Wave-U-Net Architecture for Audio Source Separation Task

A report over the implementation of the actual paper on Wave-U-Net [1]

Harsh Kumar
Electrical Engineering
Indian Institute of Technology, Kanpur
harshku@iitk.ac.in

Shorya Kumar
Electrical Engineering
Indian Institute of Technology, Kanpur
shoryak@iitk.ac.in

***Abstract*—This is a report based on our implementation and analysis of WaveUNet model [1] for audio source separation. This model works on time domain information to produce separated sources in time domain. We used this to model for a two source separation problem in which we aim to separate songs into vocals and accompaniment respectively. We used the MIR-1K dataset to train our model.**

I. Introduction

Source Separation is the process of isolating sounds from individual sources in an auditory mixture of multiple sounds. We call each sound heard in a mixture a source.
Mathematically, we assume that a mixture signal $y(t)$ is composed of N sources, $x_n(t)$, for n=1...N, such that
$y(t) = \sum_{i=1}^{n} x_i(t)$
Given $y(t)$, we want to recover one or more $x(t)$'s. In our case we assume $n = 2$ and want to separate a song into vocals and accompaniment respectively. The authors of the paper use MUSDB dataset for training , while we tried to train the model on given MIR-1K dataset.

II. Domain Approach

Many previous works focus on analysis of the spectograms of the audio files and train models to work in the frequency domain. This includes the magnitude and phase information of the time series in the frequency domain. Though it is one of the most popular approach, the authors of this paper have come up with an end-to-end model to train on the time series data itself. The advantages have been listed below.

- Models for separation usually operate on magnitude spectrum, which ignores phase information.
- The STFT output depends on many parameters, such as the size and overlap of audio frames, which can affect the time and frequency resolution. Ideally, these parameters should be optimised with parameters of the separation model to maximise performance for a particular separation task. In practice, however, the transform parameters are fixed to specific values. [1]
- Secondly, the separation model does not estimate the source phase, it is often assumed to be equal to the mixture phase, which is incorrect for overlapping partials. Alternatively, the Griffin-Lim algorithm can be applied to find an approximation to a signal whose magnitudes are equal to estimated ones, but this is slow and often no such signal exists. [1]
- Lastly, the mixture phase is ignored in the estimation of sources, which can potentially limit the performance. Thus, it would be desirable for the separation model to learn to estimate the source signals including their phase directly. [1]

As an approach to tackle the above problems, several audio processing models were proposed that operate directly on time-domain audio signals. One of them is Wave-UNet model , which adapts the U-Net architecture for 1-D time series data.

A. Some other distinctions

- Strided transposed convolutions used in previous works for upsampling feature maps are replaced with linear interpolation followed by a normal convolution to avoid artifacts.

III. Data Preprocessing

Here we mention the modifications we made to the dataset for making it compatible with our proposed Model.

- We used only the undivided 2-channel audio files present in MIR-1K, and created the input and output dataset by merging the channels and separating them respectively.
- Hence our model takes a 1-channel audio (Mono) file as input and gives 2 1-channel audio (Mono) files, one for the music accompaniment and one for the vocals.
- The audio files had lengths varying from 22 secs to 126 secs. We clipped all audio files to 1 sec audio files (sampled at 16kHz) due to machine specifications and constraints. Thus we got around 7947 1-sec input samples.

IV. Base Model Architecture

Fig. 1 describes the original baseline model architecture as given in [1]. It is an adaptation of the previously proposed U-Net architecture. It has 1D convolutions and uses down-sampling and upsampling layers to reconstruct the sources

