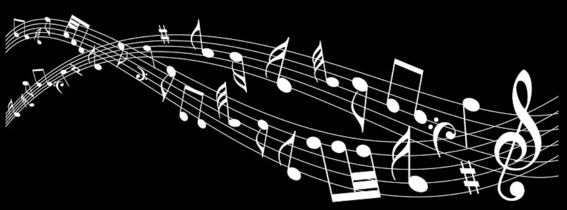
Audio Classification

CSI-4650 Parallel and Distributed Computing

By:
Jason Kauppila
Albert Hua
Bernard Kassab



Project Goal

- Given an audio sample of a chord, determine the chord type and root note of the chord.
- Two different kinds of classification:
  - Chord Type Classes
    - Major
    - Minor
    - Diminished
    - Augmented
  - Chord Root Classes
    - Ab, A, Bb, B, C, Db, D, Eb, E, F, Gb, G

Project Applications

Tool For Music Transcription

- Assists with learning and music theory
    - Great for new learners of music and useful during practice when a teacher is not around.
- Helps to analyze songs
    - Identify common chord progressions, keys, etc.
    - Could be useful for music arrangement and creating remixes of songs.

Data Collection and Preprocessing

- Audio Piano Triads Dataset
  - By Agustín Macaya Valladares
    - Obtained from Zenodo.org
  - 43,200 Audio Samples
    - .WAV format
    - 4 seconds long
- Additional Data Creation
  - 43,200 additional samples created by modifying each .wav sample from the original dataset
    - Marvel Crunch Amplifier in Mixcraft 9 Recording Studio

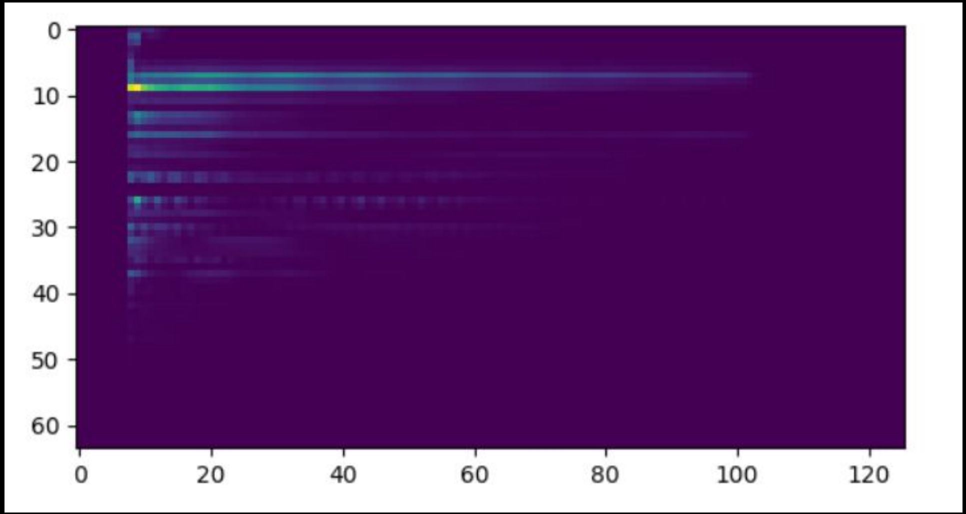
- Audio Preprocessing Pipeline
  - Audio resampled to a uniform sample rate
    - 16,000 Hz
  - Audio mixed down to mono by averaging
  - Audio adjusted to a fixed length
    - Trimming
    - Padding
  - Audio transformed into a Mel Spectrogram using TorchAudio
    - (Image representation of audio by plotting frequency over time)

G Major
piano_3_Gn_j_m_00.wav:

Modified G Major
WAV from Mixcraft 9:

Mel Spectrogram

- Can be thought of as a grayscale image.
  - 64 x 126 for our audio samples
- Parameters:
  - Size of Fast Fourier Transform (FFT) : 1024
  - Hop Length: 512
  - Number of Mel-Filterbanks: 64



Model Used

- Simple Convolutional Neural Network (CNN)
  - Optimizer: Adam
  - Loss Function: Cross Entropy
  - Network Architecture from Valerio's Velardo's Audio Classifier
    - Described as "a simple VGG-ish architecture"
      - (VGG is a model built by Google for Image Classification)
    - 4 Convolutional Blocks, Flatten Layer, Dense (Linear) Layer, Softmax Layer
    - Each Convolutional Block consists of 3 parts:
      - 2D Convolutional Layer (nn.Conv2d)
        - Kernel SIze = 3
        - Stride = 1
        - Padding = 2
        - Out Channels increase with each block accordingly
          - 16, 32, 64, 128
      - Rectified Linear Unit (ReLU) Layer
      - 2D Max Pooling Layer
        - Kernel Size = 2

Model Expectations

- The model was expected to return high accuracy results for *chord type* predictions (85-90%) which can be classified into 4 categories.

