Moving Sound Sources Direction of Arrival Classification using Different Deep Learning Schemes

by

Jana Rusrus

Thesis Submitted in Partial Fulfillment of the
Requirements for the Degree of
Master of Applied Sciences in Electrical and Computer Engineering

in the
School of Electrical Engineering and Computer Science
Faculty of Engineering

Abstract

Sound source localization is an important task for several applications and the use of deep learning for this task has recently become a popular research topic. While the majority of the previous work has focused on static sound sources, in this work we evaluate the performance of a deep learning classification system for localization of high-speed moving sound sources. In particular, we systematically evaluate the effect of a wide range of parameters at three levels including: data generation (e.g., acoustic conditions), feature extraction (e.g., STFT parameters), and model training (e.g., neural network architectures). We evaluate the performance of multiple metrics in terms of precision, recall, F-score and confusion matrix in a multi-class multi-label classification framework. We used four different deep learning models: feedforward neural networks, recurrent neural network, gated recurrent networks and temporal Convolutional neural network. We showed that (1) the presence of some reverberation in the training dataset can help in achieving better detection for the direction of arrival of acoustic sources, (2) window size does not affect the performance of static sources but highly affects the performance of moving sources, (3) sequence length has a significant effect on the performance of recurrent neural network architectures, (4) temporal convolutional neural networks can outperform both recurrent and feedforward networks for moving sound sources, (5) training and testing on white noise is easier for the network than training on speech data, and (6) increasing the number of elements in the microphone array improves the performance of the direction of arrival estimation.

Acknowledgments

I would like to express my sincerest appreciation to my thesis supervisor, Professor Martin Bouchard, for his constant guidance, encouragement, and insightful feedback throughout my research journey. His passion and dedication to his work served as an inspiration and motivation for me to strive for excellence. I am also deeply grateful for the valuable education I received through his Adaptive Signal Processing and Digital Signal Processing courses, which provided me with a solid foundation for this work. I am truly honored to have been a part of his research team. I would also like to extend my deepest appreciation to my supervisor, Professor Shervin Shirmohammadi, for his assistance and support from the beginning of my graduate studies which helped me reach the successful completion of this degree. Thank you so much for your support.

I would like to express my gratitude to Professor Peter Galko for his outstanding course on Stochastic Processes. His knowledge of the subject matter and exceptional ability to explain complex concepts in an accessible manner were truly impressive. I am deeply grateful for all that I learned from his teachings.

I am deeply indebted to my parents for their unwavering support throughout my graduate studies. Without their endless love, encouragement, and guidance, I would not have been able to achieve this accomplishment. They have been my most dedicated supporters and I cannot thank them enough for everything they have done for me. I want to express my love and appreciation for them.

I would also like to express my deepest appreciation to my husband Sa'di, who has been my source of support throughout this journey. I am grateful for his encouragement and for being by my side through every step of the way.

I would like to express my heartfelt gratitude to my sisters and brothers, for their encouragement throughout my journey.

Finally, I want to dedicate this work to my daughter Shireen, who is the light of my life. Thank you for filling my life with joy.

Table of Contents

List of Figures

List of Tables

List of Acronyms

CBRNN	Convolutional Bidirectional Recurrent Neural Network
CNN	Convolutional Neural Network
DL	Deep Learning
DNN	Deep Neural Network
DOA	Direction Of Arrival
FCN	Fully Convolutional Network
FNN	Feedforward Neural Network
GCC-PHAT	Generalized Cross Power-Spectrum Phase Analysis Technique
GRU	Gated Recurrent Unit
ILDs	Interaural Level Difference
ITDs	Interaural Time Difference
NMSE	Normalized Mean Squared Error
ML	Machine Learning
MSE	Mean Squared Error
MUSIC	MUltiple SIgnal Classification
NN	Neural Network
RNN	Recurrent Neural Network
RT	Reverberation Time
SELU	Scaled Exponential Linear Unit
SNN	Self-Normalized Neural Network
SNR	Signal to Noise Ratio
SSL	Sound Source Localization
STFT	Short-Time Fourier Transform
TCN	Temporal Neural Network

Chapter 1 Introduction

1.1 Motivation

Sound source localization (SSL) has been a subject of intense research over the last decades, due to its major importance for audio applications. For instance, applications such as source extraction, speech enhancement, human-robot interaction, noise reduction, room acoustic analysis and speech/sound recognition from multiple microphone signals, in devices such as telephones, laptops, teleconferencing systems, and hearing aids. It is also used in other applications such as drone detection.

