Digging Deeper into Automatic Music Tagging

We agree that all members have contributed to this project (both code and report) in an approximately equal manner.

1st Richa Banthia
University of Bristol
ng20035@bristol.ac.uk
RICHA BANTHIA

2nd Austin Long
University of Bristol
mf19511@bristol.ac.uk
AUSTIN LONG

3rd Sakshi Saraff
University of Bristol
zn20342@bristol.ac.uk
SAKSHI SARAFF

I. Introduction

MIR (Music Information Retrieval) has become an important area of research in computer science with the increase in the size and relevance of music databases. This has led to more sophisticated developments in musical databases. Several audio and video streaming companies like Netflix, Spotify and Amazon Prime have invested huge amounts of time and money in MIR to extract meaningful information. Audio analysis is useful for enhancing user experience, for example - song lyrics transcription, serving recommendations and music auto-tagging. Deep architectures and feature learning are prominently used nowadays to avoid traditional approaches in MIR which rely heavily on extensive prior knowledge and manual effort for extraction of features. The research paper "End-to-End Learning for Music Audio", by Sander Dieleman and Benjamin Schrauwen, delves into the challenges of content-based Music Information Retrieval (MIR) using End-to-End Learning [1]. End-to-End learning refers to the training of a network for complex tasks like audio tagging using raw audio directly without any preprocessing. Their primary goal was to explore the efficacy of learning from raw audio as an alternative to learning from spectrograms. We replicated the architecture and results mentioned in the paper and improved upon them with group normalisation and dropout in section VIII-A. We expanded upon our exploration by implementing Sample CNN convolution network in section VIII-B.

II. Related Work

Lee et al. proposed SampleCNN, a CNN-based deep-learning model for automatic music tagging [5]. SampleCNN learns representations from much smaller grains of raw input data as compared to previous developments. This is achieved by using a kernel size and stride length of between 2 and 5 in the initial convolution layer. To ensure that the model is sufficiently expressive, additional hidden layers are introduced so that features can be learned hierarchically. These hidden layers are made up of components, which are comprised of a single convolution layer and a single max pooling layer. It was found that SampleCNN could produce results that were comparable to those of spectrogram-based approaches, with both methods producing an AUC of 0.90 when evaluated against the MagnaTagATune dataset. We note this to be one of the earliest examples of end-to-end learning from raw audio producing comparable results to spectrogram-based approaches.

Won et al. introduced a deep learning model which employs a harmonic convolution layer to capture harmonic characteristics [6]. The architecture, named HarmonicCNN, reintroduced domain-specific knowledge into state-of-the-art audio auto-tagging approaches. A harmonic is an integer multiple of a signal's fundamental frequency. Harmonic characteristics can be captured by a band-pass filter, which allows only frequencies that are within a certain range of a given center frequency to pass. The harmonic convolution layer learns a collection of band-pass filters, one for each harmonic. The center frequencies of these filters are initialised with quarter-tone intervals, and the parameters which determine the widths of the bands are initialized based on equivalent rectangular bandwidth (ERB) values. A mel-spectrogram representation of audio is used as an input to this harmonic convolution layer, and its output is processed by a 2D CNN. HarmonicCNN gave an AUC score of 0.91 when evaluated against the MagnaTagATune dataset, outperforming SampleCNN. We observe that these two architectures are often used as benchmarks in modern work, such as the recent paper by Lee et al. [4].

Huang et al. proposed MuLan, an acoustic model that links music audio directly to natural language music descriptions [3]. MuLan's architecture consists of two separate embedding networks for audio and text. These networks are trained using a contrastive multi-view coding loss function, which penalises dissimilarity between the embeddings generated by each network for a particular data point. The audio embedding network uses Resnet-50 or Audio Spectrogram Transformer (AST) architectures, and the text embedding network uses Bidirectional Encoder Transformer (BERT) with base-uncased architecture. MuLan is trained on a large dataset of 44 million music recordings that were gathered by extracting 30-second clips from internet music videos. These recordings are associated with text gathered from comments, video descriptions, and other metadata. We see this paper as an example of modern deep learning advancements being applied to music auto-tagging. A noisy but extremely large corpus of data is scraped from the web, transformer-based architectures are leveraged for the embedding networks, and a joint embedding between natural language and some other modality is created. As a result of these advancements, MuLan gives an AUC of 0.92

when evaluated against the MagnaTagATune dataset.

