A Transformer-Based Approach to Music Separation

Mary Yang, Sue Yang, Elliott Zackrone, and Xiaoqing Zhou
12 March 2023

Abstract

Music Source Separation is defined as the isolation of individual instruments from a mixture of sounds. Source separation is commonly used as a tool for preservation of historical recordings, musical transcription, or preprocessing for further analysis. In this project, we introduce two transformer-based architectures for music separation, MSTransformer and MSTU. Both methods achieve decent, but not state-of-the-art, evaluations using the MUSDB18 dataset [7]. However, we contribute further experimentation and insight to the models' behavior using more data and parameters, to better inform future research with transformer-based music separation.

Introduction

A source is defined as an individual signal (e.g. voice, instrument, etc.), which has been combined with other sources to create a mixture. Since the introduction of Deep Learning, source separation has been dominated by RNNs and CNNs. RNNs perform very well, as they capture a long range of information by recursively analyzing the input, but scale poorly and are impossible to parallelize on the sequence (e.g. estimate must be generated sequentially). Conversely, CNNs are easily parallelizable, but do not capture as much information as RNNs, due to the limited sizes of their kernels.

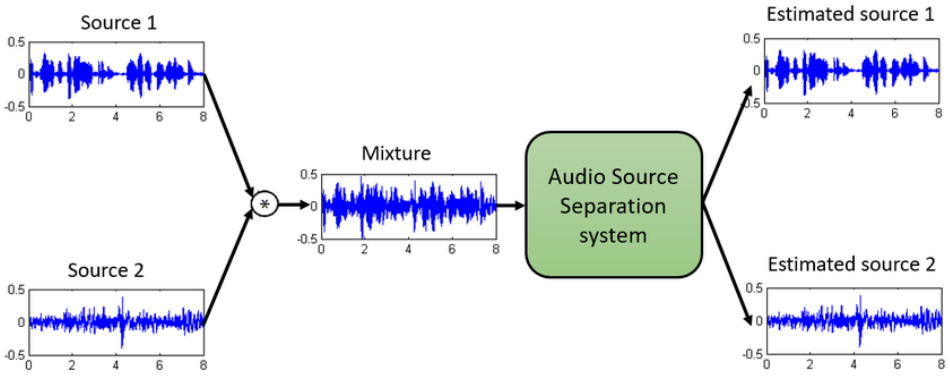

Figure 1: Audio Source Separation Visualizer [9]

Transformers present a compromise that is able to capture long ranges of information using attention mechanisms, without needing to recursively analyze the entire input [1]. Then, the encoded attention values are passed through a parallelizable, feed-forward network. Our project introduces two transformer-based neural networks for music separation, MSTransformer and MSTU (Music Separation Transformer-UNet). MSTransformer is a purely transformer-based architecture, whereas MSTU combines the transformer encoder with the UNet architecture to generate better estimates. Prior attempts to use transformers for music separation have

necessitated external training data to meet non-transformer performance. In our project, we restrict ourselves to our specific dataset, to analyze the behavior of transformer-based architectures without additional support.

Related Work

As further discussed in the Mathematical Background section, there are two primary ways to represent data for music separation: the waveform and the spectrogram. Both representations encode the same audio, but with different features (a deeper understanding is not necessary for this section). Most deep learning models for music separation only separate a single source [2, 3, 4, 6]. Consequently, distinct models must be trained for each individual source.

As priorly mentioned, modern music separation is dominated by RNNs and CNNs. A popular approach for encoding audio (both waveform and spectrogram) is the UNet. For instance, Wave-U-Net is a waveform CNN that uses upsampling and downsampling (with learned parameters), combined with convolutional layers, to learn features from the mixture and separate a source [3]. Downsampling is the process of reducing the sample-rate (bits per sample) of a signal via compression. Wave-U-Net repeatedly downsamples the input waveform, computing higher-dimensional features on an increasingly coarse time scale. Then, the encoded features are combined with output from previous layers, to upsample the signal and generate an estimated source (e.g. the output of the last downsampling block is passed to the first upsampling block). In doing so, the model is able to learn long range temporal relations via downsampling, while maintaining short term contextual information via upsampling. Although this architecture performs well, and is used as a basis for future models, its performance stagnates with outliers and larger inputs. Quiet segments of audio, labeled as “outliers" by the authors, are difficult to recover during resampling, creating residual noise. Additionally, large inputs may require additional resampling, which may necessitate too much memory usage. Other models, such as SepFormer [4], avoid this limitation by dividing the input into a sequence of chunks before passing it through a transformer. Then, the model augments the chunked output to create a separated waveform.

