

Deep Audio Isolation

Isolating Vocals from Complex Musical Tracks.

APS 360 Final Report

April 13, 2022

Group 47

Abhinav Sanjeeva Prasad

Bahareh Farhadi

Boxuan Wang

Jay Mohile

I. INTRODUCTION	4
II. BACKGROUND AND RELATED WORK	4
III. DATA PROCESSING	5
1: Preliminaries	5
2: MUSDB Dataset Collection and Preprocessing	6
3: MedleyDB 2.0 Dataset Collection and Preprocessing	7
IV. ARCHITECTURE	8
Problem Scoping	8
Model	9
Improvement 1: Awareness	10
Improvement 2: Pre-FC	10
Complex Data	11
V. BASELINE MODEL & RESULTS	11
Baseline Model	11
Baseline Results	12
VI. MODEL RESULTS	12
1: Quantitative Result	12
Loss	12
Accuracy	14
2: Qualitative Results	14
VII. EVALUATION OF MODEL ON NEW DATA	15
VIII. DISCUSSION	17
IX. ETHICAL CONSIDERATIONS	17
1: Copyrights and Licence	17
2: Inappropriate Use	18
X. PROJECT DIFFICULTY	18
1: Domain Knowledge	18
2: Generative Modelling	18
3: Limited Datasets	20
Augmentation 1: Loopback	20
Augmentation 2: Noise Generation	21
REFERENCES	23
APPENDIX I. BASELINE MODEL	26
APPENDIX II. CODE	27
APPENDIX III. SAMPLES PRODUCED FROM THE MODEL	29

Deep Audio Track Isolation

Given an input audio track, consisting of various vocals/instruments, isolate one or more of those tracks.



I. INTRODUCTION

Given an audio recording consisting of many mixed tracks (e.g vocals, instruments, background), it is often desirable to isolate one or more of those tracks individually. This has clear applications in fields like music/video-production, where it is useful to adjust individual tracks from a composite.

Generalising, the same capability could be applied to broader domains - such as active noise cancellation (or even augmentation), or as a pre-processing step to video-calls, voice-assistants, etc.

This problem is complicated because component tracks overlap in time and frequency. Thus, there is no analytic way to isolate them from each-other. Historically, statistical techniques such as ICA have been used [1], and recently a wide array of CNN and RNN based approaches have risen.

As such, given a source-track containing several instruments, vocals, etc, a deep-learning model is proposed to extract a specific component track. While the architecture used is technically agnostic to the specific type of track isolated (e.g vocal, guitar, piano), we've focused our efforts on vocal isolation specifically.

II. BACKGROUND AND RELATED WORK

Hybrid Demucs [2] is an open-source platform that isolates various stems (drums, guitar, etc) of a mixed input. It used MusDB and an internal dataset for training [3].

This platform is a modification of the U-Net convolutional architecture, inspired by Wave-U-Net [4]. This modification resulted in increased complexity of the U-Net encoder/decoder, which is the primary challenge with this model [5].

According to Figure 1, this model provides the highest Signal-to-Distortion Ratio, SDR, a measure of sound quality, of 7.7 dB amongst other models that were trained on the same dataset, MusDB.

Model	Domain	Extra data?	Overall SDR
Wave-U-Net	waveform	no	3.2
Open-Unmix	spectrogram	no	5.3
D3Net	spectrogram	no	6.0
Conv-Tasnet	waveform	no	5.7
Demucs (v2)	waveform	no	6.3
ResUNetDecouple+	spectrogram	no	6.7
KUIELAB-MDX-Net	hybrid	no	7.5
Hybrid Demucs (v3)	hybrid	no	7.7
MMDenseLSTM	spectrogram	804 songs	6.0
D3Net	spectrogram	1.5k songs	6.7
Spleeter	spectrogram	25k songs	5.9

Figure 1: Accuracy of models trained on MusDB measured in SDR [2]

III. DATA PROCESSING

1: Preliminaries

There are two training datasets, MUSDB and MedleyDB 2.0, containing 150 and 74 tracks respectively [3][6]. Figure 2 is the input data structure to our neural-network, and is the target of data preprocessing. Processed datasets are stored on Google Drive.

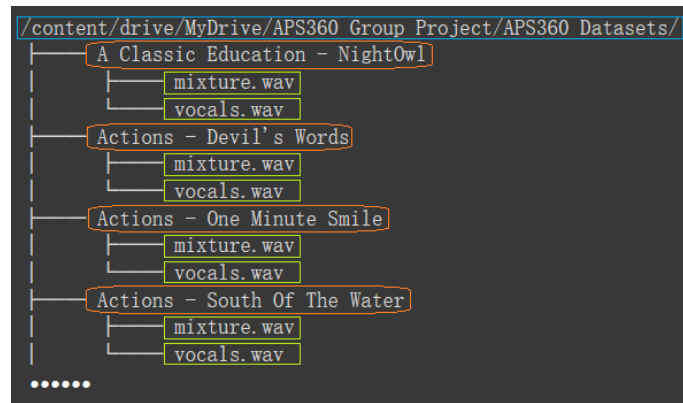


Figure 2: Data structure to feed into our neural-network

2: MUSDB Dataset Collection and Preprocessing

We obtained MUSDB through pytorch on Google Colab [3]. The structure of the dataset is already very similar to our goal. We made the following observations:

1. The stems are provided as a dictionary in {label:stem_audio_data} format for each stem.
2. Each stem audio data has the shape (num_samples, num_channels).

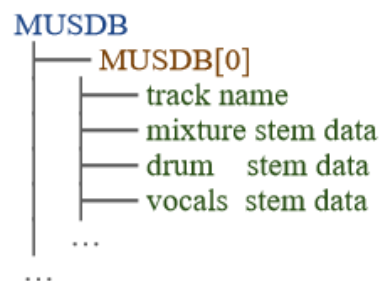


Figure 3: MUSDB data structure

The only parts of interest are the ‘mixture’ and ‘vocals’ stems. We looped through the tracks in the dataset, extracted the needed stems, and stored them as mixture.wav and vocals.wav files. We used the stempeg.write.write_audio function to accomplish the task [7].

