Music Genre Classification

Shay Sirek
Omer Zamir
Omri Shtamberger

Submitted as final project report for the Deep Learning,
COLMAN, 2021

Contents

1 Introduction

The project is about classifying music input into one of 10 different genres. We chose this problem since we were interested in working with audio input and its representation.

1.1 Related Works

1. GTZAN [1] Dataset
2. Understanding the Mel Spectrogram
3. Music Genre Classification using Convolutional Neural Network with Pytorch
4. Tranfer learning with Efficient Net
5. Convolutional Gated Recurrent Neural Network Incorporating Spatial Features for Audio Tagging
6. Supplementary material for IJCNN paper "Musical Artist Classification with Convolutional Recurrent Neural Networks"

2 Solution

2.1 General approach

Our general approach was finding a visual representation for the audio input and using CNN [2] to process it. We found that the best overall audio representation for that approach is using Mel-Spectogram [3], and found the GTZAN dataset [4] that includes 1000 instances of 30 seconds wave audio files divided to 100 wave files for each genre.

We found a pre-processed dataset in Kaggle where every wave audio file was processed into one Mel-Spectogram image file. We started working with the pre-processed data and during the process we noticed that we were missing data. To solve this issue we processed each audio file into 10 Mel-Spectograms by sampling intervals of 3 seconds.

After we chose our data representation, we had to find which model works best for our problem. Our options were training a basic CNN model from scratch, making use of transfer learning [5] from an existing network, or using some layers of RNN [6].

We wanted to test each model option and compare their performance. We planned to use a basic CNN model as a baseline for performance of all models, and later found through transfer learning from different networks each time, which gave the best performance. We considered adding some RNN layers after convolution for selected best performing models to see how it affects the result.

2.2 Design

Our code is in Jupyter Notebook on Google's Colab platform [7] using GPU runtime.

We started with simple CNN model built of three layers of convolution with 3X3 filters, ReLU activation function, and max pooling with 2X2 kernel. Then, we tested a bigger CNN model built of four convolution layers with twice as much filters at every layer. We experimented adding GRU [8] module to the four layers CNN model and we did not observed any improvements.

In addition, we considered the Transfer Learning strategy. After a thorough research, we found that EfficientNet-b0 model fits our task, and added a pool layer and linear layer for classification on top of the extracted model features. We then observed a better accuracy. We also tried adding GRU module to this model and we did not observed any improvement.

We also tried using VGG11 [9] net for transfer learning. We found it is more accurate than our four layers CNN model but less accurate than EfficientNet-b0 [10] based model.

We had to research about audio representations. We found Mel-Spectogram to be the best fit for our task, and used Mel-Spectogram images with resolution of 288X432 as the input for all of our models.

While working on the different models, we used the same train and validation set for evaluation. The validation set was used as a *dev set* for tuning hyper-parameters. Finally, the test set was used for final evaluation and models comparison.

Besides running out of GPU memory, we've had no technical issues.

3 Experimental results

We split our dataset into three subsets: train, validation and test. We chose a split ratio of 60:20:20 respectively. We chose accuracy as a measurement metric [11] because we were dealing with classification task. We also used precision and recall to see the full picture.

Following are tables of our experimental result:

3.1 Training time

Model	Epochs	Training Time
Baseline 3L CNN	15	18:34
4L CNN	15	22:09
4L CNN with GRU	15	23:19
EfficientNet-b0	15	49:47
EfficientNet-b0 with GRU	15	51:35
VGG11	15	50:10

3.2 Accuracy

Model	Train	Validation	Test
Baseline 3L CNN	0.832	0.679	0.673
4L CNN	0.972	0.864	0.871
4L CNN with GRU	0.969	0.847	0.849
EfficientNet-b0	0.974	0.909	0.917
EfficientNet-b0 with GRU	0.824	0.771	0.798
VGG11	0.992	0.898	0.908

