VoiceFilter: Source separation using Speaker Embedding

Aditya Sharma 170050043 Suraj 170050044 Rohan Abhishek 170050078

1 Introduction

The main objective of the speaker separation models is to segregate voice signals of different speakers from a sample of mixed audio. There are Voice to text systems which are already giving high accuracy results. But these systems often do not work in multi-speaker scenario. Also there are cases when you are only interested in the audio signals of a single or multiple speakers but decoupled and separated. In this case too, separation models and speaker dependent separation models are highly useful.

2 Task Definition

Problem Statement: Separating the voice of a target speaker from multi-speaker signals.

Input Output Behaviour: The input to our system will be 2 audio files: *reference_audio* and *noisy_audio*.

Reference_audio is used to determine a `d-vector` for the reference/target speaker whose audio is to be extracted from the *noisy_audio*. The output will be the audio of the target speaker extracted from the *noisy_audio* after removing background noise and audio from other speakers, if any.

The main neural network is given 2 inputs: a d-vector of the target speaker and magnitude spectrogram computed from the noisy audio. The neural network returns a mask to be applied to the *noisy_audio* to obtain the *enhanced_audio*.

3 Methodology

There are two major methods for separating voice of a target speaker from multi-speaker signals. We have followed Method II.

3.1 Method I

Initially extracting all audio signals(for each speaker) and then decide which output corresponds to the target speaker. This method suffers with these common problems:

- The loss function must be Permutation Invariant in case of multiple speakers.
- The system must estimate the number of speakers present in the noisy output, otherwise the outputs will not be good.
- Choosing the output to select for the target speaker (speaker verification) on each output may not be efficient for multiple interfering speakers.

3.2 Method II

Using the `d-vector` of the speaker as input along with the Noisy Audio spectrogram to generate a mask. This mask is then applied to the Noisy Audio spectrogram to get the target audio.

`d-vector:` Speaker embedding vector, obtained using a reference audio of the target speaker.

- This system(as per the results from the authors) outperforms the Permutation Invariant loss based system. This shows that not only our system solves the two aforementioned issues, but using a speaker embedding also improves the capability of the system to extract the source of interest (with a **higher SDR**).

3.3 Model Description

Our project includes two models.

- Speaker Encoder: This is used to produce the speaker embedding(d-vector) from a reference audio.
- VoiceFilter system: This is the main system used to generate the mask which is used to generate the target audio.

3.3.1 Speaker Encoder

We have used a pre-trained model provided in the unofficial implementation by Seung-won Park (MINDs Lab, Inc), which consists of the following details.

- The speaker encoder is a **3-layer LSTM** network trained with the **generalized end-to-end loss**.
- It takes as inputs log-mel filterbank energies extracted from windows of `1600 ms`, and outputs speaker embeddings, called `d-vectors`, which have a fixed dimension of `256`.
- To compute a `d-vector` on one utterance, we extract sliding windows with `50%` overlap, and average the L2-normalized d-vectors obtained on each window.

3.3.2 VoiceFilter System

As shown in the figure 1, here's the description for the model of VoiceFilter System

- A magnitude spectrogram is computed by applying STFT (Short-time Fourier transform) on the Noisy Audio.
- The VoiceFilter system neural network takes input the `d-vector` of target speaker and spectrogram computed of the Noisy Audio. This network returns a mask to filter the audio.
- The mask is multiplied element-wise to the input spectrogram to obtain the masked spectrogram.
- The masked spectrogram undergoes Inverse STFT to produce the enhanced audio version of the target speaker.

