Report

10955005 張可晴　　109550139 楊竺耘

Video Link:
https://drive.google.com/drive/folders/1Na7mOf5jwlxn8eDAGt8HIGWmLOGGirgG?usp=sharing

1. Background Overview of the Selected Topic

The topic we have chosen is "Classification of speech signals." This technique refers to speech recognition by analyzing and extracting features from them. The analysis of speech signals has significant applications in areas such as speech processing and speech recognition.

In traditional approaches, statistical models are widely used for classifying new signals. Examples of such models include Hidden Markov Models (HMM) and Gaussian Mixture Models (GMM). These models are used to statistically analyze speech signals, construct models based on the statistical data, and employ techniques like maximum likelihood estimation and Bayesian inference to implement classifiers based on the aforementioned statistical models.

However, in recent years, there have been significant breakthroughs in the field of constructing speech signal classifiers with advancements in machine learning and deep learning techniques. Nowadays, machine learning-based methods are predominantly used for model construction, particularly focusing on Convolutional Neural Networks (CNN) and Recurrent Neural Networks (RNN). These two types of networks greatly enhance the accuracy and performance of speech signal classifiers. Furthermore, models based on CNN and RNN can be adaptive, meaning they can automatically learn and extract useful features directly from raw speech signals and perform classification based on these features.

The applications of speech signals are widely used. In speech recognition, speech signal classification techniques are used to identify specific words from speech signals. For speech emotion recognition, these techniques are employed to differentiate between different types of emotions. Moreover, this technology can be applied in the medical field to create voice-based disease diagnosis, effectively identifying disease characteristics present in a speech segment and consolidating potentially related diseases.

In conclusion, the classification of speech signals holds a significant position and has extensive applications in the field of speech processing and speech recognition. From traditional statistical models to modern machine learning methods, continuously progressing technologies enable us to classify speech signals more accurately.

2. Description of the Selected Papers

2.1 A Speech Classification System

Speech classification indicates the technology of automatically categorizing audio signals. This paper [1] focuses on statistical-based speech classification. Such methods are beneficial for training classifiers to handle new tasks or special situations, so collecting appropriate training data is essential. Therefore, the author lists the speech classifiers used in different

circumstances, such as central storage and automatic analysis, prioritizing messages, and searching for persons in text and speech messages.

In addition to discussing the use of speech recognition in various scenarios, this paper also introduces different types of speech classification systems, such as SCOOTY (Speech Classification Online and Offline Technology) and MELANIE (Medav Language Intelligence Environment).

2.2 Classification Techniques for Automatic Speech Recognition (ASR) Algorithms used with Real Time Speech Translation

Speech recognition and speech translation systems can be mainly composed of two different types of systems: Automatic Speech Recognition (ASR) systems and Machine Translation (MT) systems. The former involves feature extraction followed by classification to finish speech recognition. Different techniques are used for classification, such as Dynamic Time Wrapping (DTW), Hidden Markov Model (HMM), and Dynamic Bayesian Network. On the other hand, MT systems can be categorized into statistical-based methods, rule-based methods, or a combination of both.

Based on the above introduction, we can understand that both ASR and MT systems offer various options for speech recognition and translation. Therefore, finding the best feature extraction and classification methods, combined with a suitable MT system, is a crucial and challenging task. The purpose of this paper [2] is to identify the most suitable ASR classification algorithm for an MT system primarily used for real-time translation from English to Arabic. The author collected 110 sentences as training and testing data and explored a total of 9 different combinations of environments and technologies, calculating their accuracy to determine the optimal one.

2.3 Speech Recognition with Deep Recurrent Neural Networks

Recurrent neural networks (RNNs) are highly useful network models for sequential data, such as handwriting data, acoustic signals, and so on. Before this paper [3] came out, there had already been some studies and experiments showing that applying RNN to handwriting recognition can get state-of-the-art outcomes, while the results from employing RNN models to speech recognition were not good.

In this paper, the authors conduct research about utilizing deep, bidirectional Long Short-term Memory RNNs and training with Connectionist Temporal Classification (CTC). In the end, they get a 17.7% error rate on the TIMIT dataset.