3.3 Precision

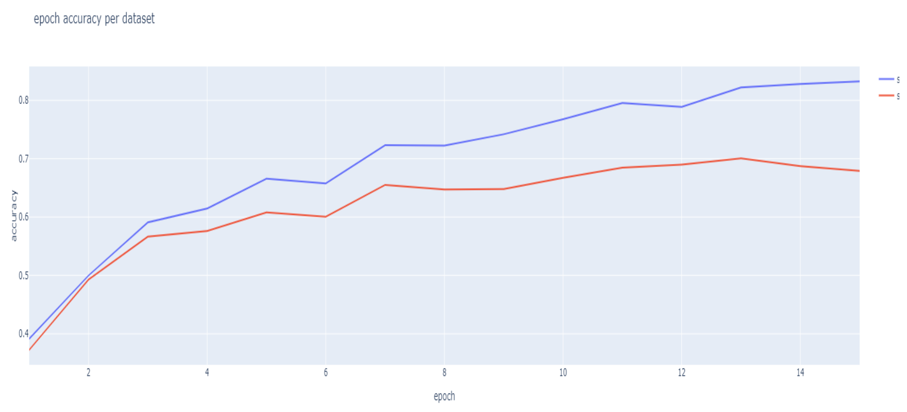
Model	Train	Validation	Test
Baseline 3L CNN	0.848	0.698	0.697
4L CNN	0.972	0.867	0.872
4L CNN with GRU	0.970	0.853	0.860
EfficientNet-b0	0.975	0.917	0.921
EfficientNet-b0 with GRU	0.836	0.785	0.810
VGG11	0.993	0.906	0.915

3.4 Recall

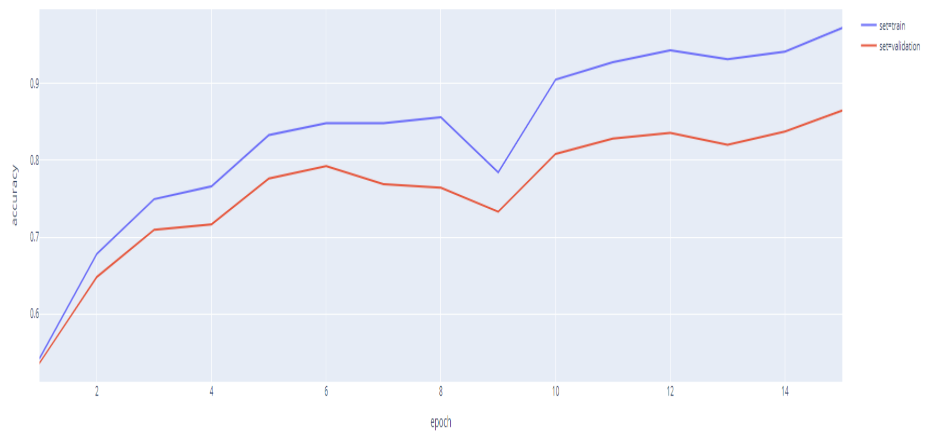
Model	Train	Validation	Test
Baseline 3L CNN	0.830	0.681	0.677
4L CNN	0.972	0.864	0.869
4L CNN with GRU	0.969	0.847	0.848
EfficientNet-b0	0.973	0.909	0.918
EfficientNet-b0 with GRU	0.825	0.769	0.797
VGG11	0.992	0.899	0.909

Following are our training plots (Blue is Train set, Red is Validation set):

