Voice Isolation from Songs

MAKHOUL Rayan
ULB - VUB Brussels, Belgium Rayan.Makhoul@ulb.be

OGUNGBURE Semilogo
ULB - VUB Brussels, Belgium Semilogo.Olusola.Ogungbure@ulb.be

PRIEELS Lucas
ULB - VUB Brussels, Belgium Lucas.Prieels@ulb.be

TROUILLEZ Benoît
ULB - VUB Brussels, Belgium Benoit.Trouillez@ulb.be

***Abstract*—This project explores the extraction of singing voice (vocals) from musical mixtures, i.e. songs. Three papers are studied, each proposing novel approaches using deep learning models and their respective algorithms are implemented for comparison. The first paper trains a convolutional neural network to estimate the ideal frequency binary masks to get the vocals from the real-world musical mixtures. The second paper applies the U-Net architecture to decompose mixed spectrograms into their constituent sources, achieving state-of-the-art performance. The third paper introduces the Wave-U-Net, an adaptation of the U-Net architecture for time-domain source separation, achieving comparable performance to spectrogram-based methods. Due to time constraints and the lack of GPU among the group members, unfortunately only the U-Net architecture gave satisfying results. For this architecture, parameters have been modified to study their impact on the quality of the vocals extraction from the songs. They highlight which parameters value allow to achieve the best possible results.**

***Index Terms*—Voice isolation, extraction, neural network, MUSDB18 dataset, spectrograms, CNN, U-Net, Wave-U-Net.**

I. Introduction

Voice isolation is a complex task in the field of music source separation, and this project focuses on a machine learning project tackling this challenge. The objective of the project is to develop techniques for extracting the vocals from music *mixtures* using deep learning methods. To achieve this, the study explores and implements approaches from three influential papers in the field, which serve as benchmarks for comparison and evaluation.

The first implemented algorithm contains layers of convolutional neural network followed by layers of fully-connected neural network. The CNN layers allow to gain an insight from the neighborhood in the frequency bins while the fully-connected layers further processes this.

The U-Net algorithm is a renowned deep learning architecture highly effective in extracting vocals from audio mixtures. It is based on an encoder-decoder architecture which sequentially performs convolutions decreasing the size of the image, followed by expansions of the image to get a new image of the same dimensions as the first one.

The third algorithm is Wave-U-Net, which is a modified version of the U-Net model. The Wave-U-Net architecture is specifically designed to capture temporal dependencies in audio signals, making it well-suited for voice isolation tasks. The project aims to achieve high-quality separation results and improve the fidelity of the extracted vocal tracks.

The evaluation of the implemented models uses performance metrics based on the SNR representing the relative importance of the computed signals with respect to artefacts and errors. They represent the effectiveness of extraction of vocals from music *mixtures*.

The second section of this report presents the current state of the art linked to those three algorithms. The following section details how we have been implementing them. The fourth section finally highlights the results of those implementation as well as the impact of different parameters on the efficiency of vocals extraction.

II. State of the art

In order to address the challenge of voice isolation, various architectures have been developed and explored in recent years. In this section are presented an overview of three state-of-the-art architectures for voices isolation including convolutional deep neural network architecture, U-Net architecture and Wave-U-Net architecture.

A. Convolutions deep neural network

. One notable architecture that has shown remarkable success in voice separation is the fully Connected Deep Neural Network (FCDNN). The FCDNN architecture, introduced by Simpson et al. in their paper "Deep Karaoke: Extracting Vocals from Musical Mixtures using a Convolutions Deep Neural Network" [1]. The goal of the algorithm is to compute frequency masks from mixture spectrograms. These frequency masks are then applied on the mixture spectrograms to get a new reconstructed spectrogram as close as possible to the target vocals spectrogram. This process is shown on Figure 1. The vocals song is then finally got back by applying the inverse short-term Fourier transform on the reconstructed vocals spectrogram.