In recent times, there has been growing interest among researchers to address the challenge of detecting the Direction of Arrival (DOA) of moving sound sources, owing to its numerous practical applications. For example, in the field of robotics, autonomous vehicles, human-computer interaction, it is necessary to accurately locate and track moving sound sources to perform tasks such as speech recognition, noise reduction, and object detection. In some applications, acoustic sources moving with large angular speed can be found. For example, in the field of hearing aids with head movement, it is important to detect the direction of acoustic sources moving with large relative angular speed, which can be created due to head movements.

Although SSL is a long-standing and widely researched topic, it remains a very challenging problem to date. Traditional SSL methods are based on signal processing techniques. Although they have shown notable advances in the domain over the years, they are known to perform poorly in difficult yet common scenarios where noise, reverberation, and several simultaneously emitting sound sources may be found. In the last decade, the potential of data-driven deep learning (DL) techniques for addressing such difficult scenarios has received an increasing interest. As a result, an increasing number of SSL systems based on deep neural networks (DNNs) have been proposed in recent years. Recent work shows that DNNs boost the accuracy when applied to audio data. Due to this fact, they have been explored significantly in recent research.

In this thesis, we focus on the problem of moving acoustic sources detection and direction of arrival estimation, and we systematically analyze the effect of a wide range of parameters at three levels including: data generation (e.g., acoustic conditions), feature extraction (e.g., STFT parameters), and model training (e.g., neural network architectures).

1.2 Literature Review

Traditionally, several signal processing-based algorithms have been developed for the task of SSL such as Multiple Signal Classification (MUSIC) which is an algorithm that works by estimating the noise subspace of the received signals using eigenvector decomposition, and then using this estimate to obtain the DOA of the sources by finding the peaks in the spectral density function. By comparing the DOA estimates from multiple microphones or sensors, the location of the sound source can be determined. In SSL applications, the MUSIC algorithm can be used to enhance the performance of speech recognition systems by improving the accuracy of speaker localization. It can also be used in other areas such as radar and sonar systems, where it is used to estimate the location of targets based on their signal emissions [1]. The Generalized Cross Power-Spectrum Phase Analysis Technique (GCC-PHAT) which is a widely used method for estimating the time delay between two signals or the DOA of a sound source relative to an acoustic array. It works by cross-correlating two microphone signals in the frequency domain and computing the phase difference between them. This phase difference is then used to estimate the time delay between the two signals, which can be used to calculate the DOA of a sound source. The PHAT weighting function is applied to the cross-correlation function to normalize it and remove the amplitude information [2]. Such methods were developed under the free-field propagation model, which can lead to severe performance degradation in reverberant and noisy scenarios. To overcome these degradations, different algorithms using various neural network (NN) architectures and different sets of features have been considered in recent years.

In some works, short-time Fourier transform (STFT) features have been used [3]-[12], while in some other works features such as eigenvectors [13], interaural time difference (ITDs) and interaural level difference (ILDs) [14]-[16], acoustic intensity [17],[18] or cross-correlation in frequency bands (CCFB) [19] have been used. The authors in [20] also proposed an end-to-end CNN model to extract features from raw signals.

