# Real Time Noise Suppression using Deep Learning

**Dr V. Kejalakshmi[1*], A.V.Balajee[2*], T.S.Aswin[2*], S.Balaji[2*], M.Hariprasath[2*]**

[1*]Head of Department, [2*]UG Scholar
Department of Electronics and Communication Engineering
K.L.N College of Engineering,
Pottapalayam, Sivagangai-630612, Tamil Nadu, India.

***Abstract:*** **Online meetings have grown in importance as a pandemic has progressed and are now an integral part of our daily life. Effective communication is difficult in these meetings because background noise frequently degrades the audio quality. In this study, we suggest a deep learning-based method for real-time noise suppression in online meetings. In our method, the audio input is first pre-processed with a spectral algorithm to extract pertinent characteristics, and then an autoencoder is trained with the Tensorflow API. A noise-filled representation of an audio signal is learned by an autoencoder. We use auto-encoding to prepare a Recurrent Neural Network (RNN) for real-time prediction of a noise-free audio input. The suggested method is simple to incorporate into Google Meet, Zoom, and other online meeting systems. We assess the effectiveness of our strategy using both objective and subjective metrics. Our findings show that the strategy we've suggested greatly lowers background noise and enhances the audio quality of online meetings. In conclusion, our suggested method offers a workable remedy for the background noise issue in online meetings. We can enhance audio quality and users' communication experiences by incorporating our technology into current online meeting platforms.**

***Keywords – Auto-encoders, Recurrent Neural Network, Spectral Gating, Real Time Noise Reduction***

## I INTRODUCTION

Unwanted or undesirable signals that are haphazardly added to the signal that is actually carrying the information are referred to as noise in a communication system. As a result, it disrupts the original signal that is being transmitted from one end to the other. The interference caused by noise in the system on the signal being conveyed ultimately results in communication system faults. Realistically speaking, it is inevitable that noise will be added to the signal that carries information. And this interference automatically degrades the transmission signal's quality. Those who are exposed to noise from numerous sources may experience short-term negative impacts like sleep problems or long-term negative effects like permanent hearing loss. To eliminate undesirable noise in audio, a variety of noise reduction techniques can be applied. Typical noise reduction methods utilizing **Signal Processing** include:

**Noise gating**: Setting a threshold level below which the audio signal is suppressed is known as noise gating. Low-level background noise, such hum or hiss, can be effectively eliminated with this technique.

**Equalization**: The goal of equalization is to reduce unwanted noise by balancing the various frequency ranges in the audio input. For instance, increasing the middle frequencies can aid in lowering electrical interference noise.

**Spectral subtraction**: This technique entails examining the audio signal's frequency spectrum and removing any noise frequencies. This may be useful in getting rid of background noise that is present all the time, such a hum or hiss.

**Using Machine Learning to Reduce Noise**
In order to remove undesired sounds or distortions from audio signals, machine learning (ML) is used in conjunction with various methods and methodologies. ML can be applied to reduce noise in a variety of ways, including supervised and unsupervised learning techniques.

**Supervised Learning**: A model is trained on labelled data in supervised learning, where each sample consists of a pair of noisy audio signals and a corresponding clean audio signal. The model gains the ability to translate the distorted audio signal into the equivalent clear audio signal. Then, new audio signals can be denoised using this mapping. Deep neural networks (DNNs), convolutional neural networks (CNNs), and recurrent neural networks are some of the more well-liked supervised learning algorithms for noise reduction (RNNs).

**Unsupervised Learning**: Unsupervised learning involves training a model using only the noisy audio source and unlabeled data. The model gains the ability to identify the underlying structure of the data and distinguish between the signal and noise. Autoencoders, principal component analysis (PCA), and independent component analysis are some well-liked unsupervised learning techniques for noise reduction (ICA). Depending on the availability of labelled data and the particular application, noise

