Audio Source Separation via Convolutional Neural Networks

Henry Hamilton
hjhamilt@uci.edu

Pranav Banuru
pbanuru@uci.edu

Sojung Yun
sojungy2@uci.edu

Melodye Nguyen
melodyen@uci.edu

Keisun Luc
knluc@uci.edu

CompSci 172B: Neural Networks & Deep Learning
University of California, Irvine
June 2023

1 Introduction

The problem of audio source separation is defined as isolating an audio of interest in an environment of many other noises and stimuli. Audio source separation has many practical uses such as: noise cancellation during phone calls through a phone or headphones, hearing aids for the hard of hearing/deaf, and more. According to the Signal Separation Evaluation Campaign (SiSEC), a regular event that evaluates the current progress of source separation through participants' algorithms, in 2018 said the TAU1 algorithm is deemed the most successful in separating the voice stem from 13 professionally mixed songs. However, there was a certain amount of dependence on song, suggesting there are some improvements to be made to generalize across genres. Our goal in this project is to focus on the separation of vocals from a mixed track [6].

2 Data

The dataset we used was MUSDB18[3], a dataset containing 150 full-length music tracks of different genres. It also includes each tracks' isolated instrument stems: vocals, bass, drums, and a group of other instruments. The dataset consists of 100 tracks for training and 50 tracks for testing. The SiSEC used this dataset for the event and tested the algorithms on separating just the vocals.

We also worked with a smaller version of this dataset, denoted as MUSDB18-7. This dataset consists of a 7 second clip from selected songs from the full dataset.

In both datasets, the data is stored in mp4 files, and each song has a sample rate of 44100. The repository which contained this dataset also contained some basic functions for converting the mp4 files into waveform data. See (1) for a visualization of a single audio file.

During our research, we examined a number of different models, each of which required the input data to be in different forms. Thus, each architecture described below will contain its own data cleaning and extraction section for improved readability.

37th Conference on Neural Information Processing Systems (NeurIPS 2023).

3 Methods & Architectures

3.1 U-Net

Data Preprocessing

The raw audio signals, `X_train_pre`, were transformed into spectrograms utilizing the Short-Time Fourier Transform (STFT) from the `librosa` library. STFT breaks down an audio signal into small, possibly overlapping segments, and then computes the Fourier transform on each, transitioning our one-dimensional audio data into a two-dimensional representation. The complex numbers resulting from the STFT were converted to magnitude values to simplify interpretation and further transformed into decibels (dB) for practicality, given the wide dynamic range of audio signals.

Data Resizing

The spectrograms obtained varied in size due to the different lengths of the original .wav data. To uniformly feed these into a neural network, the `scipy` library's `zoom` function was used to resize each spectrogram to a standard shape. Each spectrogram was resized to have 256 frequency bins and 128 time steps, striking a balance between resolution and computational efficiency.

However, this resizing method has limitations. Longer audio signals are compressed into the same shape as shorter ones, potentially leading to loss of information, feature distortion, and inconsistency. The opposite effect is also true for shorter audio signals, which get stretched.

Model Construction

Our model utilized the U-Net architecture, a convolutional neural network commonly used for image segmentation. It consists of an encoder, a decoder, a bottleneck, and skip connections. The encoder progressively downsamples the input, the decoder progressively upsamples the input, and the skip connections concatenate upsampled features with corresponding encoder features to help in fine detail reconstruction. It takes in spectrograms, and outputs spectrograms.

Model Training

The model was trained in a supervised manner using the Adam optimizer and the `mean_squared_error` as the loss function. Early stopping was implemented for efficient training, with a patience of 90 for the first 600 epochs and 30 for the subsequent 400. The model reached a training loss of ~52.3594 and a validation loss of ~112.2233 after approximately 1000 epochs.

Results

The model was evaluated by making predictions on the test dataset. An example of one prediction spectrogram is displayed here: (2).

In this predicted spectrogram, our model appears to have captured significant features. However, it could be misleading as the y-axis is divided into 200 uniform bins, which doesn't align with how humans perceive frequencies. We addressed this by transforming the y-axis to a logarithmic scale ranging from 0 Hz to 16,384 Hz. This is shown here: (3).

