CS429/529 - Project 3: Neural Networks

Names: Kaneez Fatima, Sathvik Quadros

1 Data Preparation

Structured Data

The structured data for this project was prepared in the same was it was prepared for the previous Logistic Regression project, i.e., through the use of the Librosa library and multiprocessing. However, we ended up actually using a couple less features for this project than the previous one, as we saw that disabling the features lead to better results when using them with the MLP. The extracted features were all organized in a CSV file in the same method that it is generally done for similar structured data.

Spectrogram Generation

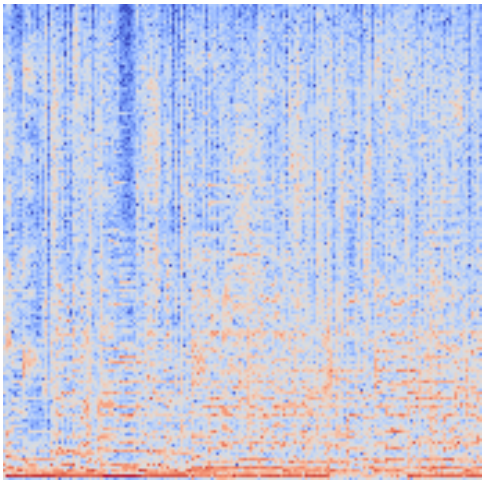
In order to create the spectrograms to be analyzed we once again used Librosa, but in combination with Matplotlib. The created spectrogram images were of size 173x172, as we wanted them to be dimensionally close to a square (as square images are highly recommended for ResNet50), not too big (in order to make training faster), and still contain a sufficient quantity of information.

As we cannot simply store images in a csv files, we went ahead and put them in a directory tree as shown below:

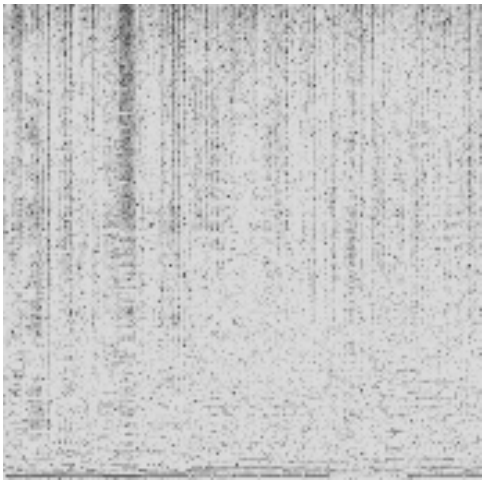
```
data
├── spectrograms
│   ├── test
│   │   ├── layl
│   │   │   └── unknown
│   │   ├── linear
│   │   │   └── unknown
│   │   └── log
│   │       └── unknown
│   └── train
│       ├── layl
│       │   ├── blues
│       │   ├── classical
│       │   └── ...
│       ├── linear
│       │   └── ...
│       └── log
│           └── ...
└── ...
```

In the above directory tree, directories `layl`, `linear`, and `log` stand for log and y-axis log, linear, and log respectively, which are what is used to group spectrograms of the respective types.

We chose to preserve multiple color channels, as while country.00000.au's log spectrogram looks like this with color:



If the color is removed, it looks like this:



It is immediately obvious from looking at this that a massive amount of data is lost by converting the image to grayscale, and in order to avoid the ramifications of doing such an action we chose to keep the color channels.

In order to make training and testing simple, we simply named the spectrograms the file name of the music file they're of, but with .png appended to the end. That is to say, country.00000.au's spectrogram has the name "country.00000.au.png".

2 Multilayer Perceptrons (MLPs) Implementation

Training and Hyperparameter Tuning

From the previous project doing logistic regression, we got the following results for the accuracy given specific pairs of learning rates and numbers of epochs:

LR \ Epochs	10k	50k	100k	250k	500k
0.01	0.55	0.64	0.66	0.67	0.68
0.05	0.64	0.67	0.68	0.68	0.66
0.1	0.66	0.68	0.68	0.66	0.66
0.8	0.68	0.66	0.66	0.66	skipped
2.0	0.67	0.66	0.66	skipped	skipped
10.0	0.62	0.64	0.65	skipped	skipped

Looking at this table, we were able to see that a learning rate up to even 0.8 allowed us to reduce the number of epochs significantly without compromising on accuracy when doing logistic regression. Based on this information we decided to start at 0.8, before iteratively selecting a new learning rate in order to increase the accuracy.
We used a generic MLP model with a batch size of 32 and hidden layers of sizes 90, 70, 50, and 30, and came all the way down to a learning rate of 0.01, with a weight decay of 0.001, converging at around 100 epochs.