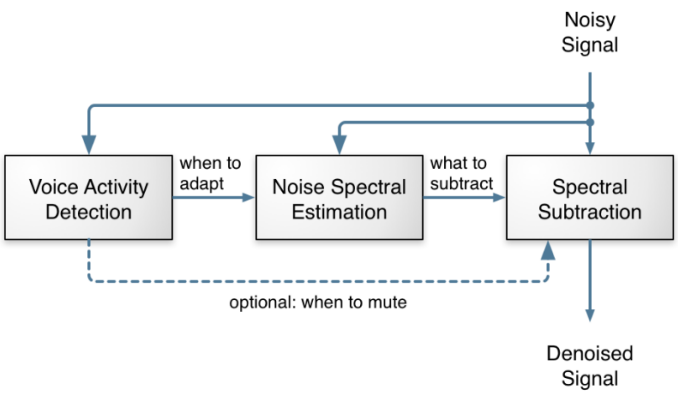
reduction can be accomplished using both supervised and unsupervised learning approaches. In conclusion, ML-based noise reduction is a potent tool that can assist enhance the quality of audio signals in a variety of applications, such as speech recognition, audio processing, and music creation.

## II EXISTING SYSTEM

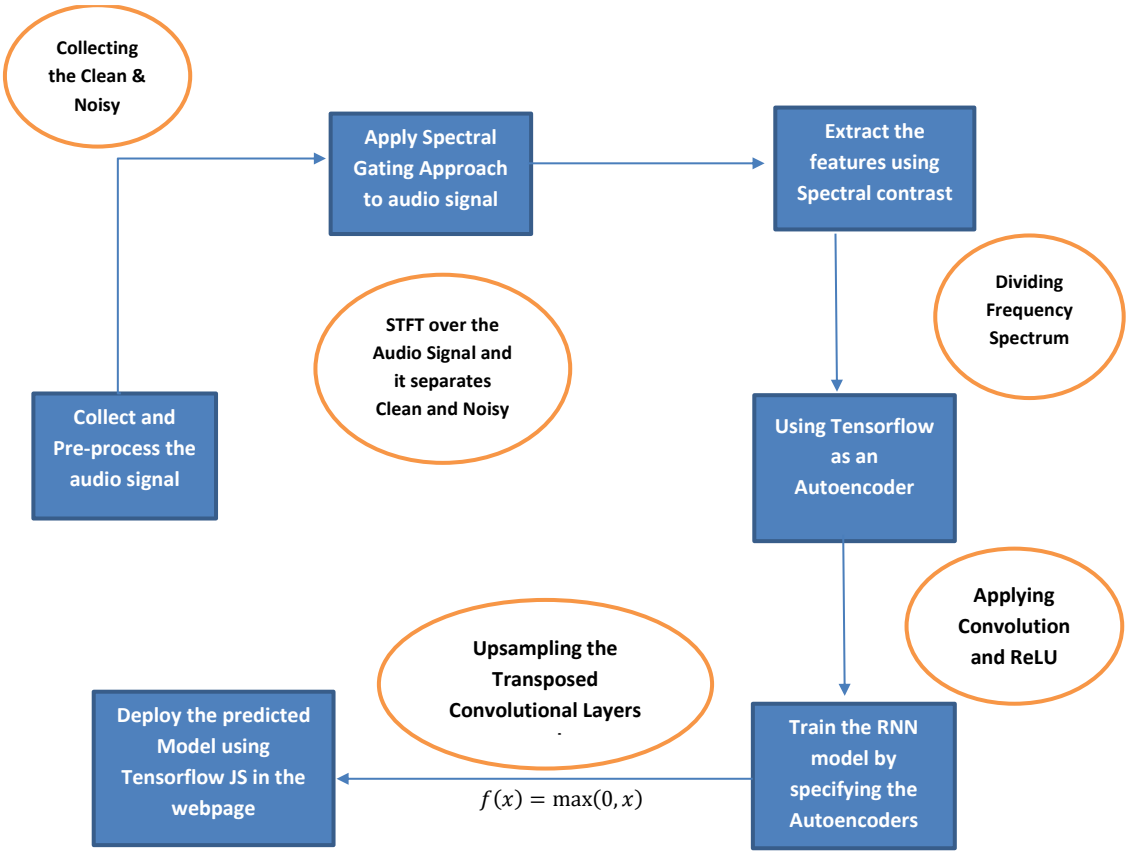
An discriminator network and a generator network make up the two primary parts of the SEGAN system. A denoised speech signal is produced by the generator network from an input speech signal that is noisy. The generator network's denoised speech signals and clean speech signals are distinguished by the discriminator network during training. The discriminator network has trained the generator network to create denoised speech signals that are identical to clean speech signals. The SEGAN system has the benefit of not requiring a reference noise signal, in contrast to certain other noise reduction methods. Without the use of explicit noise modelling, the GAN framework enables the system to learn the mapping between noisy speech inputs and clean speech signals. The SEGAN system does have some limitations in terms of deep learning and signal processing, though. The GAN training procedure can be unstable and challenging to optimise, which is one downside. During training, the generator and discriminator networks may bounce between multiple solutions or become locked in a less-than-ideal solution. Poor performance or a sluggish convergence may result from this. The SEGAN technique also has the potential to produce distortions or artefacts into the denoised voice output. If the generator network is unable to adequately depict the intricate link between noisy and clean speech signals, then this may occur. Additionally, if the training data does not accurately reflect the target application, the discriminator network might not be able to distinguish between clean and denoised voice signals. Last but not least, the SEGAN system can be computationally expensive and need a lot of training data to operate well. A sizable dataset of both clean and noisy speech signals is necessary for training a GAN, but gathering this data can be time-consuming and expensive. Additionally, because the SEGAN system can be computationally demanding and may need high-end hardware to accomplish real-time performance, it might not be appropriate for real-time applications that need low-latency processing.