A hybrid-transformer architecture, known as HTDemucs, uses both waveform and spectrogram representations of a mixture to encode separate sources, which are combined to create an estimate. HTDemucs is composed of two UNets (corresponding to the waveform and spectrogram), each with five downsampling blocks and five upsampling blocks. At the “bottleneck” (between the last downsampling block and first upsampling block), HTDemucs applies a cross-domain transformer encoder. This transformer uses the downsampled input of both UNets to further encode features from both representations (waveform and spectrogram). After upsampling, the two UNets aggregate their results to generate an estimate. Although HTDemucs overcame the weaknesses of many other models, it still underperformed relative to state-of-the-art RNNs. To provide its transformer more contextual information, HTDemucs was trained using 800 extra songs, which resulted in the current top SDR score for MUSDB18 [5, 7] of 9.00.

Data Collection

Our project uses the MUSDB18 dataset from SigSep [5]. MUSDB18 comprises 150 full-length audio tracks (~10 hours of audio), each with 5 sources: mixture (all sources), vocals, drums, bass, and other. The dataset includes 18 different genres, such as pop, rock, electronic, and classical. MUSDB18 is widely used for training and evaluating models for music source separation, which provides a useful baseline to compare our model performance against other architectures. The sample-rate of each track is 44.1 kHz (e.g. each second of audio duration is represented by a 44100-dimensional array).

As priorly mentioned, previous attempts of using transformers for music separation have underperformed due to their relatively small datasets. Although MUSDB18 is a considerably large dataset (4.4 GB), there are only 150 tracks that the model can use to learn musical features. To augment this dataset, we borrow the solution from Open-Unmix, a model developed by SigSep [6]. Open-Unmix randomly samples from each source at random points in a track, and combines these samples to generate a new mixture. This solution creates more data points for training, and provides new random patterns between sources in each mixture. For evaluation, we use the normal MUSDB18 validation dataset, without random mixing, to provide a better comparison against other baseline models. In Results, we provide a comparison of model performance as we increase the number of random samples per track. We further discuss how the data is processed (and why) in the Method section.

Method

Mathematical Background

Waveform and Spectrogram

There are two representations used for audio in source separation: the waveform and the spectrogram. The waveform is a 2D representation of audio which defines the amplitude of a signal at a given time. In comparison, the spectrogram is a 3D representation of audio which defines the volume (dB) of a given frequency at a given time. There are several equations used to alternate between these representations. In this project (with MSTransformer), we use the (Discrete) Short-Time Fourier Transform, which converts a waveform to a spectrogram as,

$$\mathbf{STFT}\{x[t]\}(\omega, \tau) = \sum_{t=-\infty}^{+\infty} x[t]w[t-\tau]e^{-i\omega t}$$

where τ is a given time, ω is a given frequency, $x[t]$ is the waveform amplitude at time t, and $w[t]$ is a window function (e.g. Hann window).

The window function is non-zero for a short interval of time, and zero otherwise. Consequently, the data is grouped into chunks of time, known as frames (which usually overlap). For reference, a matrix representation of a waveform with a shape of (44100) may have a corresponding spectrogram shape of $(169, 1024)$ (Note: this shape is dependent on specified parameters, such as window length). In both representations, the last dimension represents time (sequence length). In the spectrogram, the first dimension is the number of frequencies, or “bins”, for each frame. In Deep Learning, RNNs prefer spectrograms, as they

shorten the sequence length and number of recursive calls, whereas CNNs prefer waveforms, as they provide simpler, more informative convolutions. Notably, the output of the STFT is complex-valued. DL architectures that use spectrograms must convert these values to a different format, because most DL frameworks do not fully support complex values.

Signal-to-Distortion Ratio (SDR)

There are several metrics used in source separation, however the most popular metric is Signal-to-Distortion Ratio (SDR),

$$\mathrm{SDR} = 10 \cdot \log\left(\frac{||s_{\mathrm{target}}||_2^2}{||s_{\mathrm{estimate}} - s_{\mathrm{target}}||_2^2}\right)$$

where s_{target} is the true source, and $s_{estimate}$ is our model's estimate of the true source. Intuitively, the optimal estimate is such that $s_{estimate} = s_{target}$, which minimizes the denominator and maximizes SDR. For reference, the best performing architecture (maximum SDR) for MUSDB18 has an SDR score of 9.00 [2, 7].