The FCDNN architecture, with its fully connected layers, allows for complex relationships to be learned between the input spectrogram and the vocal mask, enabling effective vocal isolation. The success of the FCDNN in voice separation tasks highlights the capabilities of deep learning in extracting

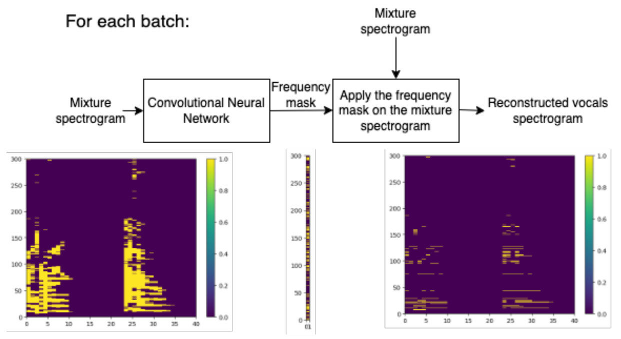

For each batch:
Mixture spectrogram
Mixture spectrogram
Convolutional Neural Network
Frequency mask
Apply the frequency mask on the mixture spectrogram
Reconstructed vocals spectrogram

Fig. 1. Overall view of the inputs and outputs of the CNN algorithm

and enhancing specific audio components. Building upon the progress of the FCDNN, researchers have proposed additional architectures to further enhance audio source separation. One such architecture is the U-Net, inspired by the FCNN architecture, which was introduced as a convolutional deep neural network architecture [3].

The CNN architecture is briefly explained as shown in fig 1:

- *Input Layer:* receives patches of mixture spectrogram representing the audio mixture (vocal and accompaniments)
- *Conv1 (Layer 1)*: applies a 3x3 convolution operation with a stride of 1, padding of 1, and 1 input channel to exytract features from the mixture spectrogram patches. Generates 16 output channels to capture relevant patterns.
- *Conv2 (Layer2):* applies 3x3 convolution operation with a stride of 1, padding of 1, and 16 input channels from Conv1 to further refine the extracted features. Aim to capture complex representations and produces a single output channel representing the frequency mask.
- *Fully Connected Layer1:* Flattens the output from Conv2 into a vector and passes it through a fully connected layer with 20,500 neurons. Performs a linear transformation and introduces non-linearity with ReLU activation.
- *Fully Connected Layer 2:* Takes the output from FC1 and passes it through another fully connected layer with 20,500 neurons, applying ReLU activation.
- *Fully Connected Layer 3:* Processes the output from FC2 through a fully connected layer with 1,025 neurons, corresponding to the number of frequency bins. No activation function is applied, resulting in a raw output representing the frequency mask.
- *Binary Mask:* The binary mask is generated by the last layer of the neural network, which consists of 1,025 units (matching the number of frequency bins in the spectrogram) and does not apply any activation function. This layer produces a raw output representing the frequency mask. The purpose of the binary mask is to highlight the vocal components in the mixture spectrogram while attenuating the background music components. It achieves this by assigning binary values (e.g., 0 or 1) to each frequency bin, indicating whether it should be considered as part of the vocals or background music.
- *Output Layer:* Consists of 1,025 units matching the number of frequency bins. Represents the predicted frequency mask that emphasizes vocals and suppresses background music.

B. U-Net

The paper *Singing voice separation with deep U-net convolutional networks* [3] is the first one to make use of the U-net architecture for audio separation. The proposed architecture is shown on figure 2. This kind of convolutional neural network has an encoder-decoder architecture where the encoder part is a ”contracting part” where conventional convolutions are performed until a bottleneck (the furthest block down on the figure). This has for effect to decrease the height and the width of the input and increasing its number of channels – such as a downsampling operation. The encoder helps capturing global context and high-level features. After that, the input is expanded until it has the same dimensions as originally. During expansion, transposed convolutions will decrease the height and width are upsampled thanks to transposed convolutions. This path helps recover spatial resolution and to localise the details. Skip connections that connects corresponding encoder and decoder modules are also present, they allow the flow of low-level information from the encoder to the decoder. This allows for the U-Net to better reconstruct fine-grained structures. In the final layer, a sigmoid activation function is implemented. As it is visible on the right of the figure, the network learns to estimate a soft mask that represents the spectrogram bins in which the vocal is more prominent than the accompaniment. This mask is then multiplied element-wise by the input mixture to obtain the final estimate.

