Piano Chord and Root Note Classifier

Albert Hua

Abstract

This report summarizes the development of a piano chord and root note classifier to distinguish between four chord types and twelve root notes using a simple convolutional neural network. The classifier was trained on a dataset of 43,200 piano chord audio samples transformed into mel-spectrograms using the TorchAudio library. The model architecture featured four convolutional blocks with ReLU activations and max-pooling layers, followed by a fully connected layer and softmax layer. Despite achieving 99% accuracy on the 4-class chord type classification task, the 12-class root note classification task plateaued at approximately 76% accuracy. Attempts to improve the latter included using a pretrained ResNet-18, introducing dropout layers, and hyperparameter tuning. Challenges such as computational constraints and overfitting were addressed, with mixed results. This paper concludes with reflections on extending the project to larger, more diverse datasets and investigating higher-quality audio samples for further exploration.

Introduction

Machine learning offers a compelling way to demonstrate performance differences between parallelized and non-parallelized computing in the form of CPU v. GPU computation time, something which was heavily emphasized in my Parallel and Distributed Comp. course. From a suggestion made by a classmate, this project took shape as an audio classification task using PyTorch and the TorchAudio library to classify samples of piano chords, a task both educationally and personally relevant to me. As someone who has previously practiced and played the piano and violin for multiple years, I found this idea to be potentially valuable for early music learners practicing independently, where the need to consult a teacher for assistance in identifying complex or uncommon chords and keys would be eliminated, streamlining the learning process and reducing frustrations when learning new pieces of music.

Data

Our group identified early on a dataset composed of 43,200 four-second piano audio samples, with each filename consisting in part of a distinct chord type and root note (https://zenodo.org/records/4740877). We extracted labels from these conveniently named .wav files using a simple Python script that spliced off the needed information from the filename and mapped it to its respective class value - numbers 1-12 for root note and numbers 1-4 for chord type. This data was converted into two CSV files, with each row consisting of the audio file name and its class value.To enhance the dataset, an augmentation script was employed, extending the existing samples' length and slicing them into smaller, uniform segments. These artificially created segments were also introduced into our dataset, essentially doubling the size of the original set. Using the TorchAudio library, our audio files were transformed into mel-spectrograms, time-frequency representations of an audio signal akin to an image, effectively enabling us to treat this as an image processing task.

Methods

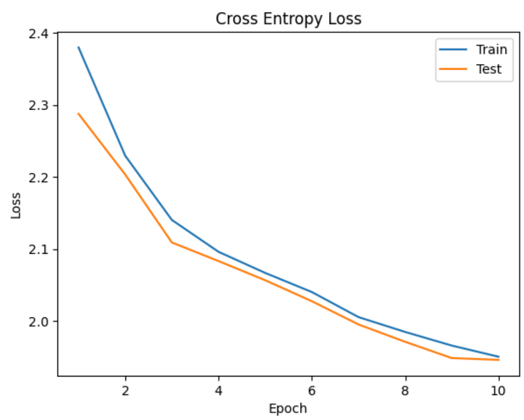
Our model consisted of four simple convolutional blocks, each featuring a 2-D convolutional layer, ReLU activation, and max-pooling. The output was flattened and passed through a linear layer followed by a softmax activation layer for a probability distribution that would be useful for prediction. This design, while simple, aligned with standard practices for image classification, ensuring compatibility with both the 4-class and 12-class tasks. Given the manageable size of our audio samples, an 80/20 training-testing split was predominantly used.

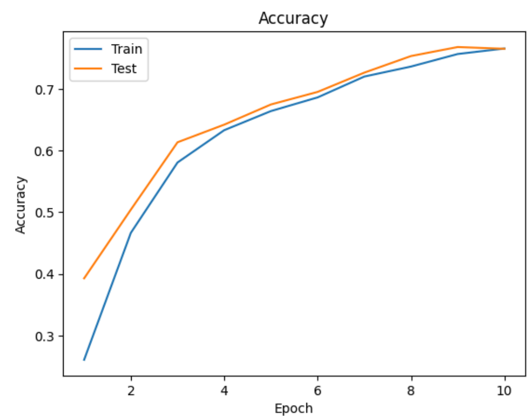
Experiments

Our model achieved a 99% test accuracy with a test loss of 0.7 on the 4-class chord classification task after 50 epochs on a GPU using the entire dataset.