Attention

Attention is a technique in machine learning that allows a model to selectively "focus" on certain parts of an input sequence [1]. It works by assigning weights to different parts of the input sequence based on their relevance to the current output. In doing so, the model can better capture long-range dependencies in the input (such as audio features).

Mathematically, the equation for attention is,

$$\mathrm{Attention}(Q, K, V) = \mathrm{softmax}\left(\frac{Q\,K^T}{\sqrt{d_k}}\right)V$$

where Q, K, and V (the query, key, and value matrices, respectively) are learned during the training process. The numerator, QK^T, is a dot product of the current query with other keys in the input, which identifies parts of the input that are related to the current state. The dot product is scaled by the square root of the embedding dimension to stabilize the softmax. Multiplying by the value matrix produces a weighted sum of values, which emphasize the most relevant parts of the input. We can independently learn multiple Q, K, and V matrices to capture different aspects of the input, known as Multi-Head Attention.

MSTransformer

MSTransformer is a spectrogram transformer-based music source separation model composed of four main sections. First, a preprocessing layer converts the input waveform into a spectrogram. Then, the embedded input passes through multiple encoder and decoder blocks to create an estimated source. Finally, the postprocessing layer converts the spectrogram to a waveform for evaluation. We provide an illustration of our model for clarity.

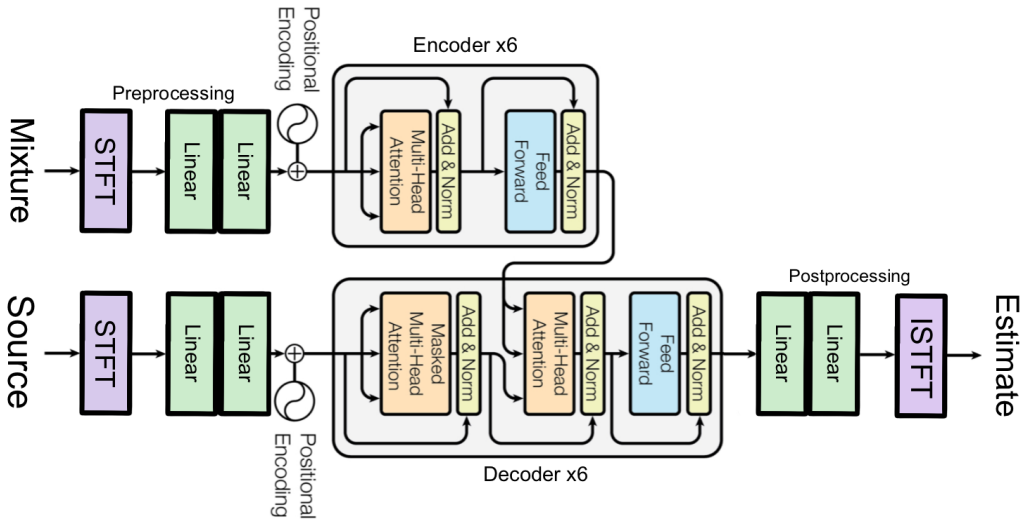

Figure 2: MSTransformer architecture. Modifies diagram from “Attention Is All You Need” [1].

Preprocessing

Our preprocessing layer begins by applying a Short-Time Fourier Transform to the waveform data to generate a spectrogram. In the STFT, we use a Hann Window with length 4096 (time per frame) and a hop length of 1024 (time between the start of frames). These parameters are not necessary to understand the architecture, but are provided for the interested reader. By definition, the output of the STFT is complex-valued, and may not match the dimension of our transformer. To simplify the input to the desired dimensions, we apply two linear layers. The first layer reduces the complex values to a single dimension, and the second layer maps the dimension of the spectrogram to the dimension of the transformer.

Encoder and Decoder

Our encoder and decoder are modeled after the architecture outlined in “Attention Is All You Need” [1]. The encoder is composed of six identical blocks, where each block has two sub-layers. Each sublayer has a residual connection, followed by layer normalization. The first sublayer applies self-attention, and the second sublayer is a simple feedforward network (linear layers). The shape of the input and output are the same, such that we are able to consecutively stack encoder blocks without additional processing. Before the first encoder block, we add a positional encoding, which provides the transformer with the relative location of each element in the input sequence.

$$PE(pos, 2i) = \sin\left(pos/10000^{2i/d_{model}}\right)$$
$$PE(pos, 2i+1) = \cos\left(pos/10000^{2i/d_{model}}\right)$$

where pos is the position in the input and i is the i-th feature.