2.4 A Hybrid SFANC-FxNLMS Algorithm for Active Noise Control Based on Deep Learning

The reason for choosing this paper [4] is because the previous three papers place emphasis on the theory related to speech classification, this article pays attention to an active noise control (ANC) model that incorporates the practical application of a Convolutional Neural Network (CNN) for speech signal classification.

Active noise control (ANC) is currently a significant development, whether it's the widely discussed active noise-canceling headphones or large-scale ANC devices used in industrial settings to protect machine operators. ANC works by detecting low-frequency environmental noise that can be heard by the human ear and using real-time computations to generate sound waves with opposite phases and equal amplitude to cancel out the noise.

Filtered-X normalized least-mean-square (FxNLMS) is an adaptive ANC algorithm that achieves low noise reduction error through adaptive optimization without requiring prior knowledge of the input data. However, due to the inherent slow convergence and poor tracking ability of least mean square (LMS), FxNLMS performs poorly in dealing with rapidly changing or dynamic noise. Its slow response to such noise may affect the user experience of noise reduction.

In current industrial applications and mature active noise control products, fixed filter-based ANC methods are predominantly used. In these methods, the coefficients of the control filter are fixed and do not undergo adaptive updates. Although this approach resolves the slow convergence and instability issues caused by the adaptive process, the performance of noise reduction can be severely compromised if a fixed filter trained for a specific noise type is used for another type of noise. To address this problem, this paper attempts to use Segment Frequency Adaptive Noise Control (SFANC), which selects filter parameters that achieve the optimal response time for different noise types. However, if this method is used without any structural modifications, it requires extensive judgment and error experiments to determine the critical parameters for filter selection, resulting in additional computational burden.

Hence, automatic learning of the key parameters for SFANC, which is based on segment frequency matching selective fixed filters, is crucial to promote the method's usage in real-world scenarios and industrial products. Considering this combination, deep learning techniques, particularly Convolutional Neural Networks (CNN), have shown excellent results in classification tasks. Therefore, we believe it is a promising approach to improving selective filtering. By replacing the filter selection module with a CNN model, the algorithm for selective filtering can automatically learn its parameters from a dataset of noise signals and select the optimal control filters for different noise types, eliminating the need for manual determination and adjustment of filter parameters based on noise type. Additionally, the CNN model implemented on a co-processor can separate the computational load from the real-time noise controller. However, while combining the segment frequency matching selective fixed filters of SFANC with deep learning approaches, particularly CNN, offers advantages in terms of response time, it still faces limitations such as noise misclassification and lack of adaptive optimization, leading to significant steady-state errors.

As mentioned above, it is challenging to achieve satisfactory noise reduction performance for all types of noise using either SFANC or FxNLMS alone. However, combining the SFANC and FxNLMS algorithms may effectively overcome the limitations of individual ANC algorithms.

Therefore, this paper proposes a hybrid SFANC-FxNLMS algorithm. In this proposed algorithm, a lightweight one-dimensional CNN (1D CNN) speech signal classifier implemented on a co-processor automatically selects filters.

3. Demo / Experiments, and Our Findings

In this section, we will talk about our implementation, experiments, and some findings on the third and fourth papers in detail.

3.1 The Third Paper

To implement the recurrent neural network on speech recognition, I refer to the framework [5] provided by the course from Udacity. We eventually trained 6 different network models for speech recognition, and then evaluated and compared their performance. Besides, the

training and testing dataset both came from the LibriSpeech dataset. For the purpose of building all the network models, we took advantage of the Keras library.

For the first model (model_0), we trained a pure RNN model to do speech recognition. The input is the audio features based on the time sequence and the RNN acoustic model will output the probability among 28 possible characters, 26 English letters, the space character, and an apostrophe, which suggests the probability that the speaker pronounces that character at that time. To be more specific, the type of RNN model we used in this part is GRU.

In terms of the second model (model_1), it is the LSTM model with a TimeDistributed layer for observing the more complex patterns and a BatchNormalization layer to speed up the training rate.