This visualization better mirrors human pitch perception. Even so, noise is present in the prediction and the Griffin-Lim algorithm, which we used to convert spectrograms back into audio, resulted in distorted output due to our resizing method. The model's testing loss, based on mean squared error (MSE), was 135.07.

3.2 Wave-U-Net

Data Preprocessing

MUSDB contains mp4 audio files with left and right sound channels. However, our AI model requires one-dimensional arrays of audio data. To address this, we performed the following preprocessing steps:

1. Audio Extraction and Splitting:
    - We separated the left and right sound channels from mp4 files. As a result, we were able to obtain two independent audio signals that could be processed independently.

2. Conversion to WAV Format:
    - We converted the isolated audio streams to WAV, which is an uncompressed, widely used audio format. This conversion ensured compatibility with the subsequent preprocessing steps and enabled the transformation into the desired 1D array format. Using WAV format, the mono audio files obtained from splitting the stereo audio were transformed into 1D arrays.

We converted the MUSDB dataset into a suitable format for training our AI model after performing these preprocessing steps. A 1D array is created from the transformed audio files, allowing for further analysis and training.

Model Construction

Wave-U-Net extends the U-Net architecture for audio source separation. It utilizes the U-Net's ability to capture context and employs skip connections for localization. Here are the components of the model:

- **Encoder Block:** Uses Conv1D layers to downsample the input, introducing non-linearity with LeakyReLU activation. Stores intermediate feature maps in 'skips'.
- **Decoder Block:** Upsamples the encoded output by performing upconvolution with Conv1D layers. Recovers spatial resolution by concatenating feature maps.
- **Wave_U_Net:** Creates the overall model by defining the input layer and calling the encoder and decoder block functions. TensorFlow Keras model returned.

Wave-U-Net adapts the U-Net's architecture to audio source separation by leveraging its architecture. It incorporates encoder-decoder paths, skip connections, and appropriate activation functions to provide high-quality separation results.

Results

Model was trained using supervised learning with MSE loss function. The training lasted approximately 200 epochs with 32 batches. Validation splits of 0.2 were used for evaluation. To prevent overfitting, early stopping was implemented with a patience of 20. A Keras callback function plotted real-time training and validation losses. In the results, the training loss is 0.0059 and the validation loss is 0.0061. It is unfortunate that the training did not result in the desired results despite these efforts.

Problems of Our Wave-U-Net

The Wave-U-Net model and training process may have several potential problems. The following are some areas where we may make mistakes:

- **Data Processing:** Audio data may require resampling and normalization. Resampling ensured consistency between the datasets by converting audio to a common sample rate. In addition, we might need to normalize the audio signals to bring their amplitudes into a standardized range. By normalizing the audio intensity, we prevent any bias toward certain intensities and allow the model to learn better patterns.
- **Model Architecture:** At first glance, the Wave-U-Net model appears reasonable. Depending on the task, it may be necessary to adjust the model based on the number of filters, kernel size, and layer configuration.
- **Loss Function:** A mean squared error (MSE) loss function is used for this model. A loss function like this is commonly used for regression tasks. Depending on the problem, it might not be the best solution.
- **Training Paramenter:** There might be a problem with the number of epochs, batch size, and learning rate. A validation split of 0.2 was used when calling fit. In other words, 20% of the training data will be used for validation. The split should have been checked to make sure it is appropriate for our dataset.

3.3 Convolutional Neural Networks - Model A

Data Preprocessing

As was stated in a prior section, the audio data initially had a sample rate of 44100. we possessed limited computational power, so we downsampled this audio to 11025. This makes each clip 4 times

smaller, while retaining the general features of the music.

The `librosa` library was used to convert the downsampled waveform data into Mel spectrograms. The details behind the Mel spectrogram is beyond the scope of our research, but the crucial thing of note is that the spectrogram is a 3-dimensional array. The x-axis represents time, the y-axis represents frequency, and the z-axis represents the amplitude of sound. However, from my observations, the range of the amplitudes were rather small, so the image produced was faint and difficult to understand. Thus, an additional function was used to convert the z-axis of the spectrogram from amplitude to decibels. An example of an amplitude spectrogram and a decibel spectrogram (4) for the same song are shown below.