A common practice in processing complex-valued features (such as STFT features) using NN is to represent the features as two arrays of real numbers. However, it is also possible to adapt the NN model to work directly with complex numbers as in [6]. Another difference among different algorithms is in how to formulate the direction of arrival (DOA) detection problem. For instance, the authors in [7], [11], [12] and [13] formulated the SSL problem as a regression task. A more

popular option is to group DOAs into "classes" and formulate the DOA detection as a classification problem. The authors in [6],[7],[11],[12],[13],[15],[16],[20] and [21] considered scenarios where only one speaker is active, while other authors such as [4],[5],[14],[18] have considered multiple speaker scenarios.

The use of convolutional neural networks (CNN) has been prevalent in image processing for their exceptional ability to detect local patterns. A popular use of this NN architecture in acoustic DOA detection is to analyze spectrograms generated by STFTs and other similar methods [4],[5],[16],[19],[20]. While these works share a similar NN model, authors have focused on optimizing different aspects of their algorithms. For example, the authors in [4] studied the effect of training the CNN model using synthesized noise signals, while the authors in [5] focused on reducing the computational cost of CNNs. The authors in [8] combined convolutional bidirectional recurrent neural network (CBRNN) with transfer learning to deal with the issue of overfitting that occurred from increased model complexity. Authors in [7] introduced a more robust and hardware friendly architecture based on a Temporal Convolutional Network (TCN). In [3],[9],[10],[13] and[18], the authors combined a CNN with a recurrent neural network (RNN). The authors in [22] used fully convolutional network (FCN), while in some other work simple feedforward neural networks (FNN) have been used [6]-[9],[18].

Recently, some works [7],[11],[12],[18],[22] have considered moving sound sources. However, some authors did the work using one moving source only: [7],[11] and [12]. Others used a high tolerance in their work to get a good performance [18]. The work in [22] was done only for noise-free scenarios, and no recurrent network architecture was considered.

In this thesis, we consider performing SSL using feedforward networks, recurrent networks and temporal convolutional networks on different datasets featuring multiple static and moving sound sources and evaluating the effect of different hyperparameters such as window size, window shifts (hop size), number of frequency bins, with different levels of diffuse-like background acoustic noise (i.e., different SNRs) and different room reverberation levels. In addition, different source angular velocities are considered.

1.3 Objectives and Organization

In recent years, DL has become the most popular approach to tackle the sound source localization problem. Most of the research done in this field is concerned about finding the best NN model that is capable of mapping a certain set of features, measured from the directional audio to the corresponding DOA. The process of finding a good model implicitly involves another important process: finding a good training dataset. Designing a good dataset requires lots of domain-specific expertise. However, the effect of different parameters is not always understood because NN models are black-box systems. This is why most of the research work in this field focuses on reporting a working setup that leads to good performance without providing an in-depth analysis of the effect of different hyper-parameters, including the ones that didn't work. The main objective of this thesis is to fill this gap, by providing a systematic study of the effect of a wide range of parameters at three stages of the training pipeline including: data generation (e.g., acoustic conditions and microphone configuration), feature extraction (e.g., STFT parameters), and model training (e.g., neural network architectures).

To achieve this objective, we formulate the problem of DOA estimation of moving sources at high-speed as a classification problem and evaluate the performance of using different hyper-parameters based on several metrics including the confusion matrix, precision, recall, F-score, and prediction plots.

In particular, Chapter 2 provides some background material related to signal processing and deep learning algorithms used in this thesis. It also describes the performance metrics that are used for evaluation. Chapter 3 describes the system implemented in this thesis. It gives detailed descriptions of the tools used to implement the datasets such as signal generator and noise generator. In addition, it provides a description of the different configurations used to implement the datasets, such as acoustic conditions (e.g., reverberation time (RT), Signal to Noise Ratio (SNR), and microphone configuration). Also, it provides the design of the neural network models used in this thesis such as feedforward neural networks (FNN), recurrent neural networks (RNN), gated recurrent networks (GRU), and temporal convolutional networks (TCN). The goal is to provide a good understanding of the trade-offs one has to make when choosing the different hyper-parameters which, in turn, explains when a certain combination of hyper-parameters might succeed or fail at capturing useful information for the task in hand.

