Classifying and Generating Instrument Sound Samples Using ResNet and DCGAN

Abstract

For this project we, Jesse Lunger and Cameron Rho, worked on using deep neural net techniques to classify audio and generate audio. We used a Residual network ResNet(50) and an adversarial generative network DCGAN. The purpose was to collect data on how changing different parameters changes efficiency within the network. We achieved moderate accuracy with our classifier, though we had less luck with our generative model; however, this work is mostly an informational piece about our findings using deep learning with PyTorch.

1. Introduction

Our goal for this project was to explore how we can use deep neural networks to perform useful operations on auditory data, with specific goals of classifying sound samples from different instrument categories and generating sound samples in a specific instrument category. Rather than focus on a finished product, we instead looked to tactics that allowed us to optimize the learning process. We believed that to truly understand the project, we needed to explore the performance characteristics of different models at a basic level and log our results.

Our objectives included:

1. Create a custom dataset and data loader that will convert audio files to MFCC's upon request.
2. Collect data from a range of different classifiers and GANS models to make an informed decision on which model would be the best option.
3. Create a DCGAN model to generate MFCC's from the original images of sound, and convert those back to wave files.
4. Design a powerful classifier model that can work with inputs of varying sizes.

1.1. Mel Spectrogram

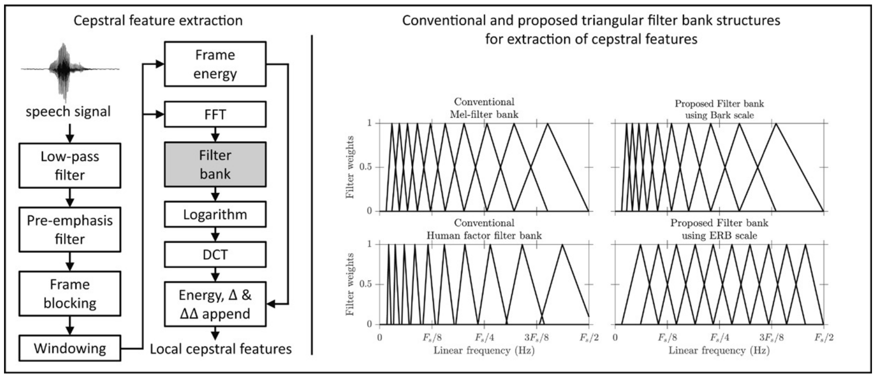
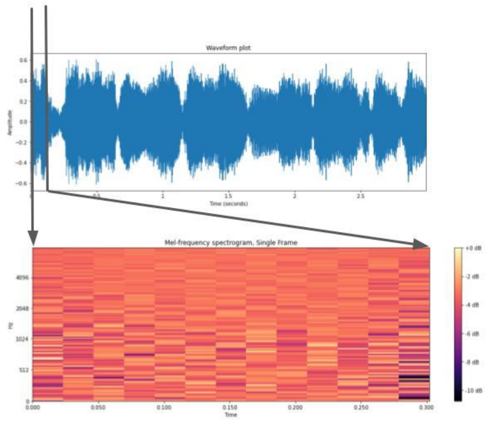

Cepstral feature extraction
speech signal
Low-pass filter
Pre-emphasis filter
Frame blocking
Windowing
Frame energy
FFT
Filter bank
Logarithm
DCT
Energy, Δ & ΔΔ append
Local cepstral features
Conventional and proposed triangular filter bank structures for extraction of cepstral features
Conventional Mel-filter bank
Proposed Filter bank using Bark scale
Conventional Human factor filter bank
Proposed Filter bank using ERB scale
Filter weights
Linear frequency (Hz)

Figure 1: Mel Spectrogram Cepstral Coefficient diagram.
(https://www.sciencedirect.com/science/article/abs/pii/S1051200420301081

A key part of sound recognition in modern sound classification is the use of Mel Spectrograms. It is used for two reasons: it converts audio wave files into two-dimensional arrays that can be used to train deep learning models and are mathematical representations of sound designed to accentuate the key frequencies humans hear. Their computation involves the following: Applying a high-frequency filter to amplify high-frequency sound as humans are better at distinguishing low frequencies. The sample, often 44100 samples/sec, is divided into overlapping frames. A Fourier transform is applied to each frame to convert signals from time domain to frequency domain. Signals are sent through a Mel filter which is a nonlinear scaling that mimics the way humans hear. A logarithm is applied to compress the dynamic range of the signal. A discrete Cosine Transform is applied to obtain the cepstral coefficients and each frame is normalized. There are many ways to represent a spectrogram, the one below is using Python's MatPlotLib, but there are many other ways to represent them.