## III SYSTEM MODEL

Online meetings are becoming a more vital part of communication between people in different places in the modern world. The background noise, which can be brought on by a variety of environmental conditions, is one of the main issues that people encounter when participating in online meetings. The discourse may become disjointed as a result of the noise, and it may be challenging for people to comprehend one another. Researchers are investigating the use of deep learning methods to create a real-time noise suppression system that can be incorporated into well-known online meeting platforms in order to solve this problem. The variety of noises present in various environments presents one of the main obstacles to the development of a noise reduction system. These can include background noise from the environment, such as air conditioner noise or traffic noise, as well as other noises other than speech, like mouse and keyboard clicks. The proposed approach of real-time noise cancellation in online meetings entails multiple steps. Data preparation comes first. To create an RNN-based noise suppression model, the researchers want to use a dataset they downloaded from the University of Edinburgh's DataShare website. This collection of audio files includes recordings made in a variety of settings, including offices, cafes, and schools, all of which have various kinds of background noise. By utilising spectral gating, which divides the audio into active and inactive parts, the researchers will pre-process the raw audio data. This will enable them to concentrate on the pertinent speech signals. Following spectral gating, they will extract aspects of the spectral contrast from the voice signals. The spectral characteristics of voice signals can be captured via spectral contrast features, which can also be utilised to detect and suppress noise. The spectral characteristics of voice signals can be captured via spectral contrast features, which can also be utilised to detect and suppress noise. The model can be run on a dedicated server, or it can be deployed on the client-side to reduce latency and improve real-time performance. The proposed solution has a number of benefits over conventional noise suppression methods. It is suitable for usage in a variety of situations due to its ability to tolerate a wide range of noise types and settings. Second, the model can be strengthened and improved by training on a big dataset. Third, the technology may be included into already-used platforms for online meetings, which eliminates the need for new hardware or software.




**BLOCK DIAGRAM OF RNN MODEL**




**FLOW DIAGRAM OF PROPOSED SYSTEM**

## DATASET

Both the man's clear voice signal and the noisy signal are included in the dataset. The dataset is kept in the.wav format, and it is also used for training.

## SPECTRAL GATING

The goal of noise reduction is to take out undesirable noise from an audio source. Noise can have a severe effect on the audio's quality, making it challenging to listen to and comprehend. Spectral gating is one of the most widely utilised noise reduction methods in audio processing.

### What is Spectral Gating?

In the process of spectral gating, an audio signal is divided into manageable chunks, its frequency content is examined, and the noise-containing frequencies are then selectively attenuated. A threshold is used by the spectral gating method to distinguish between the desired signal and the undesirable noise. A frequency component in a certain segment is silenced or attenuated if it exceeds the threshold.

