Audio denoising using STFT

Portfolio's link: Muhammad Faraz Malik's Portfolio

BY: Muhammad Faraz Malik

Date: 7th December,2024

Table of Contents

Abstract

The project "Audio Denoising Using STFT" aims to improve the clarity of audio signals by removing noise. Using Short-Time Fourier Transform (STFT) and specific filtering techniques, the project tackles audio noise problems, leveraging real-world datasets from FSD50K. The implementation includes parameter tuning, spectrogram construction, noise spectrum estimation, and the calculation of time-frequency attenuation maps. By analyzing input and output signals visually and aurally, the project showcases the effectiveness of STFT-based denoising in enhancing audio quality

Introduction

Problem Statement

Real-world audio signals often contain noise that disrupts their clarity and usability. Removing such noise is critical for various applications, including communication, entertainment, and machine learning tasks.

Importance of the Problem

Noise removal ensures better quality in audio processing systems, improves human understanding in noisy environments, and enhances the performance of automated systems relying on audio input.

Techniques to Tackle the Problem

Review of Previous Work

Past studies have employed methods like bandpass filtering, spectral subtraction, and machine learning models for noise removal. However, these techniques often struggle with balancing noise suppression and signal preservation.

Chosen Techniques

STFT, combined with attenuation map computation, was selected for its ability to provide fine-grained noise suppression without significantly distorting the original signal.

Methodology

Dataset Preparation:
The Freesound Dataset (FSD50K) was chosen due to its diversity and extensive collection of real-world audio samples. It provides a robust foundation for evaluating the denoising algorithm.

Spectrogram Construction:
Using the STFT, the audio signal is divided into overlapping time segments, creating a spectrogram representation. This allows for the analysis of both time and frequency domains simultaneously.

Noise Spectrum Estimation:
A specific time interval within the spectrogram is selected to calculate the average noise power. This forms the basis for estimating the noise profile across the signal.

SNR Estimation:
The signal-to-noise ratio (SNR) is estimated by comparing the power of the signal components against the estimated noise profile. An optional prior SNR filter is applied to improve accuracy.

Attenuation Map Construction:
A time-frequency attenuation map is computed to suppress noise while preserving the signal's integrity. Adjustable parameters such as lambda and beta provide flexibility in denoising intensity.

Inverse STFT:
The denoised spectrogram is transformed back into the time domain using an inverse STFT, resulting in a reconstructed audio signal. Overlapping windows and a normalization process ensure the smoothness of the output.

Evaluation and Visualization:
The results are evaluated by comparing the original noisy audio signal with the denoised output. Temporal plots and spectrograms are generated to visualize the improvements in signal clarity and noise suppression.

By following this structured methodology, the project effectively mitigates noise while preserving the essential characteristics of the audio signal, making it suitable for various real-world applications.

Dataset

The Freesound Dataset (FSD50K) is used, containing over 50,000 real-world audio samples. This diverse dataset ensures robust training and evaluation of the denoising algorithm.

Matlab Code

```
clc;  fprintf('--- Audio denoising --\n\n');
% Load the noisy audio signal

fprintf('-> Step 1/5: Load audio:');
[y, Fe] =
audioread("Recording.wav");    Nx =
length(y);    fprintf(' OK\n');

% Algorithm parameters
apriori_SNR = 1;   alpha
= 0.05;       beta1 = 0.5;
beta2 = 1;    lambda = 3;

% STFT parameters   NFFT = 1024;
window_length = round(0.031 * Fe);   window
= hamming(window_length);   window =
window(:);
overlap = floor(0.45 * window_length);

% Signal parameters   t_min
= 0.3;      t_max
= 2.0;
```