- Lower expected accuracy (75%) expected for *root note* predictions, which can be classified into 12 classes or categories.

- Training the model on a GPU was expected to be roughly 5 times faster compared to a CPU.

Parallelization Context

- Neural network training is inherently parallelizable, as neural network layers often incorporate frequent SIMD operations over a large batch of data (i.e. matrix multiplication, activation functions, updating weights for individual nodes during backpropagation, etc).

- The individual neurons in each layer of the NN operate on data independently, and there are no dependencies between nodes in the same layer.

Hardware Details

CPU:
Intel Core i7-10750H
- 2.60 GHz (2592 Mhz)
- 6 Cores (12 Logical Processors)

System Specs:

ASUS ROG Zephyrus M15
- 16 GB RAM
- 1 TB SSD

GPU:
NVIDIA GeForce RTX 2070 with Max-Q Design
- 8 GB VRAM
- 885 MHz Base Clock
- 1185 MHz Boost Clock
- 2304 Cores

Benchmarking Methodology

- 70/30 Train-Test Split (good for larger batches of data)
- Run the “training loop” for 5 epochs.
  - For each epoch:
    - Determine the time spent training the 70% of the data subset
    - Determine the time spent testing the 30% of the data subset
  - At the end, average the training time and the testing time over the epoch and log the results for each iteration.
- Repeat this “training loop” for a few iterations.
  - Run a total of 6 iterations, but drop the first iteration.
    - First of the six iterations acts as a “warm-up” because it was found that sometimes the hardware needs to “warm up”, so this is done to prevent outliers.
  - Calculate the average epoch time across all iterations and plot this in a graph
    - Overall average epoch time rather than average of average epoch times
- Compare 2 configurations:
  - Minimal Parallelism
    - CPU
  - Optimized for Parallelism
    - GPU

Benchmarking Methodology

Experiment 1

Use a subset of the data with 8192 samples.

- Experiment with various batch sizes:
  - [32, 64, 128, 256]
- Train both the Chord Root CNN and the Chord Type CNN

Experiment 2

Use a fixed batch size of 32 samples.

- Experiment with various data subset sizes:
  - [4096, 8192, 16384]
- Only train the Chord Type CNN

Benchmarking Results (Experiment 1)

Batch Size 32 Average Epoch Times								
	Chord Root Classifier				Chord Type Classifier			
Trial / Device	GPU Train (s)	CPU Train (s)	GPU Test (s)	CPU Test (s)	GPU Train (s)	CPU Train (s)	GPU Test (s)	CPU Test (s)
1	9.4113	19.3856	3.9830	5.0596	9.6702	19.0727	3.6116	5.0492
2	9.6397	19.3812	3.7232	4.9899	9.7752	19.0617	3.6040	5.0205
3	9.5039	19.3554	3.9851	4.9959	9.5483	19.3656	3.9571	5.0353
4	9.5741	19.4911	3.9658	4.9846	9.6773	19.2806	3.6440	5.1045
5	9.6378	19.5899	3.7473	5.0367	9.7574	19.4048	3.7029	5.0899