Similarly, the decoder is composed of six identical blocks, where each block has three sublayers. The first sublayer applies self-attention to the current output sequence (initially empty). Then, the second sublayer applies cross-attention using the output of the encoder and

the output of the first sublayer. This correlates the encoded input with the current output. Finally, the third sublayer applies a feed-forward layer, which is then passed to the postprocessing layer.

Postprocessing

Finally, the postprocessing layer converts the estimated spectrogram to a waveform. To do so, we apply two linear layers to undo the preprocessing layer. The first layer maps the dimension of the transformer to the original dimension of the spectrogram. Then, the second layer converts single-dimensional output to two-dimensional complex values. We apply ISTFT to convert the spectrogram to a waveform. However, our estimated spectrogram is unlikely to correspond to a uniform signal, such that ISTFT may discard parts of the spectrogram, resulting in a shorter waveform (e.g. an input waveform of size 44100 may produce an estimate of size 44096). To generate a consistent estimate, we pad the output to the length of the original mixture.

MSTU

MSTU is an improvement of MSTransformer, which uses a UNet architecture to encode a waveform, rather than a spectrogram. MSTU is composed of three main sections. First, the downsampling blocks encode an increasingly higher-dimensional feature representation of the input waveform on coarser time scales. We use a transformer encoder to apply multi-head attention using the high-dimensional representation. Then, the encoded features are combined with the output of previous layers, yielding multi-time-scale features for the decoded source estimate. We provide an illustration of the new architecture for clarity.

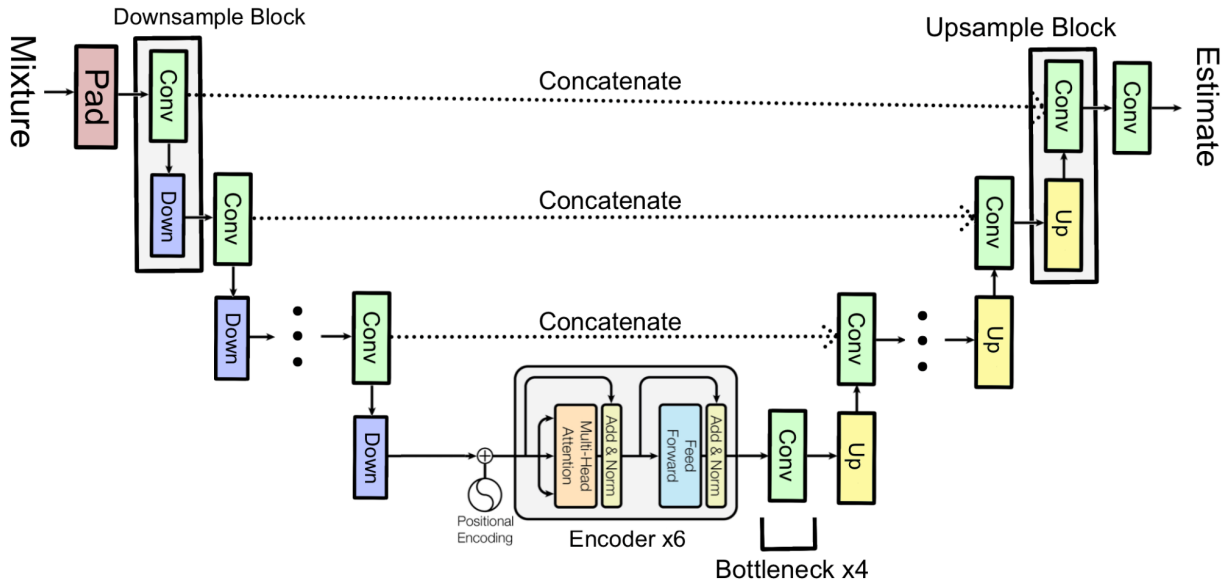

Figure 3: MSTU architecture. Combines Transformer encoder with UNet resampling.

