SORBONNE UNIVERSITÉ

Master 2 Automatique Robotique
Parcours Systèmes Intelligents

Singing Voice Separation with Deep U-Net Convolutional Networks

Implementation of Jansson et al. (2017)

Audio Signal Processing

Arezki Haddouche
Amira Yasmine Amrani

January 2026

GitHub Repository:
`https://github.com/Are-had/Singing-Voice-Separation-with-Deep-U-net`

Contents

1 Introduction

Audio source separation consists of isolating individual components from an audio mixture. In the context of music, separating singing voice from instrumental accompaniment is of major interest for numerous applications such as automatic karaoke generation, lyrics transcription, and music remixing.

This project aims to implement and train a U-Net deep neural network for singing voice separation, following the methodology proposed by Jansson et al. (2017) [1]. The approach treats the problem as an image-to-image translation task in the time-frequency domain, where a mixture spectrogram is transformed into isolated source spectrograms.

For model training and evaluation, we use the MUSDB18 dataset [2], a reference corpus for music source separation. This database comprises 150 full-length tracks (approximately 10 hours of audio) split into training and test sets, with isolated stems available for each track (vocals, drums, bass, other instruments).

The objective of this work is to reproduce the architecture and methodology of the original paper, analyze the difficulties encountered during implementation, and evaluate the model's performance using standard source separation metrics (SDR, SIR, SAR).

2 Implementation Choices

2.1 U-Net Architecture

We implemented the U-Net architecture as described in the original paper [1]. The U-Net is a convolutional encoder-decoder network initially designed for biomedical image segmentation, adapted here for audio source separation.

2.1.1 Encoder-Decoder Structure

The encoder consists of 6 convolutional layers that progressively downsample the input spectrogram while increasing the number of feature channels. Each encoder layer applies:

- 2D convolution (kernel size 5×5, stride 2)
- Batch normalization
- Leaky ReLU activation (leakiness = 0.2)

The decoder symmetrically upsamples the representation through 6 transposed convolutional layers, progressively reconstructing the output at the original resolution. Each decoder layer consists of:

- 2D transposed convolution (kernel size 5×5, stride 2)
- Batch normalization
- ReLU activation
- Dropout (50% for the first three layers)

2.1.2 Skip Connections

A key feature of U-Net is the skip connections between corresponding encoder and decoder layers at the same hierarchical level (see Figure 1). These connections concatenate high-resolution features from the encoder directly to the decoder, allowing the network to preserve fine-grained spectral and temporal details essential for high-quality audio reconstruction.

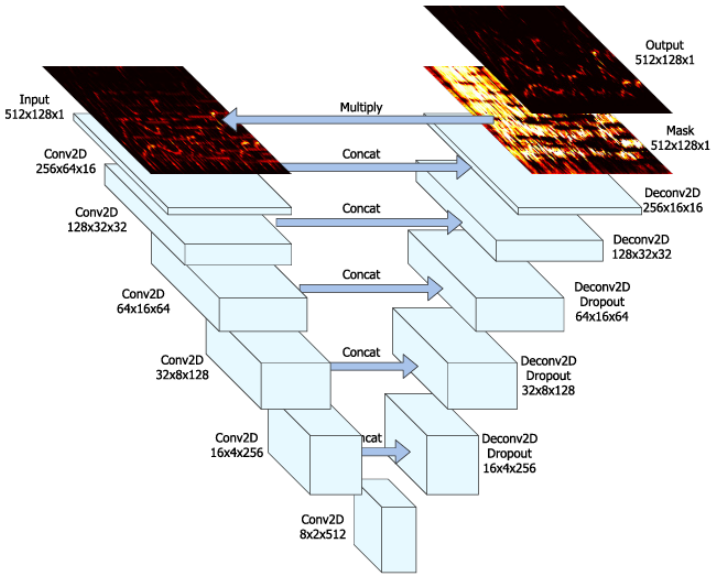

Figure 1: U-Net architecture for singing voice separation (from Jansson et al., 2017)

2.1.3 Soft Mask Output

The network outputs a soft mask $M \in [0, 1]$ with the same dimensions as the input spectrogram (1×512×128). A sigmoid activation is applied to the final layer to ensure values remain in the [0,1] range. The predicted vocal spectrogram is obtained by element-wise multiplication:

$$\hat{Y}_{\text{vocal}} = M \odot X_{\text{mixture}} \tag{1}$$

where X_{mixture} is the input mixture spectrogram and $\hat{Y}_{\text{vocal}}$ is the predicted vocal spectrogram.