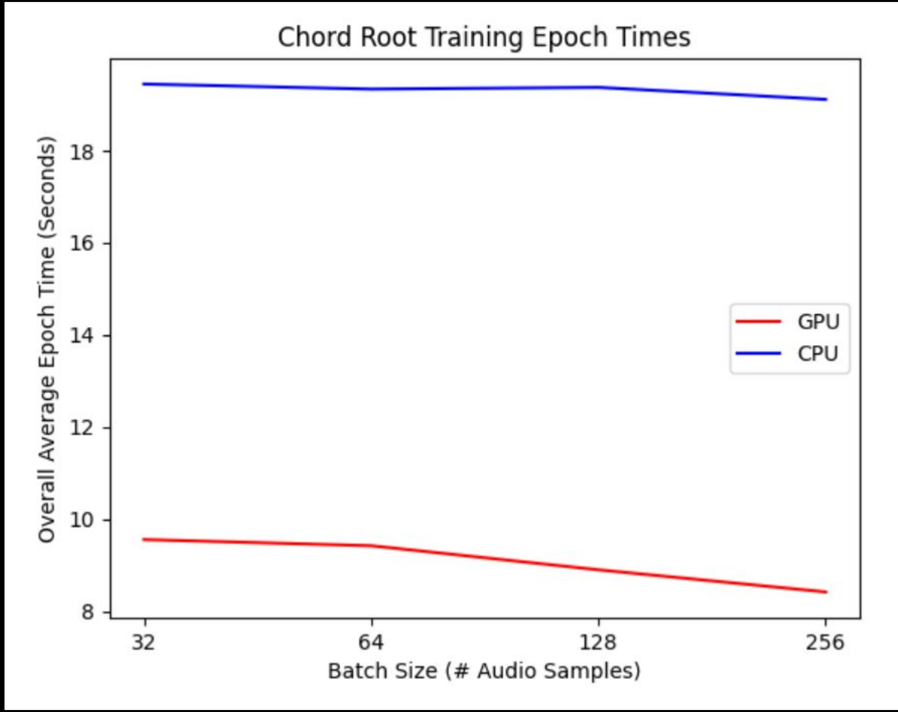
Batch Size 64 Average Epoch Times								
	Chord Root Classifier				Chord Type Classifier			
Trial / Device	GPU Train (s)	CPU Train (s)	GPU Test (s)	CPU Test (s)	GPU Train (s)	CPU Train (s)	GPU Test (s)	CPU Test (s)
1	9.3426	19.2622	3.9820	5.0799	9.3873	19.3707	4.0227	5.0211
2	9.4652	19.4598	3.8735	5.1335	9.4453	19.2382	4.0010	5.0659
3	9.4808	19.2686	4.0016	5.1122	9.3609	19.3606	3.9400	5.0377
4	9.4135	19.2974	3.8686	5.1410	9.2924	19.4411	3.9619	5.0512
5	9.3954	19.3615	3.9561	5.0386	9.3227	19.2523	4.0031	5.0265

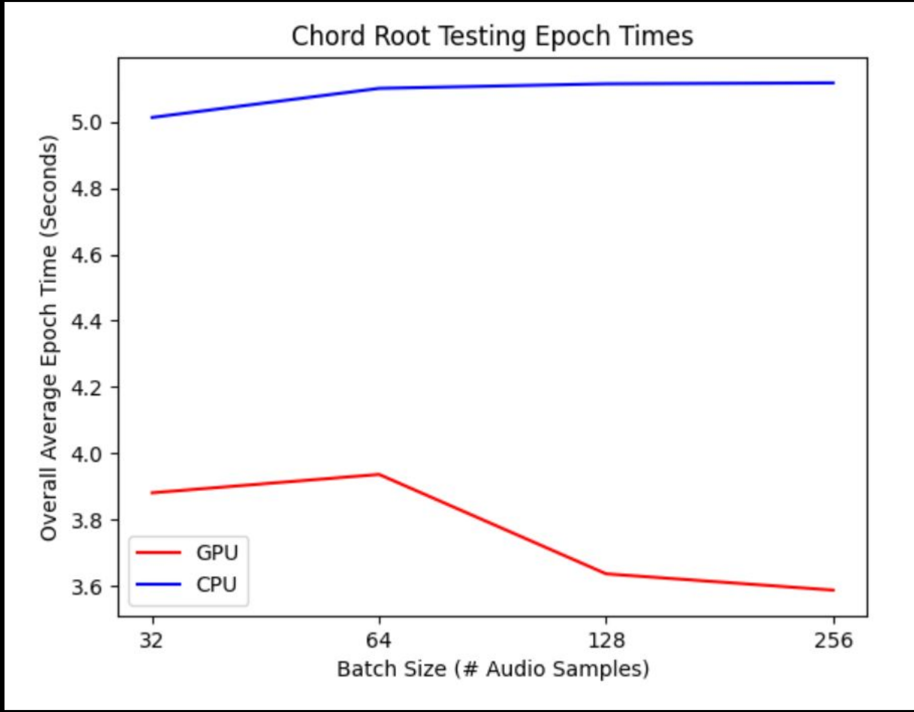
Benchmarking Results (Experiment 1)