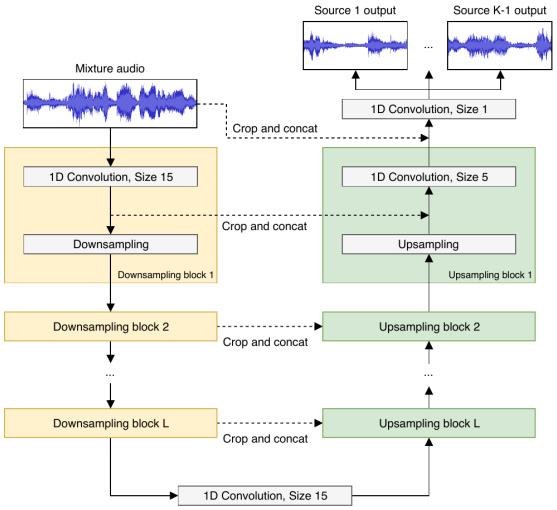

Fig. 1. WaveU-Net Model Architecture [1]

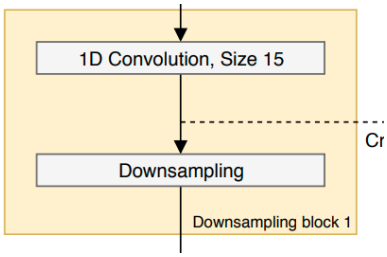

Fig. 2. Downsample Block [1]

individually from the mixture audio provided. Below are the different modules (or types of blocks) implemeneted in this model.

A. Downsampling Block (Fig. 2)

- **Convolution** : This is a general 1D convolutional layer with a fixed kernel size and fixed number of filters for each block individually. We take a stride of 1, hence the output of this layer would be L - k + 1 where L is the initial length of the 1D input to this layer and k is the kernel size. This follows no padding and an activation layer of Leaky-ReLU.
- **Downsampling** : Here the resolution of the input, given to this layer, is halved by dropping out every alternate frame/element of the input and hence the output length being half the input length.

B. Upsampling Block (Fig. 3)

- **Upsampling** : Here the resolution of the input is almost doubled by expanding the input into twice its length and filling the gaps with the mean of the values of their neighbours. Note that this is NOT done for the first and last value. So, a L length input becomes a 2L - 1 length output.
- **Crop and Concat** For the ith upsampling block, the output of the convolutional layer of the ith downsampling block is cropped to fit the input of this layer and then concatenated on the input along the channels.
- **Convolution** : This is a general 1D convolutional layer similar to the one used in the downsampling block.

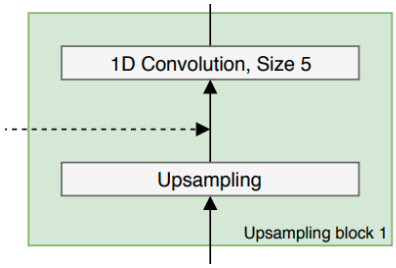

Fig. 3. Upsample Block

C. Extra convolutional Layers

- **After Downsampling** : After all the downsampling blocks, a similar 1D Convolutional Layer is implemented for a final extraction of useful features.
- **After Upsampling (Output Layer)** : After all the upsampling and concatenation at each Upsample Block, an extra convolutional layer is applied after the last upsampling block with kernel size 1, which gives 2 channels as outputs , vocals and accompaniment respectively.

V. Upsampling: Transposed Convolutions v/s Interpolation

- Upsampling artifacts are caused by problematic upsampling layers and due to spectral replicas that emerge while upsampling. Also, depending on the used upsampling layer, such artifacts can either be tonal artifacts (additive high-frequency noise) or filtering artifacts (subtractive, attenuating some bands). [3]
- Unfortunately, deconvolution can easily have “uneven overlap,” putting more of the metaphorical paint in some places than others . In particular, deconvolution has uneven overlap when the kernel size (the output window size) is not divisible by the stride (the spacing between points on the top). While the network could, in principle, carefully learn weights to avoid this, in practice neural networks struggle to avoid it completely. [2]
- When such convolutions were used as upsampling blocks in the Wave-U-Net model, artifacts were found in the form of high-frequency buzzing noise.
- Thinking about things in terms of uneven overlap is — while a useful framing — kind of simplistic. For better or worse, the model learns weights for their deconvolutions. In theory, our models could learn to carefully write to unevenly overlapping positions so that the output is evenly balanced.
- In fact, not only do models with uneven overlap not learn to avoid this, but models with even overlap often learn kernels that cause similar artifacts! While it isn’t their default behavior the way it is for uneven overlap, it’s