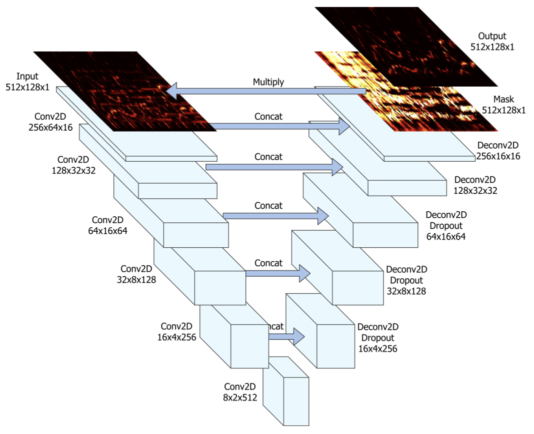


Fig. 2. U-net architecture presented in the paper [3]

C. Wave-U-Net

In 2018, the Wave-U-Net architecture for voice isolation challenge has been presented by Daniel Stoller, Sebastian Ewert and Simon Dixon in their paper *Wave-U-Net: A Multi-Scale Neural Network for End-to-End Audio Source Separation* [2]. This architecture is inspired by the U-Net

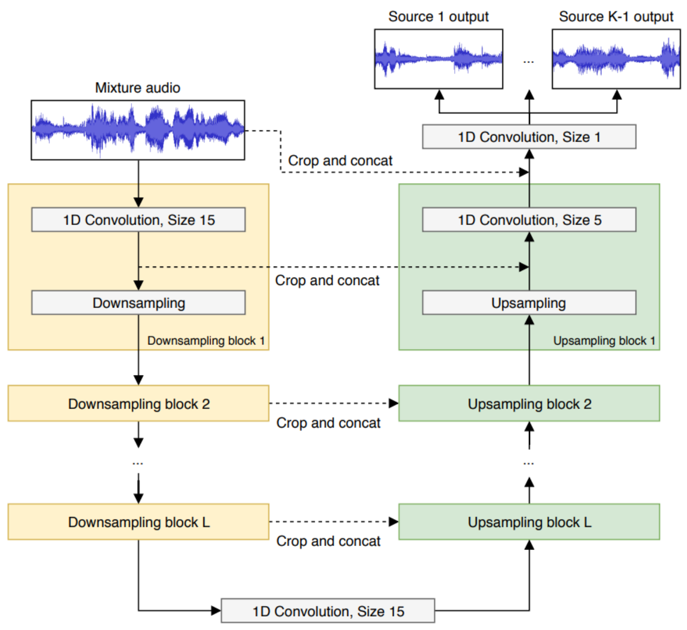


Fig. 3. Wave-U-Net architecture presented in [2]

architecture. As presented earlier, the U-Net architecture is named like that because of its U shape with the encoding and decoding parts, separated by the bottleneck block. The difference with the U-Net architecture is that the Wave-U-Net makes use of the time domain representation of the audio signal instead of its spectrogram representation, hence the *Wave* in the name that refers to the waveform representation of the signal.

The goal of the Wave-U-Net is to separate a certain mixture waveform into its K source waveforms. Those sources are typically the vocals, the drums, the bass and other sources in the mixture. An input mixture M of size $L_{in} \times 1$ passes through the network and the latter actually outputs $K-1$ source waveforms S^k of size $L_{out} \times 1$ each. The overall architecture of the Wave-U-Net is shown below.

Actually, the K^{th} source waveform is generated as being $S^K = M - \sum_{k=1}^{K-1} S^k$ as the mixture is defined as being the sum of all the source waveform in the audio. This way of computing the last source waveform as a function of the others is used to add a constraint that needs to be respected by the model. Also, one can observe that L_{in} and L_{out} are not necessarily the same as it can be possible to input a mixture of size $L_{in} = L_{out} + L_c$ where $L_c = L_{c,b} + L_{c,a}$ is the number of context samples equally taken before and after ($L_{c,b} = L_{c,a}$) the actual time window to have voice isolation on. Lastly, input and output can have multiple channels in case of stereo (M of size $L_{in} \times C$ and S^k of size $L_{out} \times C$). In our case, let's assume the simple no-conext mono case where $L_{in} = L_{out} = L_s$ and $C = 1$.

As it has already been explain in section II-B, the U-Net architecture has an encoding and decoding part. In the Wave-U-Net architecture, while the encoding part consists of L serial encoder blocks that extract the audio features at different spatial dimensionalities of the input waveform and encode them, the decoding part has the same number of decoder blocks ($= L$) that aims to reconstruct the desired predicted separated sources through those layers that make use of the encoded features and a skip connection with their corresponding encoder layer in the encoding part.