Batch Size 128 Average Epoch Times								
	Chord Root Classifier				Chord Type Classifier			
Trial / Device	GPU Train (s)	CPU Train (s)	GPU Test (s)	CPU Test (s)	GPU Train (s)	CPU Train (s)	GPU Test (s)	CPU Test (s)
1	8.8637	19.3567	3.6741	5.1241	8.9303	19.2980	3.7188	5.1140
2	9.0130	19.3337	3.5284	5.1311	8.8706	19.3066	3.6195	5.1418
3	8.8835	19.4181	3.6180	5.1340	8.8877	19.3027	3.6815	5.0855
4	8.8126	19.3741	3.6497	5.0851	8.8785	19.2405	3.6815	5.0949
5	8.9125	19.3540	3.7119	5.0992	8.8966	19.3457	3.6770	5.1063

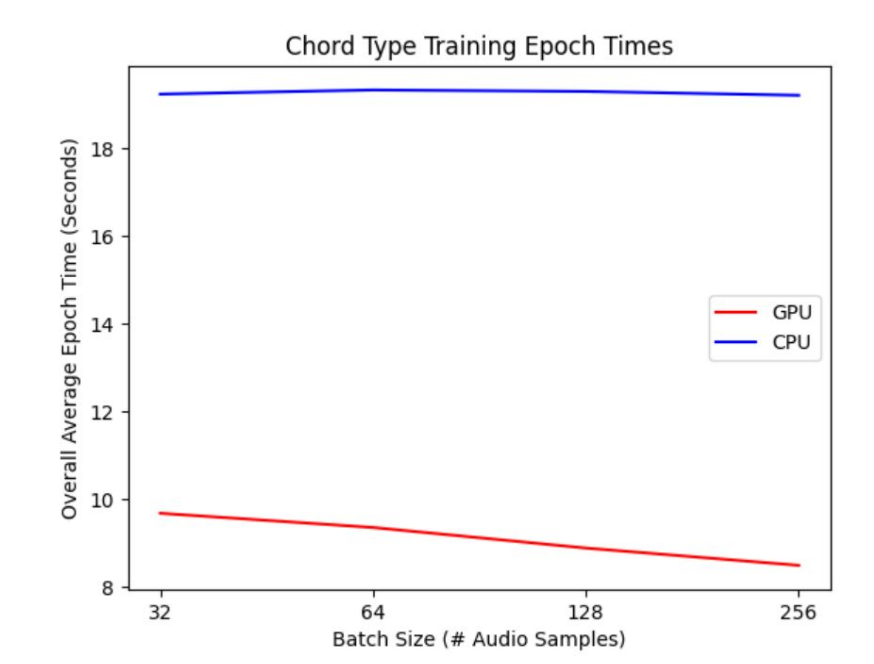
Batch Size 256 Average Epoch Times								
	Chord Root Classifier				Chord Type Classifier			
Trial / Device	GPU Train (s)	CPU Train (s)	GPU Test (s)	CPU Test (s)	GPU Train (s)	CPU Train (s)	GPU Test (s)	CPU Test (s)
1	8.4495	19.1556	3.5272	5.0889	8.5357	19.1858	3.5041	5.0630
2	8.4329	19.1454	3.5928	5.1609	8.5007	19.2503	3.4781	5.1120
3	8.3995	19.1334	3.6318	5.1417	8.5692	19.1820	3.4669	5.1077
4	8.4292	19.1246	3.5890	5.0729	8.4420	19.1983	3.5639	5.1607
5	8.3708	18.9724	3.5958	5.1233	8.4472	19.2372	3.4365	5.1247

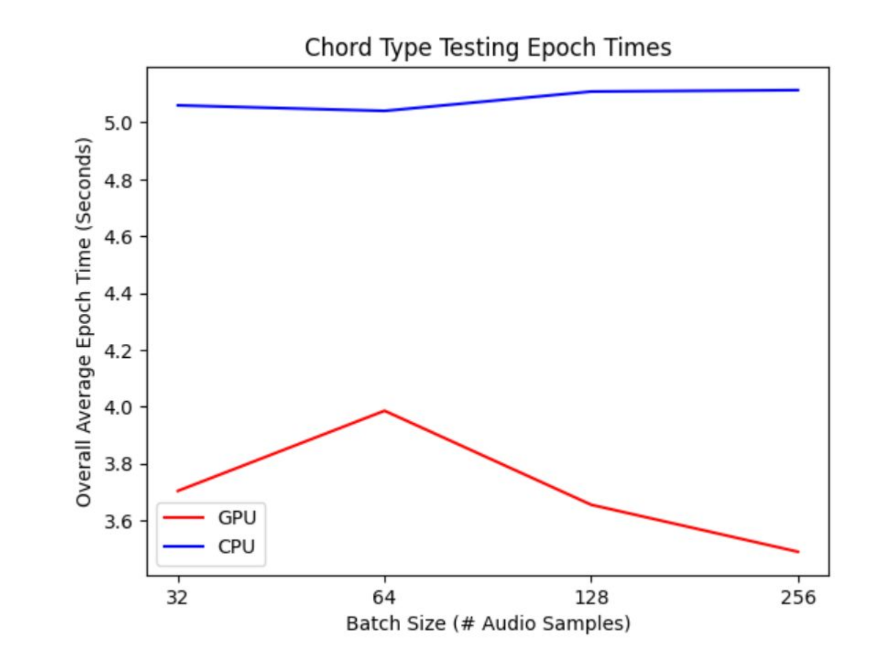
Benchmarking Results (Experiment 1)