We start our experiments using a simplified version of the problem described in Chapter 4. In this setup, we use a simple FNN model to detect DOA at a low resolution of 15^o azimuth angle. This serves as a baseline to compare with when doing further analysis as follows in Chapters 5-7. In Chapter 5, we change multiple parameters (one at a time) and include more complicated NN models that focus on local features (e.g., using CNN) and temporal features (e.g., RNN). We analyze the performance of testing with different angular speeds, sequence lengths and window sizes, at 5^o azimuth angle resolution.

Chapter 6 provides additional experiments to compare the performance when using speech versus white noise signals for the training data. The motivation behind this chapter lies in the fact that one might expect that the direction of an audio source is not related/doesn't depend on the content of that signal. To test this hypothesis, we train the NN models with a pure white noise signal and compare its results with scenarios where real speech is used. Chapter 7 provides a comparison for the performance of microphone arrays using 2 microphones and 4 microphones, with some analysis. Finally, Chapter 8 concludes the thesis and provides some suggestions for possible future work.

1.4 Contributions

The main contributions and results of this work are:

- We explored the effects of room acoustic conditions for acoustic sources detection. In particular, we studied the effect of the reverberation time, SNR levels, and static versus moving sources. For moving sources, we tested different angular speeds, different number of elements of microphone arrays and different types of training datasets.
- We analyzed the effect of the parameters of the Short Time Fourier Transform (STFT), such as hop length, number of frequency bins used and more importantly, window size.
- We focused on evaluating the performance of different types of neural networks, in particular FNN, vanilla RNN, GRU and TCN networks.

This resulted in the following publications:

1) J. Rusrus, M. Bouchard, and S. Shirmohammadi, "Direction of Arrival Estimation of Moving Sound Sources using Deep Learning," in *Proc. of 2022 IEEE International Instrumentation and Measurement Technology Conference (I2MTC*), Ottawa, Canada, pp. 1-6, May 16-19, 2022.

2) J. Rusrus, S. Shirmohammadi, and M. Bouchard, “Characterization of Moving Sound Sources Direction-of-Arrival Estimation Using Different Deep Learning Architectures”, *IEEE Transactions on Instrumentation and Measurement*, vol. 72, pp. 1-14, 2023.

Chapter 2 Background

2.1 Signal Processing

This section presents some topics in signal processing used in this thesis. In particular, it discusses microphone technology, the image method for simulating room acoustics, isotropic noise, and some mathematical tools used in this thesis such as coherence and STFT.

2.1.1 Microphone Technology

This subsection describes some topics related to microphone technology. Specifically, it presents microphone sensors and microphone arrays.

2.1.1.1 Microphone Sensors

A microphone is a sensor element that converts a sound wave into an electrical signal that can be processed, transmitted, or stored. At its core is a thin sheet of material, called diaphragm, that vibrates back and forth as a result of the air pressure caused by the sound wave. Different microphone types implement different mechanisms used to convert the vibrations into electrical signal. Also, microphones can exhibit different sensitivity to sounds coming from different directions and distances. Based on those two factors, microphones are classified into different types including: dynamic microphones, condenser microphones, large diaphragm condensers, small diaphragm condensers, ribbon microphones, shotgun microphones, lapel microphones, contact microphones, and tube microphones [23].

Polar patterns describe how a microphone is able to pick up sound from around it. The simplest polar pattern is an omnidirectional polar pattern (Figure 2.1.a). As the name suggests, the microphones are able to pick sounds from all directions equally loudly [24]. This is useful for applications that require recording ambient sound such as noise cancellation. However, in a situation where detecting a particular sound source is desired, the cardioid pattern (Figure 2.1.b) is more suitable thanks to its ability to reject background noise while picking up sound sources from the front and the sides [25]. There are also a couple of variations of the cardioid pattern called super-cardioid (Figure 2.1.c) and hyper-cardioid [25] (Figure 2.1.d) which are slightly narrower on the sides, but they tend to pick up a little more from the back. Another two polar patterns that

can be useful for picking up sources on opposite sides (e.g., two speakers in a conversation) and rejects interfering sounds coming from the sides are the 8-shape pattern (Figure 2.1.e) and the Lobar (or shutgun) pattern [24] (Figure 2.1.f). The Lobar pattern is a highly directional. The downside however is that it allows some sources from the side. It should be noted that polar patterns with high directivity can also come with the downside of amplifying spatially uncorrelated noise.