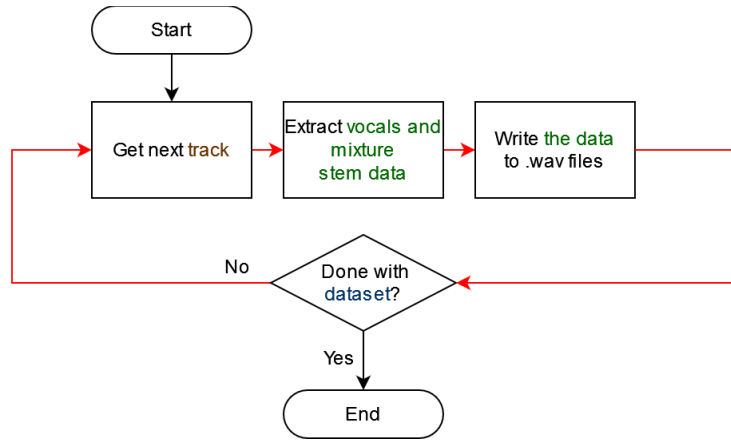


Figure 4: MUSDB data processing flow chart

3: MedleyDB 2.0 Dataset Collection and Preprocessing

We obtained the MedleyDB 2.0 dataset as a zip file on its website [6]. The data structure differs greatly from the desired structure. We made the following observations:

1. The mixture audio is readily provided as a `_MIX.wav` file.
2. The vocals audio is provided as one or several `_STEM.wav` files. We need to sum all vocal stems into one file.
3. We ignored the RAW files, for they are further broken down components of STEMS.

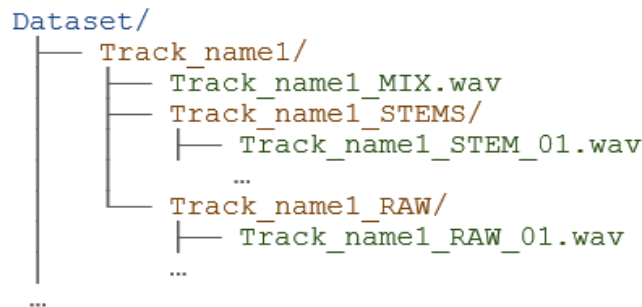


Figure 5: MedleyDB 2.0 data structure

In addition, the labels are not embedded in the dataset, but instead are provided as metadata files on the Github page [8]. The structure is as shown below. We made the following observations:

1. The files are in yaml format.
2. There are metadata files for both MedleyDB 1.0 and 2.0, which the former we did not use.

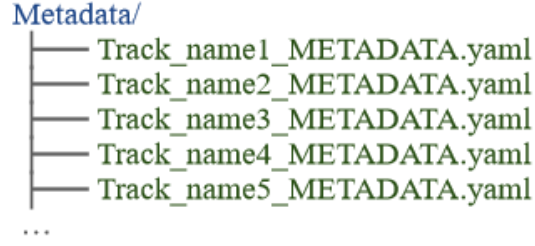


Figure 6: MedleyDB 2.0 metadata structure

To parse the metadata files and extract the needed stems, we followed a procedure similar to that of the MUSDB preprocessing, except we looped through the metadata files instead of the dataset. We found the corresponding MIX and summed STEM files, and wrote them into mixture.wav and vocals.wav. We use the PyYAML module to parse the metadata files [9]. We used `scipy.io.wavfile.write` function to write data to the files [10].

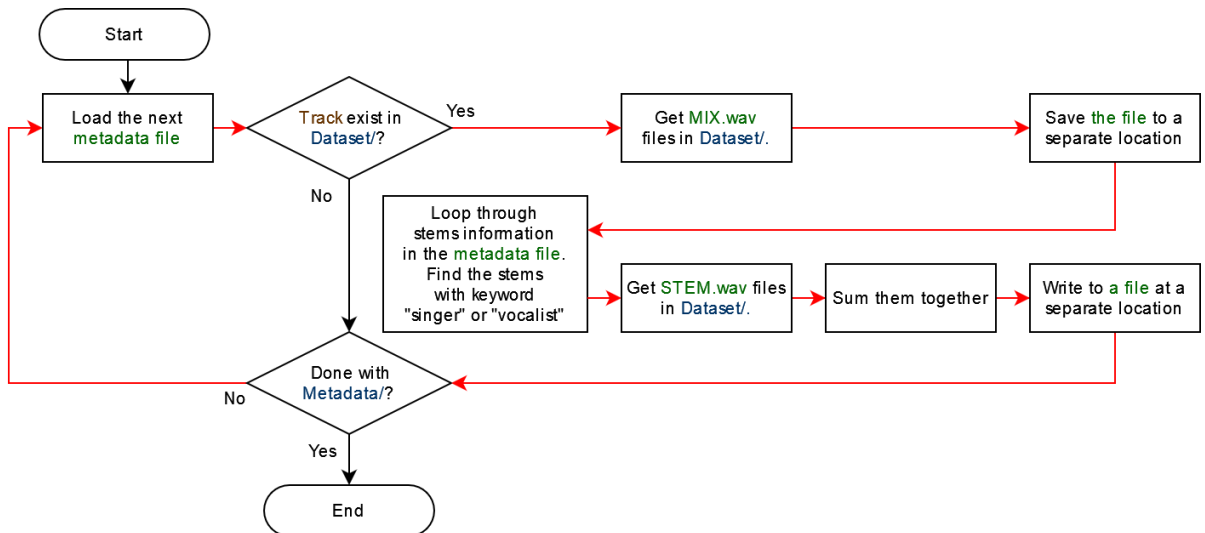


Figure 7: MedleyDB 2.0 data processing flow chart

IV. ARCHITECTURE

Problem Scoping

Several scopings of this project were discussed. These affected architecture, and included whether to:

- Restrict to a single genre of music, or support many.
- Restrict solely to music isolation, or include 'non-music' audio.
- Isolate a single type of audio (vocal), or support many targets.

Ultimately, we chose to focus our work on specially all genres of music. Further, we statically trained for vocal-isolation, instead of non-musical noise cancellation¹.