As for the third model (model_2), it has an additional 1D convolution layer added before the second model to analyze the input feature and send only the important ones to the RNN network models to do classification.

For the next one (model_3), it is the deep RNN model, which is just the RNN model, while there are 2 layers in total. Additionally, we use LSTM in this part

In the case of the fifth model (model_4), it is a bidirectional RNN network so that we can make use of both the previous information and the future one. Moreover, we also utilize LSTM for this model.

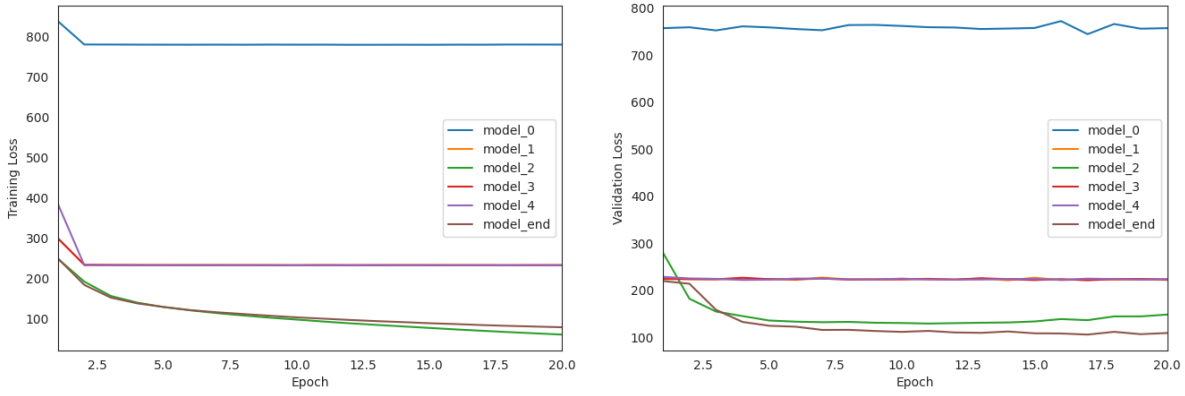

Figure 1 The training loss and validation loss of 6 different models

From Figure 1, we can observe that the pure RNN model (model_0) performs worst regardless of the training or the validating stage. It comes as no surprise since this model is too simple to handle speech recognition. Also, it is evident that the RNN with a fully connected layer, the deep RNN, and bidirectional models, model_1, model_3, and model_4, respectively, have evenly matched strength, while all of them do perform not very well. They present a little improvement over the pure one. The best one is the RNN model applying the 1D convolutional layer (model_2). Its training loss finally achieves roughly 89 and validation loss turns out to be approximately 130 in the end.

According to the above observation, we then design the last model (model_end) based on model_2. I also utilize a 1D convolutional layer at the beginning, while the RNN model is LSTM as mentioned in the paper and I make it deeper that there are 2 RNN layers. Moreover, I additionally add dropout to the RNN model with the dropout rate being 0.5 to prevent overfitting. Last but not least, we can easily find that this network model has the lowest validation loss eventually.

3.2 The Forth Paper

We refer to the previously mentioned fourth paper, "A Hybrid SFANC-FxNLMS Algorithm for Active Noise Control Based on Deep Learning," to build a complete noise reduction model that incorporates a one-dimensional CNN-based Classification of speech signals.

3.2.1 Overview of the Complete Model Architecture

During the pre-training stage, the input audio is processed by a pre-trained CNN speech signal classifier to generate id_vector, which represents the filter index to be used at each time frame.

The input audio is passed through the estimated secondary path to address the instability issue caused by the FxNLMS algorithm, which will cause oscillations and then lead to unstable active noise control.

The FxNLMS algorithm utilizes the currently selected filter from id_vector to generate a model. This model takes the input signal and the desired output (error) and loss as feedback. The loss is used to adjust the parameters of the filter, and the updated filter is used for noise reduction in the subsequent frames.

In the next frame, the selected filter from id_vector is applied as the current filter, and the process is repeated to continuously adjust the model based on the dynamically changing filter.