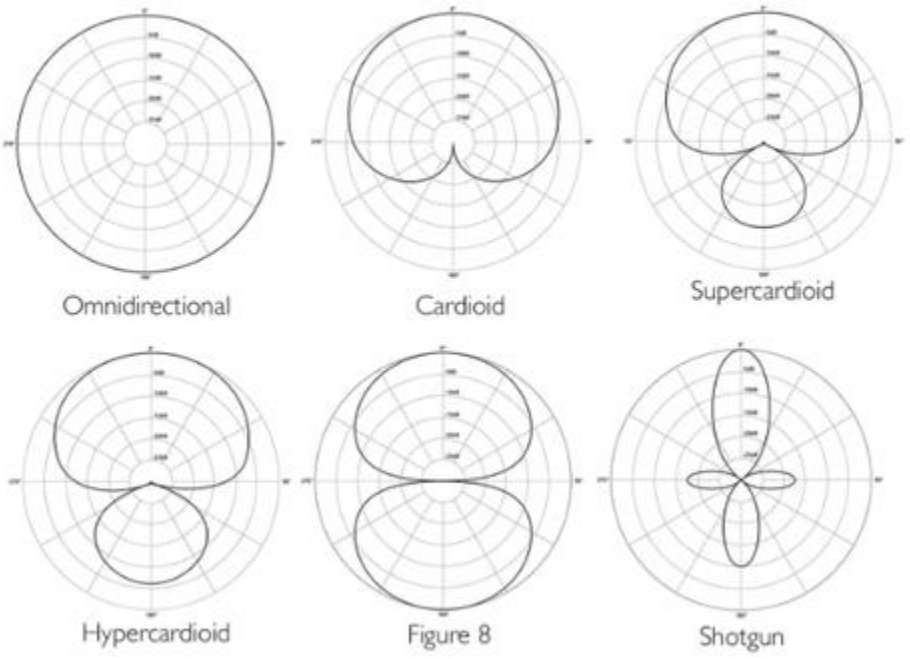


Figure 2.1 Polar patterns (a) omnidirectional, (b) cardioid, (c) super-cardioid, (d) hyper-cardioid, (e) figure of 8, (f) shotgun.

2.1.1.2 Microphone Arrays

A microphone array is a technology that utilize multiple microphones elements which work together to record the sound wave simultaneously. Microphone arrays can be designed to have as many microphones in them as needed to record the sound. The two-microphone array is the simplest configuration, with one microphone placed on the left side and the other placed on the

right side. With one microphone on each side, microphone arrays can capture a stereo recording that mimics surround sound.

Typically, the microphones in a microphone array are in fixed positions, separated from one another by an exact, predetermined distance. Each microphone is at a different distance from the source to be detected. Every time there is a sound source in the area surrounded by the microphones, the source spreads around the array in a roughly spherical fashion, so it is recorded with a slight delay by each one of the microphones. Thereby, the array is able to make precise calculations based on these time lag differences, and then determine the sound's location based upon those calculations [24].

There are many applications that require microphone arrays. They are typically used for localizing the source of a sound. Also, noise reduction and signal enhancement are typical applications. Each microphone picks up sources of noise with varying delays and volumes. By comparing the differences in noise content among the microphone recordings, specific sounds can be isolated, then amplified or removed [24].

