PRJ2: Extracting Single Voices from Musical Ensembles

Nikhil Iyer

November 2021

1 Proposal

Given an audio track of musical ensembles ranging from 2 to 5 instruments, the network is tasked with separating out the audio into tracks that contain a particular instrument.

1.1 Data

The acquired dataset is composed of recordings of 44 different musical selections. Each player has recorded their part of the musical selection independently, the musical selection is then produced by adding together the individual tracks.

The audio files are in WAV format with a sampling rate of 48kHz and a bit depth of 24bits, mono channel. The audio mixture is the direct sum of the individual tracks without individual gains. Individual track numbers are ordered following the score track order.

1.2 Prepossessing

A mixed audio track need not be complete to be useful. Given that a song may be represented by a set S of its individual parts. The useful data for training is not only the entire song but every possible non-empty subset of it. For example, some song $S_0 = \{VN, VA, TPT\}$. The set of training material for the violin T_{VN} would be the all elements of the powerset of S_0 which contain VN.

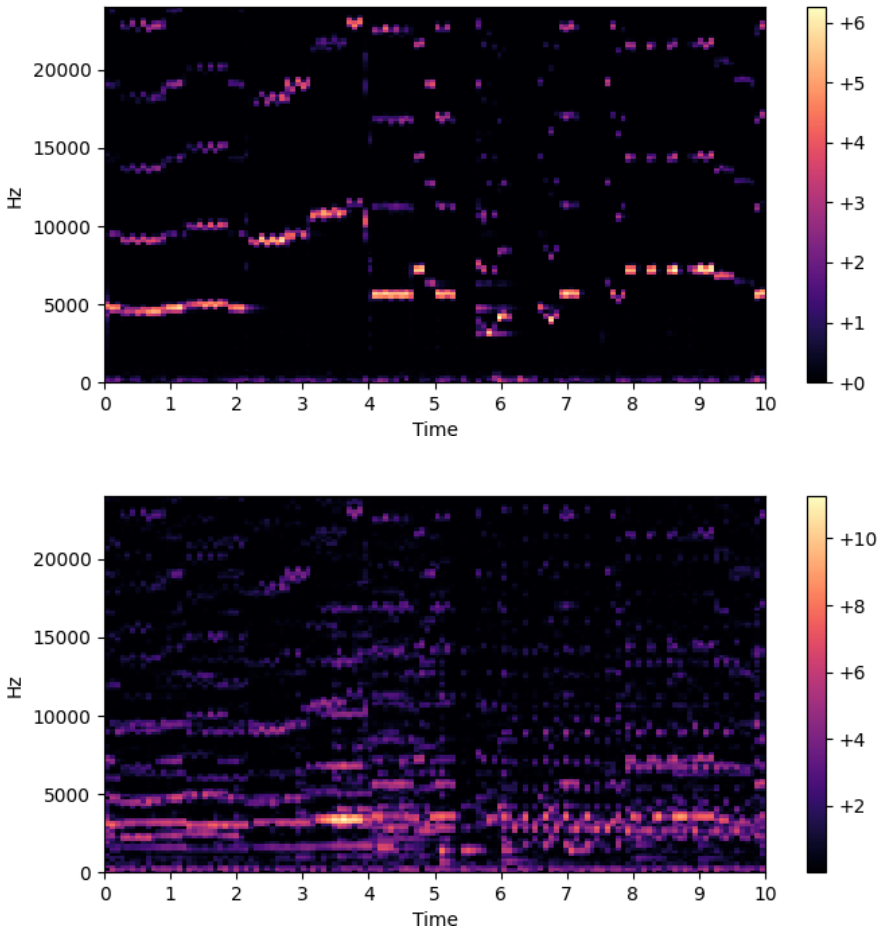
$$T_{VN} = \{\{VN\}, \{VN, VA\}, \{VN, TPT\}, \{VN, VA, TPT\}\}$$

This inflates the amount of training data and allows it to be more diverse than a mere 44 songs.

This form of the dataset was cached to the disk through a memory mapped NumPy array and read as needed during subsequent prepossessing steps.

As stated in my previous report the amount of data was intractable. A 10-second clip of audio at 48kHz is about half a million datapoints. To reduce the amount of data without losing too much information, I used the Fast Fourier Transform (FFT) to convert the 10-second clips from the amplitude-time space into 128x128 spectrograms in the amplitude-frequency-time space.