Downsampling

The downsampling layers reduce the input time dimension (length of the waveform), by using convolutional layers and max-pooling, which compute an increasingly higher-dimensional feature representation of the waveform on the coarser time scale. Each downsampling block uses a double convolution, which applies two separate convolutional layers, each followed by batch normalization and ReLU activation (denoted as “Conv” in the above diagram). These double convolutions increase the dimension of our encoding, which is also provided to future upsampling blocks as additional context. After the double convolution, we apply a max-pooling layer (with a kernel size of 2), to create a coarser time scale that divides the length of the waveform in half (denoted as “Down” in the above diagram).

To reiterate, each downsampling block divides the input length in half. If our input has odd length, then we lose information, as the fractional length is rounded down. To avoid losing information, we pad each input to the next largest power of two. This ensures that consecutive downsampling blocks will never create an odd-length input.

Another potential issue to address is Dying ReLU [10]. If previous layers learn a large negative bias, then ReLU may get “stuck”, as it outputs zero for all negative values (and the gradient is also zero). To avoid this problem, we standardize the input waveform to avoid learning large biases, and also apply dropout regularization.

Encoder and Bottleneck

The encoder and bottleneck layers of MSTU connect the downsampling and upsampling layers. The encoder is recycled from the Encoder described in MSTransformer. However, this encoder uses a waveform input, rather than a spectrogram. After the transformer encoder, the encoded input is passed to a series of double convolutions, known as the bottleneck. The bottleneck convolutions do not apply max-pooling, such that the length of the waveform does not change. The purpose of the bottleneck layer is to use the attended output from the transformer encoder to further extract the most important features and context from the input (following the intuition from HTDemucs [2]).

Upsampling

As illustrated in the above diagram, the upsampling blocks combine the output of previous downsampling convolutions (known as “shortcuts”) with the encoded input, to create a higher resolution waveform (where the length is doubled). First, the model applies a transposed convolution (denoted as “Up” in the above diagram), which decreases the feature dimension and increases the sequence length. We concatenate this output with the shortcut from the corresponding downsampling block, and apply a double convolution. The shortcut retains information from the original waveform, which improves the quality of the reconstructed audio. We repeatedly upsample until reaching the original padded length, which is followed by a final convolution, and then cropped to the original input length.

Results

Baseline Comparison

We quantitatively evaluate both MSTransformer and MSTU by comparing their SDR scores against other related architectures. We report the highest performance, using the MUSDB18 evaluation dataset, from numerous model training attempts. Both models are trained using 64 random samples per track and dropout regularization (with $P_{drop} = 0.1$). We use the same number of layers and parameters provided in the above descriptions. Additionally, we use mean-squared error to calculate our training and validation loss.

Model	HTDemucs	**MSTU**	Wave-U-Net	**MSTransformer**
SDR	9.00	**5.18**	3.25	**3.10**

Table 1: Comparison of our models (bolded) against SOTA and related architectures.

We find that MSTU outperforms MSTransformer and Wave-U-Net, but both models underperform relative to a state-of-the-art architecture, such as HTDemucs. To avoid redundancy, further analysis will primarily involve MSTU, as the results are more informative. We qualitatively analyze an informative example source, to observe patterns that are challenging for our model to estimate.

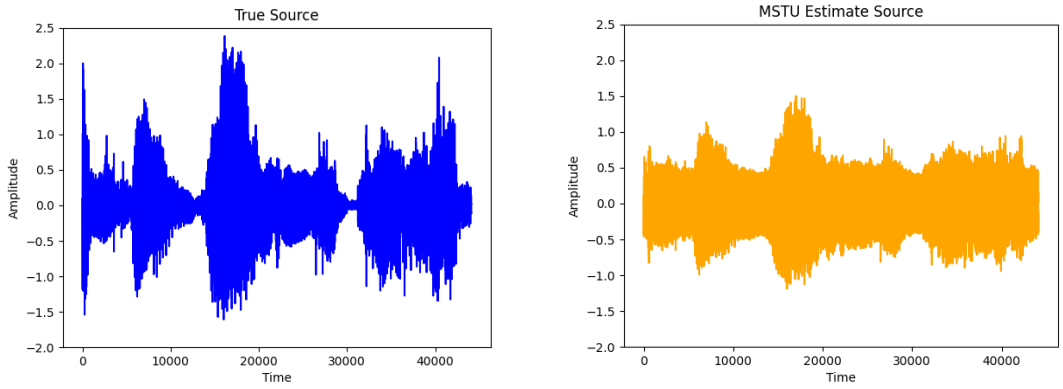

Figure 4: A true source (left) from a mixture, and the estimate (right) generated by MSTU.