2.1.4 Loss Function

Following the original paper, we use the L1 loss (Mean Absolute Error) to train the model:

$$\mathcal{L}(X, Y; \Theta) = ||f(X, \Theta) \odot X - Y||_{1,1} \tag{2}$$

where $f(X, \Theta)$ is the predicted mask, Y is the target vocal spectrogram, and $||\cdot||_{1,1}$ denotes the element-wise L1 norm. The L1 loss is preferred over L2 (MSE) for audio applications as it better preserves spectral details and is less sensitive to outliers.

2.2 Preprocessing and Data Storage

2.2.1 Preprocessing Pipeline

To optimize training time, we adopted a two-stage approach where spectrograms are computed once and saved to disk, rather than being computed on-the-fly during training. The preprocessing pipeline was implemented in Jupyter notebooks (.ipynb) and consists of the following steps:

1. Load audio tracks from MUSDB18 (44100 Hz, stereo)
2. Convert to mono by averaging channels
3. Downsample to 8192 Hz (as per the original paper)
4. Apply Short-Time Fourier Transform (STFT)
5. Extract magnitude and phase components
6. Apply per-song normalization
7. Save spectrograms as NumPy arrays (.npy files)

This approach provides several advantages:

- STFT is computed only once, significantly reducing training time
- Clear separation between feature extraction and model training
- Reproducibility: identical spectrograms across training runs
- Flexibility to experiment with different normalization strategies without recomputing STFT

2.2.2 STFT Parameters

Following the original paper, we use the following STFT parameters:

- Sampling rate: 8192 Hz (downsampled from 44100 Hz)
- FFT window size (n_fft): 1024
- Hop length: 768 samples
- Number of frequency bins: 513 (reduced to 512 by removing DC component)

These parameters result in spectrograms where each frame represents approximately 125 ms of audio, with an overlap of about 94% between consecutive frames, ensuring good time-frequency resolution.

2.2.3 File Organization

The preprocessed spectrograms are organized in a structured directory hierarchy:

```
spectrograms/
 musdb18/
    train/
       mixture/
          trackA_spec.npy
          ...
       vocal/
          trackA_spec.npy
          ...
       phase/
           trackA_phase.npy
           ...
    valid/
       [same structure]
    test/
        [same structure]
```

Each spectrogram is saved with dimensions (513, n_frames) where n_frames varies depending on the track duration. The phase information from the mixture is preserved for audio reconstruction during inference.

2.3 Normalization Strategy

2.3.1 Initial Approach and Problem

Initially, we implemented a per-spectrogram normalization using min-max scaling:

$$\text{spec}_{\text{norm}} = \frac{\text{spec} - \min(\text{spec})}{\max(\text{spec}) - \min(\text{spec})} \tag{3}$$

This normalization was applied independently to each spectrogram (mixture and vocal) during preprocessing. However, this approach led to severe convergence issues (see Figure 4). The training and validation losses stagnated around 0.004-0.007 with large fluctuations and no clear improvement over 100 epochs.

Root cause: This normalization strategy created inconsistencies between mixture and vocal spectrograms, as each was normalized with different min/max values. Additionally, different songs were mapped to the same [0,1] range regardless of their actual amplitude, making it impossible for the network to learn coherent patterns across the dataset.

2.3.2 Final Solution: Per-Song Normalization

Following the approach used in the reference implementation, we adopted a per-song normalization strategy:

$$\text{norm} = \max(\text{spec}_{\text{mixture}}) \tag{4}$$

$$\text{spec}_{\text{mixture}}^{\text{norm}} = \frac{\text{spec}_{\text{mixture}}}{\text{norm}}, \quad \text{spec}_{\text{vocal}}^{\text{norm}} = \frac{\text{spec}_{\text{vocal}}}{\text{norm}} \tag{5}$$

Key advantages:

- Both mixture and vocal are normalized with the *same* factor (max of mixture)
- The relative amplitude ratio between mixture and vocal is preserved
- Simpler formulation (division only, no subtraction)

This change dramatically improved convergence. As shown in Figure 5, both training and validation losses now decrease smoothly from ∼0.006 to ∼0.003 over 100 epochs, demonstrating effective learning without overfitting.