The output of this model is intended to be a binary mask of the same size as the input spectrogram. The mask would contain a `1` to denote the existence of vocal sound and a `0` to denote the absence of vocal sound, for every frequency at every time unit. To create this mask, the vocal spectrogram is first normalized to have values from `0` to `1`. Then any values which exceed the average for the clip are set to `1`, while others are set to `0`. An example of a vocal spectrogram and its corresponding mask (5) are shown below. The masks, when applied to the mix, should isolate the vocals by setting all non-vocal pixels to `0`.

Model Construction

The initial model, which will hereby be referred to as "Model A" is a basic CNN which takes the full mixture spectrograms and attempts to output the binary masks for each input. The full architecture can be found at (6). It is important to note that the inputs for this model come from a subset of the MUSDB18 dataset, called MUSDB18-7. These clips are all exactly 7 seconds long. This was done because the model cannot accept inputs of varying size. There are 80 examples total. The loss function used was binary cross entropy with an adam optimizer. Training was done with 700 epochs and a batch size of 32.

Results

(7) shows the loss curve for the model described. It seems to function quite well, and from the validation loss, we can see the model doesn't appear to be overfitting either. To determine the accuracy of this model, we multiplied the output masks with their corresponding amplitude mix spectrograms, which isolated the vocals. The `librosa` library was then used to reconvert the vocal spectrogram into waveform data. Finally, each predicted vocal waveform was compared with the true vocal waveforms from the dataset and MSE was calculated. This eventually resulted in a MSE of 0.34.

3.4 Convolutional Neural Networks - Model B

Data Preprocessing

For the second CNN model, the data was cleaned similarly to the first, following all the steps detailed in the previous "Data Preprocessing" section. However, for this second model, there are few extra steps. After obtaining the spectrograms and masks, each spectrogram is then broken down into ~0.5 second chunks. For each of these smaller clips, a mask of the middle frame is created. (13 time frames roughly translates to ~0.5 seconds, so the mask is for frame 6). This is the new input and output. To visualize it better, view (8). Splitting clips into smaller ones increased the number of training examples up to ~22000. The reason for the large amount of training data also stems from the fact that the full dataset may be used now, rather than the 7 second one.

Model Construction

This new model, labeled "Model B", uses a modified version of the previous architecture. Now, it outputs a 1D array rather than a 2D one, so it uses a Flatten layer to perform this task. There are also additional convolutional layers. The full architecture is detailed at (9). The optimizer was adam once again and binary cross entropy loss was used. Training was done with 100 epochs and a batch size of 32.

Results

The loss curve is visualized at (10). There is no validation loss on this graphic due to some technical issues. Evaluating this model is a bit more difficult. Since it only outputs a single frame's mask, we had to implement a sliding window of sorts that would iterate through the input spectrogram and generate the mask frame by frame. After doing so, we used the previously described evaluation method, and this model resulted in a MSE of 0.08. Shown at (11) is an example of a true vocal spectrogram from the testing data and a predicted vocal spectrogram from the model. Along with the noticeably lower error, this method is able to generate masks for inputs of any size, rather than being limited to 7 second clips like Model A.

4 Conclusion

Based on the comprehensive evaluation and comparison of the four models: U-Net model, Wave-U-Net model, CNN Model A, and CNN Model B, we yielded the best results from CNN Model B. This one outperformed the others in practical implementation and output accuracy.

There are several factors that contribute to this. CNN Model B was able to effectively manage the challenge of handling differently sized inputs. It displayed an impressive training loss curve, and had low loss when making predictions on the testing data. This indicates that the CNN Model B has good generalization capability when compared to the output of the three other machine learning models we implemented and is not overfitted.

Specifically, the mean squared error (MSE) of CNN Model B was low for the testing data set, coming to a value of 0.08. Model B showed significantly lower values for MSE than Model A, U-Net and Wave-U-Net. Moreover, Model B performed better than Model A since it was not just limited to 7 second clips, meaning it could generate masks for full length tracks.