F*igure 2: Waveform to MFCC segment*

1.2. Dataset

The dataset we used was the IRMAS dataset from Universitat Pompeu Fabra Barcelona, a compilation of musical audio excerpts, each annotated with the predominant audible instrument. The instruments include: human voice, clarinet, cello, saxophone, violin, flute, organ, piano, acoustic guitar, electric guitar, and trumpet. There are further subdivisions within the dataset, but those were not part of our focus.

The dataset presented a challenge as it had a huge variance of sound and unequal numbers of samples for each of its classes, with the largest difference in size being roughly 30%. It could have up to 5 overlapping instruments per sample. The entire IRMAS dataset was not necessary for our project purposes and so we only used the training portion. We partitioned it: 72% training, 18% validation, and 10% testing.

1.3. Data Manipulation

Converting a wave file to a Mel Spectrogram was very costly, this is even more so if the user wishes to have a high amount of MFCC features. Converting all data (7 GB) into MFCC upon each use was both not feasible and went against our other goal of using limited compute to optimize hyperparameters. We wanted a dataset that would lazily evaluate this upon request. We also did not want the model to train over an entire three-second sound clip that would lead to overfitting. From our research, the most optimal time segment for sound classification was 20-500 milliseconds. This meant that the dataset could not be subdivided purely by indexing, see figure 3. Our solution was to use three queues: training, validation, and testing. These queues feed into three segment pools that consist of training, validation, and testing segments. When the data loader requests an item, depending on its index it will pull from one of those three pools, if a pool is empty, the queue that feeds it will be activated to break a wave file into the designated amount of MFCC segments.

We also needed a system that could give us some kind of feedback that it was learning correctly within a reasonable amount of time. We need to further partition our data loaders beyond just training, validation, and testing. To do this we used a function that would take our custom dataset, clear its existing pools, reset its iterators, and then make copies of the dataset while readjusting the iterators. The reason for this is that when a dataloader requests information from a dataset, it copies its current state. So any object that is mutable within that dataset will not change between uses.

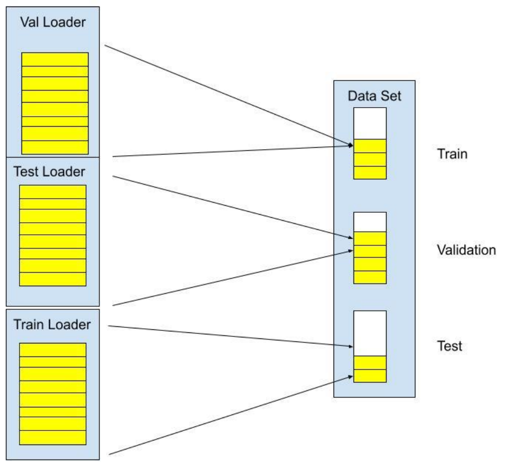


Figure 3: Training-Validation-Testing Split Visualization.

2. Details of the Approach

We constructed three different models, a Linear, CNN, and ResNet(50) model, and measured their performance over five partitions that made up ¼ of our data.