2.4 Patch Generation

2.4.1 DataLoader Implementation

The DataLoader is responsible for loading preprocessed spectrograms and extracting patches of 128 frames for training. Since saved spectrograms have variable lengths (depending on track duration), we implement adaptive strategies:

- **If n_frames $<$ 128:** Zero-padding is applied to reach 128 frames
- **If n_frames $\geq$ 128:** Random cropping extracts a 128-frame window

The random crop strategy serves as data augmentation, allowing the network to see different temporal segments of the same track across epochs.

2.4.2 Overlap Problem

The assignment mentions a high overlap rate between consecutive 128-frame spectrograms. With hop_length=768 and patches of 128 frames, systematic sequential extraction would create minimal diversity in the training data.

Our solution: We use **random cropping** instead of sequential extraction. For each training sample, a random starting position is chosen within the valid range [0, n_frames - 128]. This approach:

- Eliminates systematic overlap between training samples
- Provides natural data augmentation
- Ensures the model sees diverse temporal contexts

2.5 Audio Reconstruction

During inference, the model predicts a soft mask which is applied to the mixture spectrogram to obtain the vocal magnitude. To reconstruct the time-domain audio signal, we combine this predicted magnitude with the phase of the original mixture:

$$\text{STFT}_{\text{vocal}} = \hat{Y}_{\text{vocal}} \cdot e^{j\phi_{\text{mixture}}} \tag{6}$$

where $\hat{Y}_{\text{vocal}}$ is the predicted vocal magnitude and ϕ_{mixture} is the phase of the mixture.

The complex spectrogram is then converted back to audio using the Inverse Short-Time Fourier Transform (ISTFT):

$$y_{\text{vocal}}(t) = \text{ISTFT}(\text{STFT}_{\text{vocal}}) \tag{7}$$

Finally, if necessary, the audio is resampled from 8192 Hz back to 44100 Hz to match the original sampling rate. This simple phase reconstruction approach, while not optimal, proves effective in practice as demonstrated by the original paper.

3 Difficulties Encountered

3.1 Overfitting on 7-Second Dataset

3.1.1 Initial Observations

We began our experiments using the MUSDB18 7-second subset to enable rapid prototyping and iteration. Initially, when monitoring only the training loss, the results appeared promising: the loss decreased steadily, suggesting successful learning (see Figure 2). Encouraged by these results, we performed inference on a test vocal track.

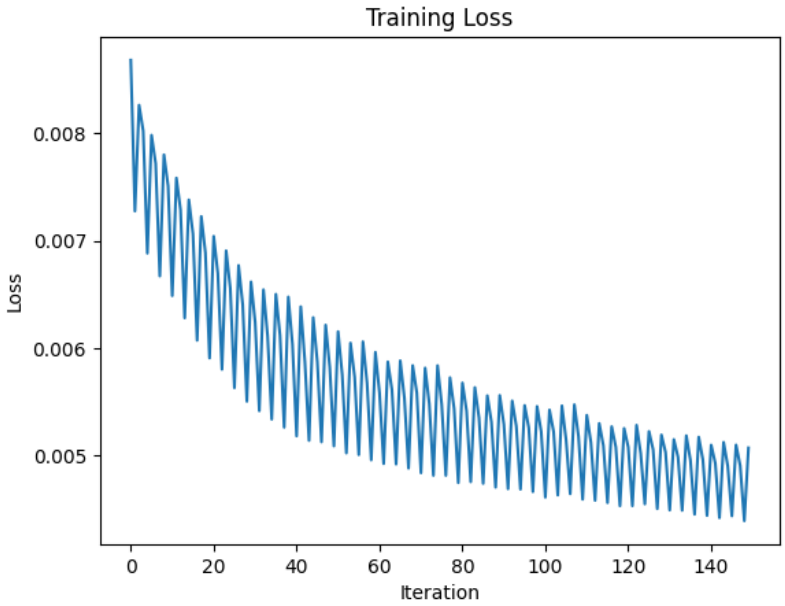

Figure 2: Training loss only on 7-second dataset (misleading)

However, upon listening to the separated vocal output (vocal_predicted_7s.wav), the quality was poor with significant artifacts and incomplete separation, revealing a critical problem despite the decreasing training loss.

