An Application of Convolutional Neural Networks to Music Genre Classification

Benjamin Steiner (44199)

Olly Mapps (52086)

A report submitted to the Department of Statistics of the London School of Economics and Political Science

April 18, 2023

Abstract

In this report, we explore the GTZAN dataset for music genre classification. Our dataset is split into 10 genres of music and we will try to learn the differences between them. For each audio file, we are also provided with an image, which has a time series element, and metadata. We are therefore able to compare the performance of CNNs for visual feature extraction, RNNs for temporal feature extraction, and MLPs for evaluating the recordings metadata.

A focus of this report is also on the ensemble methods described in Section 2.4 which allow us to use various types of data in a single model. We postulate that by using the various forms of the data, and different model architectures, we can extract more meaningful information from each recording, whether that be temporal, visual or numerical features. In fact, we find that these ensemble models do perform better. As a baseline, we first implement a CNN, RNN, and MLP and these achieve test accuracies of 67.7%, $>$ 70%, and 38.8%, respectively. As our first ensemble, we train a CNN-RNN model which attains a test accuracy of 68%. However, the CNN-MLP performed the best by far, with an accuracy of 81%.

Contents

Chapter 1

Introduction

1.1 The GTZAN Dataset

Our dataset contains audio recordings of 1000 songs, each 30 seconds long and grouped evenly into 10 genres: Blues, Classical, Country, Disco, Hiphop, Jazz, Metal, Pop, Reggae, and Rock. For each song, there is a Mel-spectrogram and associated metadata. Each instance of metadata contains auditory characteristics of the recording, such as the tempo. In contrast, the Mel spectrogram is a way of visualising what frequencies are more intense on the melodic scale at a given time.

1.2 Problem Setting

Given an audio file, we aim to predict the genre of the music. The novelty of our problem arises from the fact that there are no well-established neural network architectures to deal with audio signals. Therefore, we must first transform our data into a format that is interpretable by our neural network. We achieve this by a natural application of signal processing using spectograms. Spectograms are a visual representation of the intensity of frequencies as time progresses. Now that our data has been converted to an image, we can apply the standard Convolutional Neural Network (CNN) to learn features of our data. While these differences are hard to quantify, we show in Figure 1.1 that there are noticeable differences between genres.

1.3 Model Selection

We first replicate two models used in Pun and Nazirkhanova (2021): the Multi-Layer Perceptron (MLP), and the CNN-MLP ensemble. Due to the time series nature of our data, we also examine the applicability of the Recurrent Neural Network (RNN). We consider a sole RNN and CNN, and a CNN-RNN in two ways. Firstly, we consider it sequentially whereby the output of the CNN is fed into an RNN, and also as an

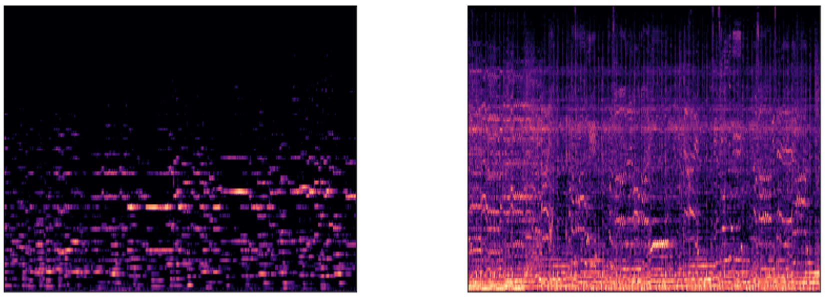


Figure 1.1: Left: A spectogram from a piece of classical music. Right: A spectogram from heavy metal music. We observe that the heavy metal spectogram is far busier than the classical spectogram.

ensemble CNN-RNN model where the networks learn in parallel. An important part of this report is the exploration of the CNN-RNN and CNN-MLP ensemble methods. We hypothesise that by combining two baseline models, we are able to exploit certain features from each model to attain a better overall model.