The above estimate illustrates several errors of MSTU. The overall shape of the waveform (e.g. location of peaks, width of bands, etc.) is correct, but not as clearly defined. The peaks of the true waveform are less intense, whereas the troughs (amplitude near zero) experience significant noise. Additionally, the model struggles to estimate short, intense peaks, such as the peaks near 0 and 40000. Consequently, the estimate sounds noisy, as if the source was recorded through an old microphone (an example will be provided in the project presentation).

In the original Wave-U-Net paper, the authors observed similar noise near troughs. Quantitatively, the SDR score shows that MSTU provides a good estimate of the source. However, qualitatively, the downsampling and upsampling from the UNet creates residual noise,

especially in quieter segments. Overall, MSTU still provides significant improvement over Wave-U-Net and MSTransformer.

Data Augmentation

Additionally, we undergo further experimentation to show potential improvements and provide a better understanding regarding the behavior of MSTU. Our first addendum correlates the SDR score with the number of random samples per track. In Data Collection, we discussed how we augment our training dataset by randomly sampling from each source, and combining the samples to create a new mixture. In this section, we analyze how the number of random samples per track affects the SDR score.

We perform all computations on our individual laptops. Consequently, the training and evaluation process requires a significant amount of time, which increases linearly with the size of the dataset. To train and evaluate the models in a reasonable amount of time, we reduce the number of training epochs. Consequently, the models do not fully converge, which causes their SDR scores to underperform relative to the baseline comparison. However, the overall trend is still informative.

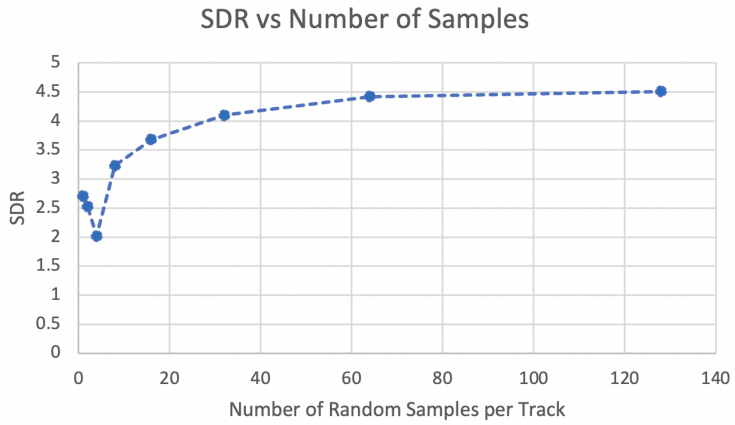

Figure 5: Comparison of SDR score against number of random samples per track.

We observe that the SDR initially decreases, and then monotonically increases beyond $n = 4$ samples. This initial decrement contradicts previous research, which states that performance should increase logarithmically based on the volume of training data [11]. We hypothesize that this error is a consequence of our random sampling. We resample from the same tracks, which may be interpreted as noise by our model, causing the initial decrease. As the number of samples per track continues increasing, the model is able to fit to the new samples, and the performance logarithmically increases as expected.

Model Complexity

Our second addendum correlates the training and validation loss with the number of parameters in our model. We increase the number of parameters in our model by: increasing the number of resampling layers, increasing the number of channels in the resampling layers, increasing the number of encoder blocks, and increasing the number of bottleneck layers. We

train multiple models with an increasing number of parameters, and compare their respective mean-squared errors.

This experiment is victim to the same restrictions as the former. To train and evaluate the models in a reasonable amount of time, we reduce the number of training epochs and random samples per track.