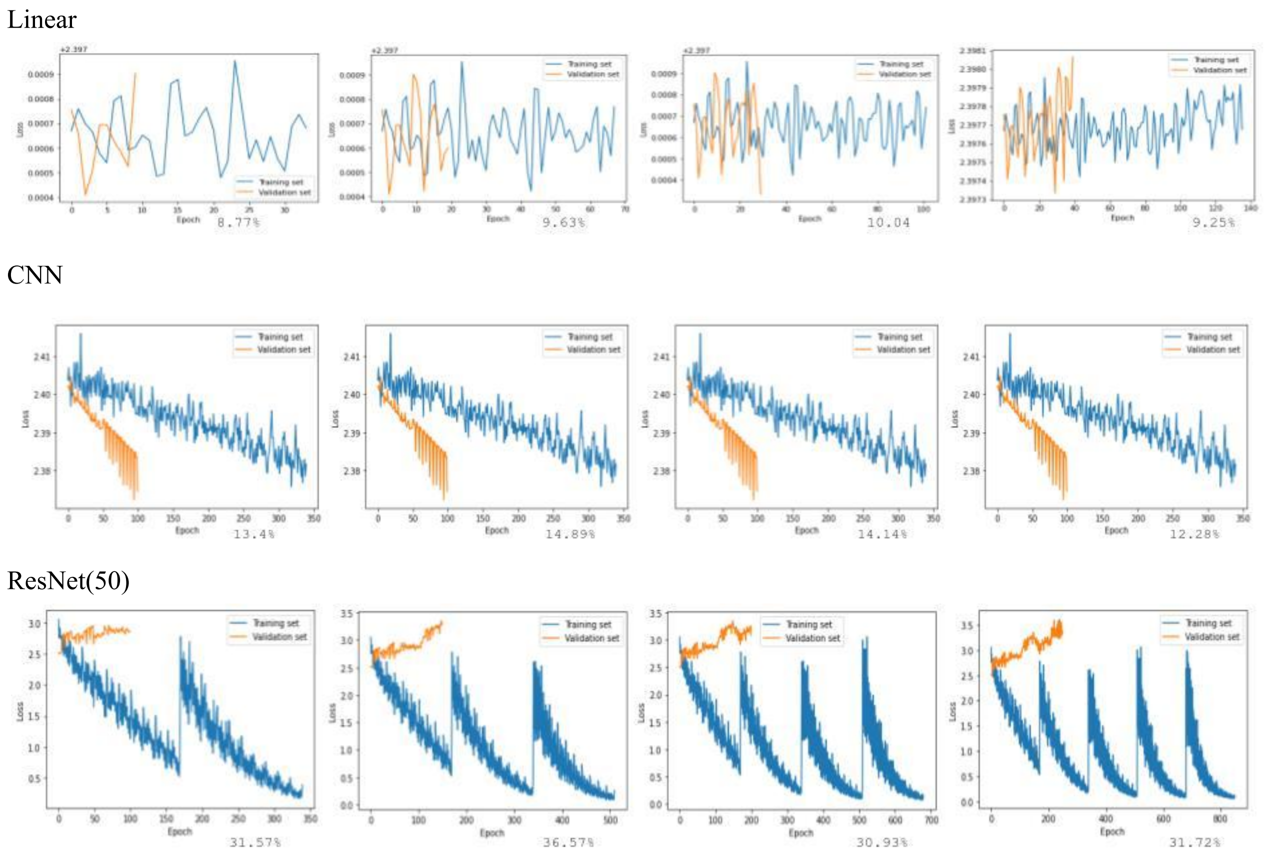


Figure 4: Training and validation loss plots for each type of model we implemented.

It was clear the Linear net was not learning, and its predictions were random. The CNN kept the validation loss low, but its accuracy was unremarkable. What caught our eye was the performance of the ResNet which after training on

1/10 of our data had greater than 30% accuracy, its downside was the validation loss clearly showed it was overfitting rapidly.

2.1. Classification Model ResNet

Once we had decided on a ResNet we needed one that could work with many different shapes. We used an adaptation of Nauman's "how to write a ResNet from scratch" tutorial to build our ResNet, but gave it average pooling so that it could process any 2d matrix. This fed into our primary objective of allowing for parameters involving the shape of our data to be altered without problems during the classification process. Batch size and segmentation were dependent on each other. Without segmentation that batchsize would need to be reduced to 60, with data segmented into 1/10 size pieces, batch size could be increased to 512. We used Pytorch's SGD optimizer (weight decay= .001, momentum= .09), and cross-entropy loss. For the structure of our model please refer to figure 5.