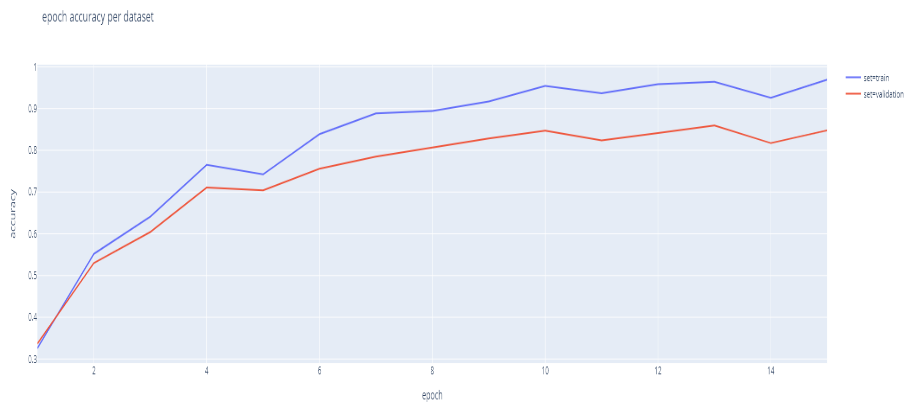
3.5 3L CNN accuracy



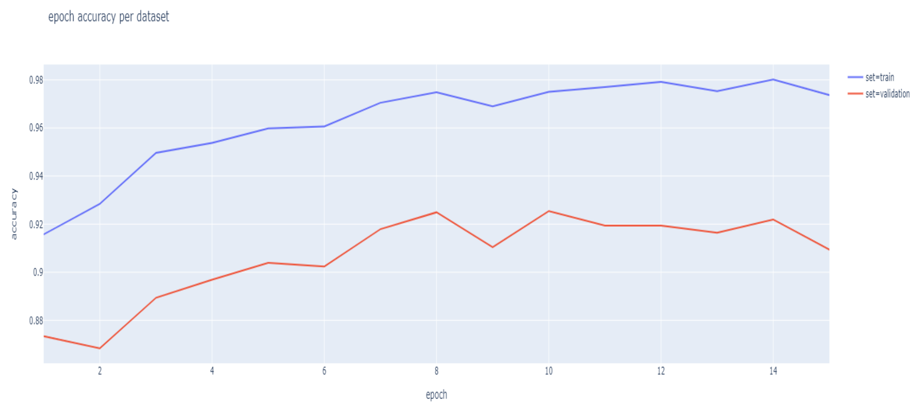
3.6 4L CNN accuracy



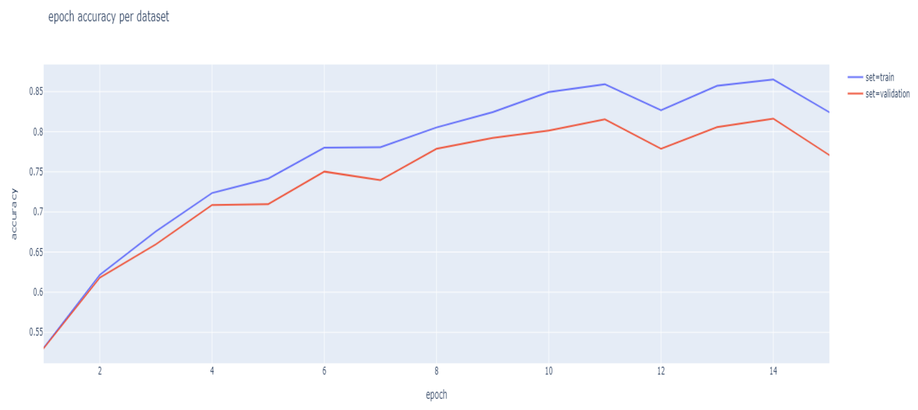
3.7 4L CNN + GRU accuracy



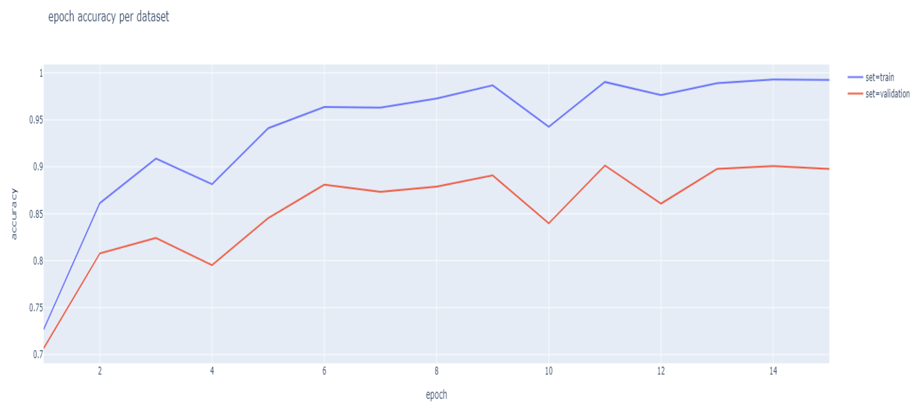
3.8 EfficientNet accuracy



3.9 EfficientNet + GRU accuracy



3.10 VGG11 accuracy



4 Discussion

We observed that 4L CNN model is better performing overall than our baseline 3L CNN. We assume it is caused by the higher number of filters in each layer, and the additional layer compared to the baseline.

When comparing the results of our own 4L CNN model we could see that it performed similarly to the transfer learning based models. This could be caused by fine-tuning on the transfer learning models, which means we do not freeze the pre-trained network weights and it might have hurt the original network's work.

These results could also mean that model's layer depth has no performance influence for this task as much as the number of filters at each convolution layer, in other words layer size could be more influential than network depth.

In addition, the results of 4L CNN with and without including GRU indicate that the model without GRU performs slightly better overall, the same goes for the transfer learning model based on EfficientNet. The transfer learning model based on VGG11 performed slightly better than our 4L CNN since it's a bit deeper and it is pre-trained.

Our task is similar to time series prediction, since spectogram is a visual representation of the spectrum of frequencies of a signal as it varies with time. Thus we considered adding Recurrent layer. The results of using GRU as a RNN module were worse than without it in all models we tested. We assume that it is so because the GRU layer is after the convolution layers. We think that changing the order of the layers such that our GRU layer would happen before our convolution layers could result better. Due to lack of time we did not test that assumption.