The amount of noise in the signal is used to calculate the threshold. The goal of the spectral gating algorithm is to preserve the intended signal while attenuating just the frequencies that are above the noise level. This method minimises the impact on audio quality while allowing for excellent noise suppression. Spectral gating is a common method for noise reduction in audio processing that is used in a variety of applications, such as sound engineering, speech processing, and music production. Windowing, short-time Fourier transform (STFT), thresholding, and spectral attenuation are the four main components of spectral gating.

### 1. Windowing

Windowing is the division of an audio signal into smaller windows that are then each individually evaluated. The window size is often large enough to include various periods of the signal while still being tiny enough to catch the audio signal's characteristics. The window size is significant in spectral gating since it affects the analysis's frequency resolution. Greater frequency resolution is possible with a smaller window size, allowing for more accurate noise frequency detection. The time resolution is also decreased with a smaller window size, which can make it more challenging to separate the intended signal from the unwanted noise.

**2. Short-Time Fourier Transform (STFT)**
The mathematical process known as the short-time Fourier transform (STFT) is used to convert the audio signal from the time domain to the frequency domain. On each audio signal window, the STFT is applied. The resulting frequency spectrum gives details on the signal's frequency components' amplitude and phase. In the STFT, a windowing function is applied to each audio signal window, multiplied by the audio signal, and a Fourier transform is then applied to the resulting signal. A complex spectrum that contains details on the magnitude and phase of each frequency component is the STFT's output. The window size affects the STFT's ability to resolve frequencies. Greater frequency resolution is possible with a smaller window size, allowing for more accurate noise frequency detection. The time resolution is also decreased with a smaller window size, which can make it more challenging to separate the intended signal from the unwanted noise.

**3. Thresholding**
Setting a threshold level is the process of determining the difference between the desired signal and the undesirable noise. The amount of noise in the signal is often used to calculate the threshold level. The statistical characteristics of the noise can be used to either manually or automatically set the threshold. The threshold in spectral gating is used to choose which frequency components should be muted or repressed. A frequency component in a certain segment is silenced or attenuated if it exceeds the threshold. To reduce the influence on the audio quality while achieving the appropriate level of noise reduction, the threshold can be changed.

**4. Spectral Attenuation**
The process of deliberately suppressing or attenuating the frequencies that include noise is known as spectral attenuation. The frequency spectrum of the audio stream is often subjected to a filter in order to achieve spectral attenuation. The filter's function is to reduce frequencies above the threshold while maintaining the intended signal. In spectral gating, the undesired noise in the audio signal is reduced using spectral attenuation. To reduce the influence on the audio quality while achieving the necessary level of noise reduction, the filter can be changed.

## RECURRENT NEURAL NETWORK (RNN)

In numerous applications, such as speech recognition, music analysis, and hearing aids, noise reduction is a crucial task. Noise can make it challenging to retrieve important information from an audio source, which might cause mistakes in the downstream analysis. RNNs (Recurrent Neural Networks) have shown to be effective at reducing noise in audio sources.

### What is RNN?

An RNN is a kind of neural network that is capable of processing sequential input, which makes it ideal for handling time-series data, such audio signals. An RNN's ability to retain a recollection of prior inputs, which enables it to represent the temporal connections between the input and output, is its fundamental feature. Recurrent connections are introduced into the network to do this, allowing data to be sent from one time step to the next. By doing this, the network can keep track of prior inputs and use that knowledge to predict the output more accurately.