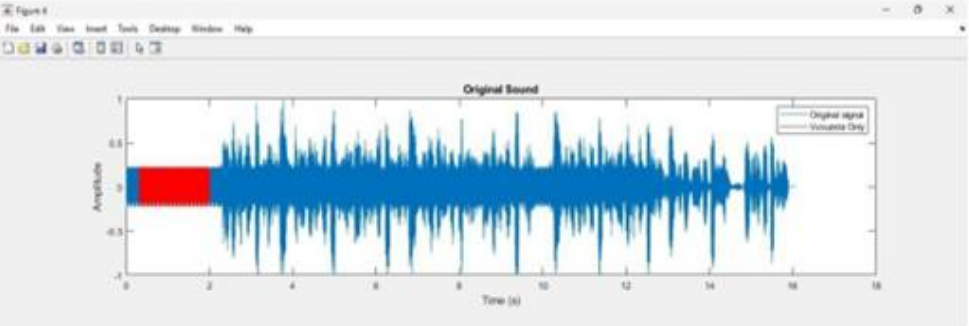
```
% Construct spectrogram    fprintf('-> Step 2/5:
Constructing spectrogram -');
[S, F, T] = spectrogram(x + i * eps, window, window_length - overlap, NFFT, Fe);
[Nf, Nw] = size(S);
fprintf(' OK\n');

% Noisy spectrum extraction    fprintf('->
Step 3/5: Extract noise spectrum -');    t_index
= find(T > t_min & T < t_max);   mgntopwr =
abs(S(:, t_index)).^2;   noisyspec =
mean(mgntopwr, 2);   acrsallwndows =
repmat(noisyspec, 1, Nw);   fprintf(' OK\n');

% Estimate SNR    fprintf('-> Step 4/5:
Estimate SNR -');   absS = abs(S).^2;
SNR_est = max((absS ./ acrsallwndows) - 1, 0);   if apriori_SNR
== 1
   SNR_est = filter((1 - alpha), [1 - alpha], SNR_est);   end      fprintf('
OK\n');

% Compute attenuation map    fprintf('-> Step 5/5: Compute TF
attenuation map -');    an_lk = max((1 - lambda * ((1 ./ (SNR_est +
1)).^beta1)).^beta2, 0);    STFT = an_lk .* S;    fprintf(' OK\n');

% Compute Inverse STFT    fprintf('-> Computing
Inverse STFT:');   ind = mod((1:window_length) -
1, Nf) + 1;
output_signal = zeros((Nw - 1) * overlap + window_length, 1);

for indice = 1:Nw        left_index =
((indice - 1) * overlap);
index = left_index + [1:window_length];       temp_ifft
= real(ifft(STFT(:, indice), NFFT));
   output_signal(index) = output_signal(index) + temp_ifft(ind) .* window;   end
fprintf(' OK\n');

% Normalize the output signal   output_signal = output_signal /
max(abs(output_signal)); % Normalize to [-1, 1]
```

```
% Clip the output signal to prevent exceeding amplitude limits   output_signal(output_signal
> 1) = 1;     output_signal(output_signal
< -1) = -1;

% Display temporal signals and
spectrogram    figure   subplot(2,1,1);
t_index = find(T > t_min & T < t_max);
plot([1:length(x)] / Fe, x);   xlabel('Time
(s)');   ylabel('Amplitude');   hold on;
noise_interval = floor([T(t_index(1)) * Fe:T(t_index(end)) * Fe]);    plot(noise_interval
/ Fe, x(noise_interval), 'r');   hold
off;
legend('Original signal', 'Vuvuzela Only');   title('Original
Sound');

% Show denoised signal   subplot(2,1,2);
plot([1:length(output_signal)] / Fe, output_signal);    xlabel('Time
(s)');
ylabel('Amplitude');   title('Sound without
Vuvuzela');

% Show spectrogram of original sound
t_epsilon = 0.001;    figure
S_one_sided = max(S(1:length(F) / 2, :), t_epsilon);
pcolor(T, F(1:end/2), 10 * log10(abs(S_one_sided)));   shading
interp;   colormap('hot');
title('Spectrogram: Speech + noise');
xlabel('Time (s)');
ylabel('Frequency (Hz)');

% Show spectrogram of denoised sound   figure
S_one_sided = max(STFT(1:length(F) / 2, :), t_epsilon);
pcolor(T, F(1:end/2), 10 * log10(abs(S_one_sided)));   shading
interp;   colormap('hot');   title('Spectrogram:
denoised');   xlabel('Time (s)');
ylabel('Frequency (Hz)');
```

