ENSC 429

Sampling Vocals Using Deep Learning Methods to Isolate Human Voices

Darren Fok 301461164

Chris Sim 301448214

Tony Chen 301458138

Nicholas Lagasse 301451354

Group 3

Table of Contents

Abstract

In music production, sampling is the process of isolating certain audio clips such as vocals or instrumentals from a previously made piece of music to use a new track. In certain scenarios, a producer may want to obtain a clean recording of a person's voice with no backing instrumentals. For this purpose, we present a vocal isolation program that removes backing instrumentals and other noise from an .wav audio clip. Using a U-Net convolution neural network that we trained with various songs along with their acapella variants, the total system produces a model that can attenuate non-human vocals in each signal. Although the final result is not effective to use as an actual tool, it provides a good starting point to further iterate on.

Introduction

Although sophisticated stem separation technology already exists within the online space, there is much knowledge to be gained by recreating the technology with digital signal processing techniques and machine learning.

For the purposes of regression, we trained the model on eighty different songs along with their acapella counterparts to help the model differentiate between vocals and instrumentals. One of the main challenges was deciding the way to store the spectrogram dataset objects, as different formats varied the results of training with some yielding better results than others. Initially, we stored frequency data of our sample audio files in .xlsx files with separate sheets for the Real and Imaginary components, but we later switched to .npy array files for reading and writing efficiency. For the model itself, we found it useful to edit certain parameters within the training block itself such as the size of the convolutional blocks to minimize distortion and clarity of the final audio file. We made extensive use of many different scientific and numeric python packages in this project such as Numpy, Librosa and PyTorch for various applications such as the Convolutional Neural Network model and transform scripts.

Although somewhat successful, more work and research are needed to fully develop a model that can be used in a production setting. This could be implemented through a more sophisticated Neural Network or simply through more rigorous training methods and/or improved data formats.

Methodology and Analysis

Methodology

Throughout the implementation process, various setups were implemented and trialed, with any errors or unexpected results prompting a re-design in our methodology.

The first design implementation included the use of data in a real and imaginary format, each on a sheet within a .xlsx file. This implementation took noticeably longer, with it taking 4 hours to complete 20 epochs of training, and the resulting model landing at a SDR of -33 dB, with an output audio file that contained a lot of noise.

After further research, it was determined that .xlsx files are not optimal for training datasets. This is due to the metadata, formatting and rounding that .xlsx files contain, and consequently, the storage size of the format [1]. Rather than maintaining this format, the training data files were switched over to a ".npy" format. These ".npy" files are binary-formatted files that contain arrays, as well as any nested arrays, that is native to the NumPy library. As such, it is optimized for it, allowing for faster reading and writing [2].

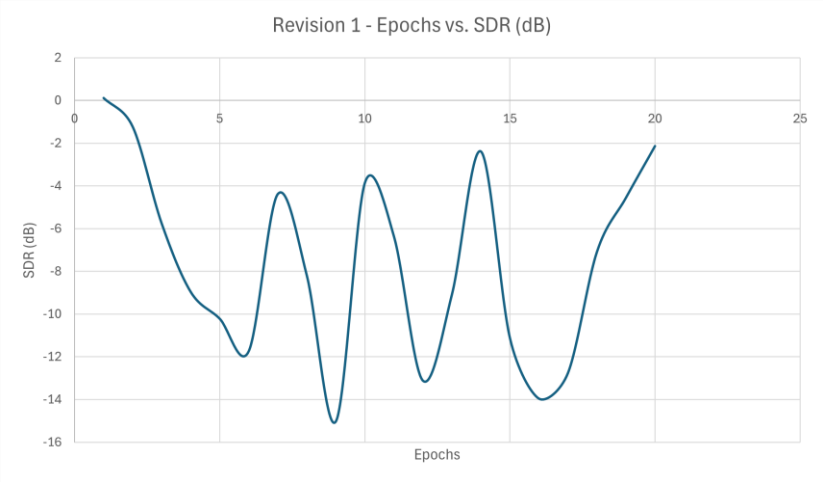

Figure 1: Results of training on .npy files containing real and imaginary values

However, even after the file formatting change, the model still was not performing as well as desired. Although the training process sped up substantially, it continued to produce negative SDR values within each epoch, all of which were not stable, and fluctuated greatly. Thus, it was deduced that the format of the file may have something to do with the model's poor performance.