III. Dataset

We use the MagnaTagATune dataset to train our model to perform automatic music tagging. MagnaTagATune is used for MIR-related tasks such as artist and instrument identification, genre and mood classification, auto-tagging and playlist Generation. The dataset contains $25,863$ 29s long music clips annotated with a combination of 188 tags with a sample rate of 12kHz. We take $21,355$ clips for our model, of which we use 72.23% for training, 7.20% for validation, and 20.56% for testing. The relative distributions of the train, validation, and test data are shown in figure 1 with the 50 most frequent tags being used as labels. The length of a given bar is indicative of the aggregated sample count across the three datasets. As MagnaTagATune is a multi-label dataset, the total count for each tag is incremented by 1 for each sample labelled with it.

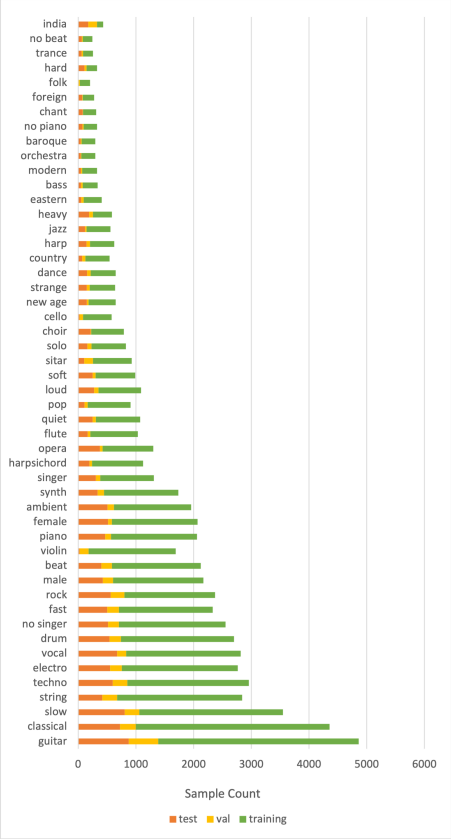

Fig. 1: The sample counts of the top 50 tags of the MagnaTagATune for train, test and validate data

IV. CNN Architecture

The CNN architecture is shown in figure 2. The basic implementation of the model excludes group normalisation (dark green) and dropout (pink) layers, as these layers are the basic extensions added to our model to improve its performance. The basic implementation consists of three convolution layers, one of which is a strided convolution, all with 32 output channels. The length of the kernels for the two convolution layers is 8, whilst the max-pooling layers have a kernel size of 4 as shown in figure 2. The ReLU activation function is applied after every convolution layer and after the first fully connected layer. The activation function after the second fully connected layer is sigmoidal, allowing the model to output a probability score for each class. In figure 2, N and C stand for neurons and channels and are the output of the specified layer. The raw audio clip shown is a 2.9s clip, which is obtained by dividing each 29.125s clip into 10 subclips. This combined with a minibatch of 10 samples, creates a batch of 100 subclips. After the final activation layer, the audio is again reshaped, and the subclips' predictions are averaged to compute tag predictions.