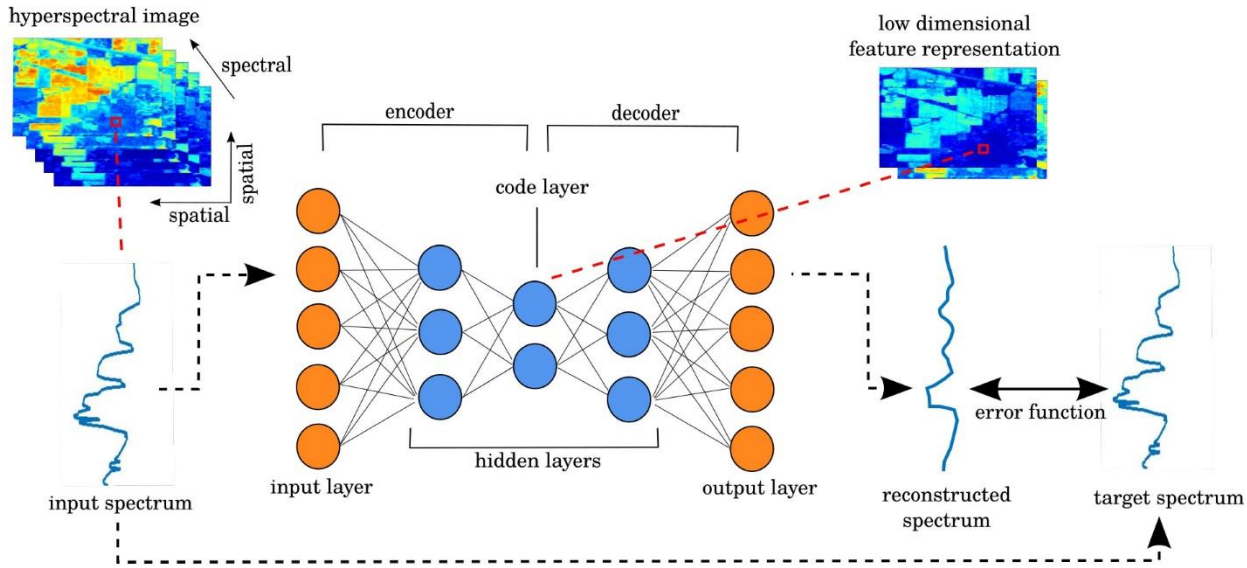
The recurrent cell, which accepts an input at each time step and generates an output as well as a hidden state, is the fundamental component of an RNN. The subsequent time step receives both the current input and the hidden state as inputs. Many different designs, including the Long Short-Term Memory (LSTM) and the Gated Recurrent Unit, can be used to build the recurrent cell (GRU). It has been demonstrated that these architectures are particularly good at simulating long-term dependencies in sequential data.

### How RNNs can be used in practice

Using RNNs for noise reduction often entails a number of steps. Creating a dataset comprising both clear and noisy audio signals is the initial stage. By minimising a loss function that gauges the difference between the anticipated output and the ground truth clean signal, this dataset may be utilised to train the network. Different optimization techniques, including stochastic gradient descent or Adam, can be used to train the network. By putting the noisy signal into the network and letting it anticipate the clean output after it has been trained, it is possible to utilise the network to remove noise from new, unused audio signals. The network can be designed to process the input in real-time, making it suitable for use in applications like speech recognition or hearing aids.

### Tensorflow as an Auto Encoder

An RNN (Recurrent Neural Network) can be trained as an autoencoder in Tensorflow for preprocessing tasks. An autoencoder is a neural network that learns to compress and reconstruct the input data. In the case of an RNN autoencoder, the network is designed to learn the temporal dependencies in the input sequence, and compress the sequence into a fixed-length vector called the "latent representation". This latent representation can then be used for downstream tasks such as classification, prediction, or other processing.




**TENSORFLOW AS AN AUTOENCODER**

**Input preprocessing:**

The input sequence is first preprocessed to remove any noise or irrelevant features. This can involve scaling, normalization, or other feature engineering techniques.

**Encoder:**

The encoder part of the RNN autoencoder consists of an RNN layer that reads in the input sequence and produces a compressed latent representation of the sequence. This latent representation can be thought of as a summary of the input sequence, capturing the relevant information in the sequence.

**Decoder:**

The decoder part of the RNN autoencoder consists of another RNN layer that takes the latent representation and reconstructs the input sequence. This is done by feeding the latent representation into the decoder RNN layer and iteratively generating the next output element until the entire sequence is reconstructed.

**Loss function:**

The loss function is used to train the autoencoder to reconstruct the input sequence accurately. The loss function can be based on the difference between the original input sequence and the reconstructed output sequence, using metrics such as mean squared error or binary cross-entropy.