Chapter 2

Methodology

2.1 Data Transformation

As mentioned earlier, we are unable to directly use the audio files as input to our CNN. However, we are able to obtain useful Mel-spectograms via a Fourier transformation, but we are graciously already provided with these images. We convert these images into grayscale as we care solely about the intensity of each pixel rather than the actual colour. Doing this will reduce the number of input channels for each convolutional layer, reducing the overall training time. We visualise an example grayscale image in Figure 2.1:

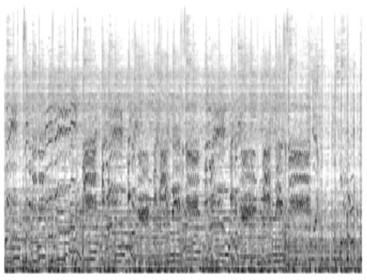


Figure 2.1: An example grayscale Mel spectogram.

For our metadata, we standardise our columns to follow a *Normal(0, 1)* distribution. We do this due to the large differences in the order of magnitude between columns. For example, the variable rms_var has order of magnitude of 10^{-3} whereas spectral_centroid_var has an order of magnitude of 10^{5}.

2.2 Data Augmentation

An immediate limitation we encountered was the relatively small number of audio files per genre (100 files for each of the 10 genres). In order to improve the performance of our models, we will first augment our data by splitting each spectogram vertically into 5 evenly-sized images. This corresponds to splitting each 30-second audio file into 5 6-second audio files. We claim that each instance of our new dataset should still be representative of the original image it came from.

The reader may wonder whether it is appropriate to split time series data in this way. We argue that, while a valid concern, each smaller image will still be highly correlated with the other splits from the same image. Therefore, we will not be losing too much information, and so the overall increase in data we now have should outweigh the information loss. *Ceteris paribus*, for the CNN model we found that the augmented dataset on average attained a test accuracy 10% higher than the original dataset, backing up our previous claim.

2.3 Baseline Methods

In this section, we describe the models we consider to be our baseline. This achieves two goals: firstly, for comparison to the more complicated models later on; secondly, a more subtle point is that we can perform exploratory analysis of these models by examining questions such as "do more convolutional layers improve performance?" or "does dropout improve performance?". While the exact answers may not be appropriate for the ensemble models, we can generalise to help us later on.

2.3.1 CNN

As a baseline model, we implement a standard CNN with 5 convolutional layers and 2 Fully Connected (FC) layers, and using a kernel size of 3×3 with a stride length of 1 for Convolutional layers. Between each layer we use a ReLU activation function and connect these to the Max Pooling layers. After flattening to a 1×784 tensor, we feed this through our FC layers with 64 and then 10 neurons.

While tuning the hyperparameters, we observed the following: that the of use dropout was necessary to prevent some of the overfitting. Therefore, we included dropout in each of our convolutional layers, using $p = 0.2$. We also saw that a small number of FC layers improved our results. Furthermore, we found that a small stride length and kernel size performed well. These seem like reasonable observations because the regularisation through dropout and ReLU activation counteracts increased complexity caused by the small kernel size and stride. In context, this means that with our relatively small amount of data we can perform more complex calculations for each image, while still retaining a reasonable overall training time.

We also examined the methodologies used in Choi et al. (2017) which compare kernel sizes. In particular, we analyse the differences between using a row vector or column vector as our kernel (e.g a 3×1 or 1×3 kernel). The consequence of this is that we are, in essence, collapsing one axis which will help us determine whether there is more information stored in the frequency or time axis of the spectogram. However, we found an indistinguishable difference between the two, so we opted for a square 3×3 kernel in our final model.

2.3.2 Metadata MLP

We are given two datasets of metadata: one for the 30-second audio files, and another for 3-second audio files which have been split from the original files. We actually found that the 30-second audio files performed better, and for this, we used an MLP with 4 FC layers, with 57, 32, 16, and 10 neurons in each layer. We used the ReLU activation function and a small amount of dropout ($p = 0.25$ in two of the layers).

2.3.3 RNN

Since our data has a time series aspect to it, we now consider the effects of time. As a baseline, we first considered a simple, single-layer RNN. However, this did not achieve sufficient performance to reasonably consider as a baseline. To improve performance, we implemented a bi-directional RNN with Gated Recurrent Activation Units (GRUs), as suggested in Cho et al. (2014). The benefit of using GRUs is twofold: firstly, through the use of an update and reset gate, the unit can remember information from long ago in the time series and, secondly, the GRU is far more computationally efficient than, say, a Long Short Term Memory (LSTM) unit, which uses three gates. We also found that adding bidirectional functionality improved performance. That being said, the performance of the RNN on its own was unremarkable. We hypothesise that it should be used in tandem with another model, so that the RNN can extract the time-series data, and the other could extract defining characteristics about the track.

