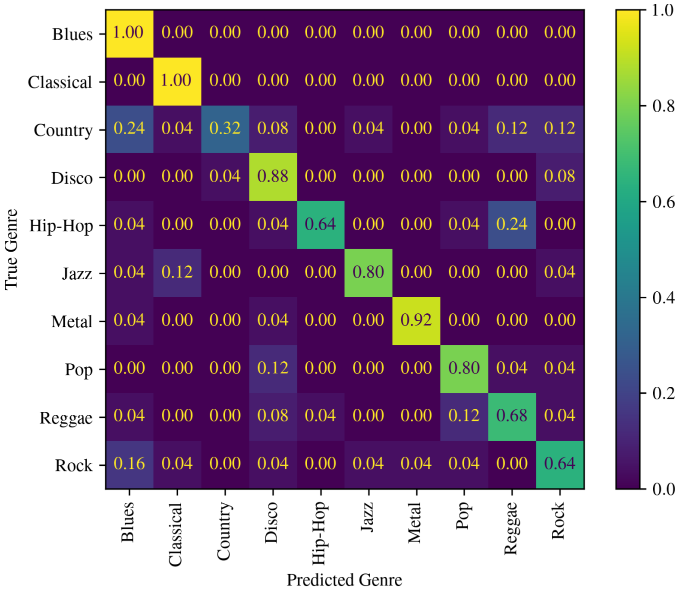


Fig. 7: Confusion matrix displaying the performance of my improved network with batch normalisation on the test set after 200 epochs for a single training run. Classification was performed using the maximum probability method. Format is the same as for Figure 2.

the per-class accuracy when the the network is trained using the GTZAN dataset. Moreso, I have conducted an extensive qualitative analysis to gain an insight into potential reasons for this discrepency.

In the latter part of this paper, I have also proposed an extension to the architecture with the inclusion of two batch normalisation layers. I found that the inclusion of these layers offered an advantage over the original architecture by reducing the time required for convergence, as well as providing a modest improvement to accuracy.

Future work should focus on eliminating the stark discrepency in per-class accuracy. Further extensions should also be considered with the aim of reducing the degree to which the network overfits. Data augmentation provides a potential avenue to explore in this respect.

## 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] C. Liu, L. Feng, G. Liu, H. Wang, and S. Liu, “Bottom-up broadcast neural network for music genre classification,” 2019. [Online]. Available: https://arxiv.org/abs/1901.08928

[4] 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.

[5] 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.”

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

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

[8] 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

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