While microphone arrays allow for enhanced awareness of a system, they also create more room for error and over-complication. There are a few drawbacks to consider and address when using microphone arrays. For example, it can increase the processing time, e.g., shifting a recording across another by small time increments and calculating the difference at each point to determine time lag requires a very large number of small operations, so it can take a long time [25].

2.1.2 Image Method

A wave is a disturbance that travels through a medium from one location to another one. Any complex sound field can be described as a superposition of a number of simple sound waves. The wave equation is a second-order partial differential equation that describes the propagation of waves through the free field as a function of position and time. The frequency domain counterpart of the wave equation is called the Helmholtz equation [26].

Simulating sound wave propagation in a closed environment is more complicated than free field due to the destructive effect caused by the walls. Figure 2.2 illustrates some of these effects including reflections, refraction, diffraction, absorptions, and scattering. To simulate the room

acoustic effect, the Room Impulse Response (RIR) is used. For every pair of points (representing the position of source and microphone) in the room, the RIR describes the transformation applied to a sound wave as it travels between the sound source and the microphone.

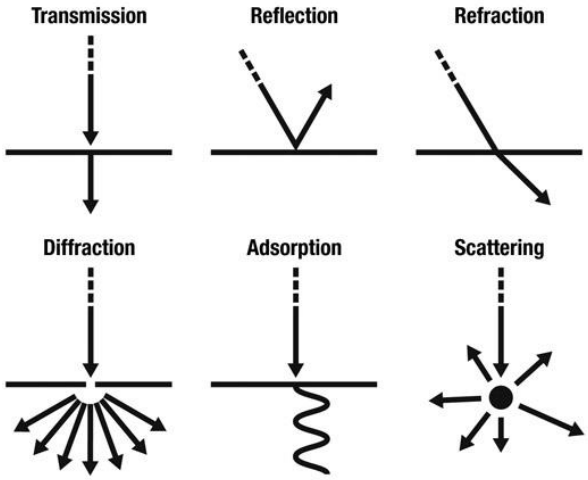


Figure 2.2 Different phenomena affecting sound wave inside a room.

Simulating room acoustic can be modeled using three different methods: ray-based modeling [27], wave-based modeling [27], and statistical modeling [28]. The ray-based methods, such as ray tracing and the image source method, are the most widely used. The wave-based methods, such as the Finite Element Method (FEM), and Boundary Element Method (BEM) are computationally expensive and more demanding, and they can be useful only in simple cases. The statistical modeling methods, such as statistical energy analysis, are used in high-frequency noise analysis and acoustic designs and they cannot model the temporal behavior of the sound field [28].

In the image source method, the process is simplified by assuming that sound propagates only along straight lines or rays. Sound energy travels at a fixed speed, corresponding to the speed of sound, along these rays. The energy in each ray decreases with the total distance that the ray has traveled. Rays are perfectly reflected at boundaries. When a ray is reflected, it creates a secondary source behind the boundary surface. This source is located on a line perpendicular to the wall, at the same distance from it as the original source, as if the original source has been mirrored on the surface as shown in Figure 2.3. This new image source represents a perfect reflection path, in that the distance along the straight line between the receiver and the image source has the same length as the path from the real source to the receiver, reflected in the boundary. If the source is reflected

in a single boundary, this represents a first-order reflection. A ray that is reflected from several boundaries is represented by a higher-order image source, which has been mirrored in each of those boundaries in turn.

All sources, original and images, emit the same impulsive source signal at the same time. The total impulse response (i.e., sound pressure against time) is found by summing the signals from each source, delayed, and attenuated appropriately depending on the distance between that source and the receiver, which is equivalent to the length of the specular reflection path. The frequency response of the signal from each image source will additionally be modified depending on the characteristics of each boundary in which that source was reflected. In the real world, not all energy is perfectly reflected at a boundary. Some energy will be randomly diffused in non-specular directions. The image-source model is not capable of modeling this phenomenon, though this is not particularly problematic. Consider that, once scattered, sound energy cannot become unscattered. The conversion from incoming energy to scattered energy is unidirectional, so repeated reflections cause the ratio of scattered to specular energy to increase monotonically.