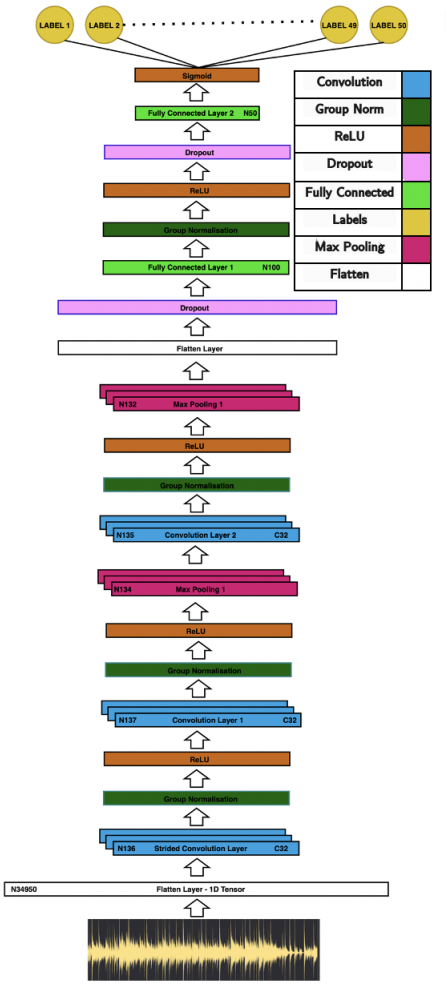

Fig. 2: The CNN Network

V. Implementation Details

The original paper lacks clarity on the size of batches and the method of input normalisation. We use our best judgment to resolve these ambiguities. Furthermore, Dieleman et al. do not specify the number of filters in the strided convolution, the momentum used with SGD, or the learning rate. We tune these hyper-parameters ourselves.

The original paper states that "minibatches of 10 examples" are used, but it is not clear what an "example" is in this context. In our interpretation, an example is a set of 10 subclips, so our batches are of size $(10, 10, 1, 34950)$.

We initialise our hyper-parameters with random values so as not to assume prior knowledge. Then we perform a grid search wherein we optimise one hyper-parameter, fix it, and move onto the next. Models are trained for 30 epochs, and evaluated against the validation data every epoch. The AUC scores presented are taken from the epoch which produces the best performance on the validation dataset. The best model during the training is saved and later used for testing.

We tried forgoing input normalisation; using standardisation, in which we subtract the mean and divide by the standard deviation; and using min-max normalisation, in which we subtract the minimum and divide by the range. It is evident that some form of input normalisation is necessary, as the model without input normalisation gave an AUC of 0.53. Of the two remaining methods, min-max normalisation performs significantly better, producing an AUC of 0.72 as opposed to 0.71.

We find that AUC increases with the number of filters in the initial layer. We expected this increase in AUC to come at the cost of longer training times, but this did not occur. However, the number of parameters increased, suggesting that additional computations were being done in parallel. As we are not subject to the same hardware limitations as the original authors, we do not select the largest number of filters, as this would have been infeasible for them. Instead, we use 32 filters, maintaining consistency with the other convolution layers. This gave us an AUC of 0.81.

Finally, we tuned the learning rate and then momentum. In a preliminary experiment, we tried the following possibilities for learning rate: $[0.1, 0.05, 0.01, 0.005, 0.001]$. We found that 0.01 and 0.005 performed the best, so we ran another experiment where we tried $[0.005, 0.0075, 0.01]$. A learning rate of 0.0075 performed best, yielding an AUC of 0.81. Then we tuned momentum, trying the following possibilities: $[0.90, 0.91, 0.92, 0.93, 0.94, 0.95, 0.96, 0.97, 0.98, 0.99]$. The value 0.95 performed best, with an AUC of 0.82.

VI. Quantitative Results

Affect of Length and Stride on AUC		
Length	***Stride***	***AUC (raw audio)***
1024	1024	81.07%
1024	512	81.66%
512	512	81.62%
512	256	82.08%
256	256	82.44%
256	128	81.83%
128	128	81.69%

TABLE I: The average AUCs of the basic Convolution Neural Network architecture over five trials for different lengths and strides for the strided convolution

We tested our model on the five pairs of length and stride that were mentioned in the original paper. We extended the table that they presented in an effort to find a configuration that produced better results. Table I shows the AUC scores produced by evaluating our model on testing data. Our best result, 82.44%, was for a length and stride of 256.

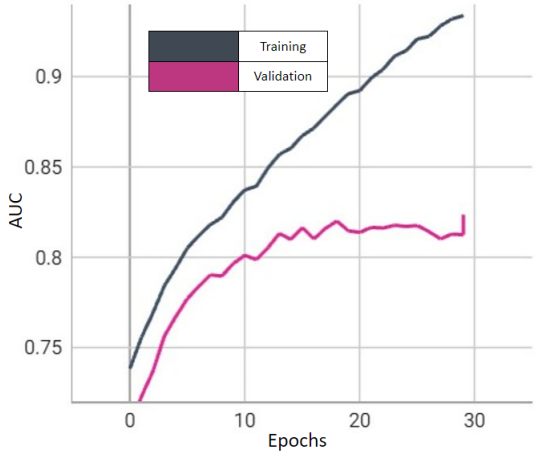

Fig. 3: A graph showing the relationship between AUC and number of epochs during training.