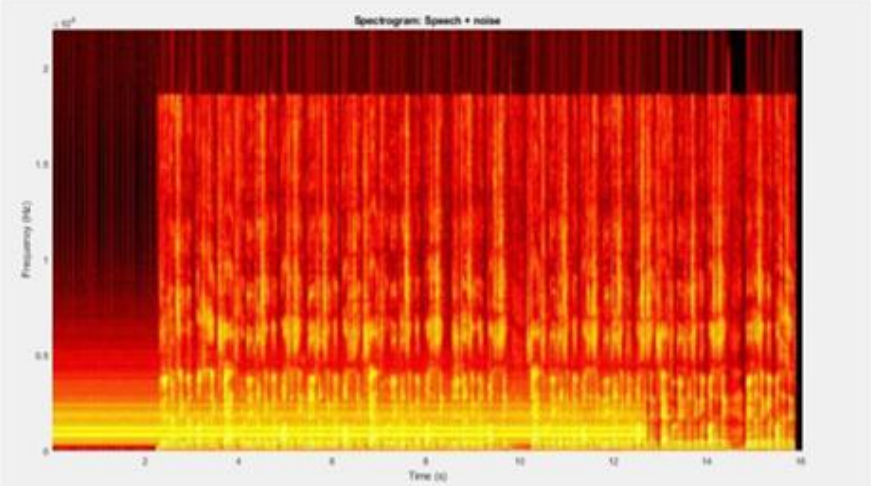
% Listen to results audiowrite('Recording_processed.wav', output_signal, Fe); fprintf('Finished! Denoised audio saved as "Recording_processed.m4a".\n');

Output

Original Audio in time domain

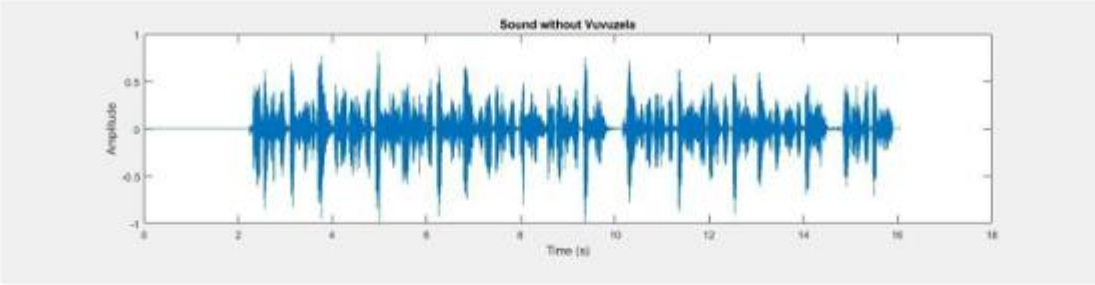


Original Audio's Spectogram

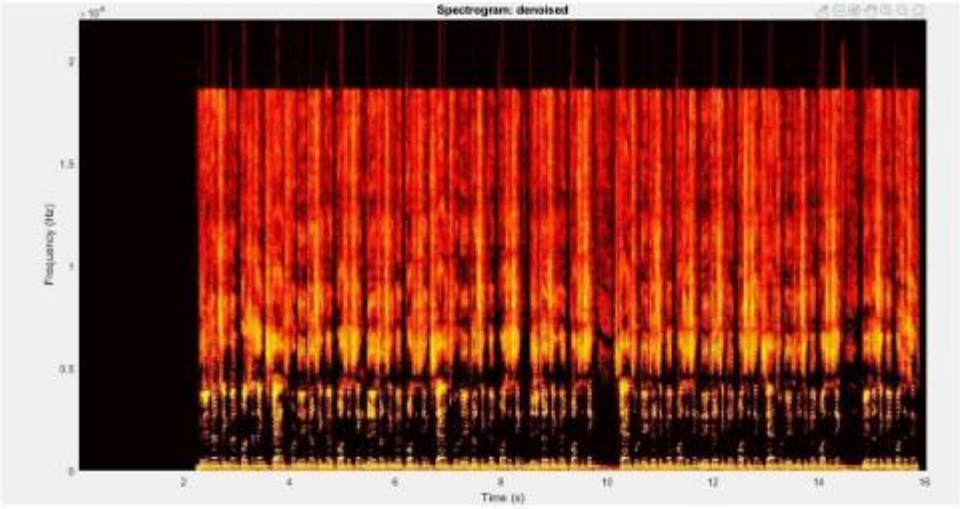


8

Denoised Audio in time domain



Denoised Audio Spectogram



Conclusion

The STFT-based method effectively removed noise while preserving the integrity of the original signal. Future work can explore adaptive parameters and neural network approaches for further improvement.

References

"Short-Time Fourier Transform and Its Applications in Signal Processing." *ScienceDirect*. Available at: https://www.sciencedirect.com.

"Single-Channel Noise Reduction in the STFT Domain Based on the Bifrequency Spectrum." *IEEE Xplore*. Available at: https://ieeexplore.ieee.org/document/6287826.

"A Speech Distortion Weighted Single-Channel Wiener Filter Based STFT-Domain Noise Reduction." *IEEE Xplore*. Available at: https://ieeexplore.ieee.org/document/10208040.