After a further round of research, it was decided upon to format the data by taking the magnitude of the waveform and then taking the logarithm of the magnitude. Although the real and imaginary format is valid, taking the logarithm magnitude of the waveform allows

for a more compact file, as it simplifies to one impact channel, rather than two [3], acts as a dynamic compressor that can boost representation of smaller values that contain crucial information [4], and is known to boost the training performance of neural networks [5]. Furthermore, well-known vocal isolator programs containing neural networks, such as Spleeter, also utilize the logarithm magnitude format [6].

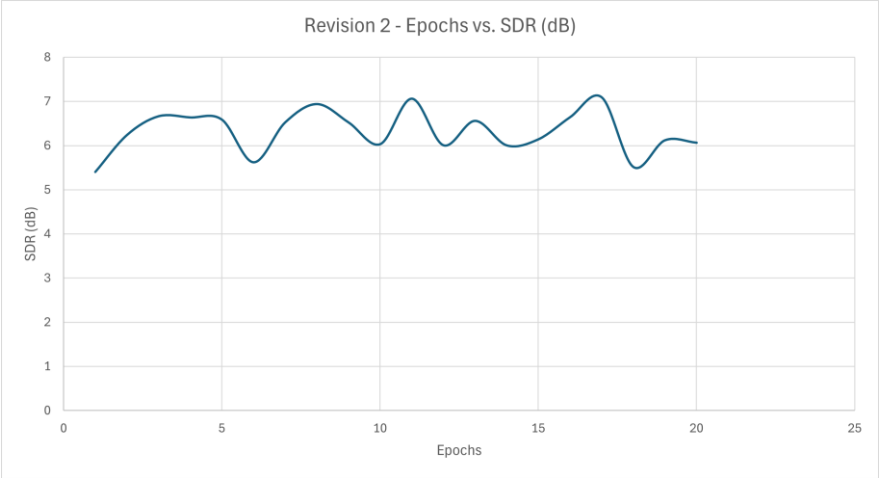

Figure 2: Results of training on .npy files containing log-mag of the waveform

As seen in figure 2, the change resulted in not only a more consistent SDR across all epochs, but most importantly, a consistent positive SDR. This resulted in an audio file that was recognizable from the original audio file, although with slightly distorted instrumentals, and intact vocals.

Implementations

We utilized a U-Net architecture designed for spectrogram-based vocal isolation, implemented with the use of the PyTorch library. This network consists of four encoder blocks and four decoder blocks, with skip connections between corresponding levels. Each of the blocks contains two convolutional 2D layers, each with batch normalization and ReLU activations. The encoder uses max-pooling to down-sample, while the decoder uses transposed convolutions to up-sample. All this together results in a 1x1 convolution which maps to a single-channel output.

To use a Convolutional Neural Network to separate vocals from unwanted noise, we needed a dataset object for the model to be trained on. Because we are attempting to generate a new signal, we must be able to gather both the frequency and time information of our data. Since the regular Fourier Transform only contains frequency information, we must use the **Short-Time Fourier Transform**.

$$STFT\{x[n]\} = \sum_{n=0}^{N-1} x[n]w[n-m]e^{-i\omega n}$$

The Short Time Fourier Transform is similar to the original Fourier transform, however it also contains a window function ω. This window function segments the time signal into chunks, and then applies the Fourier Transform onto each individual portion [7]. Doing this, we can get the frequency spectrum of the signal at certain points in time, giving us a two-dimensional piece of information we can then use to train our model.