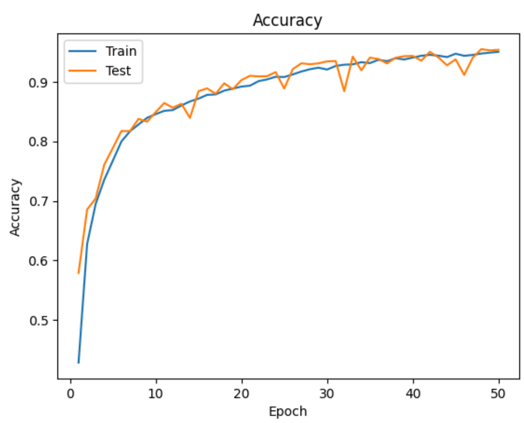
In contrast, the 12-class root note task encountered challenges, stabilizing at 75% accuracy and a 1.9 test loss after 10 epochs. A Pretrained ResNet-18 model initially showed improvement within 20 epochs but exhibited severe overfitting after 50 epochs. Additional experiments included testing varying batch sizes ranging from 8 - 256 in multiples of 2, testing different learning rates between 1e-5 and 1e-8, and introducing incremental dropout layers between the convolutional and ReLU layers across each of the four CNN blocks, starting from a p-value of 0.2 in the first block and increasing in increments of 0.1 until the fourth block at 0.5. Using a CPU for a subset of 4,000 samples, training accuracy improved from 55% to 95%, and testing accuracy rose from 59% to 96% over 1000 epochs, although the results were less conclusive due to computational limitations and technical difficulties which resulted in the training loop terminating early, preventing the data from being graphed using pyplot. Group members using GPUs reported training and testing accuracies of 76%, with consistent loss values plateauing across epochs at roughly 1.7.

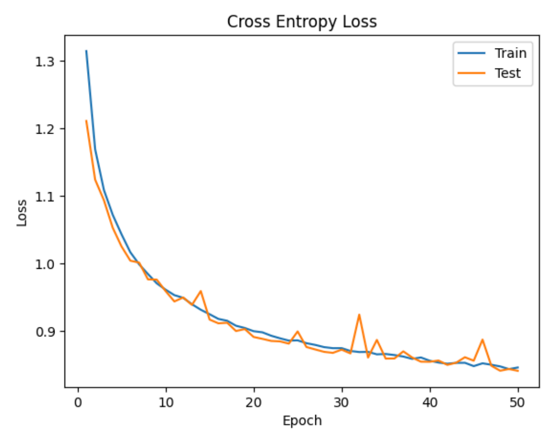
Loss and Accuracy Graphs for Root Prediction (12 classes)





Loss and Accuracy Graphs for Chord Prediction (4 classes)





Epoch Logging

(used since training loop was terminated early, shortened to the first 20 and last 10 epochs)

Using device cpu
There are 3482 items in the training dataset
There are 614 items in the testing dataset
Epoch 1 | Train Loss 2.0992 | Train Accuracy 0.5477 | Test Loss 2.1092 | Test Accuracy 0.5911 | Elapsed Time 128.4212s
Epoch 2 | Train Loss 2.0917 | Train Accuracy 0.5564 | Test Loss 2.1320 | Test Accuracy 0.6005 | Elapsed Time 120.6192s
Epoch 3 | Train Loss 2.0920 | Train Accuracy 0.5582 | Test Loss 2.1250 | Test Accuracy 0.5974 | Elapsed Time 75.4209s
Epoch 4 | Train Loss 2.0939 | Train Accuracy 0.5565 | Test Loss 2.1200 | Test Accuracy 0.5958 | Elapsed Time 71.2798s
Epoch 5 | Train Loss 2.0947 | Train Accuracy 0.5537 | Test Loss 2.1163 | Test Accuracy 0.6115 | Elapsed Time 68.5900s
Epoch 6 | Train Loss 2.0956 | Train Accuracy 0.5594 | Test Loss 2.1231 | Test Accuracy 0.5927 | Elapsed Time 68.5254s
Epoch 7 | Train Loss 2.0865 | Train Accuracy 0.5659 | Test Loss 2.1146 | Test Accuracy 0.6052 | Elapsed Time 69.2897s
Epoch 8 | Train Loss 2.0826 | Train Accuracy 0.5702 | Test Loss 2.1099 | Test Accuracy 0.6052 | Elapsed Time 69.9182s
Epoch 9 | Train Loss 2.1014 | Train Accuracy 0.5570 | Test Loss 2.1045 | Test Accuracy 0.6271 | Elapsed Time 69.9932s
Epoch 10 | Train Loss 2.0825 | Train Accuracy 0.5605 | Test Loss 2.1188 | Test Accuracy 0.5865 | Elapsed Time 70.0640s