Benchmarking Results (Experiment 1)





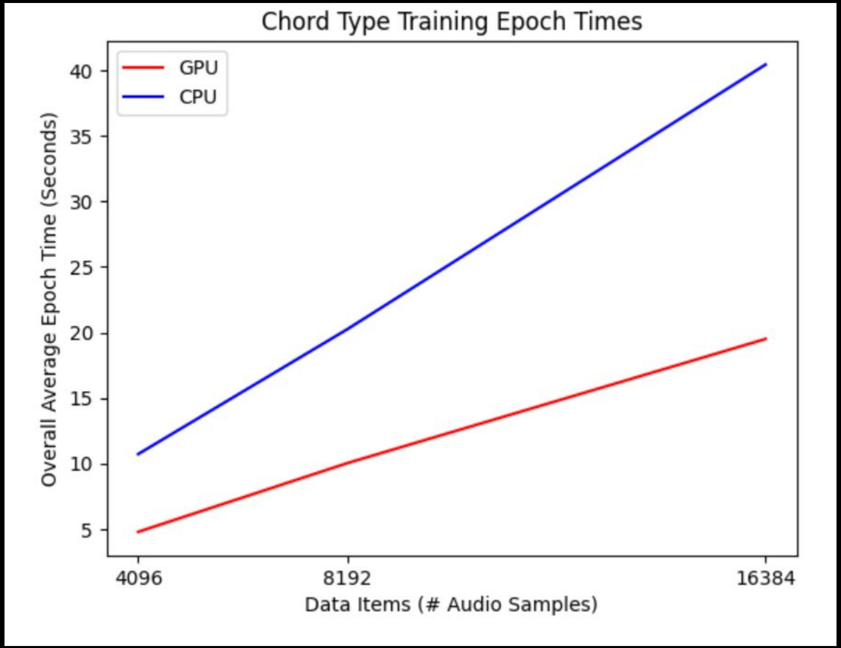
Benchmarking Results (Experiment 2)

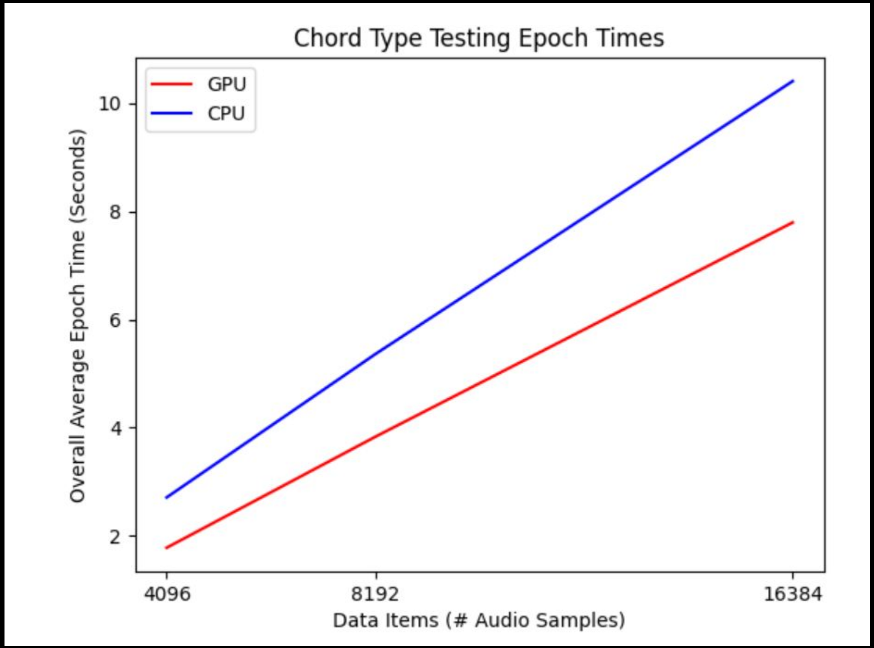
4096 Data Items Average Epoch Times				
	Chord Type Classifier			
Trial / Device	GPU Train (s)	CPU Train (s)	GPU Test (s)	CPU Test (s)
1	4.7119	10.4518	1.7478	2.6436
2	4.7665	10.6183	1.7800	2.7240
3	4.7991	11.2149	1.7798	2.7781
4	4.7946	10.9964	1.7931	2.8069
5	4.8317	10.3049	1.7938	2.5917