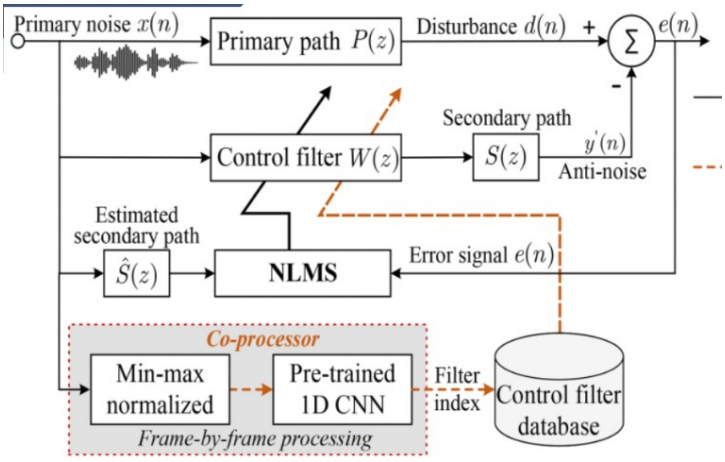

Figure 2 Complete model architecture

3.2.2 One-Dimensional CNN Speech signal classifier

The purpose of this speech signal classifier is to serve as the pre-training phase for the fixed filters. It assigns different labels to different types of noise, where each label corresponds to the filter index most suitable for that noise frequency band. In our 1D CNN architecture, each residual block consists of two convolutional layers followed by a linear rectified linear unit (ReLU) activation. The residual structure is used for easy optimization, and shortcut connections are employed to add the input to the output of each residual block. In addition, a wide receptive field (RF) is used in the first convolutional layer, while narrower RFs are used in subsequent layers to leverage both global and local information.

To pre-train the one-dimensional CNN model, we randomly generated 80,000 broadband noise tracks with varying frequency bands, amplitudes, and background noise levels. Each track has a duration of 1 second and is considered as the primary noise source to cause

interference. The class label of each noise track corresponds to an indicator of the filter's ability to achieve optimal noise reduction performance against interference. We synthesized the noise and performed tests and training on the one-dimensional CNN model. Firstly, the synthesized noise dataset was divided into three subsets: noise tracks for training (80,000), validation (2,000), and testing (2,000). During training, the Adam optimization algorithm was used, and glorot initialization was applied to prevent gradients from becoming too large or undergoing sudden dramatic changes. To mitigate overfitting caused by limited training data at the beginning, the one-dimensional CNN was designed to be a lightweight network.

3.2.3 Building the Speech Signal Classifier

The implementation in the code can be divided into three main parts.

The first part of the code (refer to Figure 3) begins by defining a convolutional block. Then, a convolution layer is added along with a batch normalization layer and ReLU activation. Following that, a pooling block is incorporated to reduce the dimensionality by four. Finally, a forward pass is performed on the convolutional layer and the pooling layer. The architecture created in this part is illustrated in Figure 4.

```
class CNN(torch.nn.Module):

    def __init__(self, channels, conv_kernels, conv_strides, conv_padding, pool_padding, num_classes=15):
        assert len(conv_kernels) == len(channels) == len(conv_strides) == len(conv_padding)
        super(CNN, self).__init__()

        # create conv blocks
        self.conv_blocks = torch.nn.ModuleList()
        prev_channel = 1

        for i in range(len(channels)):
            # add stacked conv layer
            block = []
            for j, conv_channel in enumerate(channels[i]):
                block.append(torch.nn.Conv1d(in_channels=prev_channel, out_channels=conv_channel, kernel_size=conv_kernels[i], stride=conv_strides[i], padding=conv_padding[i]))
                prev_channel = conv_channel
                # add batch norm layer
                block.append(torch.nn.BatchNorm1d(prev_channel))
                # adding ReLU
                block.append(torch.nn.ReLU())
            self.conv_blocks.append(torch.nn.Sequential(*block))

        # create pooling blocks
        self.pool_blocks = torch.nn.ModuleList()
        for i in range(len(pool_padding)):
            # adding Max Pool (drops dims by a factor of 4)
            self.pool_blocks.append(torch.nn.MaxPool1d(kernel_size=4, stride=4, padding=pool_padding[i]))

        # global pooling
        self.global_pool = torch.nn.AdaptiveAvgPool1d(1)
        self.linear = torch.nn.Linear(prev_channel, num_classes)


    def forward(self, inwav):
        for i in range(len(self.conv_blocks)):
            # apply conv layer
            inwav = self.conv_blocks[i](inwav)
            # apply max_pool
            if i < len(self.pool_blocks): inwav = self.pool_blocks[i](inwav)
        # apply global pooling
        out = self.global_pool(inwav).squeeze() # [batch_size, 256, 1] [batch_size, 256]
        out = self.linear(out) #[batch_size, 15]
        return out
```