Model

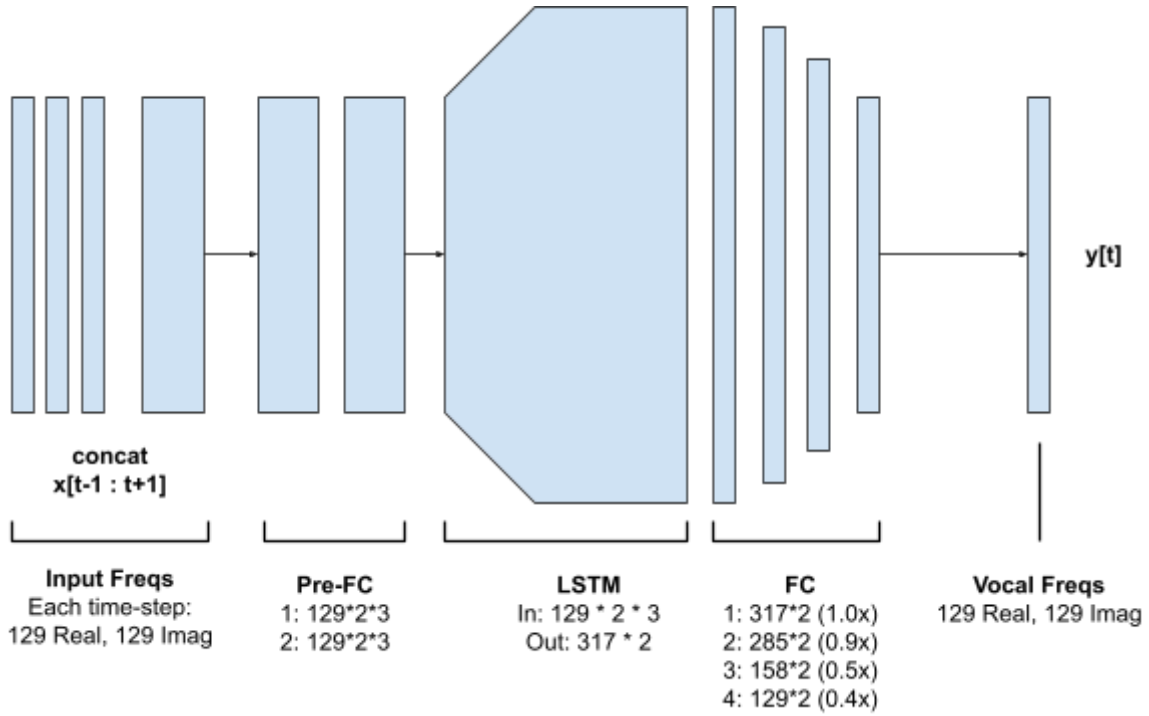


Figure 8: Model architecture.

Audio consists of variable-length temporal data, which suggests the application of an RNN. Our approach uses a bi-layer LSTM as its foundation.

An input track consists of N samples, 44100 per second of audio. Due to the high sample-rate, it is notoriously difficult to train in the time-domain. As such, we group samples into *windows* and perform fourier-transforms on each (collectively known as *short-time-fourier-transform*). This generates a vector of frequency magnitudes at each window-step (henceforth called time-step for simplicity).

¹ Although, as an unintended consequence, the model has shown itself to be quite effective at isolating spoken-word from unrelated background music.

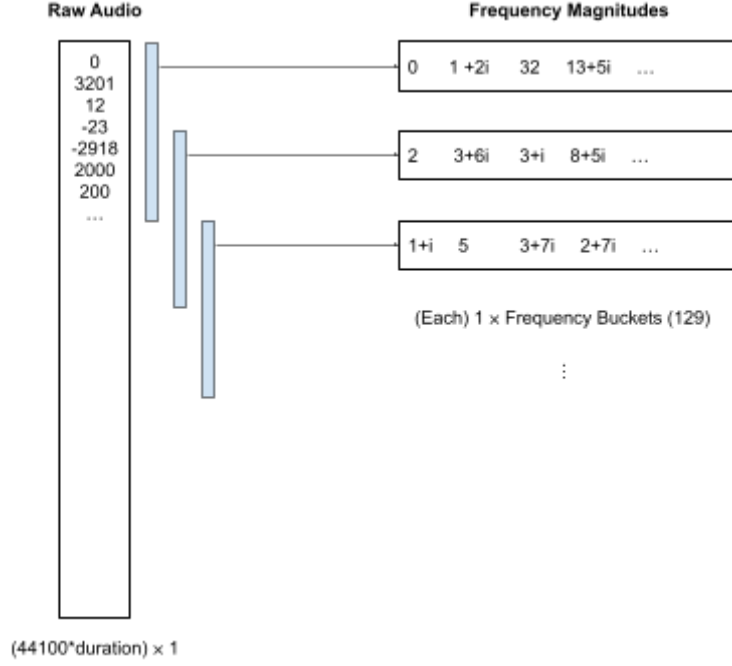


Figure 9: Raw audio to frequency-magnitude vectors

These frequency-magnitudes are fed to the LSTM, which extracts features from the audio at each time-step. These unknown features are passed through a 3-layer FC network to recover an analogous frequency-magnitude vector, per time-step, representing a prediction for the vocals alone.

Improvement 1: Awareness

Despite windowing, audio is a sparse form of data. A given frequency can belong to both instruments and the human voice, and without context it is impossible to tell which.

As such, we increased available context through *awareness*: providing multiple timesteps of input (frequency-magnitude vectors) at each timestep of prediction.

Improvement 2: Pre-FC

Building on the sparseness problem, we had success inserting an additional FC network ahead of the LSTM. This served as a way to "mix" the input-frequencies, effectively generating feature-crosses within the same dimensionality.

Complex Data

Another complication of audio data is complex numbers. Fourier-transforms produce vectors containing both real/imaginary components in each element. As Pytorch does not natively support complex-operations, some indirection is needed.

After exploring various approaches, we settled on doubling all input/output dimensions. That is, if our model originally consumed/generated 100 input frequency-magnitudes per timestep, it would now handle 200 (half real, half imaginary). Internally these values are always real, and are recombined as complex during post-processing. By converting our ground-truth "labels" to this representation, loss of gradient-data is avoided.