Epoch 11 | Train Loss 2.0828 | Train Accuracy 0.5642 | Test Loss 2.1280 | Test Accuracy 0.5849 | Elapsed Time 70.7448s
Epoch 12 | Train Loss 2.0865 | Train Accuracy 0.5692 | Test Loss 2.0985 | Test Accuracy 0.6443 | Elapsed Time 70.2982s
Epoch 13 | Train Loss 2.0868 | Train Accuracy 0.5666 | Test Loss 2.1218 | Test Accuracy 0.6068 | Elapsed Time 71.3662s
Epoch 14 | Train Loss 2.0860 | Train Accuracy 0.5695 | Test Loss 2.1047 | Test Accuracy 0.6302 | Elapsed Time 70.6788s
Epoch 15 | Train Loss 2.0731 | Train Accuracy 0.5800 | Test Loss 2.1128 | Test Accuracy 0.6271 | Elapsed Time 71.1799s
Epoch 16 | Train Loss 2.0813 | Train Accuracy 0.5813 | Test Loss 2.1159 | Test Accuracy 0.6130 | Elapsed Time 71.2882s
Epoch 17 | Train Loss 2.0888 | Train Accuracy 0.5795 | Test Loss 2.1084 | Test Accuracy 0.6177 | Elapsed Time 70.4178s
Epoch 18 | Train Loss 2.0744 | Train Accuracy 0.5871 | Test Loss 2.0924 | Test Accuracy 0.6552 | Elapsed Time 70.9770s
Epoch 19 | Train Loss 2.0717 | Train Accuracy 0.5858 | Test Loss 2.0926 | Test Accuracy 0.6479 | Elapsed Time 71.3912s
Epoch 20 | Train Loss 2.0712 | Train Accuracy 0.5913 | Test Loss 2.0887 | Test Accuracy 0.6583 | Elapsed Time 70.9501s
Epoch 989 | Train Loss 1.6896 | Train Accuracy 0.9567 | Test Loss 1.7708 | Test Accuracy 0.9641 | Elapsed Time 75.4098s
Epoch 990 | Train Loss 1.6931 | Train Accuracy 0.9529 | Test Loss 1.7699 | Test Accuracy 0.9609 | Elapsed Time 74.5006s
Epoch 991 | Train Loss 1.6915 | Train Accuracy 0.9554 | Test Loss 1.7711 | Test Accuracy 0.9594 | Elapsed Time 84.3837s
Epoch 992 | Train Loss 1.6953 | Train Accuracy 0.9500 | Test Loss 1.7718 | Test Accuracy 0.9625 | Elapsed Time 84.2663s
Epoch 993 | Train Loss 1.6962 | Train Accuracy 0.9493 | Test Loss 1.7688 | Test Accuracy 0.9672 | Elapsed Time 77.1520s
Epoch 994 | Train Loss 1.6918 | Train Accuracy 0.9549 | Test Loss 1.7700 | Test Accuracy 0.9625 | Elapsed Time 76.2418s
Epoch 995 | Train Loss 1.6887 | Train Accuracy 0.9566 | Test Loss 1.7713 | Test Accuracy 0.9547 | Elapsed Time 79.0843s
Epoch 996 | Train Loss 1.6965 | Train Accuracy 0.9475 | Test Loss 1.7710 | Test Accuracy 0.9625 | Elapsed Time 86.3408s
Epoch 997 | Train Loss 1.6903 | Train Accuracy 0.9555 | Test Loss 1.7698 | Test Accuracy 0.9594 | Elapsed Time 84.0905s
Epoch 998 | Train Loss 1.7006 | Train Accuracy 0.9435 | Test Loss 1.7714 | Test Accuracy 0.9578 | Elapsed Time 84.1408s
Epoch 999 | Train Loss 1.6968 | Train Accuracy 0.9477 | Test Loss 1.7752 | Test Accuracy 0.9516 | Elapsed Time 81.4269s
Epoch 1000 | Train Loss 1.7051 | Train Accuracy 0.9400 | Test Loss 1.7706 | Test Accuracy 0.9609 | Elapsed Time 83.2257s

Conclusion:

Results were as expected: predictions for chord type reached much higher accuracies compared to predictions for the root note, which were much more inconsistent.

For my own personal learning, this was my first exposure to audio processing using PyTorch and it's something I found to be very intriguing given my previous background in music. I plan on looking more into the features of the TorchAudio library and its uses beyond a simple 4- class and 12-class classification problem.

Some potential courses of action that I could take include training the model consistently over winter break (preferably with a GPU as I had issues using Google Colab over the course of this project and eventually resorted to training on a CPU). I could perhaps introduce larger audio samples consisting of multiple chords and/or higher quality audio samples to test how the model would perform under these conditions, and if any modifications would need to be made as a result. Conveniently, I have a sibling who currently practices and plays the piano consistently, so I could ask them to generate audio samples for me to train on, which could eliminate the need to augment an existing dataset for artificial samples which may prove to be unreliable or redundant.

Overall, this project was a fundamental learning experience. In the process of creating, training, and tuning a simple CNN, I found myself needing to review and consult class materials including previous homework assignments to see how CNNs were set up and trained effectively, and I can confidently say that I have a deeper understanding of how Neural Networks function and the various methods used to develop and train them.