Below are the spectrograms for the violin (top) and mixed (bottom) audio.



2 Benchmarking

Because many of the examples are simply identity functions (mapping a solo onto itself) the benchmark I chose to use for this problem was simply the loss of the identity function.

The loss function was Mean Squared Error and the loss of a dataset was average loss per example.

For each instrument the loss of the identity function was:

Instrument	Loss
VN	0.8873
VA	0.4913
VC	0.6317
DB	0.5702
FL	0.6404
OB	0.5867
CL	0.4938
SAX	0.6676
BN	0.7336
TPT	0.8125
HN	0.9484
TBN	0.8535
TBA	1.0705

3 Initial Architecture

The final model architecture was based on suggestions from Minh, and my own knowledge of music theory.

Minh suggested training two models. One for detecting the presence of the target instrument, the second for extracting its voice. That was a great idea so what I ended up doing was training a single model with a split computational graph.

Initial Architecture

Layer Type	Output Shape
Preparer	
Input	(-1, 1, 128, 128)
Conv2D	(-1, 8 64, 64)
ReLU	(-1, 8, 64, 64)
Conv2D	(-1, 8, 64, 64)
ReLU	(-1, 8, 64, 64)
Detector	
Preparer	(-1, 8, 64, 64)
Conv2d	(-1, 1, 32, 25)
Linear	(-1, 64)
ReLU	(-1, 1, 64, 1)
Re-Generator	
Preparer×Dectector	(-1, 1, 64, 64)
ConvTranspose2D	(-1, 1, 129, 129)
ReLU	(-1, 1, 128, 128)

The first Conv2D has 16 filters, with 5x5 kernels, 2x2 stride, and 2x2 padding. The second Conv2D has 16 filters, with 3x3 kernels, 1x1 stride, and 'same' padding. The intent is to allow the Preparer to distinguish and highlight useful patterns.

The third Conv2D has 1 filter, with a 2x16 kernel, a 2x2 stride, and no padding, The kernel size was determined by inspecting the spectrogram and counting the max pixels between resonances. The stride prevents overlap on the time-axis. The Linear layer which directly follows the Convolution serves to collapse the 32x25 image into a Timeline of 64x1.

The output of the Preparer is multiplied with the Timeline then fed into the Re-Generator which upscales the processed image to the proper output shape. Note the ConvTranspose2D only produces outputs of odd dimension so its last row and column were simply truncated.

4 Training and Tuning

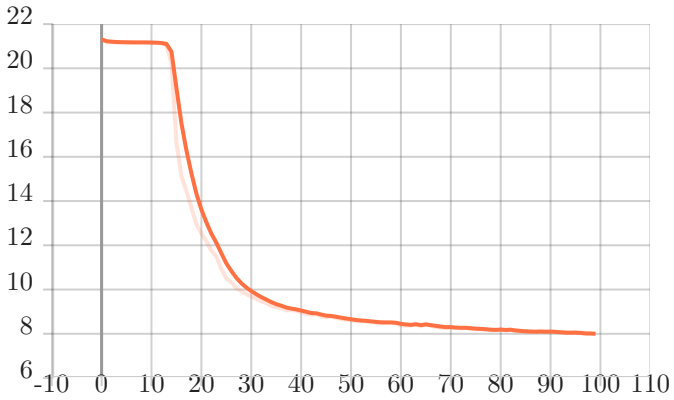


Figure 1: VN_0

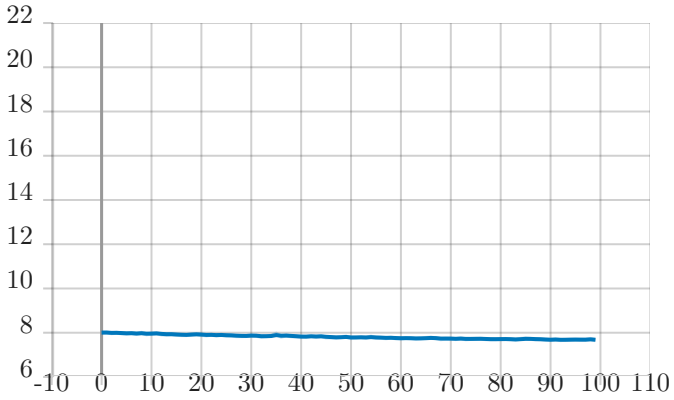


Figure 2: VN_1

The initial run (Fig. 1) was rather successful especially in comparison with the fully connected network from the last project. The loss decreases very smoothly excepting the small plateau at the beginning. Another set of 100 epochs was run after training halted (Fig. 2) to see if anything cool would happen (discover new minimum?). After the first 100 epochs the model did not learn much.