When defining the number of hidden units, we considered sizes 32, 64, 128 and 256. We found that 64 provided the best performance in both training and validation. With too many neurons, performance metrics would rise and then sharply collapse at some point during the training, likely due to over-fit. With too few neurons, we do not see any convergence. After the Bi-GRU layer, we feed the output into a fully connected layer with 10 neurons, this would then produce our output.

2.4 Ensemble Methods

As briefly mentioned earlier, we will consider the effects of combining multiple models together to create a new architecture. With the various data types, we can use appropriate baseline architectures and then merge these outputs together before feeding this through the final FC layers.

2.4.1 CNN-MLP

By combining the image data and metadata, we now train a CNN-MLP. Each instance of data will now consist of the Mel-spectogram and the associated metadata. We can think of our network as two networks which we connect after some training. We input the image to the convolutional layers, and the metadata to a separate MLP. After the final convolutional layer in the CNN and FC layer in the MLP these networks are connected. We then feed the concatenated data through some final FC layers to obtain our output.

To maximise the performance of the ensemble, it was necessary to make some changes to the individual networks used as our baseline. In order to make use of the final FC layers in the ensemble model, we increased the output sizes of each sub-network to 1×100. We then concatenate these two together to obtain a 1×200 tensor, which we feed into the final FC layers. This ensures that we maximise the information extracted from the respective models. We also found that increasing the complexity of the MLP improved the performance of the ensemble, without causing too much over-fit. This is likely due to the large dropout on the CNN balancing out any tendencies for over-fit.

2.4.2 CNN-RNN

As mentioned earlier, we hypothesised that an RNN coupled with another network could provide a nice balance between feature extraction, and time series consideration. We opted to pair it with the CNN since they would then operate on the same image input. We considered both a parallel structure, as in Yang et al. (2020), and then a sequential structure, with the CNN output being fed into the RNN, as shown in Ashraf et al. (2023) and Choi et al. (2017). We found improved performance in the sequential structure, as in this case, the CNN can extract the key features of the image and the RNN can use its internal memory to learn the temporal properties of these features. If ran in parallel, we found the RNN struggles with extracting this information when fed the raw image input.

As in the CNN-MLP case, to optimise performance, it was necessary to alter the respective networks. Interestingly, the main change that gave us improved performance was the removal of the bi-directional layer in the RNN, and its replacement with 3 single layer GRUs. This replacement was made largely due to the computational intensity of a bi-directional layer, but we also found it improved our model's predictive accuracy and training behaviour. Along with this, we increased the number of neurons in the final FC layer of the CNN to 128. This produces an input of a 1×128 tensor for the RNN. We found that this magnitude gave the best balance between data preservation and complexity for the input.

We also experimented with increasing the output size of the CNN further and then implementing a 2×2 Max Pooling layer at the start of the RNN to reduce the input size. However, this led to negligible differences in computational efficiency and more importantly, predictive power.

2.5 Summary

In Table 2.1 we briefly summarise the models we used. The stride and kernel represent their values in the convolutional layers. We used the ReLU activation function and Cross Entropy loss for all of our models.

	CNN	MLP	RNN	CNN-MLP	RNN-CNN
FC Layers	3	4	1	7	3
Convolutional Layers	5	N/A	N/A	5	5
Dropout	Low	Medium	Medium	High	None
Stride	1	N/A	N/A	1	2
Kernel	3×3	N/A	N/A	3×3	2×2

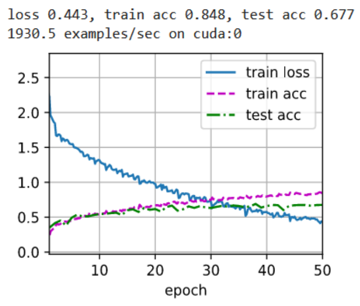
Table 2.1: An overview of the models we used.

Chapter 3

Numerical Evaluation

3.1 Baseline Methods

For our baseline CNN, we achieved a test accuracy of 67.7%, as shown in Figure 3.1. We do observe some overfitting after all epochs, but not excessive. We also find that using many convolutional layers performed better, but that any more FC layers actually decreased the accuracy. Finally, we opted for Max Pooling layers over Average Pooling in order to retain some of the dimensions and prevent our images becoming over-simplified.



```
loss 0.443, train acc 0.848, test acc 0.677
1930.5 examples/sec on cuda:0
```

Figure 3.1: The training results of our CNN.