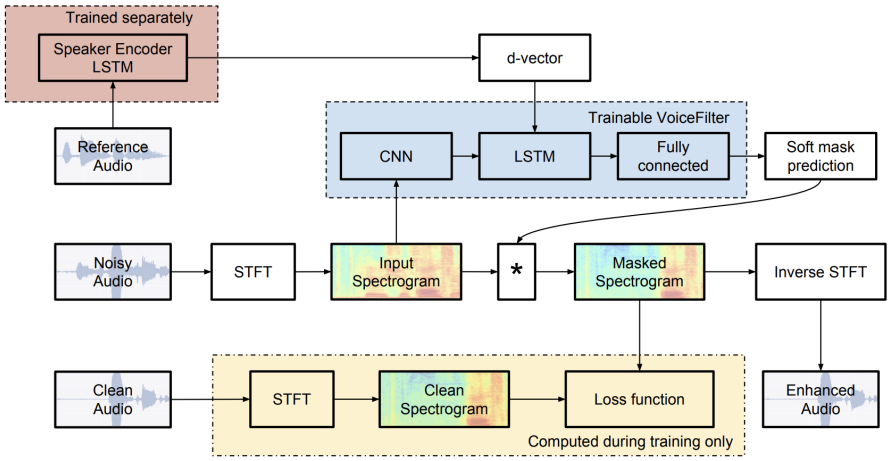


Figure 1: System Architecture of VoiceFilter System

Layer	Width		Dilation		Filters / Nodes
	time	freq	time	freq	
CNN 1	1	7	1	1	64
CNN 2	7	1	1	1	64
CNN 3	5	5	1	1	64
CNN 4	5	5	2	1	64
CNN 5	5	5	4	1	64
CNN 6	5	5	8	1	64
CNN 7	5	5	16	1	64
CNN 8	1	1	1	1	8
LSTM	-	-	-	-	400
FC 1	-	-	-	-	600
FC 2	-	-	-	-	600

Figure 2: Parameters of the VoiceFilter Network

The VoiceFilter network is composed of **8 convolutional** layers, **1 LSTM** layer, and **2 fully connected** layers, each with **ReLU** activations except the last layer, which has a **sigmoid** activation. The values of the parameters are provided in Figure 2.

4 Implementation Details

The whole project is implemented on Python Jupiter Notebooks.

```python
import librosa
import numpy as np

import torch
import torch.nn as nn
from tensorflow.keras.utils import Sequence
```

```python
def get_dvector(wave):
  mel_spec = audio.get_mel_spec(wave)
  dvec = embedder(torch.from_numpy(mel_spec).float())
  dvec = dvec.detach().numpy()
  return dvec
```

```python
class HyperParams:
  n_fft= 1200
  num_freq= 601 # n_fft//2 + 1
  sample_rate= 16000
  hop_length= 160
  win_length= 400
  min_level_db= -100.0
  ref_level_db= 20.0
  preemphasis= 0.97
  power= 0.30
  embedder_window= 80
  data_audio_len= 3.0
  embedder_num_mels= 40
  embedder_lstm_hidden = 768
  embedder_emb_dim = 256
  embedder_lstm_layers = 3
  embedder_window = 80
  embedder_stride = 40
  model_lstm_dim = 400
  model_fc1_dim = 600
  model_fc2_dim = 601 # num_freq
```

```python
class SpeechEmbedder(nn.Module):
    def __init__(self, hp):
        super(SpeechEmbedder, self).__init__()
        self.lstm = nn.LSTM(hp.embedder_num_mels,
                            hp.embedder_lstm_hidden,
                            num_layers=hp.embedder_lstm_layers,
                            batch_first=True)
        self.proj = LinearNorm(hp)
        self.hp = hp

    def forward(self, mel):
        # (num_mels, T)
        mels = mel.unfold(1, self.hp.embedder_window, self.hp.embedder_stride) # (num_mels, T', window)
        mels = mels.permute(1, 2, 0) # (T', window, num_mels)
        x, _ = self.lstm(mels) # (T', window, lstm_hidden)
        x = x[:, -1, :] # (T', lstm_hidden), use last frame only
        x = self.proj(x) # (T', emb_dim)
        x = x / torch.norm(x, p=2, dim=1, keepdim=True) # (T', emb_dim)
        x = x.sum(0) / x.size(0) # (emb_dim), average pooling over time frames
        return x
```

Figure 3: Library imports and hyperparameters of the model

4.1 Evaluation Metrics

Mainly, there are 2 metrics which can evaluate our output. We have used SDR in our experminents.

4.1.1 WER (Word Error Rate)

- Calculate WER for *noisy_audio* and *output_audio* w.r.t the test of the target speaker.
- WER should be reduced in multi-signal scenarios, which means the output audio is good.
- WER should remain same for single speaker scenarios, which means that the system is not degrading the performance and recognizing the speaker correctly.