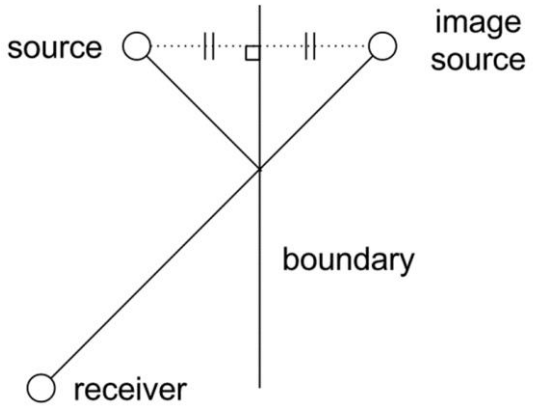


Figure 2.3 Image sources are found by reflecting the source position in a boundary.

2.1.3 Isotropic Noise

A noise signal is an unwanted signal that corrupts the desired signal. Noise encountered in real-life scenarios can often be approximated as spherically or cylindrically isotropic noise. A spherically isotropic noise, also known as three-dimensional (3D) diffuse noise field, has been

shown to be a reasonable model for a number of practical scenarios such as in an office or a car. Cylindrically isotropic two-dimensional (2D) diffuse noise fields are especially useful when, for example, the ceiling and floor in an enclosure are covered with a highly absorbing material.

Two common types of noise are: spatially white noise and isotropic noise. The former describes a case where the noise signal at any two different points in space are always uncorrelated (e.g., sensor noise or thermal noise, non-acoustic noises), while the later describes a case where the power of the incoming noise is equal from all directions (e.g., ambient background acoustic noise). Besides being isotropic, it is often assumed that the noise field is spatially homogeneous, i.e., the physical properties of the sound do not depend on the absolute position of the sensor, and are time invariant.

Generating isotropic noise is required in the signal processing field for simulating purposes such as beamforming, source localization and adaptive noise cancellation. It can be generated using large number of uncorrelated noise sources that are uniformly spaced on a sphere or a cylinder. Another way to generate the noise is using the convolution of two uncorrelated noise signals that result from a spherically isotropic noise field [29]. The characteristics of the noise field is described by a spatial coherence function.

2.1.4 Mathematical Tools

This subsection discusses two important mathematical tools used in this thesis: coherence and STFT.

2.1.4.1 Coherence

A coherence is a statistical measure and a function related to cross-correlation that shows the degree that two ergodic signals are related through a linear process. In the frequency domain, when only the magnitude is considered, it is a real quantity with values between 0 and 1 (where 1 means one signal can be completely predicted from the other through a linear filter). It is found by the ratio of the cross power spectral density divided by the auto power spectral densities as:

$$C(\omega) = \frac{|S_{xy}(\omega)|^2}{S_{xx}(\omega)S_{yy}(\omega)} \tag{2.1}$$

Where $C(\omega)$ is the coherence, $S_{xx}(\omega)$ is the auto power spectral density of signal $x(t)$, $S_{yy}(\omega)$ is the auto power spectral density of signal $y(t)$, and $S_{xy}(\omega)$ is the cross power spectral density between two signals $x(t)$ and $y(t)$. When x and y are uncorrelated (e.g., y is a noise process not derived from x), the sample coherence converges to zero at all frequencies, as the number of samples or blocks used in the average goes to infinity [30].

A common use for the coherence function is in the validation of input/output data collected in an acoustics experiment for purposes of system identification. For example, $x(t)$ might be a known signal which is input to an unknown system, such as a reverberant room, say, and $y(t)$ is the recorded response of the room. Ideally, the coherence should be 1 at all frequencies. However, if the microphone is situated at a null in the room response for some frequency, it may record mostly noise at that frequency. This is indicated in the measured coherence by a significant dip below 1 [30]. Coherence is also used to characterize the sound field of spherically or cylindrically isotropic diffuse noise, so its measure can allow to validate if synthetically produced diffuse noise field signals at different microphone locations are valid or not.