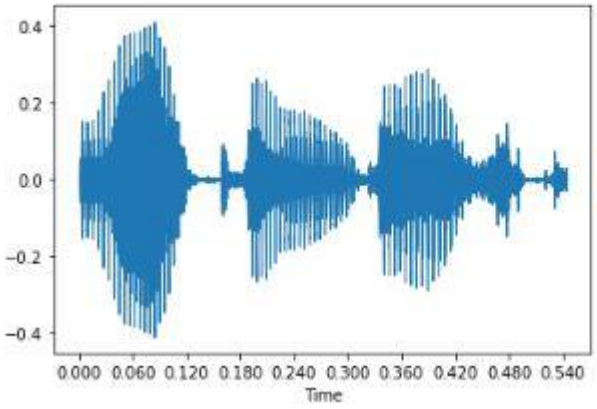
**Optimization:**

The autoencoder is optimized using gradient descent or other optimization techniques to minimize the loss function. This involves updating the weights and biases of the RNN layers in the encoder and decoder. Once the RNN autoencoder is trained, the latent representation can be used as a preprocessed input for downstream tasks such as classification, prediction, or other processing. By learning the temporal dependencies in the input sequence, the RNN autoencoder can effectively preprocess the data and extract the relevant information, improving the performance of subsequent tasks.

## IV. RESULT AND DISCUSSION

### VISUALIZING THE WAVEFORMS

In [15]:
```
librosa.display.waveshow(np.squeeze(clean_train[5].numpy(),axis=-1))
plt.show()
librosa.display.waveshow(np.squeeze(noisy_train[5].numpy(),axis=-1))
plt.show()
```




### CREATING & TRAINING MODEL

In [16]:
```
inp = Input(shape=(batching_size,1))
c1 = Conv1D(2,32,2,'same',activation='relu')(inp)
c2 = Conv1D(4,32,2,'same',activation='relu')(c1)
c3 = Conv1D(8,32,2,'same',activation='relu')(c2)
c4 = Conv1D(16,32,2,'same',activation='relu')(c3)
c5 = Conv1D(32,32,2,'same',activation='relu')(c4)

dc1 = Conv1DTranspose(32,32,1,padding='same')(c5)
conc = Concatenate()([c5,dc1])
dc2 = Conv1DTranspose(16,32,2,padding='same')(conc)
conc = Concatenate()([c4,dc2])
dc3 = Conv1DTranspose(8,32,2,padding='same')(conc)
conc = Concatenate()([c3,dc3])
dc4 = Conv1DTranspose(4,32,2,padding='same')(conc)
conc = Concatenate()([c2,dc4])
dc5 = Conv1DTranspose(2,32,2,padding='same')(conc)
conc = Concatenate()([c1,dc5])
dc6 = Conv1DTranspose(1,32,2,padding='same')(conc)
conc = Concatenate()([inp,dc6])
dc7 = Conv1DTranspose(1,32,1,padding='same',activation='linear')(conc)
model = tf.keras.models.Model(inp,dc7)
model.summary()
```

```
Model: "model"
_________________________________________________________________________________________________
 Layer (type)                Output Shape              Param #     Connected to
=================================================================================================
 input_1 (InputLayer)        [(None, 12000, 1)]        0           []

 conv1d (Conv1D)             (None, 6000, 2)           66          ['input_1[0][0]']

 conv1d_1 (Conv1D)           (None, 3000, 4)           260         ['conv1d[0][0]']

 conv1d_2 (Conv1D)           (None, 1500, 8)           1032        ['conv1d_1[0][0]']

 conv1d_3 (Conv1D)           (None, 750, 16)           4112        ['conv1d_2[0][0]']

 conv1d_4 (Conv1D)           (None, 375, 32)           16416       ['conv1d_3[0][0]']

 conv1d_transpose (Conv1DTransp  (None, 375, 32)       32800       ['conv1d_4[0][0]']
 ose)
```