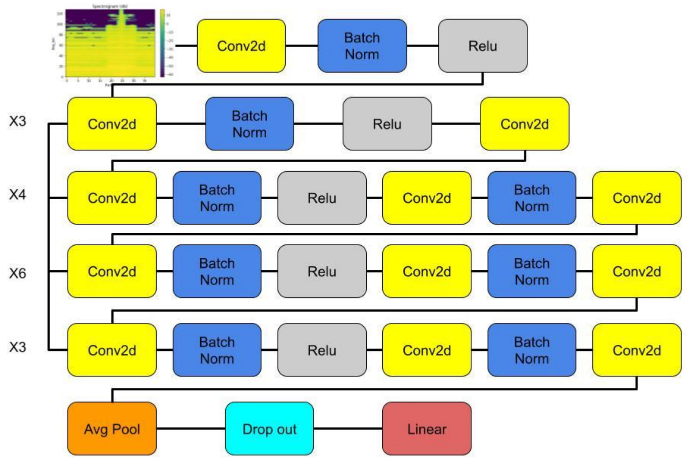


Figure 5: ResNet Classification Model Structure.

Unfortunately, this ResNet was not doing much better than the previous one. On average it could get 37% accuracy after 5 epochs of training on all our training data, but past that and it would start to overfit. However, we had one last trick up our sleeve, the ResNet was training on a segmented dataset. Our thought process was that if we could process a segment of 1/10th the sample size at 37% then surely we could recombine the segments to get a more accurate prediction. We did this by using a softmax probability distribution from the output of each segment and used that to build a secondary image. We could have chosen to sum the probabilities and take the max, but we were curious what would happen if we gave those probabilities to a second ResNet to see if there would be any benefit. We were able to boost our accuracy by an additional 10%, resulting in 47% accuracy. We expected a greater yield but ultimately the problem was again overfitting. The model was focusing on some classes while ignoring others.

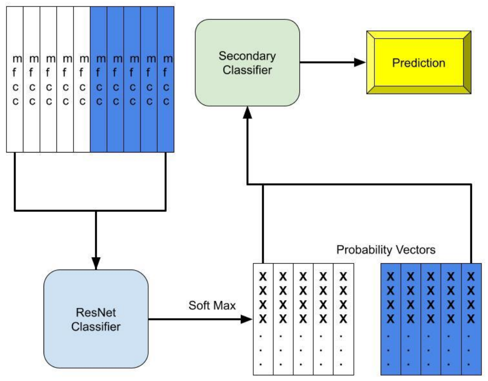


Figure 6: Dual Model with Softmax Structure

2.2. DCGAN Model

We also created a DCGAN to generate audio samples in specific instrument categories. We modified the data loaders to only contain samples of violin audio. We then used this data to train a DCGAN operating with 128x128 MFCC segments in batches of 50. The generator and discriminator were composed of 6 convolutional layers, with batch norm and ReLU activation for the generator and batch norm and SELU activation for the discriminator. BCELoss was used for the loss function, and an Adam optimizer was applied to the model with a beta1 value of 0.5. The model was trained for 20 epochs with a learning rate of 0.001.

We started out using leaky ReLU activation in the discriminator, but we switched to using SELU on a recommendation from Steven Walton. Additionally, we initially tried training the DCGAN on segments similar to the ones we used in the ResNet. However, this produced such short audio samples that we switched to using full audio frames for the input data.

3. Results and Discussion

We were able to establish that epochs greater than thirty in almost all models lead to overfitting. A learning rate of .001 had universally better results in each of our test cases. In approximately 90% of our testing, the datasets that used segmentation produced models that outperformed or did equally as well as models without segmented data. Dropout helped with some models like the CNN, but it could not prevent the Resnet from overfitting unless it was taken to such a high level that it started losing accuracy. We found that the optimal dropout was (.5). In general having a high number of features in our spectrograms never hurt our performance, but beyond thirty it didn't greatly change our results. This may change with larger datasets with more labels.