In order to better understand what happens in the neural network, one can check what happens between each layer. Also, it is interesting to know that, since the Wave-U-Net is inspired by the U-Net architecture, the steps in the forwarding are really similar to what happens in the U-Net architecture with a few minor differences such as the data that are 2D spectrograms patches instead of 1D waveform window. Let's first have a look at an encoding block at level l. It takes as input a data block of shape $[L_s/2^{l-1}, N_{c,l-1}]$ where the first element is the number of samples and the second is the number of convolutional channels which is $N_{c,0} = 1$ for the mixture input and $N_{c,l>0} = F_c l$ for the lower layers. F_c is defined as the number of extra filters per layer. After the 1D convolution in the encoder block l, one obtain a data block of shape $[L_s/2^{l-1}, N_{c,l}]$ that is stored for the skip connection with the corresponding decoder block at layer l. A downsampling process is then applied to make the time scale coarser and pass to the next lower encoder block data of shape $[L_s/2^l, N_{c,l}]$. Between the encoding and decoding part, a bottleneck layer is used and provide the lower level decoder data of shape $[L_s/2^L, N_{c,L+1}]$. Concerning the decoding part, the decoder block l (l going from L to 1) takes as input data of shape $[L_s/2^l, N_{c,l+1}]$ and first apply upsampling though linear interpolation in order to get data of shape $[L_s/2^{l-1}, N_{c,l+1}]$. Afterwards, the concatenation on the second dimension (i.e. number of convolutional channels) with the corresponding encoder layer l skip connection is done and data has shape $[L_s/2^{l-1}, N_{c,l+1} + N_{c,l}]$. The 1D convolution layer decreases the number of convolutional channels and produces data of shape $[L_s/2^{l-1}, N_{c,l}]$. Therefore, after the last decoder layer ($l = 1$), the data is in the form $[L_s, F_c]$. As a last step, the output layer concats the $[L_s, F_c]$ data block with the skip connection with the input mixture in order to obtain a $[L_s, F_c + 1]$ data block that passes through a last 1D convolutional layer which produces the predicted $K-1$ source waveforms; the data has shape $[L_s, K-1]$. Finally the K^{th} source is computed as explained earlier and concatenated in order to obtain the final predicted output of size $[L_s, K]$ that contains the K predicted separated source waveforms from the input mixture of size $[L_s, 1]$.

The Wave-U-Net architecture has shown really good performance and that the latter could even outperform the U-Net architecture for comparable settings [2].

III. Algorithms

As frequency content of a song or an isolated voice can be a good tool in order to assess performance, spectrogram calculation has been implemented and is explain in the first subsection. Moreover, two of the three presented architectures make use of spectrograms as data for the neural network.

In the framework of this voice isolation project, we decided to explore the different architectures presented in the section II. The architectures and algorithms we implemented for each of the three approaches are therefore explained latter in this section.

A. Constructing spectrograms

The input of the neural network algorithm is a *spectrogram*, a graph showing the frequency content of the song at different points in time. An example of log-scale spectrogram from one of the songs in the dataset is shown on Figure 6. The figure shows the time in x-axis and the different frequency bins in y-axis: the more a frequency bin is part of the song at the given time the more the corresponding point is red. Notably, the right part of the graph being blue shows that the song has been padded with zeros at the end because it was shorter than 3 minutes. Also, it is clearly visible that the extract of the song between 1:40 and 2:10 doesn't contain any vocal component.

Spectrograms are calculated using short-time Fourier transforms (STFT). It consists to split the sequence corresponding to the song in many overlapping shorter sequences and compute the Fourier transform of each of these sequences. The spectrogram is then the absolute value of this STFT and is most often plotted in log scale.

B. Convolutional deep neural network

The implementation of this algorithm is presented on Figure 5 and each step is detailed in the following paragraphs.

The first steps of the training phase are to import the songs, normalize their duration, and compute the spectrograms of the mixtures and vocals as explained in the previous sub-section. Then, the spectrograms are cut into batches of a defined number of values (typically 20 spectrogram values, corresponding to around 0.2s of song as a single spectrogram value is computed from many audio samples because of the STFT) and stored in numpy arrays. A single frequency mask is applied on each batch, meaning that the audio samples which are part of this batch are treated in the same way. In the contrary, samples which are part of different batches might have different frequency masks.