8192 Data Items Average Epoch Times				
	Chord Type Classifier			
Trial / Device	GPU Train (s)	CPU Train (s)	GPU Test (s)	CPU Test (s)
1	9.7752	19.6253	3.6991	5.2198
2	9.9045	20.2630	3.5383	5.4081
3	10.0638	20.5733	3.8818	5.4014
4	10.1760	20.6922	4.1350	5.4339
5	10.2084	20.0712	3.8994	5.3302

16384 Data Items Average Epoch Times				
	Chord Type Classifier			
Trial / Device	GPU Train (s)	CPU Train (s)	GPU Test (s)	CPU Test (s)
1	19.5169	40.4109	7.7995	10.4037
2	19.4677	40.3710	7.9375	10.5222
3	19.4545	40.5494	7.7952	10.3976
4	19.5734	40.3926	7.7440	10.3600
5	19.4931	40.4989	7.6865	10.3430

Benchmarking Results (Experiment 2)





Conclusions From Experiments

Experiment 1

- Training and testing time trends appear consistent between Chord Root and Chord Type models.
- CPU training and testing times stay pretty much the same when the batch size changes.
- GPU training time decreases slightly but noticeably with an increase in batch size.
  - Testing time decreases as well after a slight spike with a batch size of 64.
- CPU average training time remains approximately twice that of GPU average training time.
- CPU average testing time is approximately 1.25-1.45 times that of GPU average testing time depending on batch size.

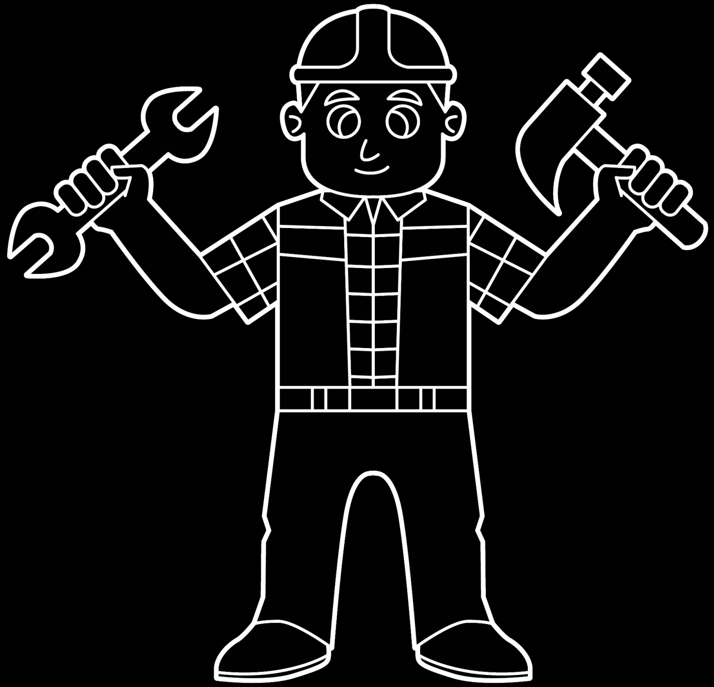
Experiment 2

- When the size of the dataset doubles:
  - CPU training and testing time approximately doubles
  - GPU training and testing time approximately doubles
- CPU average training time remains approximately twice that of GPU average training time.
- CPU average testing time is approximately 1.3-1.5 times that of GPU average testing time.