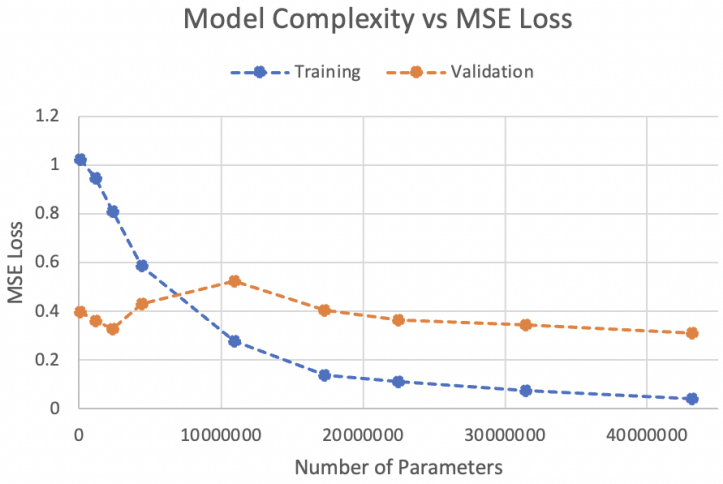

Figure 6: Comparison of MSE loss and model complexity.

We observe that the validation loss initially decreases, then increases, and then decreases again as the number of parameters increases. This observation mirrors a phenomenon observed in modern Deep Learning research, known as “Double Descent” [8]. Although our observation does not match the exact criteria for double descent (training loss does not reach zero), the results are still indicative of the benefits of larger models for music separation. Our environment restrictions prohibit further experimentation, but we hypothesize that increasing parameters would continue the decreasing end behavior. The results suggest that future music separation research involving CNNs and Transformers (and more GPUs), could observe increased performance by increasing model complexity.

Conclusion

In this paper, we proposed two transformer-based architectures for music separation, MSTransformer and MSTU. Although our architectures do not achieve state-of-the-art results, we contribute experimental analysis of methods to improve future transformer-based architectures, such as artificial data augmentation and increased parameterization. Previous research involving transformer-based music separation, such as HTDemucs, required additional audio to achieve state-of-the-art results [2]. By randomly sampling new mixtures, we are able to increase performance, without introducing new tracks. We also provide new experimental results to illustrate how model performance increases as parameters increase. Future research using larger datasets and more parameters would necessitate better development environments. However, if these larger, transformer-based architectures are successful, they would be easily parallelizable, in comparison to their RNN counterparts.

Contributions

Elliott Zackrone: Introduction, related work, data collection, mathematical background, model implementations, model descriptions and diagrams, baseline comparison, data augmentation, model complexity, audio generation, and presentation preparation.

Xiaoqing Zhou: References searching, related work, model description and implementations, experiment analysis and diagrams.

Sue Yang: Review of related work, description of data in data collection, searching for related work, working on data preliminary analysis and report-writing.

Mary Yang: References searching, introduction, review of related work, description of metadata, experimental analysis, and presentation preparation.

References

[1] Ashish Vaswani, et al. "Attention Is All You Need". *CoRR* abs/1706.03762. (2017).

[2] Rouard, Simon et al. "Hybrid Transformers for Music Source Separation." (2022).

[3] Daniel Stoller, Sebastian Ewert, & Simon Dixon (2018). "Wave-U-Net: A Multi-Scale Neural Network for End-to-End Audio Source Separation." CoRR, abs/1806.03185.

[4] Cem Subakan, Mirco Ravanelli, et al."Attention is All You Need in Speech Separation." abs/2010.13154. (2020).

[5] Rafii, Zafar et al. "The MUSDB18 corpus for music separation." (2017).

[6] Fabian-Robert Stöter, et al. "Open-Unmix - A Reference Implementation for Music Source Separation". *Journal of Open Source Software* 4. 41(2019): 1667.

[7] Meta AI. *Papers With Code - MUSDB18 benchmark (music source separation)*. The latest in Machine Learning. (2022).

[8] Preetum Nakkiran, Gal Kaplun, Yamini Bansal, Tristan Yang, Boaz Barak, & Ilya Sutskever (2019). Deep Double Descent: Where Bigger Models and More Data Hurt. *CoRR, abs/1912.02292.*

[9] Duong, Thanh Thi Hien, et al. "Multichannel audio source separation exploiting NMF-based generic source spectral model in Gaussian modeling framework." Latent Variable Analysis and

Signal Separation: 14th International Conference, LVA/ICA 2018, Guildford, UK, July 2–5, 2018, Proceedings 14. Springer International Publishing, 2018.

[10] Lu Lu (2020). Dying ReLU and Initialization: Theory and Numerical Examples. *Communications in Computational Physics, 28(5), 1671–1706.*

[11] Chen Sun, Abhinav Shrivastava, Saurabh Singh, & Abhinav Gupta (2017). Revisiting Unreasonable Effectiveness of Data in Deep Learning Era. *CoRR, abs/1707.02968.*