During training, we need to know the input of the neural network (mixture spectrograms) and the targeted output (frequency mask). To get the frequency mask from the songs, we need to compare the mixture spectrograms with the vocals spectrogram. For each frequency, we need to decide whether it should be kept for the target vocals: the criteria we decided to use is to check whether the vocals have a larger value than 20% of the value of the non-vocals. If this is the case, the

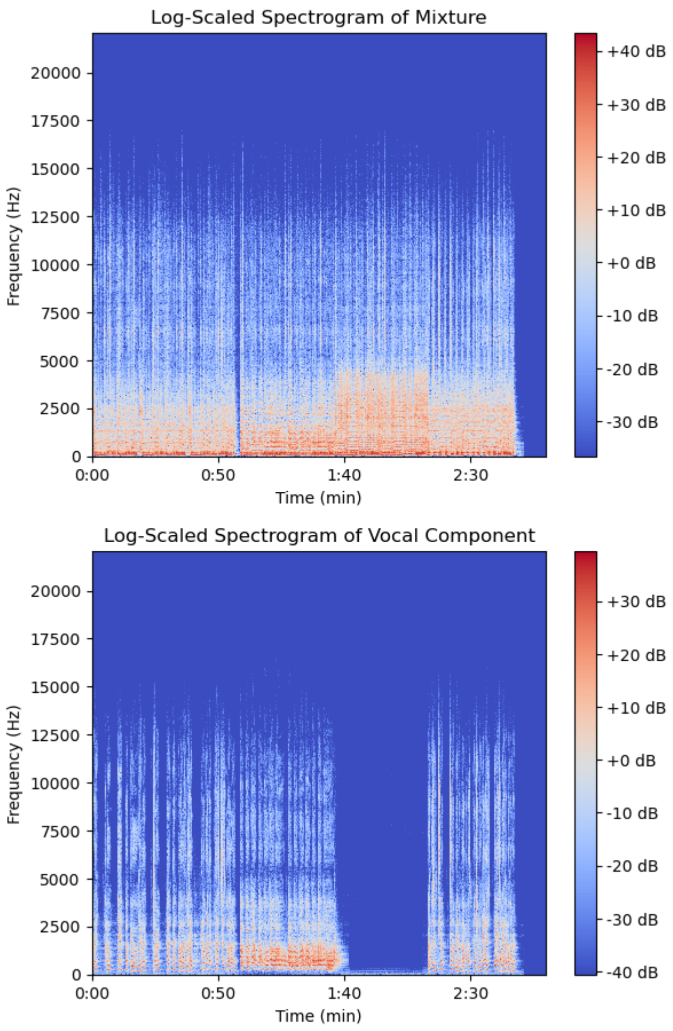
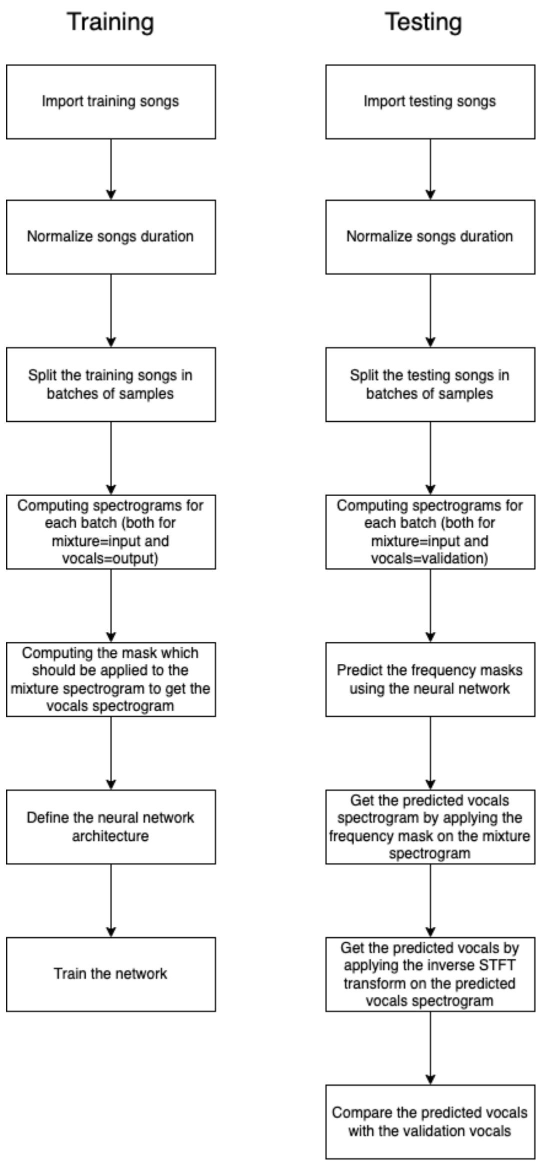


Fig. 4. Example of a spectrogram for one of the songs in the MUSDB dataset: upper image shows the spectrogram of the mixture and lower image the spectrogram of the vocal component

frequency is considered as one we need to keep the singer's voice. If it is not the case, the vocals are small enough that we can drop them to drop the non-vocals and isolate the voice. This value has been chosen by trial and error, checking which one is giving the best results.