Fig. 3 shows the relationship between AUC and the number of epochs during the training of our best configuration, which used a length and stride of 256. The navy curve represents the AUC produced by evaluations against training data. We see that AUC increases in proportion to number of epochs. However, the magenta curve represents the AUC produced by evaluations against validation data. We see that AUC peaks at around 19 epochs, and then slowly falls off. This suggests that our model overfits the training data after a certain point. We save the model parameters after the best epoch according to evaluation against the validation data. On the far right of the graph, a peak can be seen which corresponds to the evaluation of this saved model against validation data.

VII. Qualitative Results

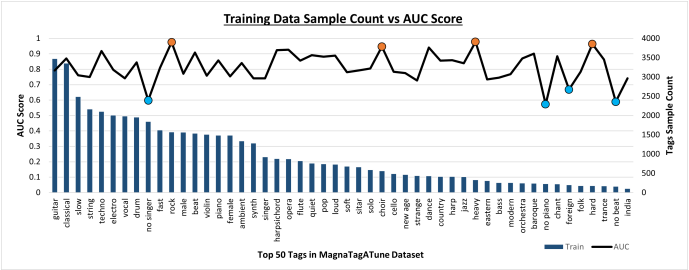

Fig. 4: The sample counts of the top 50 tags of the MagnaTagATune training dataset (decreasing in sample counts from left to right) and the AUC per label/class of the test dataset

As indicated by the light blue markers in figure 4, the model struggles to identify examples that are labelled using one or more of the following tags: “foreign”, “no singer”, “no beat”, and “no piano”. This is also evidenced by the low AUCs of the raw audios of *shira_kammen-music_of_waters-12-jackass_canyon-204-233* and *magnatune-relaxation_spa-04-secret_solution_michael_masley-146-175* which are tagged by “foreign” and “no singer”, and “no beat” and “no singer” respectively. The first clip has an AUC of 0.59, and the second clip’s AUC is 0.55. This means that the model’s performance on these two clips is close to a random guess.

On the other hand, the model performs very well on audio with tags- "hard", "rock", "choir", and "heavy". This is well represented by the AUC score of 1 for the audio *electric_frankenstein-sick_songs-08-back_at_you-88-117*, which is tagged with "rock" and "hard" amongst other tags.

Whilst the low representation of "no piano", "foreign", and "no beat" examples in the training dataset can explain their low test AUC scores, the same cannot be said about examples with the "no singer" label. We believe the reason for this is that audio with no vocals will contain fewer features, so the model will struggle to make predictions. For example, without vocals being layered on top of instrumentation, there will be fewer temporal and timbre-related features. On the other hand, with examples that are tagged with labels such as "rock" and "choir", the different instruments and vocals provide a great variety of timbre-related patterns. Feature extraction is easier, allowing for a higher AUC score for these particular examples [2].

We also observed that despite having among the highest sample counts in the training dataset, the "guitar" and "classical" tags were not among the best performers in terms of per-class AUC. For example, there are almost 3500 samples tagged with "guitar", making it the most popular tag. But the AUC score for the "guitar" tag is only 0.78. We suppose that this is a result of complex features being present in music tagged with "guitar". For example, Won et al. show that harmonic features are important for MIR tasks [6]. Guitar players sometimes use capos to change the harmonic characteristics of their instruments, making harmonic features more difficult to learn.

VIII. Improvements

A. Group Normalisation and Dropout

To further improve upon the basic architecture, we applied normalisation techniques after each convolution and dense layer but before their respective activation functions. This hastens training by optimising and stabilising the learning process and allows for better generalisation of the model. The normalisation layer is omitted from the final layer to avoid corrupting the predictions. We experimented with batch, layer, and group normalisation. According to table II, group normalisation outperforms other normalisation techniques based on the AUC scores produced by our model on the test dataset.