4.1.2 SDR (Source to Distortion Ratio)

- SDR is a very common metric to evaluate source separation systems. It is an energy ratio between the energy of the target signal contained in the enhanced signal and the energy of the errors (coming from the interfering speakers and artifacts). **Thus, the higher it is, the better**.

5 Experimental Setup

The experimental setup mainly includes the data preparation. We have used the LibriSpeech Dataset for the project.

5.1 Dataset Preparation

As described in the System Architecture above, for each sample, we need 3 audio inputs, namely the *reference audio* to calculate the *d-vector* for the target speaker, the *clean Audio* to compute the clean spectrogram which is used to calculate the training loss and the *interference audio* to produce the *noisy audio* which is the input to the *VoiceFilter* model.

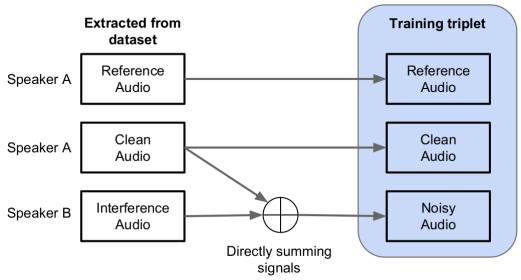


Figure 4: *Input data processing workflow*

The above figure describes the processing to generate each data sample. The *reference audio* and the *clean audio* are selected at random from the same speaker(A). The *interference audio* is then selected randomly from a different speaker(B). The *clean* and *interference* audios are directly added and the result is trimmed to the length of the *clean* audio to generate the *noisy audio*.

All the datasets (Train, Validation and Test) are prepared in the same way and the following subsections describe the statistics about the prepared data.

5.1.1 Train Dataset

LibriSpeech *train-clean-100.tar.gz* which consists audio samples from 251 different speakers is used to generate the training samples.

The train dataset prepared is divided into two separate datasets(*Train1* and *Train2*) are generated because of the size.

Train Set 1: It consists a total of 16000 samples with a total size of approx. 43GB.
Train Set 2: It consists a total of 7594 samples with a total size of approx. 20.5GB.
Approx. 1.17% training samples consists of same *reference* and *clean* audios.

5.1.2 Validation Dataset

LibriSpeech *dev-clean.tar.gz* which consists audio samples from 40 different speakers is used to generate the validation samples.
Validation Set: It consists 921 samples with a total size of approx. 2.5GB.
Approx. 1.09% validation samples consists of same *reference* and *clean* audios.

5.1.3 Test Dataset

LibriSpeech *test-clean.tar.gz* which consists audio samples from 40 different speakers is used to generate the test samples.
Test Set: It consists 950 samples with a total size of approx. 2.6GB.
Approx. 1.05% test samples consists of same *reference* and *clean* audios.

6 Experiments, Results and Discussion

This section consists the training, validation and testing of the VoiceFilter Model and the obtained results.

6.1 Training the VoiceFilter Model

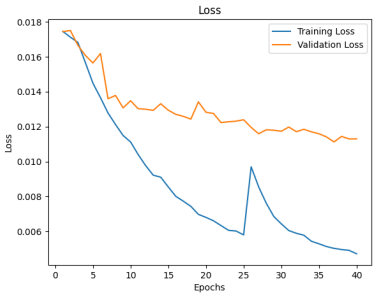
The model is trained using the training sets, *Train Set 1* and *Train Set 2*. Due to the large size of the dataset, we had very limited GPU options available. We have used **Google Colab Pro (GPU Runtime)** for the model training.

The model is trained for 25 epochs using *Train Set 1 (16000 samples)*. Each epoch took approx. 31 minutes resulting in a total of 13 hours.
After that, the model is trained for 15 epochs using *Train Set 2 (7594 samples)*. Each epoch took approx. 15 minutes resulting in a total of 3.7 hours.
In summary, the model is trained for 40 epochs resulting in 16.7 hours.

