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: a shallow architecture and a deep architecture.

II. Related Work

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

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 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.

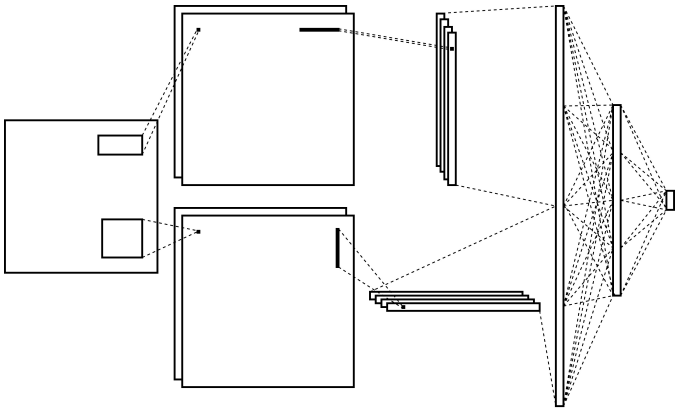

Fig. 1. Abstract representation of the shallow CNN architecture

IV. CNN Architecture (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. implemented 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 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 constructed my implementation using Python and the *PyTorch* [3] machine learning framework. I trained my network on a BlueCrystal Phase 4 GPU node, which contains two NVIDIA Tesla P100 GPUs [4].

A. Optimiser

I optimised my network using the Adam optimisation algorithm [5], 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_uniform` 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 128.

E. Classes

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

VI. Replicating Quantitative Results

Table I displays the accuracy my implementation achieved on the test dataset at both 100 and 200 epochs. My implementation achieved an accuracy approximately 3% lower than that of Schindler et al at both 100 and 200 epochs. Though I cannot determine the exact reason for this discrepency, I can speculate on several potential reasons. 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 used by Schindler et al. often has different default parameters than the $PyTorch$ library I produced my implementation in.

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

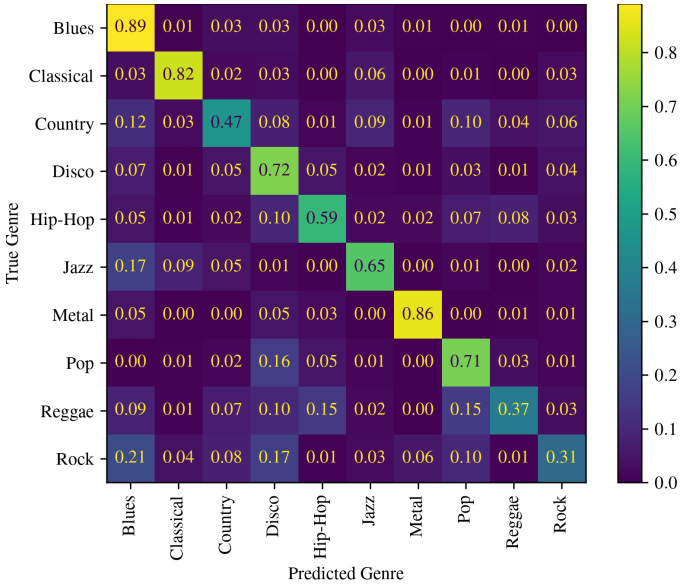

Fig. 2. Confusion matrix

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

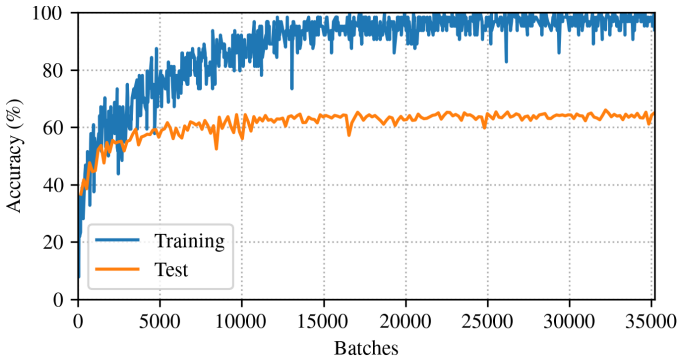

Fig. 3. Accuracy curves

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.

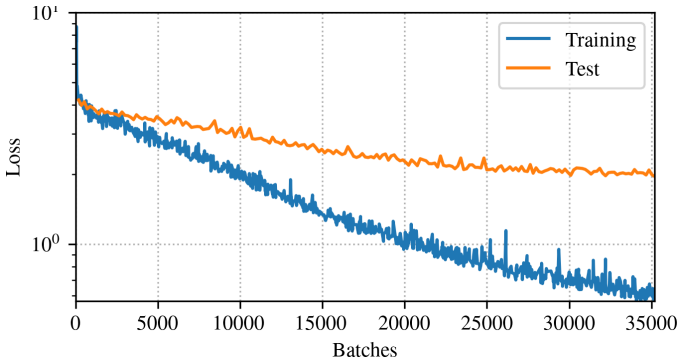

Fig. 4. Loss curves

VIII. Qualitative Results

The model achieved a high accuracy on specific genres, but a low accuracy on others, as evidenced by the confusion matrix in Figure 2. Notably, it accurately classified samples from the classical and metal genres over 85% of the time. While conversely, it only managed to accurately classify approximately 40% of samples from the rock and reggae genres.

An analysis of the spectrograms from these genres reveals potential reasons for these discrepencies in per-class accuracy.

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

Model	Epoch	Accuracy		
		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	76.30	76.56
	200	**66.55**	**77.68**	**76.64**

B. Batch Normalisation

Batch normalisation [6] 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 `torch.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. Unlike in the case of the CNN trained without batch normalisation, the difference in accuracy between 100 and 200 epochs is 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. [7] proposed that batch normalisation significantly smoothes the optimisation landscape, which induces a more predictive and stable behaviour of the gradients, thus facilitating faster training.

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

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

[6] 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

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