Architecture Design

The architecture for this model was designed iteratively. We started with the hidden layer sizes of 90, 70, 50, and 30, and once we managed to optimize the hyperparameters so that they converged quickly, we started experimenting with them. We found out that removing the last hidden layer, bringing us down to just 90, 70, and 50, allowed us to a maximum accuracy of 0.74, and as we weren't able to improve this result and were satisfied with what we had, we stopped here.

3 Convolutional Neural Network (CNN)

Architecture Design

The final design of the CNN has a total of 4 Conv2D layers, each with a kernel size of 4, a stride of 4, and a padding of 1. The existence of stride in the Conv2D layers removes the need for pooling layers, and so no additional pooling layers were used. This configuration was chosen due to it having the highest accuracy among all the combinations tested through manual kaggle resubmissions, with the following chart containing data on the accuracy of various numbers of layers, kernel sizes, and stride sizes using the Tanh activation function. For all the data in this section, a weight decay of 10^{-4} and a learning rate of 10^{-3} were used initially with learning rate scheduling:

Num. Conv2d	Kernel Size	Stride Size	Accuracy
3	4	5	0.48
4	3	4	0.38
4	4	4	0.61
4	5	4	0.59
5	4	3	0.54
6	4	2	0.59

Tanh was used as the activation function, as it performed the best out of all the activation functions for CNNs of 6 and 4 Conv2D layers. The following chart contains data on the accuracy of the 6 layer CNNs created with commonly used activation functions:

Activation Layer	Accuracy
Tanh	0.59
ReLU	0.54
Sigmoid	0.10

With this chart containing data on the accuracy of the 4 layer CNNs created with commonly used activation functions:

Activation Layer	Accuracy
Tanh	0.61
ReLU	0.32
Sigmoid	0.33

Training and Hyperparameter Tuning

After designing the CNN, learning rate scheduling and early stopping were used for efficient training. In addition to that, checkpoints were taken whenever an improvement was made in the validation loss, allowing us to have a copy of the most accurate model.

Hyperparameter tuning for this section was done only for weight decay and learning rate. Thanks to early stopping there was no need for us to worry about the optimal number of epochs, but we still set the maximum to 100 to avoid wasting time. We decided to select the learning rate and weight decay after obtaining some initial findings from having the weight decay be 10% of the learning rate:

Learning Rate	Weight Decay	Accuracy	Epochs
1	10^{-1}	0.10	16
10^{-1}	10^{-2}	0.11	8
10^{-2}	10^{-3}	0.24	18
10^{-3}	10^{-4}	0.61	26
10^{-4}	10^{-5}	0.43	35

Due to how well a learning rate of 10^{-3} and weight decay of 10^{-4} did, we ended up simply using that as our learning rate and weight decay for the CNN. A batch size of 32 was used, as we found out that the influence on accuracy was minimal to none up to that point.

4 Transfer Learning

Selection of Modality and Pre-trained Models

The data modality we chose for transfer learning was spectrogram-based, and this was due to the fact that it offered significantly richer temporal information. While feature-based learning did offer some information about the mean, max, and min values for a specific metric across a range of time, it still fails to stand up to spectrograms in terms of information across a period of time, something that is very important for music files.

Once we'd selected the modality, we needed to select the model which we would use for transfer learning. For this, we ended up choosing ResNet50. ResNet was one of the recommendations given to us in the PDF for this project, and after seeing how it mitigates the vanishing gradient problem to some extent and won the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) of 2015, we decided that it would be the best option for our project. Inception was also considered as it had won the ILSVRC of 2014, but as ResNet had won the ILSVRC of the following year, it was clear that it had beaten Inception.

Fine-Tuning and Performance Improvement

In order to fine-tune this model, we started off by simply changing the output layer size to the class size of our dataset. We initially kept all the layers frozen, and tried out different epoch sizes, learning rates, and batch sizes, but were only able to get a maximum accuracy of 0.16.

We then began unfreezing layers, while using the same learning rate, batch size, and weight decay as we did for the CNN, which was 10^{-3}, 32, and 10^{-4} respectively. The results are below:

Unfrozen Layers	Accuracy	Epochs
1	0.63	13
2	0.69	19
3	0.67	10
4	0.64	17

Analysis and Interpretation

From the table in the previous section, it is quite clear that a total of 2 unfrozen layers lead to the best results in this situation. Transfer learning led to a massive improvement over just creating a CNN, and while it may not be better when all 4 layers are frozen, after unfreezing two layers we can see that it is approximately 13% better than training a CNN from scratch.

It should be mentioned, however, that evaluating test data took significantly longer for this model than it did for the CNN. While the CNN took only around 1-3 seconds, ResNet50 ended up taking around 200 seconds.