The next thing I did was see if the model would work as well on the other instruments as well.

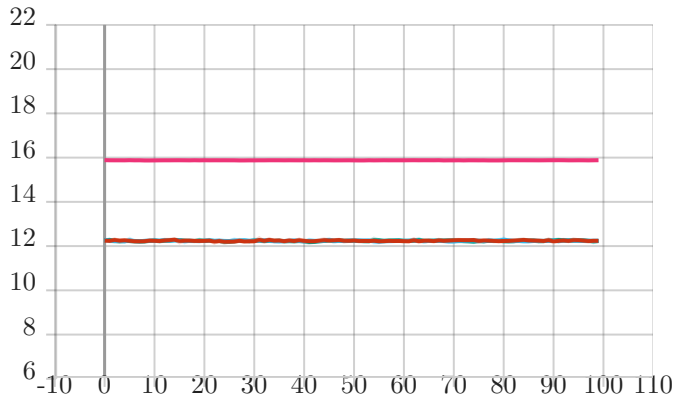


Figure 3: VC_0 & VC_1 & VC_2 & TPT_0

The model did appear to have some issue with dying nodes (Fig. 3). Specifically, printing the output revealed the model was producing a 128x128 image of zeros. I switched out all except the last ReLU for ELUs to reduce node death which resolved the node death.

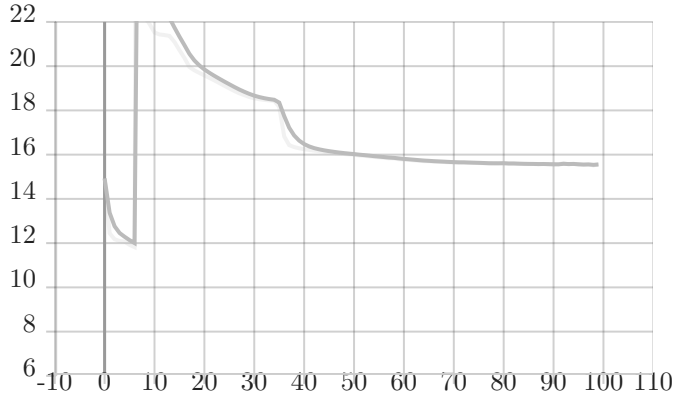


Figure 4: VN_2

There was a large unexplained discontinuity that would occasionally show up during training. Originally, I thought it was the result of truncating the output of the final ConvTranspose layer, but I made a similar model without the truncate and the issue persisted. The model only takes 5 minutes to train so I can afford to run it a couple times for a good initialization.

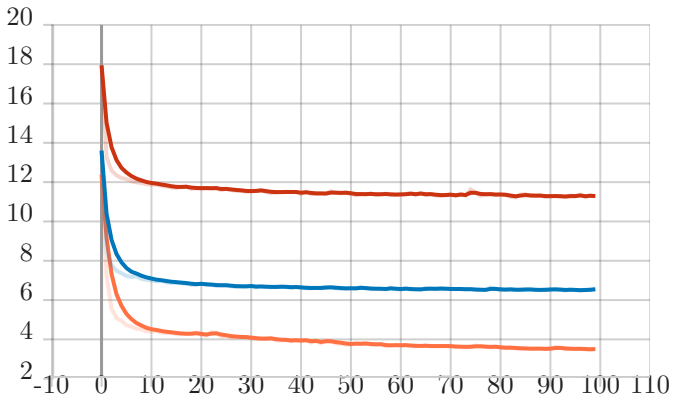


Figure 5: VN_3 & VC_3 & TPT_1

The next several runs (Fig. 5) faced no such issues and had very smooth learning curves. each approaching the same asymptote of their respective instruments.