AUC Performance for Different Normalisation Techniques	
Normalisation Technique	***Average AUC***
Batch Normalisation	81.69%
Layer Normalisation	83.66%
Group Normalisation	84.70%

TABLE II: The average AUCs of the basic Convolution Neural Network architecture over five trials with different Normalisation Techniques

Group normalisation is a technique, introduced by Yuxin Wu and Kaiming He, as an alternative to batch normalisation [9]. Whilst batch normalisation is a revolutionary technique in deep learning, it is heavily dependent on the model's batch size and increases model error as the batch size decreases. Group normalisation, on the contrary, is independent of the batch size, and divides the different channels from the prior layer into groups and normalises within these groups. The group normalisation layer computes the mean μ and standard deviation σ within a group and is formulated by

$$\mu_i = \frac{1}{m}\sum_{k\in S_i} x_k, \qquad \sigma_i = \sqrt{\frac{1}{m}\sigma_{k\in S_i}(x_k-\mu_i)^2+\epsilon},$$

where S_i is the set of neurons in a group i [9]. The inputs within a given group are then normalised based on the group's μ_i and σ_i The group normalisation layer is easily implemented and only requires the hyper-parameter tuning of one parameter - the number of groups. This parameter has to be a multiple of the number of channels in the input, so we tried multiples of 32- 2, 4, 8, and 16. We found that the model performs the best when the group size is set to 8, meaning there are 4 channels per group.

Another technique that is added to the model is dropout [8]. It is a regularisation technique which is used to prevent the model from over-fitting due to co-adaptation between neurons over multiple iterations. It randomly deactivates or drops neurons of the neural network, each with a probability p- usually implemented using a Bernoulli distribution. This creates 2^n possible "thinned" networks that the model can sample from and train during each forward propagation, with n being the number of neurons in the network. The weights are shared between the thinned networks in each forward pass. During testing time, all the neurons are activated with the outgoing weights of each neuron being multiplied by p allowing for the thinned network to be combined into one big network. Dropout helps the model generalise better to unseen data [8]. We conducted hyper-parameter-tuning on our updated model using the validation dataset and discovered $p = 0.2$ to work the best for our model. This increased the average AUC of the model to 85.06% over five trials.

We also increase the number of epochs the model trains to 40, as the model takes longer to converge with the updated model, whose architecture can be seen 2. This leads to around a 0.5% increase in the AUC score, making the average AUC of the upgraded model around 85.54%

B. SampleCNN

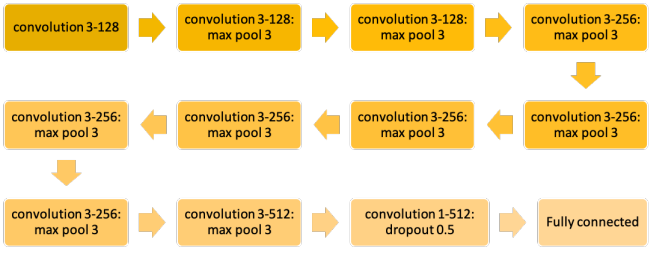

Fig. 5: Deep Network

We recreate SampleCNN as described by Lee et al. [5]. Fig. 5 outlines the architecture we implemented. We apply

the ReLU activation function and batch normalisation after each convolution layer. The only exception to this is the final convolution layer, which is effectively a fully-connected layer. To this, we apply a sigmoidal activation function. We use the SGD optimiser with 0.9 Nesterov momentum, and our learning rate is determined by a scheduler which begins at 0.01 and reduces by a factor of 5 when performance does not increase for 3 epochs. We use the binary cross-entropy function as our criterion. So far this constitutes a faithful recreation, but we have made two modifications. Firstly, we use a batch size of 10 rather than 23. This is because we were not able to access enough memory to load that many samples with this particular model. Secondly, we use an input of shape (B, 10, 1, 4). This differs from the original paper because they do not split each of their samples into sub-clips, and their data is sampled at 16KHz rather than 12KHz.

When we evaluate our recreation against testing data from the MagnaTagATune dataset, we get an AUC score of 0.86. This is significantly lower than the original paper, which mentions an AUC of 0.91. However, it matches up with the benchmark included in the paper by Lee et al., where SampleCNN gives an AUC of 0.87 [4]. We suspect that the modifications made to the original architecture resulted in the specified hyper-parameters being sub-optimal. We correct this by performing hyper-parameter tuning; we use the same process as outlined in our implementation details section V.