V. BASELINE MODEL & RESULTS

Baseline Model

We used the non-machine learning approach Blind Source Separation, BSS, using Independent Component Analysis, ICA [1]. The combination of the sound signals in an audio source is a linear mixture problem, Figure 10.

We followed Shlens' paper [1] and Shankar's article [11] that is a summarization of Jonathan Shlens' paper to implement ICA.

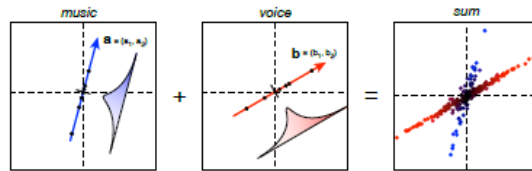


Figure 10: The sum of the two sounds show how the mixture is linear [1]

ICA assumes the vocals and music are independent of each other. Such linear mixture can be written as:

$$\mathbf{x} = \mathbf{A}\mathbf{s} \quad (1. \text{ Linear Mixture})$$

Where,

- \mathbf{A} is an unknown invertible, square matrix that mixes the components of the sources
- $\mathbf{s} = s_1(t), s_2(t), s_3(t), \dots, s_n(t)$ are the independent sources

Figure 11: Equation solved by ICA to separate vocals from music [11]

ICA's aim is to solve for A . We used matrix manipulation techniques to solve for s , given x , the linear mixture, and the solved A . The output of ICA is s . See Appendix A for the ICA implementation code copied from our Google Colab.

Baseline Results

According to Figure 12, the waveforms of the separated vocal stem from our model (prediction waveform) is closer to the ground truth waveform compared to the baseline waveform. Such comparison proves that our model produces better results than our baseline model.

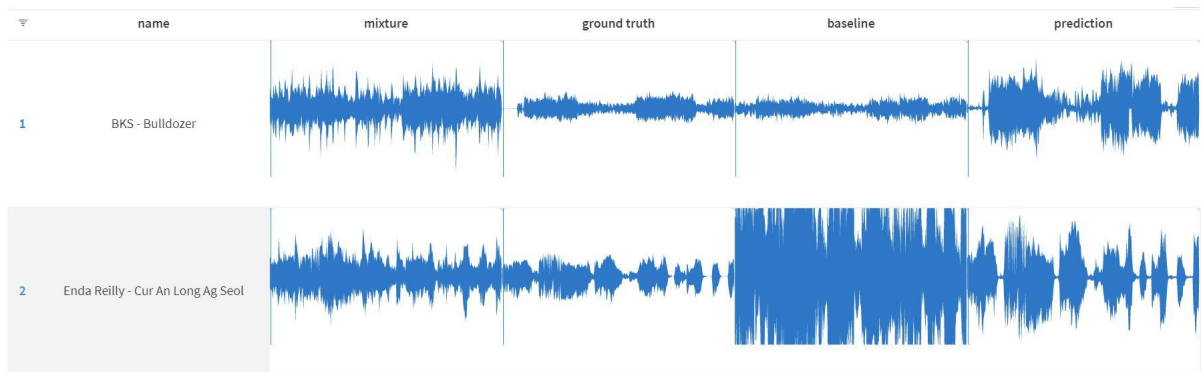


Figure 12: Comparison of the results from our model and the baseline for 2 different audio tracks

VI. MODEL RESULTS

1: Quantitative Result

Loss

Loss values were chosen as a metric for our quantitative results that were computed using Contrastive-Cosine Distance Loss as shown in Section X.

According to the various validation loss curves of different models in Figure 13, dandy-paper-201 has the minimum loss. As a result, this was chosen as our final model.

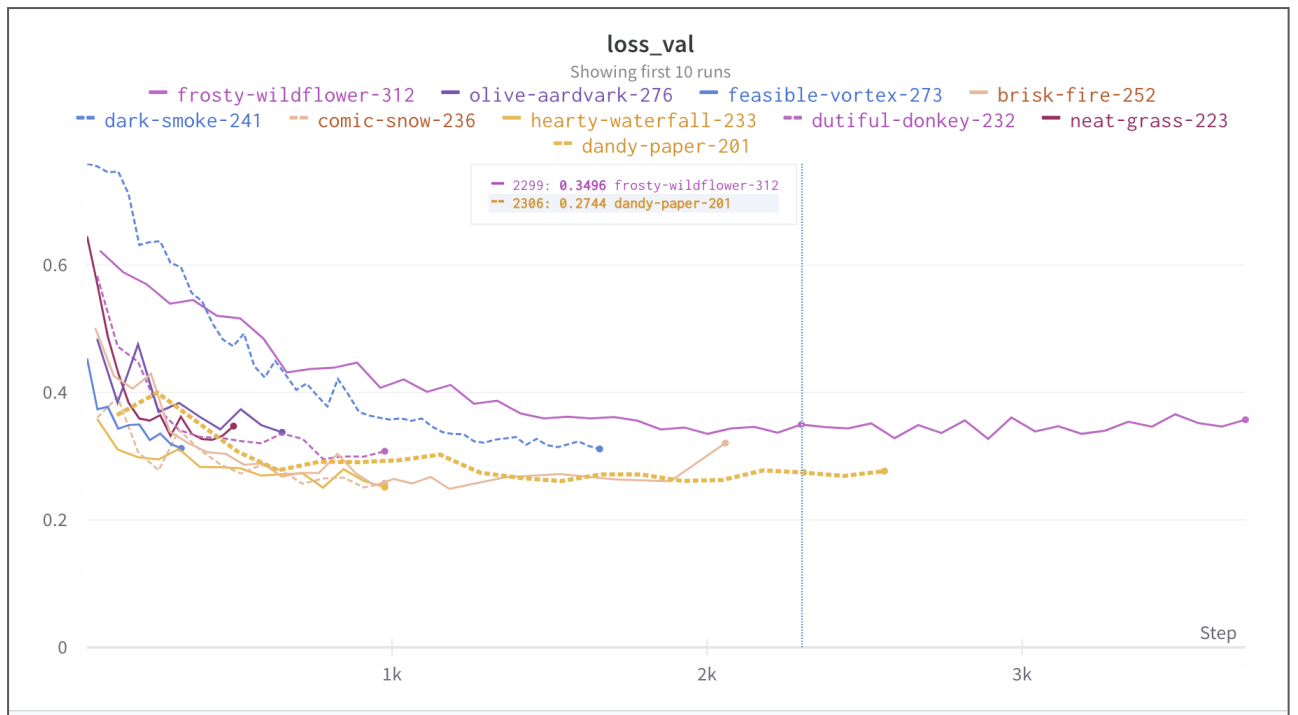


Figure 13: Plot of Validation Loss vs Number of Steps. Each curve represents a different Model

The training and validation losses of our final chosen model is shown in Figure 14. The validation and training loss are around 0.27 and 0.15 respectively.