3.1. ResNet Results

With the dual ResNets, 47% accuracy was disappointing, but for samples that have multiple classifications and a model that only classified a single instrument it wasn't unexpected. We believe that if we used softmax on our model's final outputs and took the labels with the highest values, we would have achieved a much higher level of classification. With that said, showing that we could lower the validation loss for at least 10 epochs will be necessary if we were going to draw real conclusions.

As far as segmentation, there was a noticeable difference between samples segmented at 300ms and 500ms, with 500ms often leading to a loss of about 5% in accuracy. Similarly, below 150 ms we also noticed a loss in accuracy. This is probably due to the nature of an instrument not being a continuous input but simultaneously having some fluctuation.

Shown in figure 7 is the training and validation loss of a nonsegmented audio model trained on the entire dataset using a ResNet(50). This is a common problem that has happened with non segmented data. The model begins to immediately overfit at extreme levels, resulting in 15.67% accuracy. Our reasoning for this was that our samples did not have a clear end or beginning, and each sample was very unique, which was a perfect recipe for overfitting.

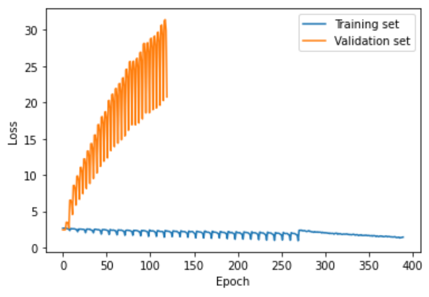


Figure 7: Nonsegmented audio data

In contrast, shown in figure 8 is the model in which each sample is cut into 1/10 its duration. While overfitting does start to occur at 10 epochs, it remains relatively level. The results had an accuracy of 37.65% and then after the secondary layer, 47.16%.

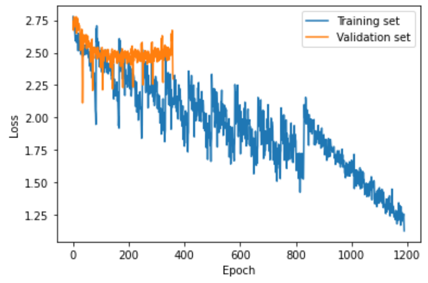


Figure 8: Segmented audio data

3.2. DCGAN Results

We were unable to produce satisfying or meaningful audio samples with our DCGAN. Part of this was due to the compression that occurs when audio waveforms are converted to MFCC. The conversion back to waveform is largely unsuccessful, as much of the data necessary for audible sounds are lost. That being said, our results were discouraging even disregarding the actual audio produced. Shown in figure 9 is the training loss from the generator and the discriminator.

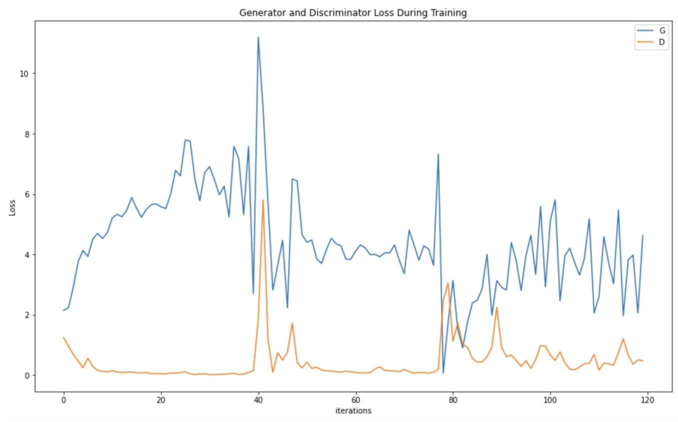


Figure 9: Training loss from the generator and the discriminator

Furthermore, when examining the spectrograms of the produced audio samples and comparing them to the spectrograms of the audio input, it was clear that what our model produced was truly static noise. The spectrograms visualizing the produced audio samples can be seen in figures 10 and 11.

One of the factors contributing to the poor generation of audio may have been an imbalance in the learning between the generator and the discriminator. The loss from the discriminator is consistently lower than that of the generator, indicating that the discriminator was quickly able to tell that all the output from the generator was fake, giving the generator very little information to learn from. In future work on this model, I would try to use regularization to slow down the learning of the discriminator; hopefully, this would allow the generator to keep up with the discriminator.