3.1.2 Diagnosis: Severe Overfitting

To diagnose the issue, we plotted both training and validation losses together (Figure 3). This revealed a clear overfitting pattern:

- Training loss: continued to decrease
- Validation loss: remained flat or even increased
- Large gap between training and validation performance

This indicated that the model was memorizing the training set rather than learning generalizable features for voice separation.

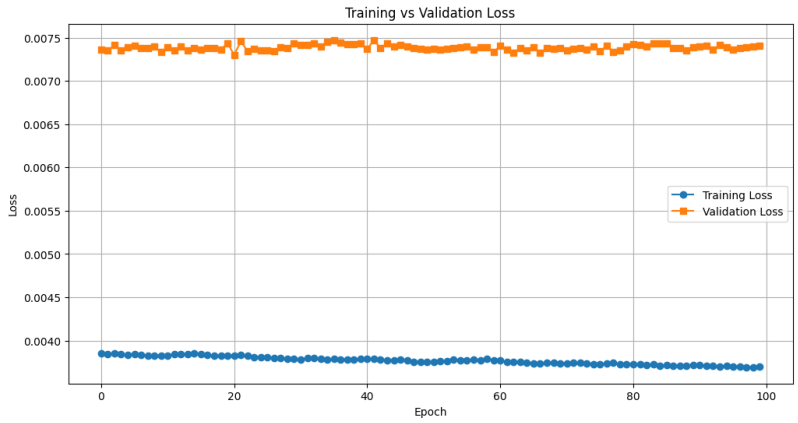

Figure 3: Training vs validation loss showing overfitting on 7s dataset

3.1.3 Root Cause Analysis

The 7-second dataset presented several limitations:

1. **Limited data:** Only 106 audio files in the training set
2. **Short duration:** Each track is approximately 7 seconds, resulting in only 70-80 frames per spectrogram
3. **Excessive padding:** To reach the required 128 frames input size, we had to pad each spectrogram with 48-58 frames of zeros ($\sim$43-55% padding)
4. **Insufficient diversity:** The model learned to predict zeros for the padded regions rather than learning meaningful vocal separation patterns

The combination of limited training samples and excessive padding created a scenario where the model could easily overfit to the small number of real audio frames.

3.1.4 Solution: Full MUSDB18 Dataset

We transitioned to the complete MUSDB18 dataset, which provides:

- **More samples:** 100 full-length training tracks (vs. 106 7-second clips)
- **Longer duration:** Full-length tracks (typically 3-5 minutes), yielding approximately 2000-4000 frames per spectrogram
- **Natural random cropping:** With sufficient frames, we can extract 128-frame patches without padding, enabling data augmentation
- **Greater diversity:** Multiple patches per song increase effective training data size

This change eliminated the padding problem and provided sufficient data for the model to learn robust vocal separation features, as demonstrated in the subsequent experiments.

3.2 Normalization Problem on Full MUSDB18 Dataset

3.2.1 First Normalization Attempt: Failure to Converge

After transitioning to the full MUSDB18 dataset, we initially applied the same per-spectrogram min-max normalization strategy (Equation 3). Despite having significantly more training data, the model failed to converge. As shown in Figure 4, both training and validation losses stagnated around 0.004-0.006 with large fluctuations and no clear downward trend, even after 100 epochs. This was particularly frustrating as we had resolved the overfitting issue but introduced a new convergence problem.