Our MLP exhibited a few curious properties, in particular that our test accuracy hardly changed depending on whether we used 2, 3, or 4 FC layers. In fact, adding too many FC layers decreased accuracy drastically. Another parameter to point out is the learning rate, which is bigger than we would typically see, but it performs much better than a learning rate of 0.01 or smaller. Once again, as long as the order of magnitude was correct, we observe only a marginal difference. For example, using a learning rate $0.1 \leq \eta \leq 0.5$ yields roughly the same result. We show the learning process below in Figure 3.2:

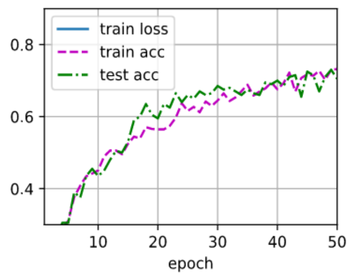
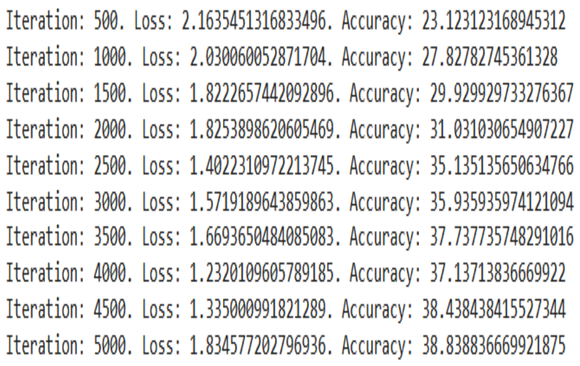


Figure 3.2: The training results of our MLP.

Our RNN model proved itself to be extremely fragile, a minor change in its structure or the algorithm's learning parameters led to wildly different training outcomes. We found that an increase in the learning rate from 0.0001 to 0.001 would cause the model to become extremely over-fit. We made use of early stopping in our training procedure so as to prevent our final model from being excessively over-fit as well. The performance of our RNN is outlined in Figure 3.3:

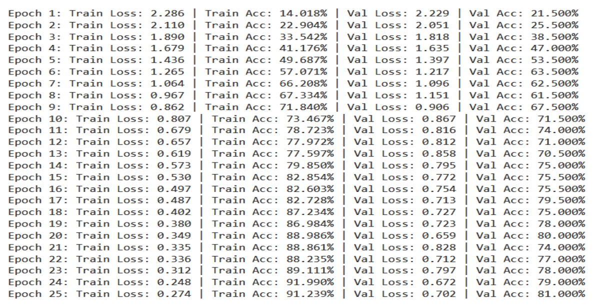


```
Iteration: 500. Loss: 2.1635451316833496. Accuracy: 23.123123168945312
Iteration: 1000. Loss: 2.030060052871704. Accuracy: 27.82782745361328
Iteration: 1500. Loss: 1.8222657442092896. Accuracy: 29.929929733276367
Iteration: 2000. Loss: 1.8253898620605469. Accuracy: 31.031030654907227
Iteration: 2500. Loss: 1.4022310972213745. Accuracy: 35.135135650634766
Iteration: 3000. Loss: 1.5719189643859863. Accuracy: 35.935935974121094
Iteration: 3500. Loss: 1.6693650484085083. Accuracy: 37.737735748291016
Iteration: 4000. Loss: 1.2320109605789185. Accuracy: 37.13713836669922
Iteration: 4500. Loss: 1.335000991821289. Accuracy: 38.438438415527344
Iteration: 5000. Loss: 1.834577202796936. Accuracy: 38.838836669921875
```

Figure 3.3: The training results of our RNN.