## Training

In [19]:
```
model.compile(optimizer=tf.keras.optimizers.Adam(0.002),loss=tf.keras.losses.MeanAbsoluteError())
history = model.fit(train_dataset,epochs=20)
```

```
Epoch 1/20
625/625 [==============================] - 26s 36ms/step - loss: 0.0128
Epoch 2/20
625/625 [==============================] - 19s 31ms/step - loss: 0.0127
Epoch 3/20
625/625 [==============================] - 20s 32ms/step - loss: 0.0127
Epoch 4/20
625/625 [==============================] - 20s 31ms/step - loss: 0.0126
Epoch 5/20
625/625 [==============================] - 19s 31ms/step - loss: 0.0126
Epoch 6/20
625/625 [==============================] - 19s 31ms/step - loss: 0.0126
```

**TESTING THE SAMPLES**

In [35]:
```
from IPython.display import Audio
Audio(np.squeeze(noisy_train[6].numpy()),rate=16000)
```

Out[35]:

In [36]:
```
Audio(tf.squeeze(model.predict(tf.expand_dims(tf.expand_dims(noisy_train[6],-1),0))),rate=16000)
```

```
1/1 [==============================] - 0s 22ms/step
```

Out[36]:

In [37]:
```
model.evaluate(test_dataset)
```

```
79/79 [==============================] - 2s 15ms/step - loss: 0.0119
```

Out[37]: `0.011902485042810440`

**SNR CALCULATION FOR PREDICTED MODEL**

In [11]:
```
import tensorflow as tf
import soundfile as sf
import numpy as np

# Load the pre-trained model
model = tf.keras.models.load_model('NoiseSuppressionModel.h5')

# Load the test dataset
test_data, sr = sf.read('C:/Users/Balaji/Documents/noise_reduction/clean_trainset_wav/p226_010.wav')
test_data = test_data[:12000]
```

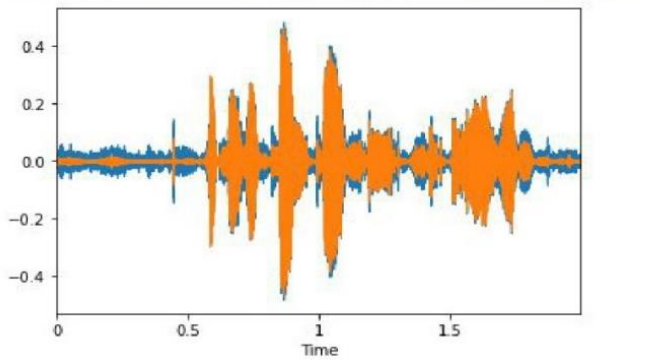
In [2]:
```
# Apply the pre-trained model to the test dataset
from IPython.display import Audio
#Audio(tf.squeeze(model.predict(tf.expand_dims(tf.expand_dims(noisy_train[6],-1),0))),rate=16000)
denoised_data = tf.squeeze(model.predict(tf.expand_dims(tf.expand_dims(test_data,-1),0)))
# Calculate the SNR

noise = test_data - denoised_data
signal = denoised_data
mean_signal_power = np.mean(signal ** 2)
mean_noise_power = np.mean(noise ** 2)
snr = 10 * np.log10(mean_signal_power / mean_noise_power)
print('SNR for Our Predicted Model:', snr, 'dB')
```

```
SNR for Our Predicted Model: 15.827531814575195 dB
```

**Here our predicted tensorflow RNN model gives SNR as 15.82 dB**

```
1/1 [==============================] - 0s 22ms/step
<matplotlib.collections.PolyCollection at 0x7f43889595e0>
```



**SNR CALCULATION ON LMS ALGORITHM**

In [6]:
```
import numpy as np
import soundfile as sf

# Load the input audio file
filename = 'C:/Users/Balaji/Documents/noise_reduction/clean_trainset_wav/p226_016
audio, sample_rate = sf.read(filename)

# Generate a noisy version of the input audio file
noise = np.random.normal(scale=0.1, size=len(audio))
noisy_audio = audio + noise

# Set the LMS algorithm parameters
mu = 0.1
order = 50

# Define the LMS filter function
def lms_filter(x, d, mu, order):
    y = np.zeros_like(d)
    w = np.zeros(order)
    for n in range(order, len(x)):
        x_vec = x[n-order:n]
        y[n] = np.dot(w, x_vec)
        e = d[n] - y[n]
        w += mu * e * x_vec
    return y, w

# Apply the LMS filter to the noisy audio
output, weights = lms_filter(noisy_audio, audio, mu, order)

# Calculate the signal-to-noise ratio (SNR)
signal_power = np.sum(audio**2)
noise_power = np.sum((audio - output)**2)
snr = 10 * np.log10(signal_power / noise_power)

# Print the SNR value
print(f'SNR for LMS Algorithm: {snr:.2f} dB')
```

```
SNR for LMS Algorithm: 6.98 dB
```