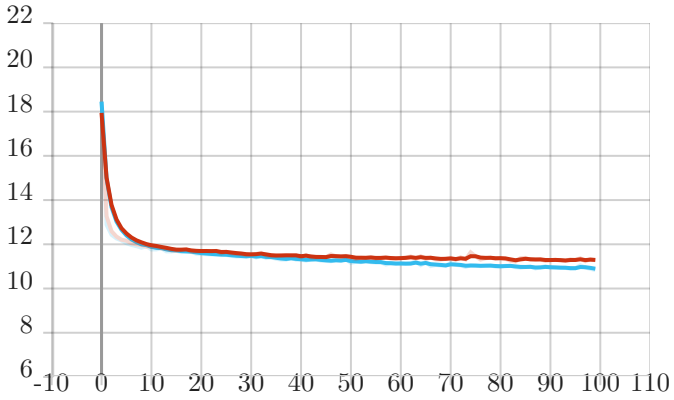


Figure 6: VN_3 & VN_4

The neural network is becoming saturated song before the training data is depleted, so I tried increasing the number of filters on the convolutional layers and the size of the linear layer. It did not have much influence on learning (Fig. 6).

The training loss of the network was significantly lower than the testing loss (Fig. 7) so I added an L2 penalty which had the desired effect of completely halting learning (Fig. **??**). Reducing the L2 coefficient led to a little more chaotic convergence but a less distance between the training and testing losses (Fig. **??**).

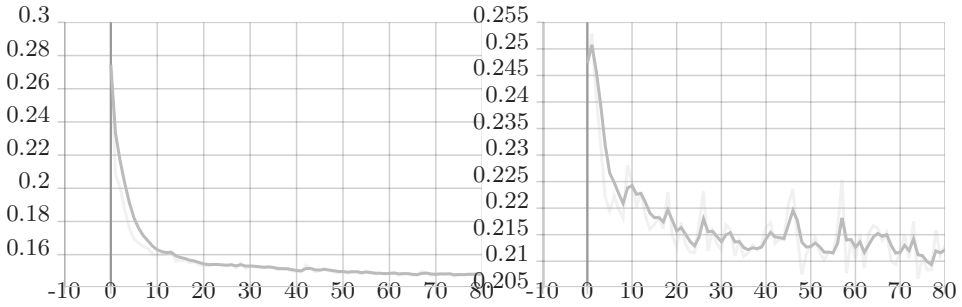


Figure 7: VN_6

Final Architecture

Layer Type	Output Shape
Preparer	
Input	(-1, 1, 128, 128)
Conv2D	(-1, 16 64, 64)
ELU	(-1, 16, 64, 64)
Conv2D	(-1, 16, 64, 64)
ELU	(-1, 16, 64, 64)
Detector	
Preparer	(-1, 16, 64, 64)
Conv2d	(-1, 1, 32, 25)
Linear	(-1, 64)
ELU	(-1, 1, 64, 1)
Re-Generator	
Preparer×Dectector	(-1, 1, 64, 64)
ConvTranspose2D	(-1, 1, 129, 129)
ReLU	(-1, 1, 128, 128)

5 Results

For each instrument the loss of the model was:

Instrument	Model Loss	Benchmarking Loss
VN	0.3879	0.8873
VA	0.3310	0.4913
VC	0.2739	0.6317
DB	0.1554	0.5702
FL	0.2734	0.6404
OB	0.2163	0.5867
CL	0.4285	0.4938
SAX	0.3377	0.6676
BN	0.3332	0.7336
TPT	0.3389	0.8125
HN	0.3096	0.9484
TBN	0.3605	0.835
TBA	0.2788	1.0705

Figures 8-11 illustrate the varying degrees of success the model had at isolating the voice of each instrument. Using the predictions as a mask it becomes possible to reconstruct the audio with only the desired voice.

6 Conclusion

After way too much time and effort the model has finally performed well enough to be called successful. The spectrograms show a clear suppression of all noises except the target voice. The model out performed the benchmark in every case (which didn't blow up). In hindsight I should have shuffled the data BEFORE partitioning it because the more complex songs ended up being over represented the the validation set. Regardless, the model still did quite well in separating the audio tracks.