Figure 3 The first part of the code

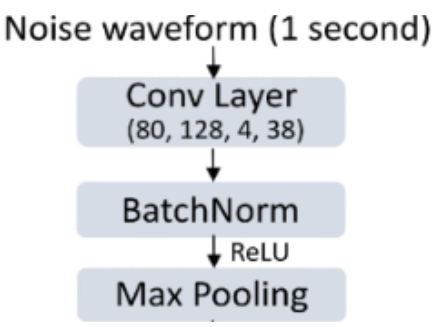

Figure 4 The first part of the architecture

The second part of the code (refer to Figure 5) is responsible for constructing a residual block as Figure 6. In addition to incorporating a convolutional layer and batch normalization layer, the main focus lies in performing repeated iterations of smaller blocks during the forward pass until the desired functionality of the larger block is achieved and then output the results.

```python
class ResBlock(torch.nn.Module):

    def __init__(self, prev_channel, channel, conv_kernel, conv_stride, conv_pad):
        super(ResBlock, self).__init__()
        self.res = torch.nn.Sequential(
            torch.nn.Conv1d(in_channels = prev_channel, out_channels=channel, kernel_size=conv_kernel, stride=conv_stride, padding=conv_pad),
            torch.nn.BatchNorm1d(channel),
            torch.nn.ReLU(),
            torch.nn.Conv1d(in_channels=channel, out_channels=channel, kernel_size=conv_kernel, stride=conv_stride, padding=conv_pad),
            torch.nn.BatchNorm1d(channel),
        )
        self.bn = torch.nn.BatchNorm1d(channel)
        self.relu = torch.nn.ReLU()

    def forward(self, x):
        identity = x
        x = self.res(x)
        if x.shape[1] == identity.shape[1]:
            x += identity
        # repeat the smaller block till it reaches the size of the bigger block
        elif x.shape[1] > identity.shape[1]:
            if x.shape[1] % identity.shape[1] == 0:
                x += identity.repeat(1, x.shape[1]//identity.shape[1], 1)
            else:
                raise RuntimeError("Dims in ResBlock needs to be divisible on the previous dims!!")
        else:
            if identity.shape[1] % x.shape[1] == 0:
                identity += x.repeat(1, identity.shape[1]//x.shape[1], 1)
            else:
                raise RuntimeError("Dims in ResBlock needs to be divisible on the previous dims!!")
            x = identity
        x = self.bn(x)
        x = self.relu(x)
        return x
```