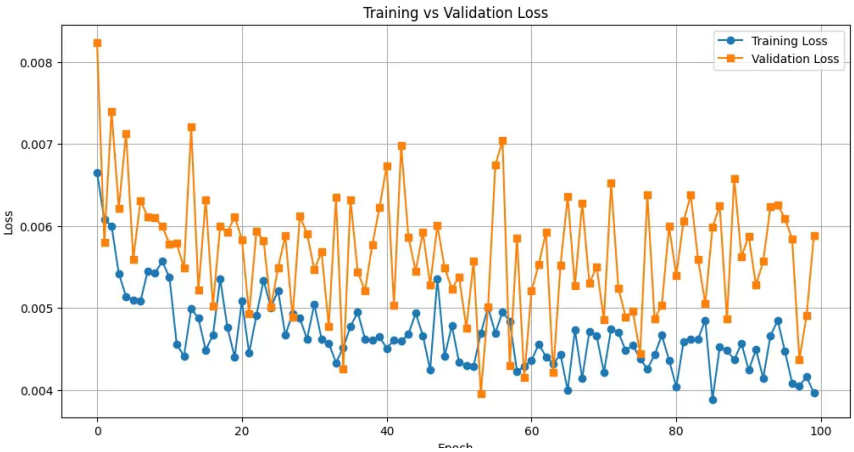

Figure 4: Training and validation loss with per-spectrogram min-max normalization. The model fails to converge despite 100 epochs of training.

The problem stemmed from normalizing each spectrogram independently with different min/-max values, causing the network to receive inconsistent training signals where the relationship between mixture and vocal magnitudes was distorted across different songs.

3.2.2 Solution: Per-Song Normalization

We modified the normalization strategy as described in Section 2.3.2 (Equations 4-5), using only the maximum value of the mixture spectrogram as the normalization factor for both mixture and vocal.

This change had an immediate and dramatic effect. As shown in Figure 5, both training and validation losses now decreased smoothly from ∼0.006 to ∼0.003 over 100 epochs, tracking each other closely with much lower fluctuations.

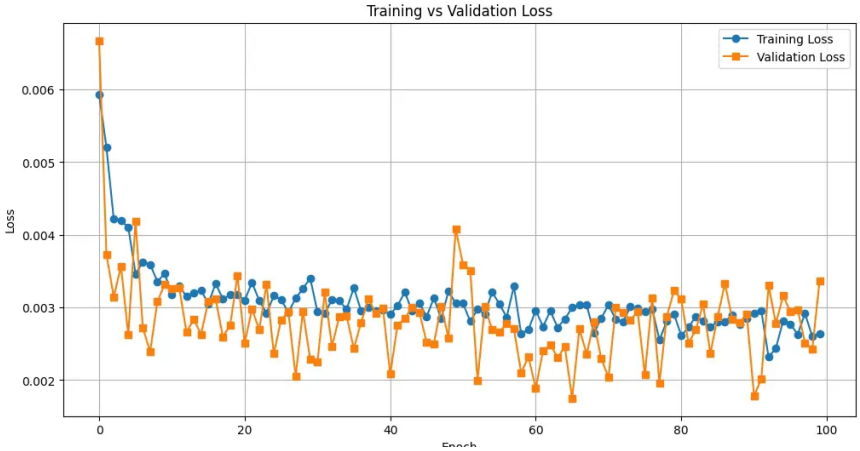

Figure 5: Training and validation loss with per-song normalization. Both losses decrease smoothly from ∼0.006 to ∼0.003 over 100 epochs.

3.2.3 Extended Training

Encouraged by the successful convergence, we extended training to 160 epochs (Figure 6). The validation loss continued to improve, reaching below 0.0024, demonstrating that the model had not yet plateaued.

The quality of separated vocals (predicted_vocal_musdb18.wav) showed significant improvement compared to earlier attempts, with cleaner separation and fewer artifacts.

This experience highlights the critical importance of normalization strategy in deep learning for audio: even with correct architecture and sufficient data, an inconsistent normalization scheme can completely prevent model convergence.