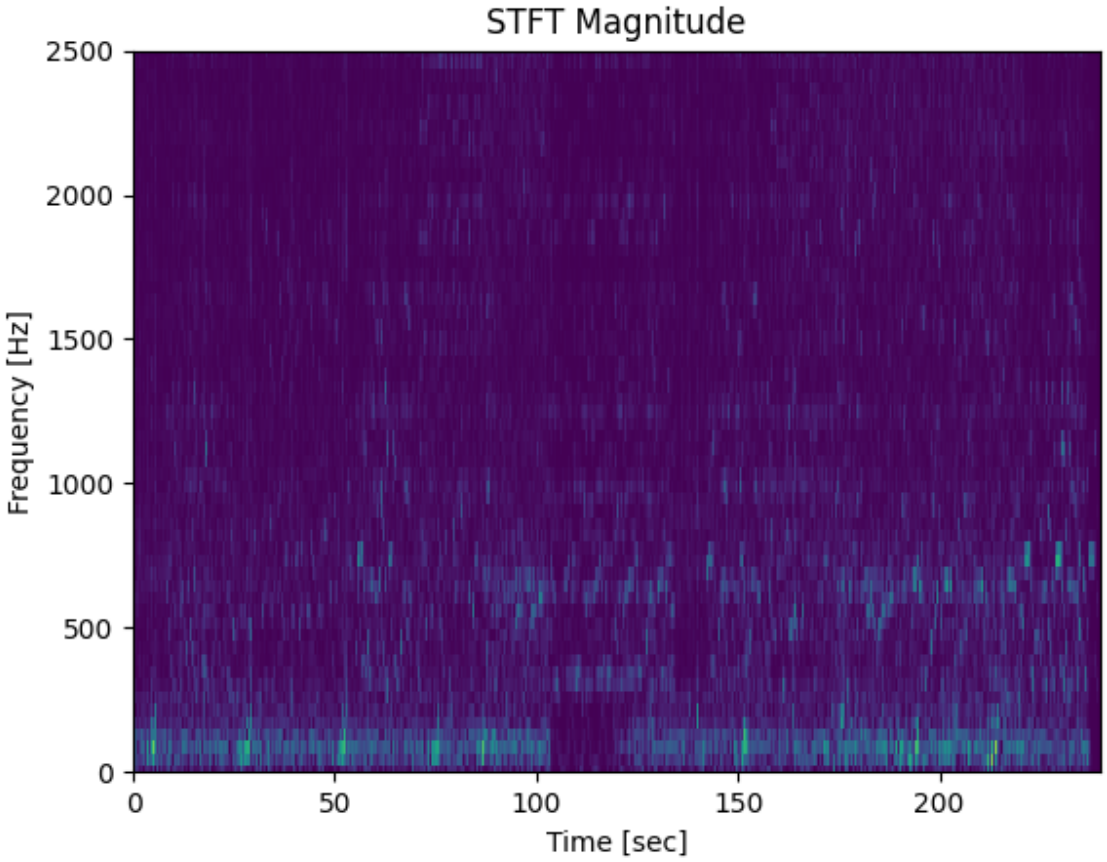

Figure 3: Matplotlib Spectrogram of Starlight by Muse

Before being fed into the U-Net model, the audio files are converted to log-magnitude spectrograms via a Short-Time Fourier Transform from the Librosa library, with parameters of "n_fft" = 1024, and "hop_length" = 512. The phase is discarded during training and then estimated using the Griffin-Lim algorithm during reconstruction. The model inputs are 2D log-magnitude spectrograms of mixed audio, with the corresponding vocal-only spectrogram being a target input.

As for the loss function, a simple L1 loss method is used between the predicted and ground truth isolated vocal spectrogram. Experiments with composite losses containing L1 and SDR were conducted, but it was found that pure L1 loss produced stable training with more desired results. As a result, SDR is not involved in the loss calculation, and is treated simply as a metric in validation.

In the training loop, each epoch has a training phase where it goes through forward and backward passes, followed by a validation phase where loss, **mean absolute error (MAE)**, and SDR are used as metrics. When looking for the best performing model, SDR was used as the primary metric. All training was performed on a single GPU.

Results and Discussion

```
20.20000000000003
Validation: Loss = 0.0903, MAE = 0.0903, SDR = 6.4394 dB
Saved prediction to predicted_epoch_19.npy
Training complete, model saved at vocal_isolator.pth
```

Figure 4: Final Epoch Results of training

After the final epoch of training on the latest iteration of the project, we found that the Loss and Mean Absolute Error were 0.0903, while the Signal Distortion Ratio settled at 6.4394 dB. Compared to the earlier model, this one had a significantly better SDR, as the previous oscillated wildly and settled at -2 dB, implying a high noise factor.

Unfortunately, even though the model performed better, it still had many issues with vocal isolation. When running an audio file through, the lower frequency instrumentals do get attenuated somewhat via muffling, but higher frequency sounds such as guitar riffs are still mostly present.

Although the progress of the project was promising, it was clear that the final result was not up to par for someone who would want to use it for its intended purpose. Given more time and resources, a more usable model would be feasible.