Figure 5 The second part of the code

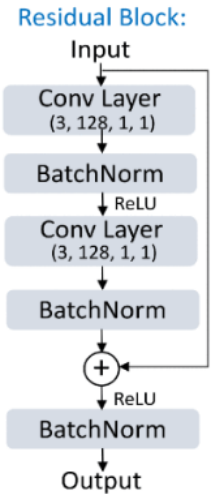

Figure 6 Res Block architecture

The third part of the code (refer to Figure 7) combines the previous two parts to construct the architecture depicted in Figure 8.

```python
class CNNRes(torch.nn.Module):

    def __init__(self, channels, conv_kernels, conv_strides, conv_padding, pool_padding, num_classes=15):
        assert len(conv_kernels) == len(channels) == len(conv_strides) == len(conv_padding)
        super(CNNRes, self).__init__()

        # create conv block
        prev_channel = 1
        self.conv_block = torch.nn.Sequential(
            torch.nn.Conv1d(in_channels=prev_channel, out_channels=channels[0][0], kernel_size=conv_kernels[0], stride=conv_strides[0], padding=conv_padding[0]),
            # add batch norm layer
            torch.nn.BatchNorm1d(channels[0][0]),
            # adding ReLU
            torch.nn.ReLU(),
            # adding max pool
            torch.nn.MaxPool1d(kernel_size = 4, stride = 4, padding = pool_padding[0]),
        )

        # create res
        prev_channel = channels[0][0]
        self.res_blocks = torch.nn.ModuleList()
        for i in range(1, len(channels)):
            # add stacked res layer
            block = []
            for j, conv_channel in enumerate(channels[i]):
                block.append(ResBlock(prev_channel, conv_channel, conv_kernels[i], conv_strides[i], conv_padding[i]))
                prev_channel = conv_channel
            self.res_blocks.append(torch.nn.Sequential(*block))

        # create pool blocks
        self.pool_blocks = torch.nn.ModuleList()
        for i in range(1, len(pool_padding)):
            # adding Max Pool (drops dims by a factor of 4)
            self.pool_blocks.append( torch.nn.MaxPool1d(kernel_size = 4, stride = 4, padding = pool_padding[i]) )

        # global pooling
        self.global_pool = torch.nn.AdaptiveAvgPool1d(1)
        self.linear = torch.nn.Linear(prev_channel, num_classes)


    def forward(self, inwav):
        inwav = self.conv_block(inwav)
        for i in range(len(self.res_blocks)):
            # apply conv layer
            inwav = self.res_blocks[i](inwav)
            # apply max_pool
            if i < len(self.pool_blocks): inwav = self.pool_blocks[i](inwav)
        # apply global pooling
        out = self.global_pool(inwav).squeeze()
        out = self.linear(out)
        return out
```

Figure 7 The third part of the code

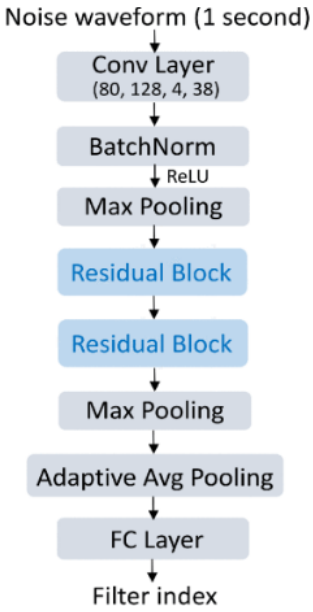

Figure 8 The third part of the architecture

3.2.4 Results

The following figure shows the line chart of train_loss and validation_loss at different time intervals during the training for 30 epochs. The learning rate was set to 0.01. It can be observed that the loss is initially unstable but becomes consistent after around 10 iterations. To elaborate, the loss values remain below 0.05, indicating the effectiveness of the speech signal classifier built using the CNN.

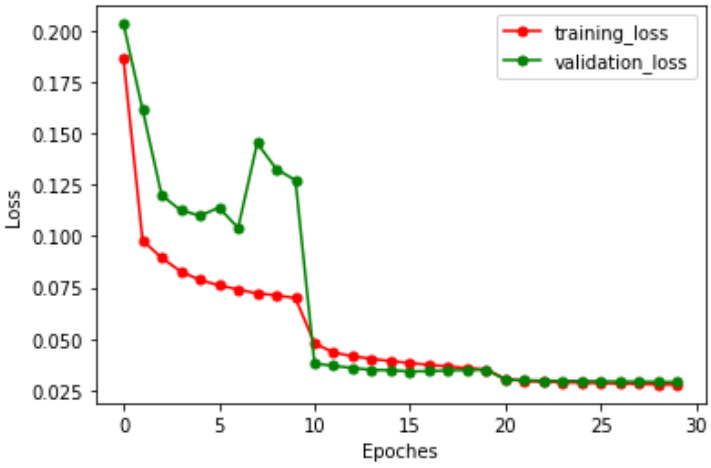

Figure 9 Line chat of train_loss and validation_loss

Figure 10 displays the performance results of the speech signal classifier on the testing set. The loss is recorded as 0.03725, and the accuracy is approximately 0.9865.

```
==========================================================
Testing model accuracy
==========================================================
Test loss: 0.037248353202085124Test accuracy: 0.9864999999999998
 The accuracy on testing datast is 0.9864999999999998
```