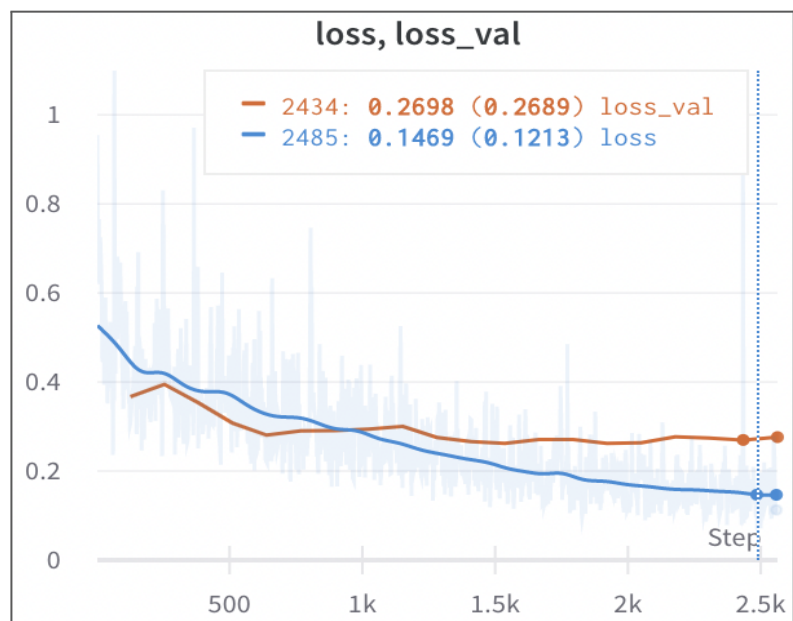


Figure 14: Plot of Loss vs Number of Steps. Orange depicts validation loss curve and blue depicts training loss curve.

Accuracy

The accuracy of the model was calculated based on the metric of votes for the three best candidate models shown in Table 1. These candidates were the models that produced three least losses according to Figure 13. Each team member listened to the multiple tracks' output produced by each of these chosen candidates. Based on their judgement of the output's quality, each member voted for the best candidate.

Table 1: Number of votes received for each chosen candidate


Candidate/Model	Number of Votes Received
hearty-waterfall-233	0
frosty-wildflower-312	1
dandy-paper-201	3

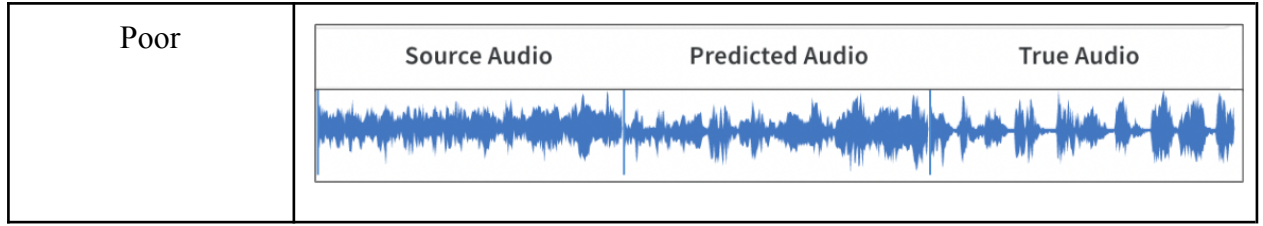
According to Table 1, dandy-paper-201 received the highest number of votes and was chosen as our final model.

2: Qualitative Results

The best way to illustrate the sample outputs (See APPENDIX III, link to the audio clips) is to show the waveforms of the audio clips. We have three waveforms for each sample shown in Table 2: S (music), P (model's predicted vocal stem) and T (ground-truth vocal stem).

Table 2: Waveforms of two Samples (not seen by the model before)

Prediction Quality	Waveforms (S, P, T)
Good	<div><div>Source Audio</div><div>Predicted Audio</div><div>True Audio</div></div>



According to Table 2, we can clearly see a significant difference between the S_{Good} and P_{Good} waveforms. This indicates that our model was able to suppress the sounds of the instruments. Moreover, we observe that P_{Good} is very similar to T_{Good} . However, P_{Poor} and T_{Poor} are not very similar.

Furthermore, from Figure 14, we can clearly see that the training and loss curves do not perfectly coincide. This suggests that the model can potentially produce a poor quality vocal stem. Nevertheless, it is important to mention that, as we will see in [Section VII](#), our model produces a high quality of isolated vocal stems from a mixed stereo/music in most of the cases.

Finally, upon further analysis by inputting various songs, we found out that our model works the best with the songs that have:

- High Cadence Beats
- Distinctive Instrumental Melodies































VII. EVALUATION OF MODEL ON NEW DATA

We evaluate the accuracy of our model by using ten new tracks, which were reserved from the dataset before the training to make sure our model has never encountered them.

According to waveforms shown in Table 3, nine of the ten separated tracks are showing shapes almost identical to the ground truth, indicating the good quality of the results². Track 6, however, differs from the expected results, showing its poor quality. Thus, we can conclude that the model produces a high quality of isolated vocal stems in most of the cases.

² We have observed this to be a good proxy for audio quality, and have validated all claims auditorily.

Table 3: Source, Predicted and True Waveforms of 10 songs unknown to the model

ID	Waveforms		
	Source Audio	Predicted Audio	True Audio
1			
2			
3			
4			
5			
6			
7			
8			
9			
10			

VIII. DISCUSSION

According to Tables 2/3 it can be concluded that overall, our model can isolate vocal stems from audio tracks with high qualities as the ground truth and predicted audio waveforms are very similar.