4 Experimental Results

4.1 Training Configuration

The model was trained with the following hyperparameters:

- Optimizer: Adam with learning rate 10^{-3}
- Batch size: 8

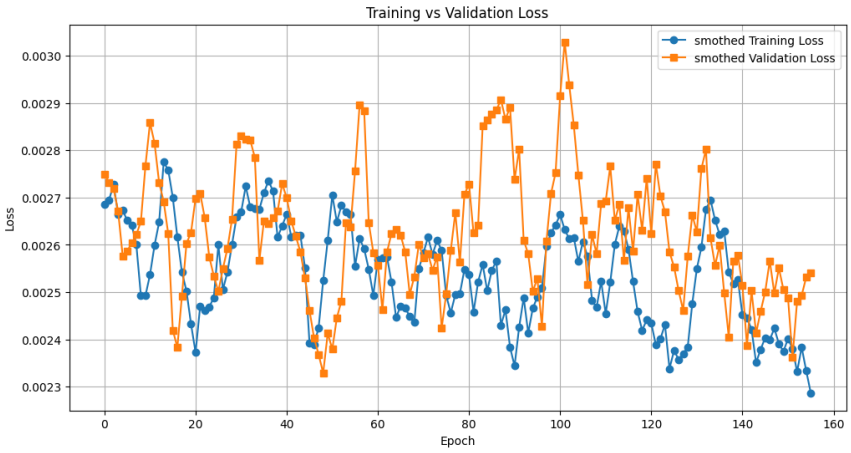

Figure 6: Smoothed training and validation loss over 160 epochs. The validation loss reaches below 0.0024, indicating continued improvement.

- Number of epochs: 160
- Loss function: L1 (Mean Absolute Error)
- Training set: 100 tracks from MUSDB18
- Validation set: 25 tracks

4.2 Convergence Analysis

After resolving the normalization issues described in Section 3.2, the model achieved stable convergence over 160 epochs. As demonstrated in Figure 6 (Section 3.2.4), the final validation loss of 0.0024 represents a significant improvement from the initial 0.006, with training and validation curves tracking closely throughout—confirming good generalization.

The continued downward trend at epoch 160 suggests that further training (500-1000 epochs as mentioned in the original paper) would likely yield additional improvements.

4.3 Separation Performance Metrics

We evaluated the model on a subset of the MUSDB18 test set (5 tracks) using standard source separation metrics computed with the `museval` library. Due to computational constraints, we calculated the average performance across these representative samples. Table 1 presents the results..

Table 1: Source separation performance on MUSDB18 test set

Metric	Score (dB)
SDR (Signal-to-Distortion Ratio)	-86.90 ± 2.99
SIR (Signal-to-Interference Ratio)	10.17 ± 7.25
SAR (Signal-to-Artifacts Ratio)	1.17 ± 0.97

4.4 Analysis and Discussion

4.4.1 SIR Performance

The Signal-to-Interference Ratio (SIR) of approximately 10 dB indicates that the model successfully learned to isolate vocals from instrumental accompaniment. This demonstrates that the U-Net architecture effectively captures the spectral patterns that distinguish singing voice from background music. The SIR performance is reasonable considering the limited training (160 epochs vs. 80,000 in the original paper).

4.4.2 SDR and SAR Issues

However, the Signal-to-Distortion Ratio (SDR) shows unexpectedly poor performance at -87 dB. This negative value indicates a technical issue rather than a learning problem, as a properly functioning model should yield positive SDR values. The most likely explanation is a mono-to-stereo conversion artifact: our model predicts mono vocals which are then duplicated to create stereo output, potentially causing phase cancellation or amplitude mismatches when compared against the stereo ground truth.

Similarly, the low Signal-to-Artifacts Ratio (SAR) of 1.17 dB suggests the presence of reconstruction artifacts, likely stemming from:

- Phase reconstruction using mixture phase rather than true vocal phase
- Resampling artifacts (8192 Hz $\rightarrow$ 44100 Hz)
- Potential length mismatches between predicted and ground truth signals

4.4.3 Comparison with Original Paper

The original paper reports median SDR of approximately 6-11 dB and SIR of 12-23 dB on various test sets. While our SIR performance is approaching the lower bound of their results, the SDR issue prevents meaningful comparison. Addressing the stereo conversion and reconstruction pipeline would be essential for proper benchmarking.

5 Conclusion

This project successfully implemented the U-Net architecture for singing voice separation as proposed by Jansson et al. (2017). The implementation includes a complete pipeline from audio preprocessing to model training and inference on the MUSDB18 dataset.