A neural network model is then defined using the PyTorch library. It consists of 2 convolutional layers followed by 3 fully connected layers. The first convolutional layer uses a 3×3 kernel with a padding of 1 to keep the same data size, and it has 1 input channel and 16 output channels. The second layer has the same kernel and padding, but 16 input channels and 1 output channel.

Then, the fully connected layers have $20\,500$ neurons because the spectrograms size are 1025 frequency bins $\times$ 20 time bins. The output layer is however 1025 because we need a single binary value for each frequency. The activation



Training
Testing
Import training songs
Normalize songs duration
Split the training songs in batches of samples
Computing spectrograms for each batch (both for mixture=input and vocals=output)
Computing the mask which should be applied to the mixture spectrogram to get the vocals spectrogram
Define the neural network architecture
Train the network
Import testing songs
Normalize songs duration
Split the testing songs in batches of samples
Computing spectrograms for each batch (both for mixture=input and vocals=validation)
Predict the frequency masks using the neural network
Get the predicted vocals spectrogram by applying the frequency mask on the mixture spectrogram
Get the predicted vocals by applying the inverse STFT transform on the predicted vocals spectrogram
Compare the predicted vocals with the validation vocals

Fig. 5. Summary of the algorithm followed for the convolutional neural network architecture

function used is ReLu for each layer.

The training process involves setting the loss function (L1 loss) and iterating over the training data for a specified number of epochs which is maximum 25. After each epoch, the loss is calculated and displayed.

Finally, the trained model is tested by performing vocals isolation on a sample mixture from the MUSDB18 dataset. Just like during training, the testing song is imported, its duration is normalized, its spectrogram is computed and split in batches. The mixture spectrogram is fed into the model, and the corresponding frequency mask is obtained, allowing to get the predicted vocal spectrogram. The predicted vocal spectrogram is then converted back to the time domain the inverse STFT transform, resulting in the reconstructed isolated vocal audio. The initial vocals extract, reconstructed vocals extract, and initial mixture extract are plotted and saved as audio files.

C. U-Net

The first step in this algorithm concerns the datasets. Following [3], each audio included in the dataset is first downsampled from 44.1kHz to 8192Hz for more efficient computation. For each song in the MusDB dataset, a Short Time Fourier Transform with a window size of 1024, hop length of 768 frames has been computed and the square of its module has been computed to get its corresponding spectrograms. After that, patches with a length of 128 frames have been extracted from the audio's spectrogram and each one of them has been normalized such that its values lie in the range $[0, 1]$. The training dataset is composed of 100 audios and the testing dataset of 50 audios, after preprocessing the audios are in the form $(nb_{patches}, 512, 128)$. $nb_{patches}$ is a parameters depending on the original audio's length. However, because of limited computational resources respectively 50 and 20 tracks from each dataset were kept. The algorithm exclusively works on the magnitude of the STFT, therefore the output of the algorithm will correspond to the magnitude of the STFT of the vocals. To reconstruct the vocal's audio, this spectrogram is out together with the phase of STFT of the input mixture.

The model structure of the U-net follows [3] meaning that there are six convolution happening in the contracting path and six in the expanding path. Each convolution is performed with a stride of $(2, 2)$ meaning that at the bottleneck, the input's height and width are downsampled by a factor 2^6 while the number of channels reach 512.

The model has been trained for 5 epochs which could be higher but was kept low because of the restricted computational resources. During training, let X be the spectrogram of the audio that is containing both the vocals mixed and instrumental components and Y the spectrogram of the target audio – the isolated vocals. The loss function used to train the model is the $L_{1,1}$ norm (which is the sum if the absolute values of the elements of a matrix) of the difference between the target spectrogram and the masked input spectrogram:

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

where $f(X, \Theta)$ is the soft mask generated by the model. The Adam optimizer is used with a learning rate $\lambda = 0.001$.

D. Wave-U-Net

The Wave-U-Net algorithm that has been implemented is described in this section. The implemented algorithm tries to follow as much as possible the steps and parameters described in [2].