3.2 Ensemble Methods

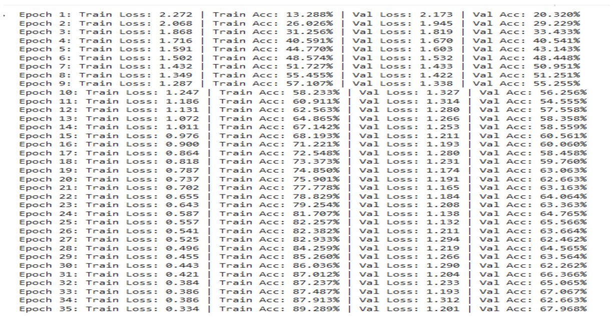
Our best-performing model was the CNN-MLP, with a final test accuracy of 81%. Ultimately this model shows that in our case, extracting temporal information from the track is not necessary to achieve high prediction accuracy, the mixture of visual and feature learning capability provided by the CNN-MLP is sufficient. The training feedback is outlined in Figure 3.4. As with our other models, it does flirt with over-fit in the later epochs, however, we deem the test accuracy high enough to justify this. We found that any more drop-out or any earlier a stop hinders the learning process of the network.



```
Epoch 1: Train Loss: 2.286 | Train Acc: 14.018% | Val Loss: 2.229 | Val Acc: 21.500%
Epoch 2: Train Loss: 2.110 | Train Acc: 22.904% | Val Loss: 2.051 | Val Acc: 25.500%
Epoch 3: Train Loss: 1.890 | Train Acc: 33.542% | Val Loss: 1.818 | Val Acc: 38.500%
Epoch 4: Train Loss: 1.679 | Train Acc: 41.176% | Val Loss: 1.635 | Val Acc: 47.000%
Epoch 5: Train Loss: 1.436 | Train Acc: 49.687% | Val Loss: 1.397 | Val Acc: 53.500%
Epoch 6: Train Loss: 1.265 | Train Acc: 57.071% | Val Loss: 1.217 | Val Acc: 63.500%
Epoch 7: Train Loss: 1.064 | Train Acc: 66.208% | Val Loss: 1.096 | Val Acc: 62.500%
Epoch 8: Train Loss: 0.967 | Train Acc: 67.334% | Val Loss: 1.151 | Val Acc: 61.500%
Epoch 9: Train Loss: 0.862 | Train Acc: 71.840% | Val Loss: 0.906 | Val Acc: 67.500%
Epoch 10: Train Loss: 0.807 | Train Acc: 73.467% | Val Loss: 0.867 | Val Acc: 71.500%
Epoch 11: Train Loss: 0.679 | Train Acc: 78.723% | Val Loss: 0.816 | Val Acc: 74.000%
Epoch 12: Train Loss: 0.657 | Train Acc: 77.972% | Val Loss: 0.812 | Val Acc: 71.000%
Epoch 13: Train Loss: 0.619 | Train Acc: 77.597% | Val Loss: 0.858 | Val Acc: 70.500%
Epoch 14: Train Loss: 0.573 | Train Acc: 79.850% | Val Loss: 0.795 | Val Acc: 75.000%
Epoch 15: Train Loss: 0.530 | Train Acc: 82.854% | Val Loss: 0.772 | Val Acc: 75.500%
Epoch 16: Train Loss: 0.497 | Train Acc: 82.603% | Val Loss: 0.754 | Val Acc: 75.500%
Epoch 17: Train Loss: 0.487 | Train Acc: 82.728% | Val Loss: 0.713 | Val Acc: 79.500%
Epoch 18: Train Loss: 0.402 | Train Acc: 87.234% | Val Loss: 0.727 | Val Acc: 75.000%
Epoch 19: Train Loss: 0.380 | Train Acc: 86.984% | Val Loss: 0.723 | Val Acc: 78.000%
Epoch 20: Train Loss: 0.349 | Train Acc: 88.986% | Val Loss: 0.659 | Val Acc: 80.000%
Epoch 21: Train Loss: 0.335 | Train Acc: 88.861% | Val Loss: 0.828 | Val Acc: 74.000%
Epoch 22: Train Loss: 0.336 | Train Acc: 88.235% | Val Loss: 0.712 | Val Acc: 77.000%
Epoch 23: Train Loss: 0.312 | Train Acc: 89.111% | Val Loss: 0.797 | Val Acc: 78.000%
Epoch 24: Train Loss: 0.248 | Train Acc: 91.990% | Val Loss: 0.672 | Val Acc: 79.000%
Epoch 25: Train Loss: 0.274 | Train Acc: 91.239% | Val Loss: 0.702 | Val Acc: 81.000%
```

Figure 3.4: The training results of our CNN-MLP.