It is surprising that the mixtures with more complicated instruments showed better results. Our expectation was to observe better results with mixtures consisting of simple instruments. However, the results proved otherwise. We conjecture that rapid instrument sound was captured more easily by the model, rather than slow instrument sounds. Even with windowing, there are ~200 samples per time-step, which may get lost in the LSTMs memory for slower beats.

According to our results the model did not do as well on low cadence beats. This is because lower sound modulation results in more similar frequencies that will possibly overlap. Therefore, the model will not be able to completely isolate the vocals in such scenarios.

We expected our model to isolate vocals from music and background sounds. However, it exceeds our expectations with regards to the following observations:

- It understands vocal echos and reverb, and isolates them when they are surrounded by instrument sounds.
- It isolates backing vocals and different singer voices, genders, etc, even within the same track.
- It is able to eliminate music even for segments with no vocals.

Overall, our results show the model performs very well for most tracks and exceeds our initial expectations and goals.

IX. ETHICAL CONSIDERATIONS

1: Copyrights and Licence

All data we used is licensed under Creative Commons (BY-NC-SA) [12]. Therefore we do not hold the copyrights to any of the audio processed by our neural-network. Moreover, the audio produced must not be used for commercial purposes [13].

2: Inappropriate Use

The user should not use our model for any malicious intent. For example, when the background audio is essential for providing context to the vocal track, separation of tracks may cause misinterpretation of the speaker/vocalist's intention.

X. PROJECT DIFFICULTY

Although our results far exceeded our initial expectations, working with audio was far more difficult than expected, compared to other mediums. This difficulty falls into three categories, described below.

1: Domain Knowledge

Firstly, it requires domain-expertise to reason with. Raw audio data is large and sparse (1000s of samples must be considered simultaneously for meaningful structure, due to Nyquist sampling [14]). As such, signal processing techniques such as STFT and ratio-masking must be developed before modelling. Even the baseline model used is complex, and requires non-trivial research to understand and implement.

This also reduces debuggability/interpretability during development. We understand audio in the time-domain, while the model understands in frequency-domain. This makes it difficult to understand/address issues during training. Furthermore, while perceptive-losses are meaningful in the time-domain, these cannot be used, as conversion from frequency to time destroys gradients.

2: Generative Modelling

In a given audio track, there is significant overlap between instrumentals/vocals. At a single time-step, each frequency has contributions from both.

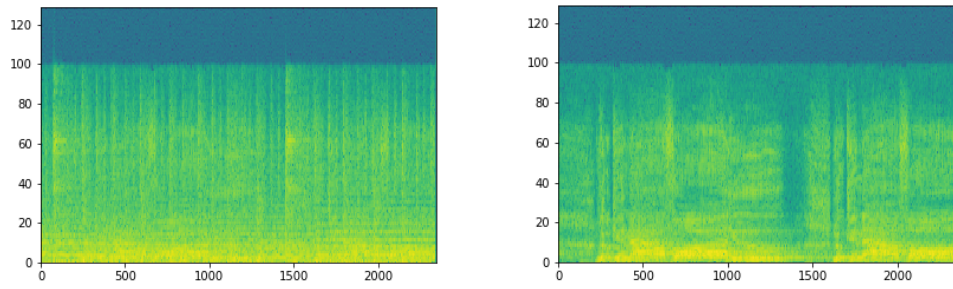


Figure 15: Signal overlap with spectrum of instruments (left) and vocals (right). Simple filtering is impossible, and context-driven generation is needed.

Thus, isolating vocals is a truly generative task: given a track, *generate* (vs filter) a vocal track.

A difficulty of generative models is imprecise loss/accuracy. First, loss must be evaluated at a per-time-step basis, while audio only gains meaningful structure across *many* timesteps. Second, audio perception is inherently subjective. Any metrics used to compare tracks are simply *proxies* for how similar they will be *perceived* as.

After experimenting with several approaches, we developed *contrastive-cosine-distance* as our loss function.

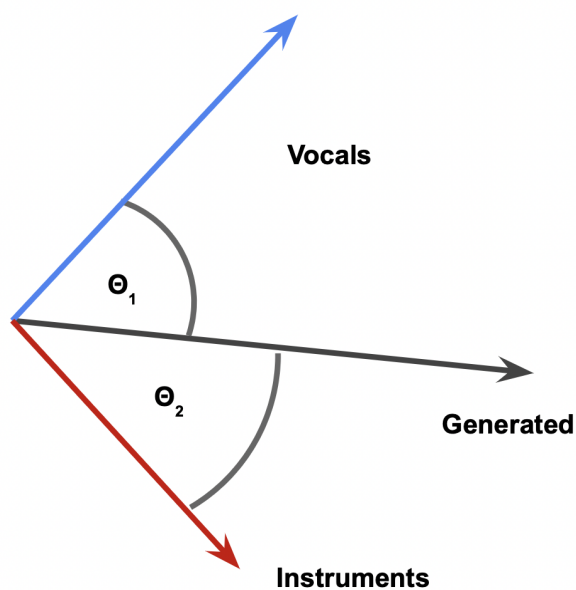


Figure 16: Contrastive cosine distance, in 2 dimensions.

In this approach, we treat frequency-magnitudes as N-dimensional vectors in some space, and minimise the angle between generated/vocals. During development, we realised that given track X and vocal Y, instrumental Z could be recovered - and its angle *maximised*. Then, we contrastively minimise one angle, while maximising the other.

3: Limited Datasets

Music is a prolific dataset, providing an infinite dataset for evaluating our model in post. However, training requires *labels*, in this case isolated vocals. This is both rare (i.e less samples than e.g ImageNet), and impossible to generate³.

As such, we had to maximise the value extracted from our data through data-augmentation techniques.

Augmentation 1: Loopback

During initial experiments, the model became good at separation when *both* vocal/instruments were present in the input, but had trouble in sections with only one. This was interesting, as it could indicate the model is *actually* separating foreground/background, melody/accompaniment, etc, rather than vocals/instruments.

We solved this through a method we developed⁴ called *Loopback*, illustrated below.