We observed much lower performance for the same models when starting working with GTZAN dataset's pre-processed Mel-Spectogram images of 30 seconds wave file from Kaggle. We later created our own pre-processed Mel-Spectogram images from the wave files in the dataset but divided it to 3 seconds to have more instances, and then received the results we have now, which are approximately 20 percent higher.

5 Code

`https://github.com/omier/music-genre-classifier`

References

[1] "Gtzan dataset - gtzan genre collection." [Online]. Available: http://marsyas.info/downloads/datasets.html

[2] "Convolutional neural network." [Online]. Available: https://en.wikipedia.org/wiki/Convolutional_neural_network

[3] "Mel-frequency cepstrum." [Online]. Available: https://en.wikipedia.org/wiki/Mel-frequency_cepstrum

[4] A. Olteanu, "Gtzan dataset - music genre classification." [Online]. Available: https://www.kaggle.com/andradaolteanu/gtzan-dataset-music-genre-classification

[5] "Finetuning torchvision models." [Online]. Available: https://pytorch.org/tutorials/beginner/finetuning_torchvision_models_tutorial.html#vgg

[6] "Recurrent neural network." [Online]. Available: https://en.wikipedia.org/wiki/Recurrent_neural_network

[7] "Google colab platform." [Online]. Available: https://colab.research.google.com/

[8] "Pytorch documentation - pytorch implementation of gru." [Online]. Available: https://pytorch.org/docs/stable/generated/torch.nn.GRU.html

[9] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 2015.

[10] L. Melas-Kyriazi, "A pytorch implementation of efficientnet." [Online]. Available: https://github.com/lukemelas/EfficientNet-PyTorch

[11] "Pytorch lightning - metrics." [Online]. Available: https://pytorch-lightning.readthedocs.io/en/stable/metrics.html