Building The Final Model

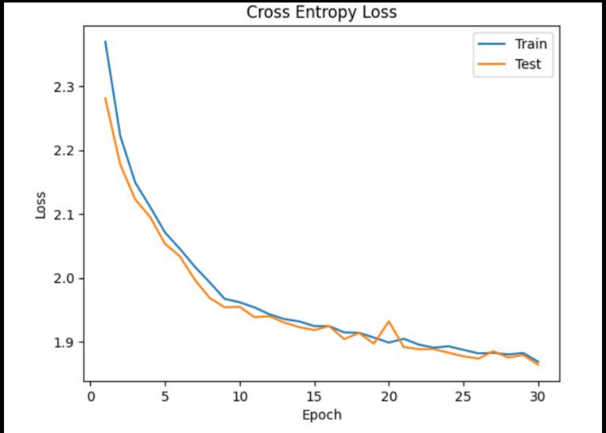
From our experiments, we discovered that using a GPU brought worthwhile execution time improvements to the training time for our classifier models.

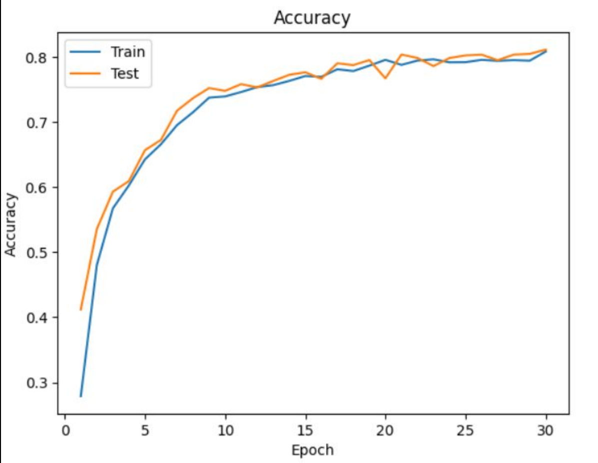
- GPU is better able to exploit the ability for tensor operations to be parallelized.
  - Appear to make up a significant portion of the training time.
- Final Model:
  - Built on GPU
  - Full Dataset
  - 30 Epochs
  - 32 Batch Size



Final Model Results

Chord Root Model



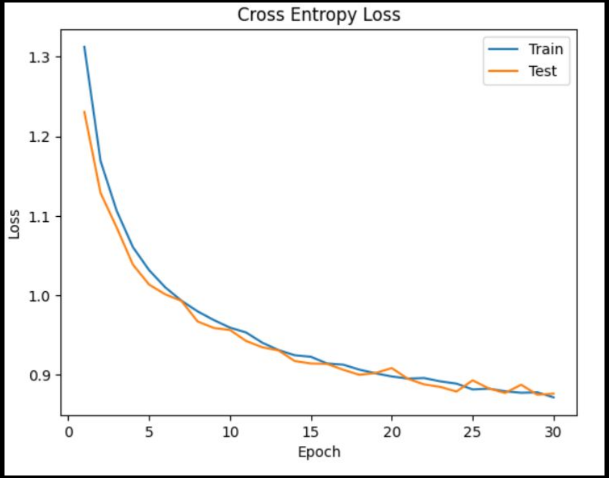


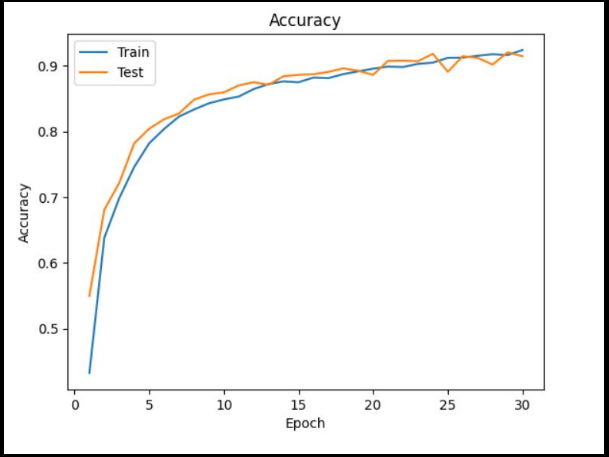
- Final Accuracy: ~81%
- Final Loss: ~1.87
- Total Training Time 7965.27 seconds
  - ~ 2.2 hours
- Average Epoch Time: 265.91 seconds
  - ~ 4.4 minutes

Final Model Results

Chord Type Model

- Final Accuracy: ~91%
- Final Loss: ~0.88
- Total Training Time: 8691.54 seconds
  - ~ 2.4 hours
- Average Epoch Time: 289.72 seconds
  - ~ 4.8 minutes