Finally, as described in Figure 3.5, our CNN-RNN model does offer improvements over the baseline RNN, but still struggles to achieve truly great performance. However, the sequential model we settled on still performs roughly 20% better than the parallel architecture, and almost 30% better than our baseline RNN. The ensemble model is vastly more robust than the sole RNN as well. We found that tuning the parameters gave us marginal improvements, as opposed to the baseline where it could destroy the network.



```
Epoch 1: Train Loss: 2.272 | Train Acc: 13.288% | Val Loss: 2.173 | Val Acc: 20.320%
Epoch 2: Train Loss: 2.068 | Train Acc: 26.026% | Val Loss: 1.945 | Val Acc: 29.229%
Epoch 3: Train Loss: 1.868 | Train Acc: 31.256% | Val Loss: 1.819 | Val Acc: 33.433%
Epoch 4: Train Loss: 1.716 | Train Acc: 40.591% | Val Loss: 1.670 | Val Acc: 40.541%
Epoch 5: Train Loss: 1.591 | Train Acc: 44.770% | Val Loss: 1.603 | Val Acc: 43.143%
Epoch 6: Train Loss: 1.502 | Train Acc: 48.574% | Val Loss: 1.532 | Val Acc: 48.448%
Epoch 7: Train Loss: 1.432 | Train Acc: 51.727% | Val Loss: 1.433 | Val Acc: 50.951%
Epoch 8: Train Loss: 1.349 | Train Acc: 55.455% | Val Loss: 1.422 | Val Acc: 51.251%
Epoch 9: Train Loss: 1.287 | Train Acc: 57.107% | Val Loss: 1.338 | Val Acc: 55.255%
Epoch 10: Train Loss: 1.247 | Train Acc: 58.233% | Val Loss: 1.327 | Val Acc: 56.256%
Epoch 11: Train Loss: 1.186 | Train Acc: 60.911% | Val Loss: 1.314 | Val Acc: 54.555%
Epoch 12: Train Loss: 1.131 | Train Acc: 62.563% | Val Loss: 1.280 | Val Acc: 57.558%
Epoch 13: Train Loss: 1.072 | Train Acc: 64.865% | Val Loss: 1.266 | Val Acc: 58.358%
Epoch 14: Train Loss: 1.011 | Train Acc: 67.142% | Val Loss: 1.253 | Val Acc: 58.559%
Epoch 15: Train Loss: 0.976 | Train Acc: 68.193% | Val Loss: 1.211 | Val Acc: 60.561%
Epoch 16: Train Loss: 0.900 | Train Acc: 71.221% | Val Loss: 1.193 | Val Acc: 60.060%
Epoch 17: Train Loss: 0.864 | Train Acc: 72.548% | Val Loss: 1.280 | Val Acc: 58.458%
Epoch 18: Train Loss: 0.818 | Train Acc: 73.373% | Val Loss: 1.231 | Val Acc: 59.760%
Epoch 19: Train Loss: 0.787 | Train Acc: 74.850% | Val Loss: 1.174 | Val Acc: 63.063%
Epoch 20: Train Loss: 0.737 | Train Acc: 75.901% | Val Loss: 1.191 | Val Acc: 62.663%
Epoch 21: Train Loss: 0.702 | Train Acc: 77.778% | Val Loss: 1.165 | Val Acc: 63.163%
Epoch 22: Train Loss: 0.655 | Train Acc: 78.829% | Val Loss: 1.184 | Val Acc: 64.064%
Epoch 23: Train Loss: 0.643 | Train Acc: 79.254% | Val Loss: 1.208 | Val Acc: 63.363%
Epoch 24: Train Loss: 0.587 | Train Acc: 81.707% | Val Loss: 1.138 | Val Acc: 64.765%
Epoch 25: Train Loss: 0.557 | Train Acc: 82.257% | Val Loss: 1.132 | Val Acc: 65.566%
Epoch 26: Train Loss: 0.541 | Train Acc: 82.382% | Val Loss: 1.211 | Val Acc: 63.664%
Epoch 27: Train Loss: 0.525 | Train Acc: 82.933% | Val Loss: 1.294 | Val Acc: 62.462%
Epoch 28: Train Loss: 0.496 | Train Acc: 84.259% | Val Loss: 1.219 | Val Acc: 64.565%
Epoch 29: Train Loss: 0.455 | Train Acc: 85.260% | Val Loss: 1.266 | Val Acc: 63.564%
Epoch 30: Train Loss: 0.443 | Train Acc: 86.036% | Val Loss: 1.290 | Val Acc: 62.262%
Epoch 31: Train Loss: 0.421 | Train Acc: 87.012% | Val Loss: 1.204 | Val Acc: 66.366%
Epoch 32: Train Loss: 0.384 | Train Acc: 87.237% | Val Loss: 1.233 | Val Acc: 65.065%
Epoch 33: Train Loss: 0.386 | Train Acc: 87.487% | Val Loss: 1.193 | Val Acc: 67.067%
Epoch 34: Train Loss: 0.386 | Train Acc: 87.913% | Val Loss: 1.312 | Val Acc: 62.663%
Epoch 35: Train Loss: 0.334 | Train Acc: 89.289% | Val Loss: 1.201 | Val Acc: 67.968%
```