In conclusion, CNN Model B's performance and adaptability makes it the premier choice for the task of audio source separation. It's crucial to note that this does not denote the potential utility of the other models in response to different scenarios or more fine-tuning, but based on the results and project needs, Model B serves as the best solution we found.

5 Contributions

We collectively provided information and support when researching/brainstorming the topic of this project. We presented and worked on the class presentation together, along with collaborating on the final report.

Melodye Nguyen

I created and formatted the presentation, as well as, executing my respective slides in class. I also created and formatted this final report in the NeurIPS 2023 style, in which I contributed in writing the introduction, data, and conclusion portions. The rest of the tasks have been collaborative efforts.

Sojung Yun

I conducted the data preprocessing, built, and trained the Wave-U-Net model. My research and knowledge of the model was provided in the presentation, which I presented in class. I contributed my results and information in the Wave-U-Net portion of the report.

Henry Hamilton

I completed the entirety of the tasks detailed in the "CNN" section of the report, which includes cleaning and reshaping the data, training and tuning the CNN models, creating the visualizations (found in the appendix), and writing the CNN portion of the report. I also contributed to writing the introduction, data, and conclusion portions.

Keisun Luc

I contributed in the making of the presentation, as well as, creating and presenting the background information portion. I provided assistance in creating and testing the models. I also contributed in writing and formatting the final report, mainly the conclusion.

Pranav Banuru

I implemented the U-Net architecture, which includes building and training the model, and computing the outputs and loss of the test data. I also contributed and reported my results and information in the U-Net portion of the report.

6 Appendix

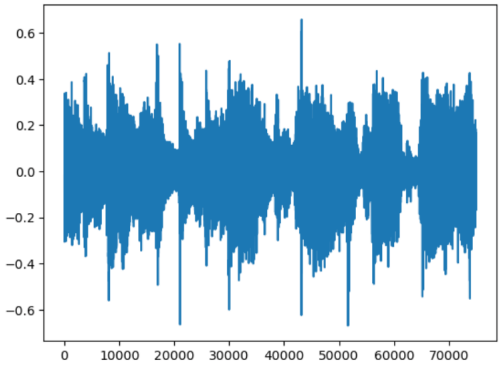

Figure 1: Visualization of single audio file.

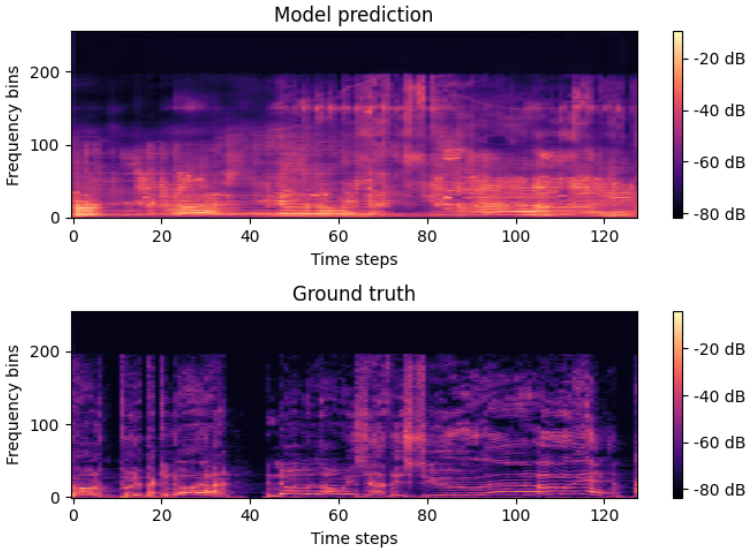

Figure 2: Vocal Spectrogram Prediction (Top) and its corresponding Ground Truth (Bottom) (uniform y-axis)