First, the training and testing datasets need to be generated. Indeed, audios are not directly used as direct inputs and targets for the neural network, they are preprocessed. In the framework of our implementation of the Wave-U-Net, inputs and outputs are waveform windows of size $L_s = 16384$. Also, we decided to test $K = 2$ number of source waveforms for the outputs, being the *vocals* and the *accompaniment* waveforms, while the input is the *mixture* waveform, being the sum of the *vocals* and *accompaniment* waveforms.
For each song in the MUSDB18 training dataset, L_s-long time windows for the three waveforms are cut are stored in a training loader. The same is done for the testing dataset. For both, waveform blocks are then shuffled between each epoch. Preprocessed datasets can be stored and ready to be loaded for future trainings and testings.

Once dataset have been preprocessed and have the suitable form for the neural network, the latter is constructed. The Wave-U-Net architecture is build. As an example, and following parameters used in [2], we set $L = 12$ layers, $F_c = 24$ extra channels per convolutional layer, $f_d = 15$ as filters size for convolutional blocks in the encoder layers and $f_u = 5$ as filters size for convolutional blocks in the decoder layers.

During the training, batches of size 16 are passing through the network at once and the maximal number of epochs can vary depending on the number of songs and the quantity of waveform windows per track we decided to consider in the dataset. Those are parameters we had to play with since it influences a lot the training complexity and computation duration. The overall goal is to consider the 100 songs with the maximum number of waveform windows per song. As the criterion, the MSE between the target $vocals$ and $accompaniment$ waveform blocks and the predicted ones is used for the loss function. The Adam optimizer is used with learning rate $\lambda = 0.0001$ with decay rates $(\beta_1, \beta_2) = (0.9, 0.999)$. Therefore, learning rate scheduling is done in order to reduce the learning rate when performance does not improve.

When the network has been trained, the latter is tested on the testing dataset.

IV. Results

Although we decided to explore the different architectures, the lack of time, the arrival of the deadline and the lack of access to a GPU to facilitate training and results analysis meant that we were unable to debug all the implementations. Therefore, CDNN and Wave-U-Net algorithms did not give fairly satisfactory results.
We spent a lot of time in troubleshooting and trying to identify the potential causes behind the inadequate loss convergence and poor performance. You can find the different codes for each architectures in the provided folder.

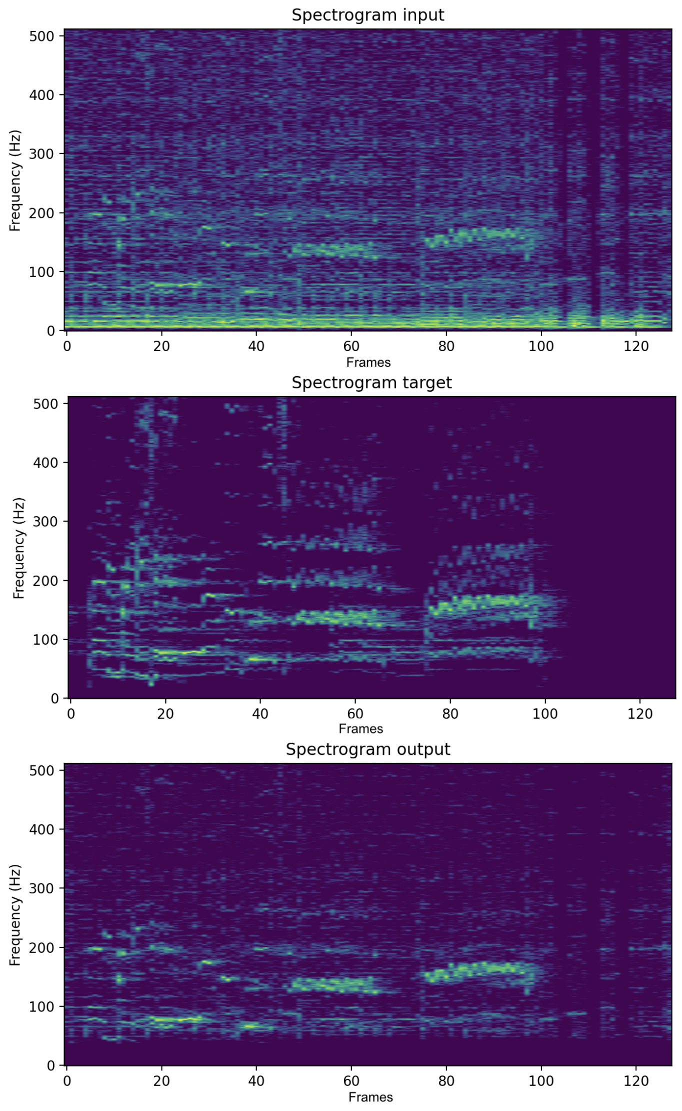


Fig. 6. Visual comparison between the input, target and output spectrograms. It is visible that the model do keep the most important features of the vocal component (the high intensity regions at the frequency bins 80, 120 and 160) that can be seen on the target patch.