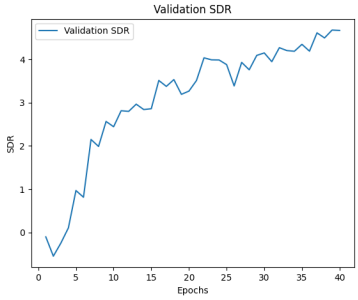
As shown in the plot below, the training loss started with a value of 0.0175 and improved to 0.0047 after 40 epochs.

6.2 Validation Results

The model is validated using the *Validation Set*. The following plots represent the results obtained.



(a) Training and Validation Loss



(b) Validation SDR

The validation loss started with a value of 0.0174 and improved to 0.0113 after 40 epochs. The validation SDR started with a negative value of -0.10 and improved to 4.68. Also, as we can see, the model after 39 epochs performs better than the final model. So, that is used for evaluating the model on the test set.

6.3 Testing Results

The model is tested on the *Test Set*. The following table represents the SDR values obtained using the model checkpoints after every 10 epochs.

As we can see in the table 1, the model which performs the best on Validation also performs the best on Test set.
We have achieved an SDR value of **4.681** on the test set, which is slightly less than the achievable value because of the limitation on resources including time.

Table 1: SDR on Test set

Number of Epochs	Test SDR
10	2.277
20	3.267
30	4.398
39	4.681
40	4.672

6.4 Sample Demo Examples

We have used the trained model to create some samples on different audio inputs and check the filtered audio manually.
Sample demo audios for the below four cases are present at this link.

6.4.1 Case 1: General Case (2 speakers, Noisy input audio)

Description: In this case, two audios(reference and clean) are selected from the primary speaker and one interference audio is selected from the secondary speaker.
The noisy audio is calculated by directly summing the clean and the interference audio.
The noisy audio is then provided as input to the VoiceFilter model returning the soft mask used to get the filtered audio.
Expected Output: The model is expected to suppress the interference audio and the filtered audio should be similar to the clean audio.

6.4.2 Case 2: Single Speaker Audio (Same reference speaker)

Description: In this case, two audios(reference and clean) are selected from the primary speaker and there is no interference, i.e., the noisy audio is same as the clean audio.
This is to test if we are able to correctly detect the target speaker or not.
Expected Output: As the input only consists only the audio signal of the target speaker, the output audio should be same as the input audio (may suppress any background noise, if present).

6.4.3 Case 3: Single Speaker Audio (Different reference speaker)

Description: In this case, two audios(reference and clean) are selected from different speakers and the noisy audio is same as the clean audio.
Expected Output: As the input does not consist any audio signal of the target speaker, the output audio should be completely suppressed and no audio should be present.

6.4.4 Case 4: Multi-signal Case (2 interference speakers)

This is similar to Case I, except that two interference audios are used. This is just to test how the model performs in cases when the noise is more than the samples used to train it.

Error Analysis: The model does not perform as efficiently as Case I because VoiceSplit is not trained for these cases explicitly. We can improve the performance in these cases by creating the training samples similarly.

7 Summary

The project shows the effectiveness of speaker dependent separation models. However there are some improvements which can easily improve the performance of the model:

1. Increasing training epochs and training on a larger dataset.
2. Adding more interfering speakers to the synthetic dataset we created. This should make the model for robust.

With the current model, we were able to achieve SDR of 4.681 as compared to the best SDR of 6.17 after 3,000 hour training with the VoiceSplit model[4] (unofficial implementation of the reference paper). For further extension, different embedding should be tested for validation accuracy. This is because different embedding types characterise different propoerties of the speaker. We can also test it for multiple embeddings. Testing on different loss functions may also help increase SDR by a significant amount.

References

[1] VoiceFilter: Targeted Voice Separation by Speaker-Conditioned Spectrogram Masking Quan Wang, Hannah Muckenhirn, Kevin Wilson, Prashant Sridhar, Zelin Wu, John Hershey, Rif A. Saurous, Ron J. Weiss, Ye Jia, Ignacio Lopez Moreno.

[2] Unofficial PyTorch implementation of Google AI's VoiceFilter system

[3] Unofficial Keras implementation of Google AI VoiceFilter

[4] VoiceSplit: Targeted Voice Separation by Speaker-Conditioned Spectrogram