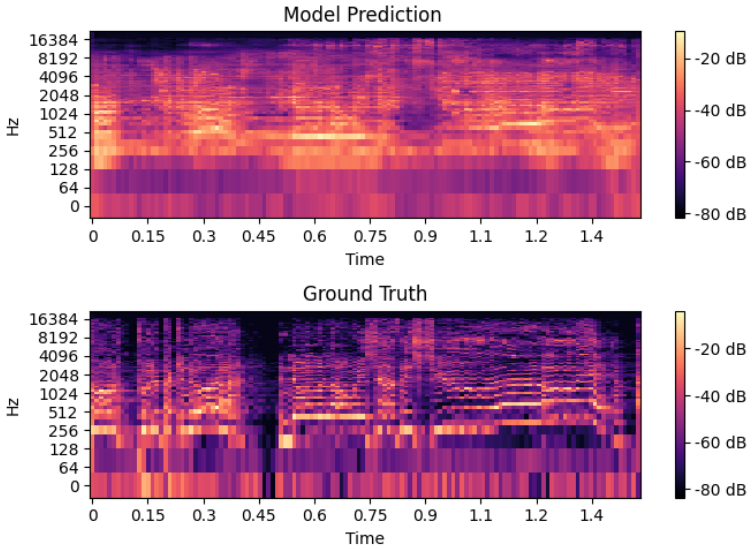

Figure 3: Vocal Spectrogram Prediction (Top) and its corresponding Ground Truth (Bottom) (log-scaled y-axis)

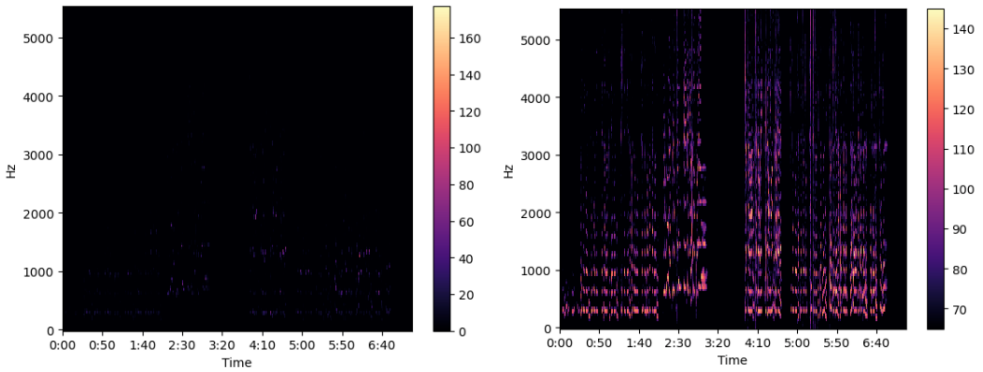

Figure 4: Amplitude Spectrogram (left) and Decibel Spectrogram (right)