One of the reasons that the model was unsatisfactory could be because of lack of training data. As mentioned before, to train the model we used about 80 songs along with their acapella counterparts. During training, we saw that the performance of the model began to plateau as more epochs ran. Running additional epochs could possibly improve the model, but the plateau was likely due to a lack of training data to go off of. Sourcing more music and acapella tracks could alleviate this issue.

However, one of the reasons this was not possible in our timeframe was due to the sheer time investment needed to train the model. Our group was limited by our hardware, and training often led to crashes and unexpected hangs.

Another possible improvement would be to create a more complex loss function. Currently, we use a simple L1 loss function procedure. It may be beneficial to use an L2 loss function, otherwise known as a Mean Square Error function as it amplifies the presence of outliers more than L1.

$$MSE = \frac{1}{n}(\sum y_i - \bar{y})^2$$

This would ultimately mean that outliers are more heavily emphasized when taking the gradient, which could lead to a better performing model.

Finally, we could also attempt to find a different way to store our spectrograms. Due to the complexity of having multiple channels when storing our spectrograms in rectangular complex form, we opted to use the Log Magnitude representation of our short time Fourier transformed audio files. This form of transform resulted in only one data channel needed, but it also does not save the phase of the complex number. To alleviate this, we used the Griffin Lim algorithm to predict the phase, but it may have been less effective than saving the phase directly and training the model to output a phase based on the input.

Conclusion

This project demonstrates a ground-up development of a Deep Learning regression model for the purpose of isolating vocals from an audio track. While the model ultimately did not meet the standards expected by the development team, it showed great progress in approaching a point where it could be useable in a music production setting. Certain additions and changes such as a larger dataset, a more detailed loss function, and a more accurate method of storing our audio spectrograms could help push towards our goals.

Ultimately, there are still options for improving our model going forward, although some would require more work and resources than others. To fulfill our project objectives completely, we could also look towards open-source audio isolation programs already on the internet. There are pre-trained models present online along with more sophisticated code and neural network training blocks to use as references.

Individual Contributions

Below is a table of contributions by each group member.

Darren Fok	Development of Transform/Spectrogram/Utility Scripts Development of U-Net and overview of other notebook functions Training of Model Training Model Revisions Proposal Write-up Final Report Write-up
Tony Chen	Dataset Sourcing Development of Training Block Training of Model Training Model Revisions Assembly of total program pipeline
Nicholas Lagasse	Dataset Sourcing Development of Train Main Function Making of Presentation Slides
Chris Sim	Development of Transform/Spectrogram/Utility Scripts Development of Spectrogram Dataset Object Proposal Write-up Final Report Write-up Demo Recording

References

[1]
A. Daneswara, “Using NPY in NumPy instead of CSV,” Analyticbox. Accessed: Aug. 04, 2025. [Online]. Available: https://medium.com/analyticbox/using-npy-in-numpy-instead-of-csv-f53865320687

[2]
T. Sarkar, “Why You Should Start Using .npy Files More Often,” KDnuggets. Accessed: Aug. 04, 2025. [Online]. Available: https://www.kdnuggets.com/why-you-should-start-using-npy-files-more-often

[3]
S. Gul, M. S. Khan, and M. Fazeel, “Single channel speech enhancement by colored spectrograms,” Oct. 26, 2023, *arXiv*: arXiv:2310.17142. doi: 10.48550/arXiv.2310.17142.

[4]
“Spoken Digit Recognition with Custom Log Spectrogram Layer and Deep Learning - MATLAB & Simulink.” Accessed: Aug. 04, 2025. [Online]. Available: https://www.mathworks.com/help/signal/ug/spoken-digit-recognition-with-custom-log-spectrogram-layer-and-deep-learning.html

[5]
K. W. Cheuk, K. Agres, and D. Herremans, “The impact of Audio input representations on neural network based music transcription,” July 21, 2020, *arXiv*: arXiv:2001.09989. doi: 10.48550/arXiv.2001.09989.

[6]
A. Défossez, N. Usunier, L. Bottou, and F. Bach, “Music Source Separation in the Waveform Domain,” Apr. 29, 2021, *arXiv*: arXiv:1911.13254. doi: 10.48550/arXiv.1911.13254.

[7]
J. Allen, “Short term spectral analysis, synthesis, and modification by discrete Fourier transform,” *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 25, no. 3, pp. 235–238, June 1977, doi: 10.1109/TASSP.1977.1162950.