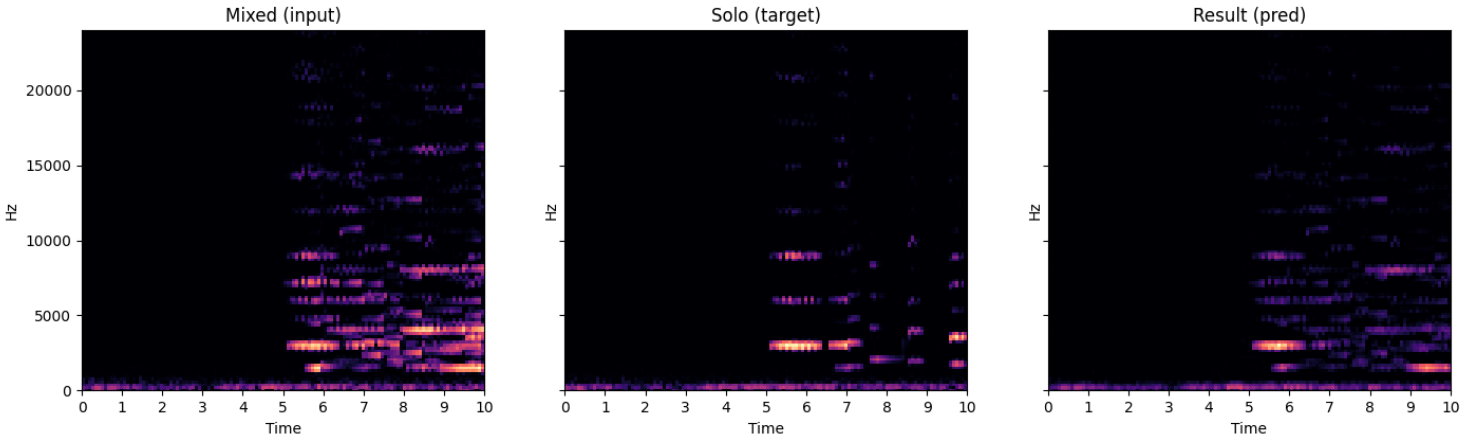


Figure 8: [VA_{Final}]

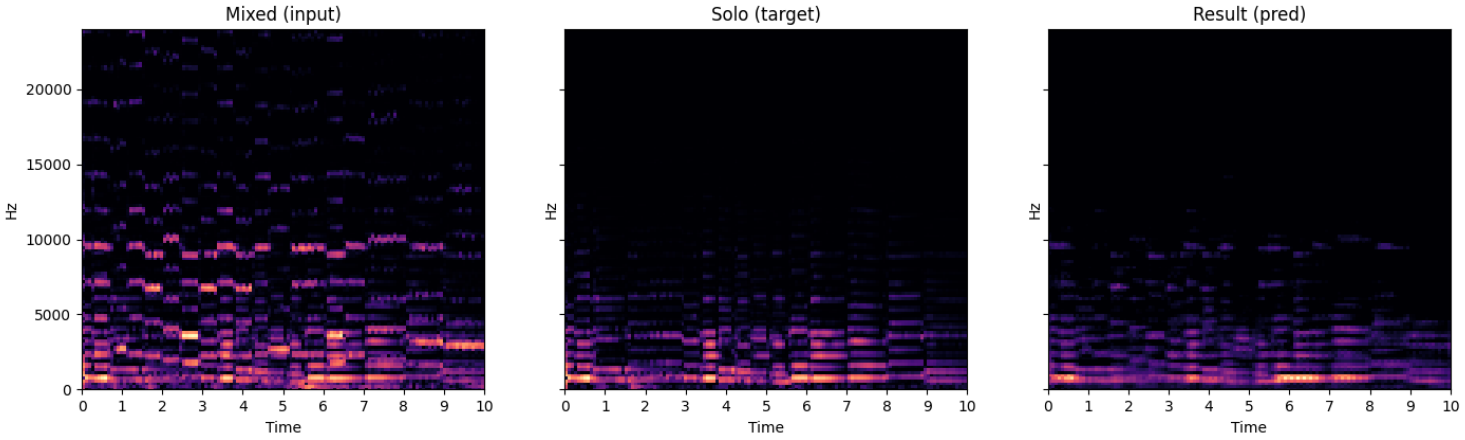


Figure 9: [DB_{Final}]