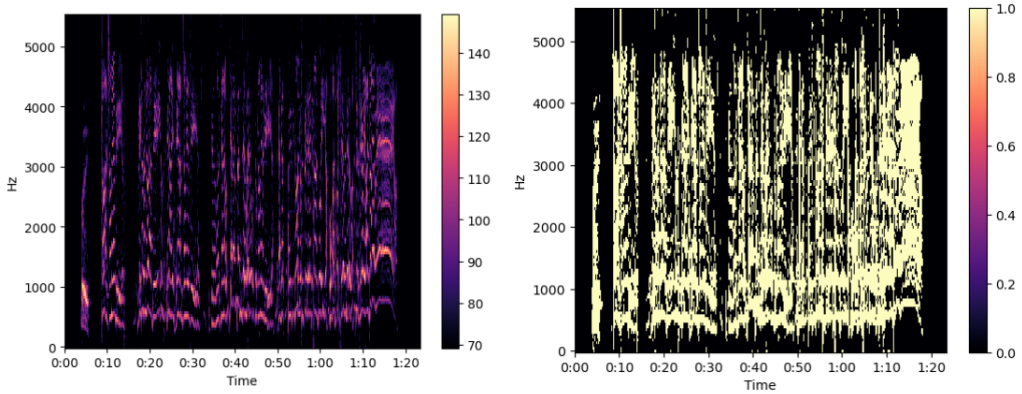

Figure 5: Vocal Spectrogram (left) and its corresponding mask (right)

```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_12 (Conv2D)           (None, 128, 294, 32)      320

leaky_re_lu_13 (LeakyReLU)   (None, 128, 294, 32)      0

conv2d_13 (Conv2D)           (None, 128, 294, 16)      4624

leaky_re_lu_14 (LeakyReLU)   (None, 128, 294, 16)      0

max_pooling2d_3 (MaxPooling  (None, 42, 98, 16)        0
2D)

dropout_3 (Dropout)          (None, 42, 98, 16)        0

conv2d_14 (Conv2D)           (None, 42, 98, 64)        9280

leaky_re_lu_15 (LeakyReLU)   (None, 42, 98, 64)        0

conv2d_15 (Conv2D)           (None, 42, 98, 16)        9232

leaky_re_lu_16 (LeakyReLU)   (None, 42, 98, 16)        0

conv2d_transpose_1 (Conv2DT  (None, 126, 294, 32)      4640
ranspose)

leaky_re_lu_17 (LeakyReLU)   (None, 126, 294, 32)      0

conv2d_16 (Conv2D)           (None, 126, 294, 1)       289

dense (Dense)                (None, 126, 294, 1)       2

=================================================================
Total params: 28,387
Trainable params: 28,387
Non-trainable params: 0
```

Figure 6: CNN - Model A: Architecture

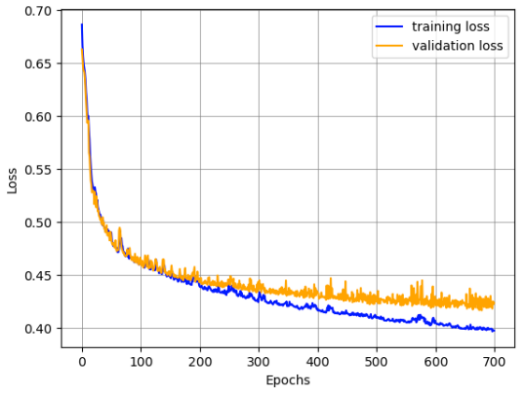

Figure 7: CNN - Model A: Loss Curve

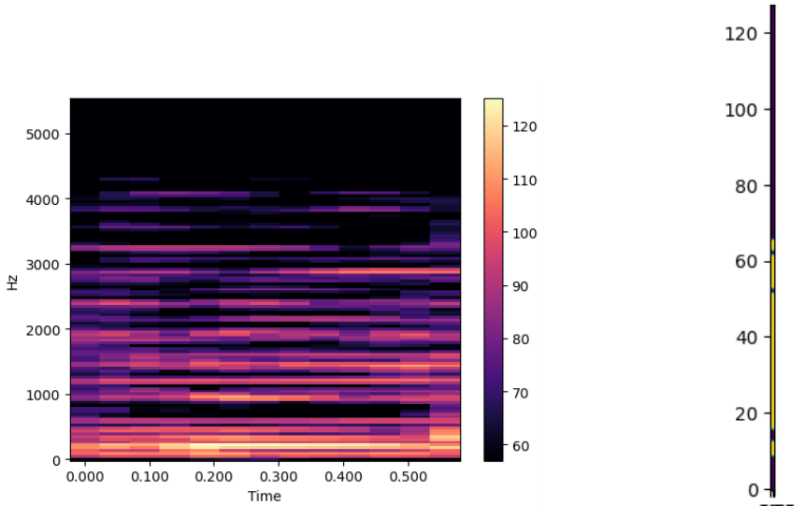

Figure 8: CNN - Model B: 0.5 Second Input (left) and Output Mask for Single Frame (right)

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 128, 13, 32)       320

 leaky_re_lu (LeakyReLU)     (None, 128, 13, 32)       0

 conv2d_1 (Conv2D)           (None, 128, 13, 16)       4624

 leaky_re_lu_1 (LeakyReLU)   (None, 128, 13, 16)       0

 max_pooling2d (MaxPooling2D  (None, 42, 4, 16)        0
 )

 dropout (Dropout)           (None, 42, 4, 16)         0

 conv2d_2 (Conv2D)           (None, 42, 4, 64)         9280

 leaky_re_lu_2 (LeakyReLU)   (None, 42, 4, 64)         0

 conv2d_3 (Conv2D)           (None, 42, 4, 16)         9232

 leaky_re_lu_3 (LeakyReLU)   (None, 42, 4, 16)         0

 max_pooling2d_1 (MaxPooling  (None, 14, 1, 16)        0
 2D)