We compared different methods of inter-layer normalisation. Specifically; group normalisation, batch normalisation, and no normalisation. We found that group normalisation performed best, with an AUC of 0.87. Next, we tested the efficacy of the scheduler and found that using no scheduler increased our AUC by 0.004. We tested 10 different values of momentum, 0.9 through to 0.99. We found that a momentum of 0.95 was optimal, with an AUC of 0.88. At this stage, we found that the model did not converge after 20 epochs. By extending the number of epochs to 30 we increased the observed AUC to 0.89.

When we evaluated this hyperparameter-tuned model against testing data, we got an AUC of 0.90. This number is similar to the performance specified by the original paper and significantly higher than the performance of our shallow CNN.

We believe that there are two main reasons why SampleCNN outperforms our shallow CNN. Firstly, the additional hidden layers learn features hierarchically, resulting in the model making predictions based on more complex features. We can provide evidence for this belief by analysing the AUC scores of the "guitar" class. In section VII we note that the "guitar" class did not perform as well as expected, and we posit that this may be because it includes complex features. When using SampleCNN, the AUC of the "guitar" class increases by 15% when the overall AUC only increases by 5%, suggesting that SampleCNN is better able to learn these complex features.

The smaller kernel size in the initial convolution layer is the second reason why SampleCNN outperforms the shallow CNN. A kernel size of 3 corresponds to 0.25 ms of audio, whereas a kernel size of 256 corresponds to 21 ms. As such, SampleCNN is able to learn much more granular features.

IX. Conclusion and Future Work

In this report, we describe our recreation of "End-to-End Learning for Music Audio" by Dieleman et al. [1]. We evaluate our implementation of their architecture against training data from the MagnaTagATune dataset, we get an AUC of 0.82. We extend our recreation with group normalisation and drop-out to get an AUC of 0.86. We find that both models struggle to perform well in certain classes, and we posit that their lack of depth may be a limiting factor. We implement SampleCNN, a deeper network described by Lee et al. and find that AUC increases to 0.90 [6]. Furthermore, per-class AUC for the problematic tags increases disproportionately, suggesting that our supposition was correct.

In future work, we could investigate the efficacy of the one-sided joint embedding described by Lee et al [4]. The cited paper uses inter-tag correlations to improve AUC by an average of $\tilde{2}\%$ on the given base architectures. We could also experiment with using mel-spectrograms as an input to our shallow CNN, this would provide an interesting point of comparison to use when evaluating both our shallow and deep networks. We could then integrate the trainable front-end module described by Won. et al [7]. into our shallow CNN. The authors state that this module exploits the inherant harmonic characteristics of audio signals; it would be interesting to evaluate the improvement this confers to the AUC of the "guitar" tag, which has unusual harmonic features.

References

[1] Sander Dieleman and Benjamin Schrauwen. End-to-end learning for music audio. In *2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 6964–6968, 2014.

[2] Zhouyu Fu, Guojun Lu, Kai Ming Ting, and Dengsheng Zhang. A survey of audio-based music classification and annotation. *IEEE Transactions on Multimedia*, 13(2):303–319, 2011.

[3] Qingqing Huang, Aren Jansen, Joonseok Lee, Ravi Ganti, Judith Yue Li, and Daniel P. W. Ellis. Mulan: A joint embedding of music audio and natural language, 2022.

[4] Jik-Soo Kim Jaehwan, Daekyeong Moon and Minkyoung Cho. Atose: Audio tagging with one-sided joint embedding. 2023.

[5] Jongpil Lee, Jiyoung Park, Keunhyoung Luke Kim, and Juhan Nam. Sample-level deep convolutional neural networks for music auto-tagging using raw waveforms. *the 14th Sound & Music Computing Conference*, abs/1703.01789, 2017.

[6] Oriol Nieto Minz Won, Sanghyuk Chun and Xavier Serra. Automatic music tagging with harmonic cnn. *International Society for Music Information Retrieval (ISMIR)*, 2019.

[7] Oriol Nieto Minz Won, Sanghyuk Chun and Xavier Serra. Data-driven harmonic filters for audio representation learning. *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2020.

[8] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958, 2014.

[9] Yuxin Wu and Kaiming He. Group normalization. *CoRR*, abs/1803.08494, 2018.