³ Versus something like image classification, where labels/bounding-boxes could be created.

⁴ It's unclear whether this is already standard practice, and if so, we do not claim novelty. However, it was developed here without outside influence.

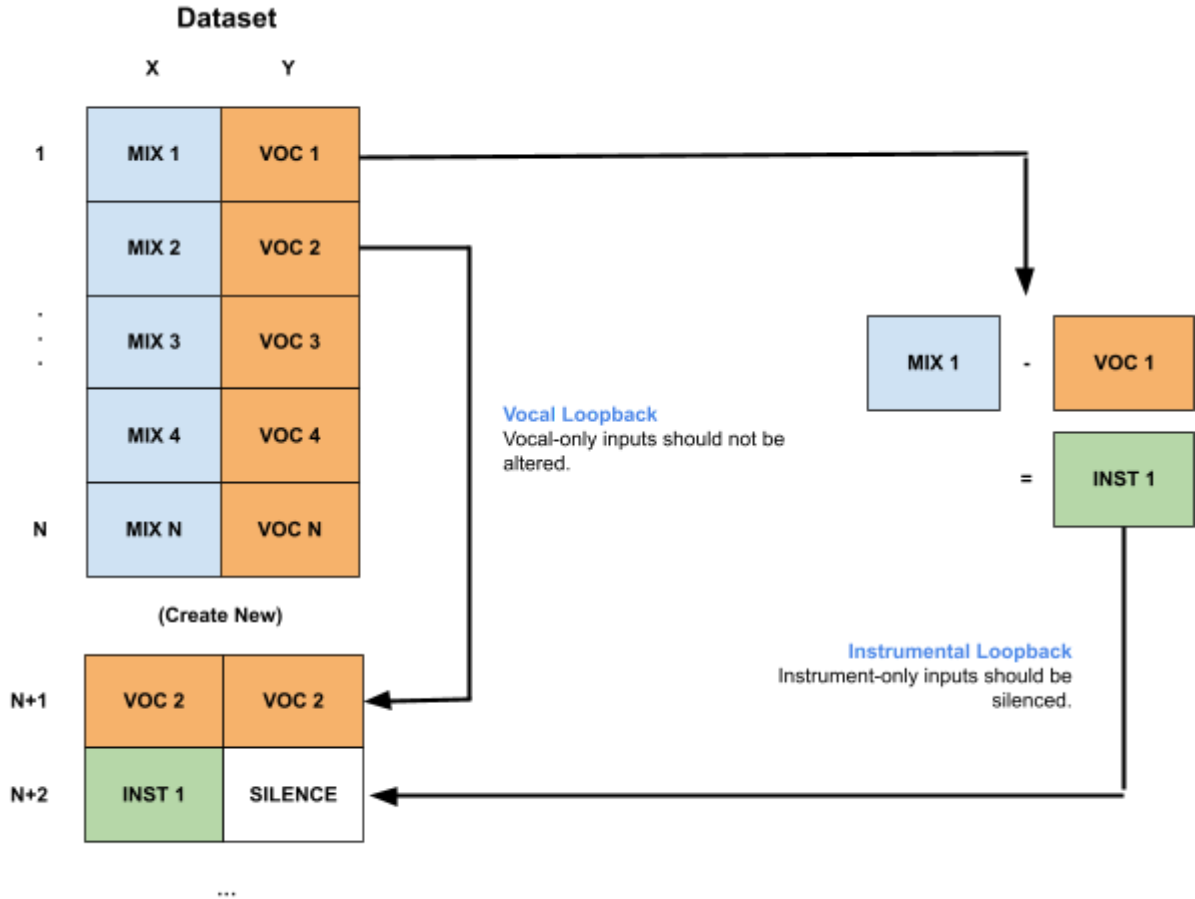


Figure 17: Dataset expansion through *Loopback*. Shown for two tracks, but performed against X% of all tracks.

Loopback can only be performed *after* the train/test/val split is created, else contamination is possible. Furthermore, the system is very sensitive to the proportion of tracks looped-back.

- **Excessive vocal-loopback:** biases towards parroting⁵.
- **Excessive instrumental-loopback:** biases towards silence.

We found 10% loopback to be optimal.

Augmentation 2: Noise Generation

Additionally, we wanted to prevent overfitting and generalise results. One strategy was the creation of noisy versions of each input track, mapping to the same output.

⁵ Mirroring input to output with no alteration.

- During each batch, multiple noisy versions of each track were created on the fly using additive-white-gaussian-noise [15].
- N white-noise profiles were generated for each of M variances (higher = more noisy), resulting in $N*M$ additional tracks/input.
- Generated per-batch, *these were unique between epochs*. In a 100-epoch run, this meant training on a dataset (effectively) $100*N*M$ times larger than our initial.

The limiting-factor was GPU-memory, which forced a compromise between batch-size and extent of noise-generation. Finally, on-the-fly generation protected against dataset-contamination.

REFERENCES

- [1] J. Shlens, *A Tutorial on Independent Component Analysis*. Mountain View, CA: Google Research, 2014.
- [2] “FacebookResearch/Demucs: Code for the paper hybrid spectrogram and waveform source separation,” *GitHub*. [Online]. Available: <https://github.com/facebookresearch/demucs>. [Accessed: 09-Feb-2022].
- [3] “# MUSDB18,” *SigSep*. [Online]. Available: <https://sigsep.github.io/datasets/musdb.html#musdb18-compressed-stems>. [Accessed: 13-Apr-2022].
- [4] “F90/Wave-U-Net: Implementation of the wave-U-net for audio source separation,” *GitHub*. [Online]. Available: <https://github.com/f90/Wave-U-Net>. [Accessed: 09-Feb-2022].
- [5] A. Défossez, *Hybrid Spectrogram and Waveform Source Separation*. 2021.
- [6] *MedleyDB*. [Online]. Available: <https://medleydb.weebly.com/>. [Accessed: 13-Apr-2022].
- [7] “Module stempeg.write,” *stempeg.write API documentation*. [Online]. Available: <http://faroit.com/stempeg/write.html>. [Accessed: 13-Apr-2022].
- [8] Marl, “Medleydb/medleydb/data/metadata at master · Marl/MEDLEYDB,” *GitHub*. [Online]. Available: <https://github.com/marl/medleydb/tree/master/medleydb/data/Metadata>. [Accessed: 13-Apr-2022].