Another factor that contributed to the model's failure could be the format of the data we used. While musical audio samples provide a great opportunity to create a robust model that contains complex information about the sound produced by different instruments in a real musical performance setting, the added complexity would certainly make it more difficult for the model to learn the basic features of the instrument's sound profile. It would most likely be beneficial to gather new data containing single sustained notes from different instruments to simplify the information the model needs to learn.

A sample of the audio produced by our model can be found at the following location:
https://drive.google.com/file/d/1E2NHojTS16EPgttndm-I7tM3E_bjXIB1/view

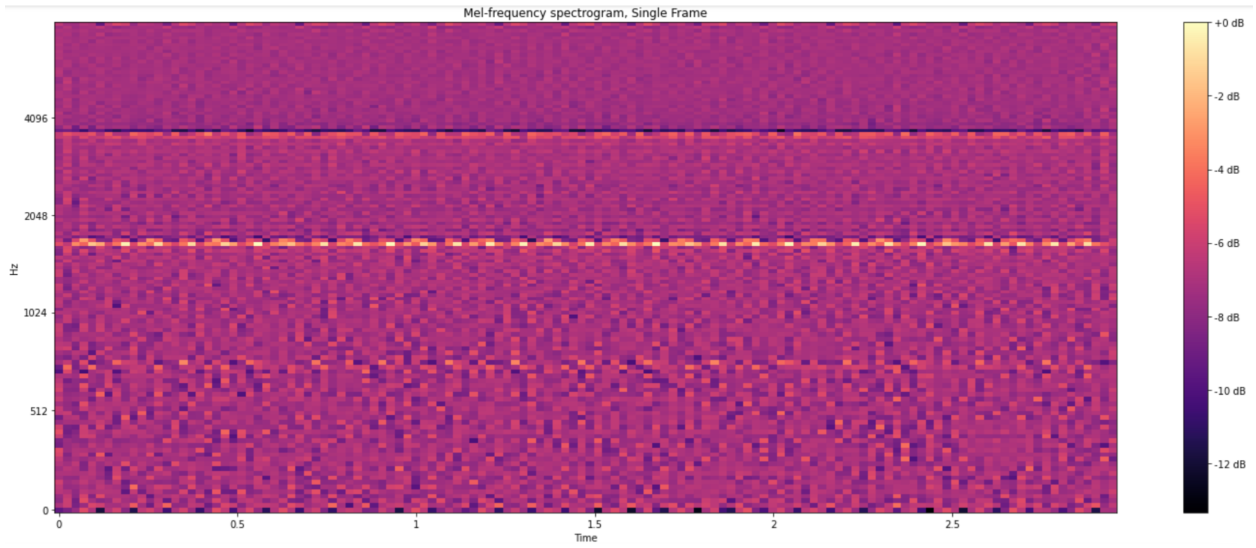


Figure 10: Mel-frequency spectrogram of an audio sample generated by the DCGAN generator.