Through iterative debugging and experimentation, we identified and resolved critical issues including overfitting on limited data and normalization inconsistencies. The transition from the 7-second subset to the full MUSDB18 dataset, combined with per-song normalization strategy, enabled successful model convergence. After 160 epochs of training, both training and validation losses decreased from approximately 0.006 to 0.0024, demonstrating effective learning without overfitting.

The model achieved a Signal-to-Interference Ratio (SIR) of approximately 10 dB, indicating successful separation of vocals from instrumental accompaniment. However, reconstruction issues—particularly related to mono-to-stereo conversion and phase estimation—resulted in poor SDR performance. These technical challenges highlight the complexity of audio source separation beyond the machine learning model itself.

Future improvements could include: (1) extending training to 500-1000 epochs to match the original paper's setup, (2) implementing proper stereo prediction or improved mono-to-stereo conversion, (3) exploring alternative phase reconstruction methods beyond using mixture phase, and (4) implementing a learning rate scheduler to further optimize convergence. Despite current limitations, this work demonstrates a solid foundation for singing voice separation and provides valuable insights into the practical challenges of implementing deep learning models for audio processing tasks.

A Code Structure and Organization

The complete implementation is publicly available at:
`https://github.com/Are-had/Singing-Voice-Separation-with-Deep-U-net`

A.1 Repository Structure

```
Singing-Voice-Separation-with-Deep-U-net/

 notebooks/
    data_preprocessing_steps.ipynb  # Pipeline exploration and testing
    process_data.ipynb              # Spectrogram generation for train/test/valid
    train.ipynb                     # Model training and loss visualization
    inference.ipynb                 # Single audio test and reconstruction
    evaluate.ipynb                  # Metrics computation on test set

 src/
    DataLoader.py        # Custom DataLoader class for patch extraction
    model.py             # U-Net architecture implementation
    utils.py             # Helper functions (STFT, normalization, etc.)
```

A.2 File Descriptions

A.2.1 Notebooks

- **data_preprocessing_steps.ipynb**: Exploratory notebook for testing the complete pipeline from raw audio to spectrograms, experimenting with STFT parameters and normalization strategies
- **process_data.ipynb**: Production preprocessing pipeline that generates and saves spectrograms for training, validation, and test sets as NumPy arrays
- **train.ipynb**: Implements the training loop, handles model checkpointing, and generates loss vs. epoch plots for convergence analysis
- **inference.ipynb**: Tests the trained model on individual audio files, performs vocal separation, and saves the reconstructed audio
- **evaluate.ipynb**: Evaluates separation performance on the test set using standard metrics (SDR, SIR, SAR) computed with the `museval` library

A.2.2 Source Code

- **DataLoader.py**: Custom PyTorch DataLoader class implementing random cropping, zero-padding, and patch extraction (512×128 patches)
- **model.py**: U-Net architecture with encoder-decoder structure, skip connections, and soft mask output layer

- **utils.py**: Utility functions for STFT computation, per-song normalization, audio reconstruction (ISTFT),

References

[1] Jansson, A., Humphrey, E., Montecchio, N., Bittner, R., Kumar, A., & Weyde, T. (2017). *Singing Voice Separation with Deep U-Net Convolutional Networks*. Proceedings of the 18th International Society for Music Information Retrieval Conference (ISMIR), Suzhou, China.

[2] Rafii, Z., Liutkus, A., Stöter, F. R., Mimilakis, S. I., & Bittner, R. (2017). *MUSDB18 - a corpus for music separation*. Zenodo. https://doi.org/10.5281/zenodo.1117372

[3] SunnerLi. (2019). *SVS-UNet-PyTorch: Singing Voice Separation with U-Net in PyTorch*. GitHub repository. https://github.com/SunnerLi/SVS-UNet-PyTorch

[4] Ronneberger, O., Fischer, P., & Brox, T. (2015). *U-Net: Convolutional Networks for Biomedical Image Segmentation*. Medical Image Computing and Computer-Assisted Intervention (MICCAI), Springer.

[5] Stöter, F. R., Liutkus, A., & Ito, N. (2018). *The 2018 Signal Separation Evaluation Campaign*. Latent Variable Analysis and Signal Separation (LVA/ICA), Springer.