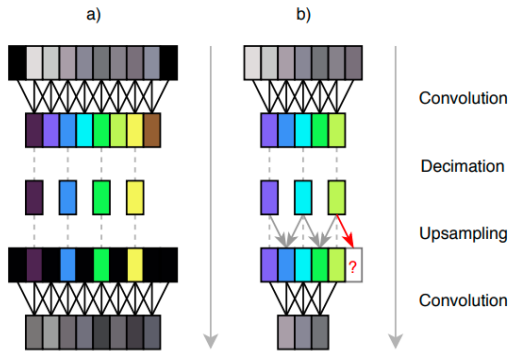

Fig. 4. Preservation of Input Context [1]

still very easy for even overlap deconvolution to cause artifacts.

VI. Proposed Improvements

- **Difference Layer** : The baseline model outputs one source estimate for each of K sources by independently applying K convolutional filters followed by a $tanh$ nonlinearity to the last feature map.

$$M \approx S_1 + S_2 + ... + S_K$$

Since the baseline model is not constrained in this fashion, it has to learn this rule approximately to avoid highly improbable outputs, which could slow down learning and reduce performance. Therefore, the authors use a difference output layer to constrain the outputs S_j such that only $K-1$ source signals are estimated and the last source is then simply computed as

$$S_k \approx M - (S_1 + S_2 + ... + S_{K-1})$$

- **Learned Upsampling** [1] : Linear interpolation for upsampling is simple, parameter-less and encourages feature continuity. However, it may be restricting the network capacity too much. The authors have proposed the learned upsampling layer. For a given Fn feature map with n time steps, we compute an interpolated feature $f_{t+0.5} \in R_F$ for pairs of neighbouring features f_t, $f_{t+1} \in R_F$ using parameters $w \in R_F$ and the sigmoid function σ to constrain each $w_i \in w$ to the [0, 1] interval:

$$f_{t+0.5} = \sigma(w) \odot f_t + (1 - \sigma(w)) \odot f_{t+1}$$

The learned interpolation layer can be viewed as a generalisation of simple linear interpolation, since it allows convex combinations of features with weights other than 0.5.

- **Prediction with proper input context and resampling** (Fig. 4) [1] : In earlier attempts, the input and the feature maps were padded with zeros before convolution, so that the resulting feature map does not change in its dimension. This simplifies the network's implementation, since the input and output dimensions are of the same size. This padding of zeroes at the boundaries results in loss of input context information at the boundaries and it becomes difficult for the model to learn output at the boundaries. As a solution, we employ convolutions without implicit padding and instead provide a mixture input larger than the size of the output prediction, so that the convolutions are computed on the correct audio context. Since this reduces the feature map sizes, we constrain the possible output sizes of the network so that feature maps are always large enough for the following convolution.

TABLE I
Model Summary

Block	Layer	Output Shape
Input		(16000, 1)
Downsample Block for $i = 1, ..., L$	Conv1D($F_c \cdot i, f_d$) Downsample	(49, 192)
	Conv1D($F_c \cdot (L+1), f_d$)	(35, 216)
Upsample Block for $i = L, ..., 1$	Upsample Crop and Concat Conv1D($F_c \cdot i, f_d$)	(7685, 24)
Output	Conv1D(1, 2)	(7685, 2)