5 Model Evaluation and Comparison

The different models achieved the following values in the corresponding metrics on the training set:

Model	Accuracy	Precision	Recall	F1-Score
MLP	0.9395	0.9401	0.9395	0.9396
CNN	0.9466	0.8729	0.9000	0.8768
ResNet50	0.9832	0.9604	0.9616	0.9579

The different models achieved the following values in the corresponding metrics on the validation set:

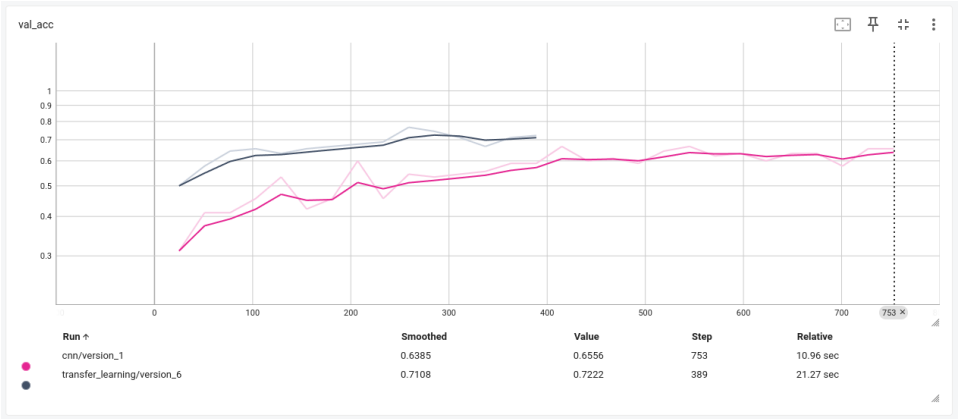
Model	Accuracy	Precision	Recall	F1-Score
MLP	0.8000	0.8073	0.8000	0.7941
CNN	0.6385	0.6472	0.6764	0.6264
ResNet50	0.7108	0.6359	0.6477	0.6145

Finally, they obtained these additional values:

Model	Train Time in seconds	Kaggle Score (Test Accuracy)
MLP	2.77	0.74
CNN	9.34	0.61
ResNet50	21.5	0.69

From the previous tables, it's quite obvious that MLP performs the best out of all the models. Not only is it the fastest to train, but it also has the highest accuracy on the validation set and the highest Kaggle score, beating the others by a sizable margin. That said, while the winner is quite obvious, there are still a few other interesting things worth mentioning:

- Both ResNet50 and CNN have a higher training accuracy than MLP, but a lower validation and testing accuracy than MLP. This may be a sign that they're overfitting, and have the potential to beat MLP under different conditions.
- While not shown in the above tables, ResNet50 takes significantly longer to evaluate the testing set. While MLP and CNN only take a few seconds at most, ResNet50 takes around 200 seconds.
- Despite CNN's lower Accuracy on the validation set, it has a higher Precision, Recall, and F1-Score than ResNet50.
- All the models have a consistently higher validation accuracy than testing accuracy. This may be a sign that there is a small gap between the content in the testing and training sets.
- Finally, looking at the below graph for validation accuracy, we can see that ResNet50 immediately starts off with a higher validation accuracy than the CNN. This is most likely due to the fact that it's a pretrained model, giving it an advantage over a model which needs to be trained from scratch.



6 Optimizations

The only significant optimization we did outside optimizing the hyperparameters for the various models in the previous sections, was through the implementation of multiprocessing in feature extraction and spectrogram generation, as there weren't many other places where multiprocessing would be implemented.

Using multiprocessing in feature extraction reduced the amount of time it took to collect the features by a significant amount, as shown in the table below (all time in seconds, on a 16 core 32 thread system):

Threads	1	4	16	32
Extract Time	4716.342	638.994	316.360	288.186

Using multiprocessing in spectrogram generation reduced the time it took to create the spectrograms by a significant amount as well, as shown in the table below (all time in seconds, on a 16 core 32 thread system):

Threads	1	4	16	32
Generation Time	1198.574	340.174	185.166	212.532

While Matplotlib is inherently thread-unsafe, multiprocessing allows us to completely ignore that limitation by spawning entirely new processes instead of having a single process with multiple threads. The decrease in performance for 32 processes may have been due to the lack of sufficient resources for each of the processes that had been created, or the presence of some other background process.

7 Other Features/Things of note

A few other features have also been implemented for this project:

- General and expandable functions for training and evaluating have been written to make the addition of new models for comparison very straightforward.
- An easy-to-use config file has been created, making it so that there's a single place which controls all modifications.