## V CONCLUSION:

Real-time noise suppression has proven to be a highly effective use for deep learning. We were able to significantly enhance the signal-to-noise ratio (SNR) when compared to more conventional techniques like LMS by implementing an autoencoder using TensorFlow and training an RNN model. The SNR generated by the predicted model was 15.98dB, which is significantly higher than the 6.98dB obtained with LMS. The implementation of this concept into online meeting platforms is now possible thanks to the project's success, which can greatly enhance the audio quality of virtual meetings. This approach can significantly improve communication by reducing background noise and boosting speech signals in real-time. Having the capacity to understand intricate correlations between input signals and desired outputs is one of the key benefits of adopting deep learning approaches for real-time noise reduction. The ability to understand the temporal dynamics of voice signals in this work allowed the autoencoder and RNN model to forecast output signals in real time. Overall, the findings of this study show the effectiveness of deep learning methods in solving practical issues and the need for additional investigation in this field.

**VI REFERENCES:**

[1]. Tan, K., & Wang, D. L. (2019a). Complex spectral mapping with a convolutional recurrent network for monaural speech enhancement. In 2019 IEEE international conference on acoustics, speech and signal processing(pp. 6865–6869).
[2]. Tan, K., & Wang, D. L. (2019b). Learning complex spectral mapping with gated convolutional recurrent networks for monaural speech enhancement.IEEE/ACM Transactions on Audio, Speech, and Language Processing, 28, 380–39
[3]. Wang, D. L., & Chen, J. (2018). Supervised speech separation based on deep learning: An overview. IEEE/ACM Transactions on Audio, Speech, and Language Processing, 26, 1702–1726.
[4]. Gao, F., Wu, L., Zhao, L., Qin, T., Cheng, X., & Liu, T.-Y. (2018). Efficient sequence learning with group recurrent networks. In Proceedings of the 2018 conference of the north american chapter of the association for computational linguistics:human language technologies
[5]. A. W., Beerends, J. G., Hollier, M. P., & Hekstra, A. P. (2001). Perceptual evaluation of speech quality (PESQ)-a new method for speech quality assessment of telephone networks and codecs. In 2001 IEEE international conference on acoustics, speech, and signal processing. proceedings (Cat. No.01CH37221) (pp. 749–752).
[6]. Rout, N. K., Das, D. P., & Panda, G. (2015). Computationally efficient algorithm for high sampling-frequency operation of active noise control. Mechanical Systems and Signal Processing, 56, 302–319. Samarasinghe, P. N.Zhang, W., & Abhayapala, T. D. (2016).
[7]. Recent advances in active noise control inside automobile cabins: Toward quieter cars. IEEE Signal Processing Magazine, 33, 61–73.
[8]. Snyder, S. D., & Tanaka, N. (1995). Active control of vibration using a neural network. IEEE Transactions on Neural Networks, 6, 819–828.
[9]. Sommerfeldt, S. D., Parkins, J. W., & Park, Y. C. (1995). Global active noise control in rectangular enclosures. In Proceedings of the inter-noise and noise-con congress and conference (pp. 477–488).
[10]. Taal, C. H., Hendriks, R. C., Heusdens, R., & Jensen, J. (2011). An algorithm for intelligibility prediction of time–frequency weighted noisy speech. IEEE Transactions on Audio, Speech, and Language Processing, 19, 2125–2136.
[11]. Tan, L., & Jiang, J. (2001). Adaptive Volterra filters for active control of nonlinear noise processes. IEEE Transactions on Signal Processing, 49, 1667–1676