- [9] "PyYAML," *PyPI*. [Online]. Available: <https://pypi.org/project/PyYAML/>. [Accessed: 13-Apr-2022].

- [10] "Scipy.io.wavfile.write," *scipy.io.wavfile.write - SciPy v1.8.0 Manual*. [Online]. Available: <https://docs.scipy.org/doc/scipy/reference/generated/scipy.io.wavfile.write.html>. [Accessed: 13-Apr-2022].

- [11] Gowri Shankar, "Cocktail party problem - eigentheory and blind source separation using ICA," *Gowri Shankar*, 18-Jul-2021. [Online]. Available: <https://gowrishankar.info/blog/cocktail-party-problem-eigentheory-and-blind-source-separation-using-ica/>. [Accessed: 13-Apr-2022].

- [12] "SigSep Audio Tracks." <https://github.com/sigsep/website/blob/master/content/datasets/assets/tracklist.csv>, 27-Aug-2018.

- [13] "Creative Commons License Deed," *Creative Commons - Attribution-NonCommercial-ShareAlike 4.0 International - CC BY-NC-SA 4.0*. [Online]. Available: <https://creativecommons.org/licenses/by-nc-sa/4.0/>. [Accessed: 09-Feb-2022].

- [14] C. E. Shannon, "Communication in the Presence of Noise," in *Proceedings of the IRE*, vol. 37, no. 1, pp. 10-21, Jan. 1949, doi: 10.1109/JRPROC.1949.232969.

- [15] Mathuranathan, "Simulate additive white gaussian noise (AWGN) channel," *GaussianWaves*, 21-Nov-2020. [Online]. Available:

<https://www.gaussianwaves.com/2015/06/how-to-generate-awgn-noise-in-matlab-oc-tave-without-using-in-built-awgn-function/>. [Accessed: 13-Apr-2022].

APPENDIX I. BASELINE MODEL

Implementation Code of ICA from our Baseline Model

```
def ICA(X, epochs = 5, learning_rate = 1e-5):

    # shape of the input
    shape_ = X.shape[0]

    # create the weights matrix with the same dimensions
    weights = np.zeros((shape_, shape_), dtype=X.dtype)
    for i in np.arange(shape_):

        # initialize the matrix to random values
        w = np.random.rand(shape_)

        # iterate for the number of epochs
        for epoch in np.arange(epochs):

            # run the objective functions
            w_new = W_new(w, X)

            # train the weights
            if(i >= 1):
                weights_i = weights[:,i]
                w_new = w_new - np.dot(np.dot(w_new, weights_i.T), weights_i)

            # calculate the improvement of the model
            improvement = np.abs((w * w_new).sum()) - 1
            improvement = np.abs(improvement)

            weights_updated = w_new

            # if the learning rate is achieved
            if(improvement < learning_rate):
                # done looping, learning rate achieved
                break;

            weights[i, :] = weights_updated

    # s = Wx
    seperated_tracks = np.dot(weights, X)

    # return s (s[0] is the source track, s[1] is the seperated vocals)
    return seperated_tracks

seperated_tracks= ICA(x_whitened, epochs=100)
print(seperated_tracks.shape)
wf.write('source.wav', sampling_rate, seperated_tracks[0].astype(np.float32))
wf.write('vocals.wav', sampling_rate, seperated_tracks[1].astype(np.float32))
```

Figure 16: Implementation code of ICA from our baseline model

APPENDIX II. CODE

Our code was organised into a series of Colabs.

Baseline Model

https://colab.research.google.com/drive/1Sx2agKtAQsXB4TNHdc5YwwD3kJ9wJe_I?usp=sharing

Data Sync

Format audio tracks for training, and import into W&B.

<https://colab.research.google.com/drive/1BUvPC02FQuX7fFd8SQtCMgH1rUn5zxWA?usp=sharing>

Toy Model

Used to build an intuition around RNN-type models, capacity, training-time, etc.

<https://colab.research.google.com/drive/185Kcdev5Gp8bi813BxpfaWgv0WwmtHwt?usp=sharing>

Final Model

<https://colab.research.google.com/drive/1RxOUjELuJFQPWm60o-qgQiz3tNnpfc6g?usp=sharing>

Fourier Transform Experiments

<https://colab.research.google.com/drive/1x--Czvai51Ecbxz3rUZ8NEtceaygN22?usp=sharing>

Spectrogram & Ratio Masking Experiments

<https://colab.research.google.com/drive/1VFtaGgWflutAa4U8mwxQe-amj1W-clwH?usp=sharing>

MUSDB Preprocessing

<https://colab.research.google.com/drive/1cJ17M4NOj33VOsdxE6Mecag9KA-S0Jo2?usp=sharing>

MedleyDB Preprocessing

https://colab.research.google.com/drive/1WmCfEKYQetWc35pwUprNxO4t7_YQBsTw?usp=sharing

Final Results Table Generation

<https://colab.research.google.com/drive/1-PckAgGDNoEf8-D8xSy0ak7hzEhPk4P8?usp=sharing>

APPENDIX III. SAMPLES PRODUCED FROM THE MODEL

Sample Outputs from the ML Model and Baseline along with the Ground Truth

https://wandb.ai/aps360/visualizations/reports/Results--VmldzoxODM2MTE5?accessToken=jwbxasf8l3l7ztc8aqn6grf2pv5fak0v6pxzq2ve8v7s1p6uqy8pfbleqzenu9qs&fbclid=IwAR2Qx-bTpKB5YXQjJgmk_00eOFJqyxoW1MZadIUqh_0JW4-ol4avJb8mgQ

Sample Outputs with more Results

<https://wandb.ai/aps360/visualizations/reports/Large-Result-Set--VmldzoxODM2MjI5?accessToken=n71amx8606pwd4633615v2gnpps1ly9ovd3e0goa7dygy6tybgfaxl2ot0daf0i7&fbclid=IwAR0GHHoD3f4AjsxDtu3uqcKLN4DHDPVWnHus5kPPQO157dr1FCIRobK4tkY>