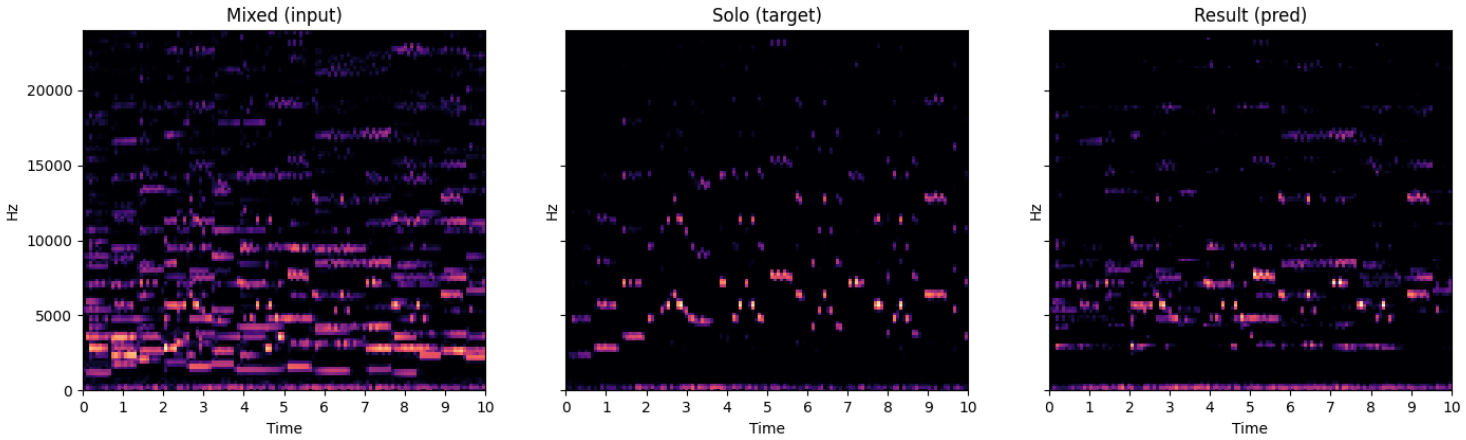


Figure 10: [$\mathrm{FL}_{\mathrm{Final}}$]

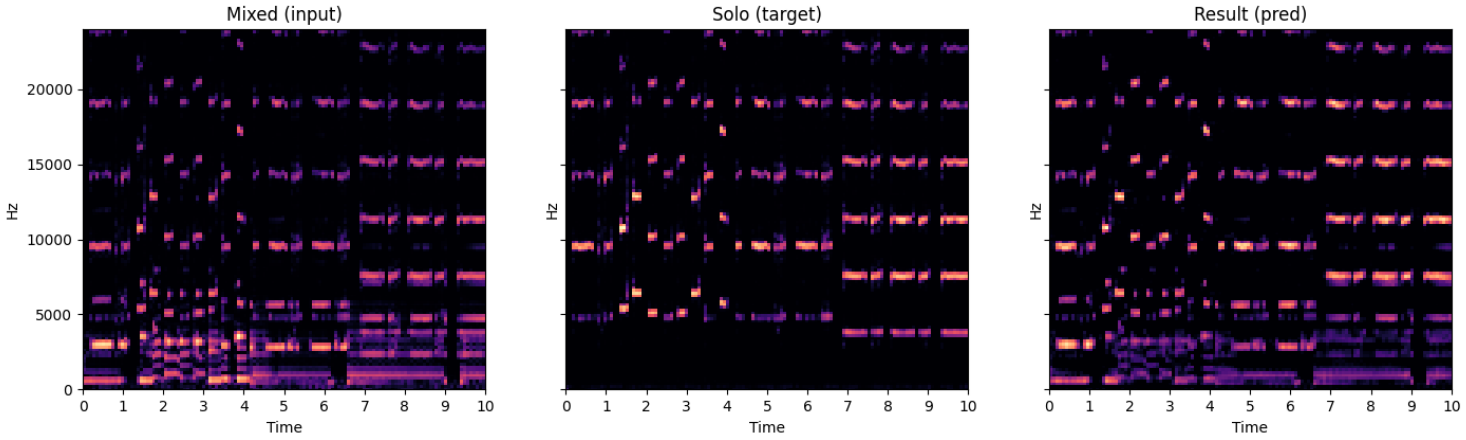


Figure 11: [$\mathrm{TPT}_{\mathrm{Final}}$]