In audio signal processing, comparison between a target signal and a predicted signal are made using metrics such as Signal-to-Distortion Ratio (SDR), Signal-to-Interference Ratio (SIR), or Signal-to-Artefact Ratio (SAR). They are evaluating the quality of the recovered signal by measuring the ratio between the signal and undesired artifacts or interference. The predicted signal s_p is considered to be the sum of the target signal s_t and three error terms: for noise e_n, interference e_i and artifacts e_a:

$$s_p = s_i + e_n + e_i + e_a$$

The SIR is calculated as

$$SIR = 10 \log 10 \frac{||s_t||^2}{||e_i||^2}$$

Number of layers of the U-Net	L1 difference
3	$1.516e^{-3}$
4	$1.308e^{-3}$
5	$9.8e^{-4}$

TABLE I
COMPARISON OF THE NEURAL NETWORK RESULTS FOR VARIATIONS IN THE KERNEL SIZE AND THE STRIDE IN THE U-NET ARCHITECTURE

and represents how much the predicted signal contains other interfering signals. The SAR represents the importance of the signal with respect to artifacts introduced by the processing algorithm

$$SAR = 10 \log 10 \frac{||s_t + e_i + e_n||^2}{||e_a||^2}$$

Finally, the SDR is the ratio of the signal amplitude with all other unwanted signals, it represents the overall quality of the prediction [4]:

$$SDR = 10 \log 10 \frac{||s_t||^2}{||e_n + e_i + e_a||^2}$$

In this paper, because of time constraints and not having a GPU to train, we were not able to obtain proper audio results with our algorithms. Therefore, we didn't have the opportunity to use these metrics directly for comparison. Instead, we relied on visual comparisons of spectrograms and calculated the L1 difference between them as a substitute measure. While this approach provides some information into the differences between the signals, it is not as insightful as analyzing metrics SDR, SIR, and SAR. As a reminder, the L1 difference between images X and Y is

$$e_{L1} = \sum_{i=1}^{N} \sum_{j=1}^{M} |X_{i,j} - Y_{i,j}|$$

An example of visualization of an input spectrogram, its corresponding target spectrogram and the output spectrogram predicted by the trained neural network is shown below.

Also, the average L1 difference has been computed for the testing dataset for different number of layers L of the architecture in order to assess the performance compared to the network complexity. These are presented in TableI.

V. CONCLUSION

After having reviewed different papers about audio voice separation, it is obvious that the use of neural networks for voice isolation purpose is more than useful. Recent presented architecture such as the U-Net and Wave-U-Net architectures or even more classical convolutional deep neural networks are all shown to be efficient in that framework.
During our project, we got the opportunity to study and implement all of them from scratch. The lack of time prevented us from correctly checking ours results with the U-net and improving CDNN and Wave-U-Net architectures to get suitable results. However, we observed good first results for the U-Net architecture and tried to vary the number of encoding and decoding layers in order to see how it can impact the performance through the L_1 loss on the predicted spectrograms.

REFERENCES

[1] A. Simpson, G. Roma, and M. Plumbley (2015), "Deep karaoke: Extracting vocals from musical mixtures using a convolutional deep neural network." in International Conference on Latent Variable Analysis and Signal Separation, pp. 429-436. Springer

[2] D. Stoller, S. Ewert, S. Dixon, "Wave-U-Net: A Multi-Scale Neural Network for End-to-End Audio Source Separation" in E-Print Archive: https://arxiv.org/abs/1806.03185

[3] A. Jansson, E. Humphrey, N. Montecchio, R. Bittner, R. Kumar and T. Weyde (2017), "Singing voice separation with deep U-Net convolutional networks" in 18th International Society for Music Information Retrieval Conference (ISMIR), pp. 745-751.

[4] E. Vincent, R. Gribonval and C. Fevotte (2006), "Performance measurement in blind audio source separation" in IEEE Transactions on Audio, Speech, and Language Processing, vol. 14, no. 4, pp. 1462-1469, doi: 10.1109/TSA.2005.858005.

APPENDIX

Repartition of the work

Semilogo and Lucas: spectrograms and convolutional deep neural network
Rayan: U-net network
Benoît: U-net and Wave-U-Net networks
The report has been written by all four of us