 dropout_1 (Dropout)         (None, 14, 1, 16)         0

 flatten (Flatten)           (None, 224)               0

 dense (Dense)               (None, 64)                14400

 leaky_re_lu_4 (LeakyReLU)   (None, 64)                0

 dropout_2 (Dropout)         (None, 64)                0

 dense_1 (Dense)             (None, 128)               8320

=================================================================
Total params: 46,176
Trainable params: 46,176
Non-trainable params: 0
```

Figure 9: CNN - Model B: Architecture

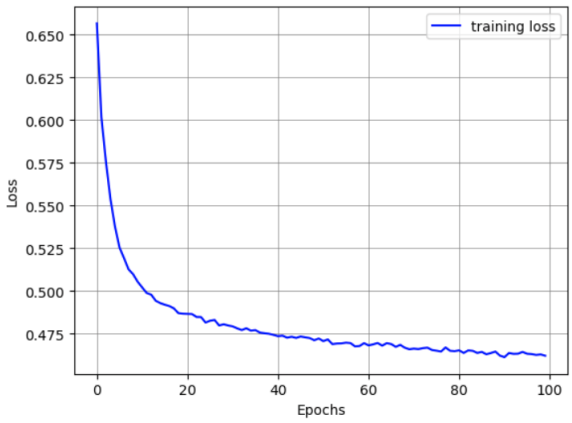

Figure 10: CNN - Model B: Loss Curve

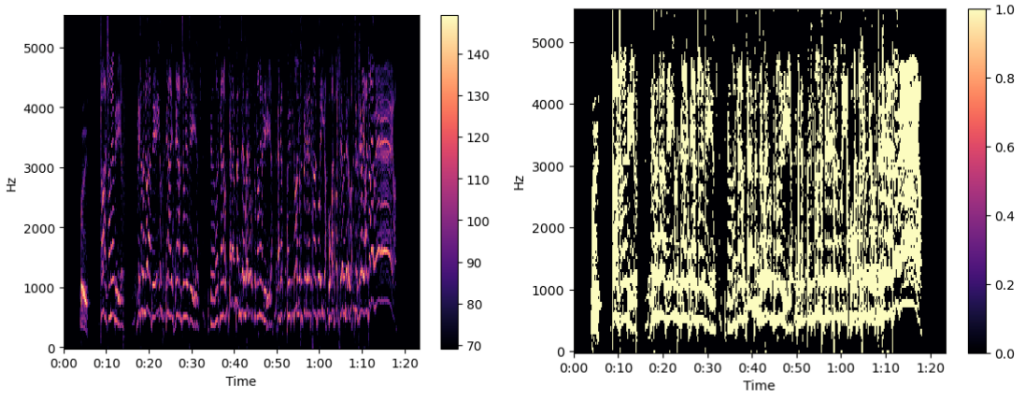

Figure 11: CNN - Model B: True Vocal Spectrogram from Testing Data (left) and Predicted Vocal Spectrogram from Model (right)

References

[1] A. Koretzky. Audio ai: Isolating vocals from stereo music using convolutional neural networks, Oct 2020.

[2] J. Mico. Breaking down the mix: Using python and neural networks to separate audio tracks, Mar 2023.

[3] Z. Rafii, A. Liutkus, F.-R. Stöter, S. I. Mimilakis, and R. Bittner. The MUSDB18 corpus for music separation, Dec 2017.

[4] D. Stoller. Wave-u-net / readme.md, Mar 2021.

[5] D. Stoller, S. Ewert, and S. Dixon. Wave-u-net: A multi-scale neural network for end-to-end audio source separation, Jun 2018.

[6] D. Ward, R. D. Mason, C. Kim, F.-R. Stöter, A. Liutkus, and M. D. Plumbley. Sisec 2018: State of the art in musical audio source separation - subjective selection of the best algorithm. In *WIMP: Workshop on Intelligent Music Production*, Huddersfield, United Kingdom, Sep 2018.