TABLE II
Model Evaluation

Type	Total Epochs	Signal-to-Distortion Ratio (SDR)			
		Median	MAD	Mean	SD
Vocal	100	-0.4329	0.5512	-0.5202	0.8152
	200	-0.8370	0.6126	-0.8407	0.8646
	300	-1.0216	0.6029	-1.0023	0.8566
	400	-1.2467	0.6391	-1.2168	0.8861
	500	-1.2849	0.6491	-1.2397	0.9009
	600	-1.3469	0.6319	-1.3037	0.8867
	700	-1.4034	0.6324	-1.3524	0.8825
	800	-1.5218	0.6511	-1.4373	0.9017
	900	-1.5303	0.6391	-1.4672	0.8926
	1000	-1.4877	0.6396	-1.4205	0.8941
	1100	**-1.5109**	**0.6411**	**-1.4456**	**0.8913**
Music	100	-0.4434	1.2433	-1.0678	3.5087
	200	-0.7363	1.3396	-1.3811	3.6966
	300	-1.0721	1.5708	-1.8242	4.1163
	400	-1.1123	1.5117	-1.8042	3.9806
	500	-1.1140	1.4792	-1.7698	3.8869
	600	-1.2270	1.5823	-1.9626	4.1572
	700	-1.3593	1.6217	-2.0919	4.2125
	800	-1.3831	1.6519	-2.1277	4.2767
	900	-1.4111	1.6945	-2.1849	4.3931
	1000	-1.3908	1.6915	-2.1794	4.3463
	1100	**-1.4449**	**1.6921**	**-2.2155**	**4.3200**

VII. Implementation

We have implemented the same Architecture as described in the paper including some of the architectural improvements mentioned such as *Proper Input Context and Resampling*

- **# Layers** : 8 (= L)
- **Kernel Size** : 15 (= f_d) for Convolutions in Downsampling Blocks and 5 (= f_u) Convolutions in Upsampling Blocks.
- **Filters** : 24 (= F_c) extra filters in each layer. The i^{th} Downsampling layer will have a Convolutional layer with

$24 * i$ filters. Similarly the extra convolutional layers after all the Downsampling Blocks will have $24*(L+1)$ filters. Similarly, the ith Upsampling layer (counted in the reverse order) will have a Convolutional layer with $24 * i$ filters too. The output convolutional layer has 2 filters.

VIII. Observations and Results

We trained the above cited model with the following hyper-parameters -

- **# epochs** : 1100
- **Batch size** : 16
- **Loss Function** : Mean Squared Error
- **Optimizer** : Adaptive Moment Estimator (Adam)

We used the SDR (Signal-to-Distortion Ratio) to evaluate our model. Follwing are the SDR values and SDR related statistics after each 100 epochs. We calculate the Median, the Mean Absolute Deviation (MAD), the Mean and the Standard Deviation (SD).

IX. Observations and Conclusions

Preferring time domain analysis over frequency domain analysis did not give us any good results. The model followed the same concepts as discussed in [1] but after the training it failed to show improvements. The metric used, Signal-to-Distortion Ratio (SDR), was observed to be decreasing after each 100 epochs, where in reality, it should have been increasing to prove that our model was working correctly. Even the audio output where noisy and no differentiation could be made. This was confusing as the loss metric used, Mean Squared Error (MSE), kept decreasing after each epoch leading us to believe that the model was training fine.

Models based on frequency domain analysis are widespread while there are very few end-to-end models that deal with time domain representation of audio directly. The results show us why. Hence we conclude that the results by this model are not satisfactory. As a team, we need to look into possible erroneous implementation of concepts and rectify them to see if the model still matches with the claimed results in [1]. A comparative study can also be done by between this model and an analogous model with frequency domain analysis to see the effect of the change of domain.

References

[1] Stoller, D., Ewert, S., and Dixon, S. (2018). "Wave-u-net: A Multi-Scale Neural Network for End-To-End Audio Source Separation," in Proceedings of the 19th International Society for Music Information Retrieval Conference (ISMIR), 334–340.

[2] Odena, et al., "Deconvolution and Checkerboard Artifacts", Distill, 2016. http://doi.org/10.23915/distill.00003

[3] Jordi Pons, et al. (2021), "Upsampling Layers for Music Source Separation", https://arxiv.org/pdf/2111.11773.pdf