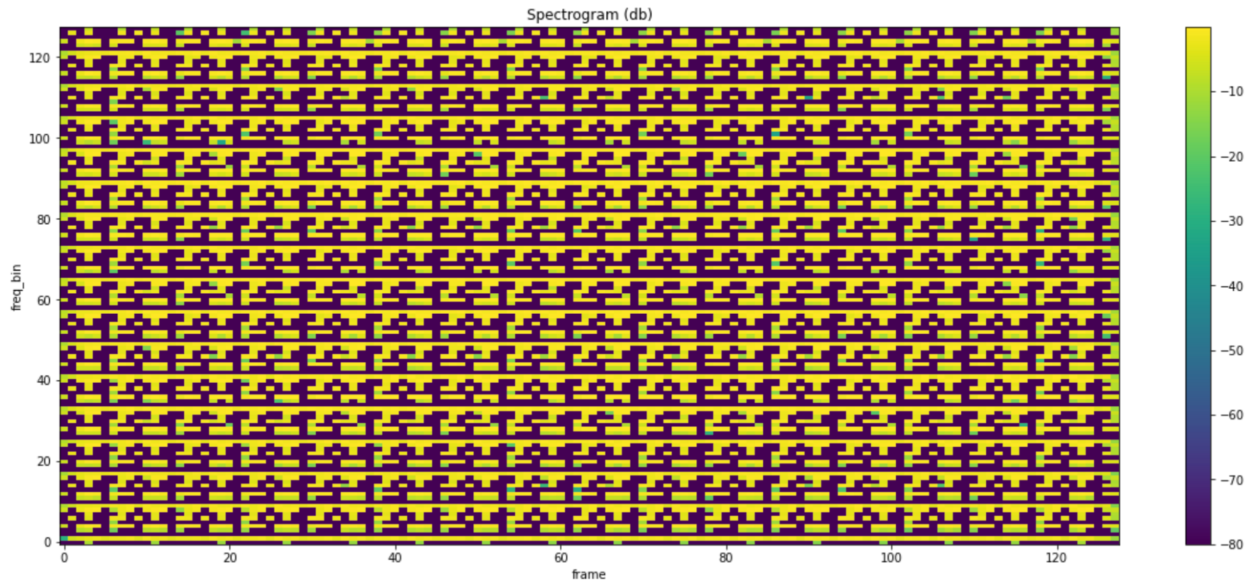


Figure 11: Working spectrogram of the same audio sample shown in Figure 10; This image is more interpretable than that shown in Figure 1, and it is apparent that the sound is chaotic and noisy.

4. Conclusion

For our project we achieved a lot in terms of collecting meaningful data for sound classification. We were able to build a system that could be rapidly tuned and changed, which we utilized to get the performance characteristics of many different models. We learned a lot about the intersection of deep learning and sound recognition, and track our progress to present a meaningful report. And having the opportunity to work with data that could be heavily altered to change the effectiveness of deep learning techniques.

For the sound classification I, Jesse, was not happy with the performance of the ResNet model. I expected to attain a minimum of 90% accuracy after the time and effort I had put into building the system. Overall working with audio presented far more challenges than I expected. There are so many hyperparameters that can be changed which can completely change a model’s performance characteristics or do nothing. It was very overwhelming. Google Colabs was both a blessing and a curse. It gave us the capability of running our code using a high-end GPU which my laptop did not have, but if we let our string idle for longer than 10 minutes, Collabs would discontinue the training. This made training models very stressful and time-consuming.

As for the sound generation, we had lower expectations for the performance of the DCGAN model. While it would have been satisfying to produce a clear sample of violin music, the results we achieved were acceptable with the resources immediately available to us. We learned much about the nature of audio data and how it differs from image data, and even though the samples we produced sounded like static white noise, it was still fun to hear the sounds that we produced.

5. Statement of Individual Contribution

Jesse Lunger: I worked on several components including finding the IRMAS dataset and creating a custom dataset and data loader to manipulate the data. Researched ways to extract MFCCs from wave files and how to split them efficiently. Built the classification models we used to collect our data and completed our final dual ResNet classifier.

Cameron Rho: My main contribution to the project was the creation and analysis of the DCGAN model we used to generate audio samples. I composed the sections of this report that concern the DCGAN model and the slides in the in-class presentation that concerned the DCGAN model.

References

[1] Bosch, J. J., Fuhrmann, F., & Herrera, P. (2014). IRMAS: a dataset for instrument recognition in musical audio signals (1.0) [Data set]. 13th International Society for Music Information Retrieval Conference (ISMIR 2012), Porto, Portugal. Zenodo. https://doi.org/10.5281/zenodo.1290750

[2] Bosch, J. J., Janer, J., Fuhrmann, F., & Herrera, P. “A Comparison of Sound Segregation Techniques for Predominant Instrument Recognition in Musical Audio Signals”, in Proc. ISMIR (pp. 559-564), 2012

[3] Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun. "Deep Residual Learning for Image Recognition." Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016.

[4] Nouman. (2018, August 20). Writing ResNet From Scratch in PyTorch. Paperspace Blog. Retrieved from https://blog.paperspace.com/writing-resnet-from-scratch-in-pytorch/