Figure 10 The loss and accuracy of the testing dataset

3.2.5 Analysis of the Active Noise Reduction Model

Based on the provided input audio, the first part represents airplane takeoff noise and the latter part suggests traffic noise, the denoising result is visualized in Figure 11. The blue portion represents the original audio, while the green portion represents the denoised result.

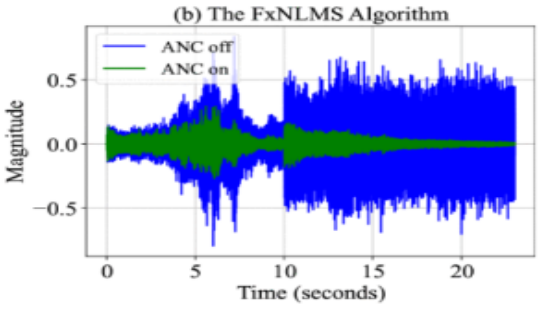

Figure 11 The result of active noise reduction

It can be observed that the noise reduction model is indeed effective. Moreover, we can focus on the period between 10 to 11 seconds, which represents a transition period between two different types of noise. Despite the significant changes in the noise frequency bands and amplitudes, noise reduction is still evident. This indicates that the denoising model remains effective even when there are substantial variations in the frequency bands and amplitudes of the noise. This effectiveness can be attributed to our earlier development of the CNN-based speech signal classifier, which accurately classifies different noise frequency bands according to specific time segments and assigns them to different filters for the denoising process.

4. DISCUSSION

From this final project, we get a deeper understanding into the classification of the speech signal. From the first paper, we can know the scenario that will use speech classification. The second one introduces us to the ASR and MT systems and the different options for us to choose. As for the third one, it shows us the application of RNN network models on speech recognition. Then, we move to learn the application of speech recognition. The final one is about the ANC models with the CNN model for classifying the noise signal. Due to these papers, I learn both the theoretical and practical parts of speech signal classification.

Apart from the theory we learn from the papers we read, we also implement the experiment related to the third and final papers. Thanks to this experience, we know how to build RNN and CNN models for speech classification.

We think that this topic is very interesting and useful for our life. It can be applied to various fields. Besides, finding out the best classification model for different scenarios is also a great challenge for this topic. As mentioned in the background, the technique of this topic grows rapidly and better. Hopefully, there will be a huge advancement in the future.

REFERENCES

[1] U. Uebler, “A Speech Classification System,” MEDAV GMBH UTTENREUTH (GERMANY), Dec. 2006. Accessed: Mar. 20, 2021. [Online]. Available: https://apps.dtic.mil/sti/citations/ADA474126

[2] H. H. O. Nasereddin and A. A. R. Omari, "Classification techniques for automatic speech recognition (ASR) algorithms used with real time speech translation," *2017 Computing Conference*, London, UK, 2017, pp. 200-207, doi: 10.1109/SAI.2017.8252104.

[3] A. Graves, A. Mohamed, and G. E. Hinton, “Speech recognition with deep recurrent neural networks,” in Proc. ICASSP, Vancouver, 2013.

[4] Z. Luo, D. Shi and W. -S. Gan, "A Hybrid SFANC-FxNLMS Algorithm for Active Noise Control Based on Deep Learning," in *IEEE Signal Processing Letters*, vol. 29, pp. 1102-1106, 2022, doi: 10.1109/LSP.2022.3169428.

[5] https://github.com/udacity/AIND-VUI-Capstone