Figure 3.5: The training results of our CNN-RNN.

3.3 Summary

In Table 3.1 we summarise our results:

	CNN	MLP	RNN	CNN-MLP	CNN-RNN
Test Accuracy (%)	67.7	> 70	38.8	81.0	68.0
Train Accuracy (%)	84.8	> 70	N/A	91.2	89.2
Batch Size	30	50	16	30	16
Learning Rate	0.01	0.1	0.0001	0.0001	0.0001
Epochs	50	50	5000	25	35

Table 3.1: A summary of our results.

Chapter 4

Conclusion

4.1 Findings

In this report, we performed a music classification task by comparing various baseline methods to state-of-the-art architectures still in development. We found that these so-called ensemble methods do in fact perform better due to their ability to exploit the various features we are interested in. We also observe that the data-splitting process significantly improved performance, but this was to be expected with the small dataset we use.

As a sort of sanity check, we also see that we do not have to change our baseline methods too much in order to build our ensemble methods. This is reassuring as it improves our confidence in the robustness of our models to generalise well to other datasets. A natural worry with a small dataset is the likelihood that we are overfitting to the dataset, so our models would not perform well once deployed to a new distribution of data.

In Pun and Nazirkhanova (2021), they achieved test accuracies of 68% and 81% for their MLP and CNN-MLP, respectively. Using different architectures, these are exactly the same results that we achieved with our models. Our MLP achieved an accuracy just over 70%, comparable to the CNN, however, the RNN does not perform well at all with only a 38.8% accuracy. Consequently, the ensemble CNN-RNN achieves an accuracy of 68% - only marginally better than the baseline CNN. Both of these results provide evidence against the use of RNNs for our task. The final ranking of our models is as follows: `CNN-MLP` > `CNN-RNN` > `MLP` > `CNN` > `RNN`.

4.2 Limitations

There are a few obvious limitations to our analysis, some of which stem directly from the dataset. Firstly, the dataset is relatively small for a complicated task such as this one. Secondly, the subjective nature of the labels - how sure can we be that there is a significant difference between Pop and Hiphop, or Jazz and Blues? There are also some genres which use music from the same composer - reducing the within-genre variation to help us learn the differences. For a more comprehensive overview, we refer the reader to Sturm (2013).

4.3 Further Work

With more time, we could consider some other ensemble methods; in particular, some more of the CNN-RNN models in Ashraf et al. (2023). These used various types of RNNs in the architecture, combining both the LSTM and Bi-GRU layers to the CNN simultaneously. These achieved test accuracies above 87%, a significant improvement on our CNN-MLP.

4.4 Team Contributions

We both agree that this project was 50% effort from both of us. Ben was responsible for the collection and splitting of data, and the baseline CNN and MLP. Whereas Olly researched the RNN and ensemble methods, before implementing and tuning them.

Bibliography

Ashraf et al. (2023). A hybrid CNN and RNN variant model for music classification. *Applied Sciences* 13(3), 1476.

Cho et al. (2014). Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078* .

Choi et al. (2017). Convolutional recurrent neural networks for music classification. In: *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2392–2396. IEEE.

Pun, A. and K. Nazirkhanova (2021). Music genre classification with Mel spectograms and CNN .

Sturm, B. L. (2013). The gtzan dataset: Its contents, its faults, their effects on evaluation, and its future use. *arXiv preprint arXiv:1306.1461* .

Yang et al. (2020). Parallel recurrent convolutional neural networks-based music genre classification method for mobile devices. *IEEE Access* 8, 19629–19637.