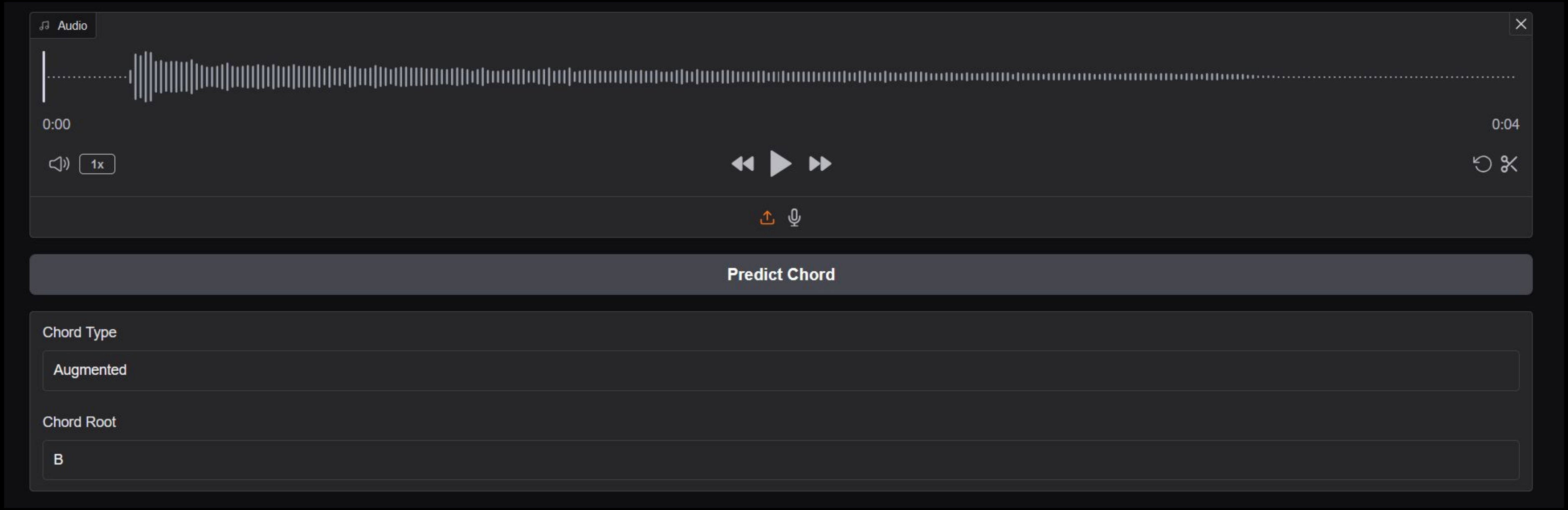
Expectations vs Actual Results

- Accuracy is close to what we expected.
- GPU speedup from CPU was not as large as we expected.
    - ~ 2X speedup instead of 5X
- There are a few possible explanations for why we did not experience the speedup we were anticipating.

Possible Reasons

- Trained on laptop GPU, not as powerful.
- Potential overhead from transferring data to GPU memory.
- Background applications taking away GPU processing power.
- Thermal limitations leading to performance throttling.

Final Gradio Application



References

- Slide 4
  - https://zenodo.org/records/4740877
- Slide 6
  - https://www.youtube.com/watch?v=SQ1iIKs190Q
- Slide 9
  - https://www.techpowerup.com/gpu-specs/geforce-rtx-2070-max-q.c3392

Pytorch Tutorials Consulted:

- https://www.youtube.com/playlist?list=PL-wATfeyAMNoirN4idjev6aRu8ISZYVWm
- https://www.youtube.com/watch?v=V_xro1bcAuA
- https://www.youtube.com/watch?v=dOG-HxpbMSw

Resources (Images)

- Slide 1
  - https://creazilla.com/media/vector/7870091/music-nots-background
- Slide 2
  - https://media.istockphoto.com/id/119121470/vector/love-music.jpg?s=612x612&w=0&k=20&c=JDEEnusEd2JwfCbTcL1zoHgGj9P7k4zRoai55Nutxbo=
- Slide 3
  - https://cdn.creazilla.com/silhouettes/7984606/musical-ear-type-ii-prismatic-silhouette-000000-md.png
  - https://openclipart.org/image/800px/298174
- Slide 11
  - https://classroomclipart.com/clipart-view/Clipart/Black_and_White_Clipart/Science/scientist_holding_beaker_jpg.htm
- Slide 19
  - https://freedesignfile.com/761791-bob-the-builder—bob-with-tools-drawing-black-and-white-clipart/