2.1.4.2 Short Time Fourier Transform (STFT)

Short time Fourier transform is a mathematical tool that is used to analyze the frequency components in a signal at different time instances. It divides the signal into short segments, optionally with overlap between successive segments, and applies the Fourier Transform to each segment to extract its frequency components. The output of magnitude STFT is a spectrogram represented as a three-dimensional plot; the x- and y- axis represent the time and frequency respectively, and the z-axis (usually represented as a heatmap) represent the intensity (amplitude or square amplitude) of the sound signal at each time-frequency bin.

STFT is important because it preserves the time information, as opposed to the Fourier Transform which extracts the frequency components presented in the signal without any indication of when

these appear (except when also considering the phase, where the timing information is encoded but it is not explicitly visible). There are three important STFT parameters: window size, hop size (window shift value), and the number of frequency bins used. Both the window size and hop size are measured in seconds. They define the size of each chunk when segmenting the signal and the amount of overlap between consecutive segments. The frequency is measured in Hertz and can be quantized into bins using a linear or logarithmic scale. The output of STFT is a two-dimensional array (a spectral matrix) of size: [F, N] where F is the number of frequency bins and N is a number of segments (frames) [31].

When choosing the STFT hyper-parameters, it is important to consider the trade-off between the time- and frequency- resolution. In particular, achieving good time resolution requires using smaller window size in the time domain. However, this negatively affect the frequency resolution. The same happen if better frequency resolution is desired, where the window size has to increase causing the time resolution to degrade [31]. Another important parameter to consider is the window shape. In practice, the rectangle window is not used because it creates discontinuity at the edges, leading to higher levels of spectral leakage. A better option is to use a bell-shaped window, such as the Hanning window given below, because it has smooth transitions on the edges [30].

$$w(n) = 0.5 - 0.5\cos(2\pi(n - Lf_s / 2) / (Lf_s)) \quad 0 \le n \le Lf_s \tag{2.2}$$

Where L is the window length in seconds and f_s is the sampling rate.

The STFT is then described as:

$$X[m,\omega) = \sum_{m=0}^{L-1} x[m+n]w[m]e^{-j\omega m} \tag{2.3}$$

STFT is a complex-valued function that can be visualized in terms of two plots: the magnitude $|X[m,\omega)|$ (spectrogram, often squared as $|X[m,\omega)|^2$) and the phase $\angle X[m,\omega)$. It is worth noting that in DOA estimation in free field the phase $\angle X[m,\omega)$ more important than the magnitude, because it corresponds to time difference of arrival (TDOA) which is directly related to the source azimuth direction.

2.2 Deep Learning

Traditional digital signal processing techniques have been used for decades to process audio signals for many applications. However, this requires a lot of domain-specific expertise. In recent years, machine learning (ML) has become more and more ubiquitous and has seen tremendous success in solving a wide range of audio use cases such as audio separation and segmentation, music generation, speech transcription, voice recognition, speech-to-text and text-to-speech translation, and audio classification. This section introduces a family of ML algorithm called deep neural networks, which have been a state-of-the-art ML algorithm for a wide range of tasks. In particular, it discusses different architectures, evaluation metrics, and learning algorithms used in this thesis.

2.2.1 Architecture

Artificial neural networks consist of a set of computational units called artificial neurons that are connected to create network. The connections in a NN are not random. They use a predesigned architecture in which neurons are grouped into layers and are connected using tunable parameters. The type of neurons and the way they are connected define the neural networks.

2.2.1.1 Feedforward Neural Network (FNN)

The oldest and most classic architecture invented in 1960s is called multilayer perceptron or feedforward neural network (FNN) or fully connected network [32]. In this architecture, neurons are grouped by layers and every neuron sends its output to every next neuron of the next layer. The following figure shows an example of this network.

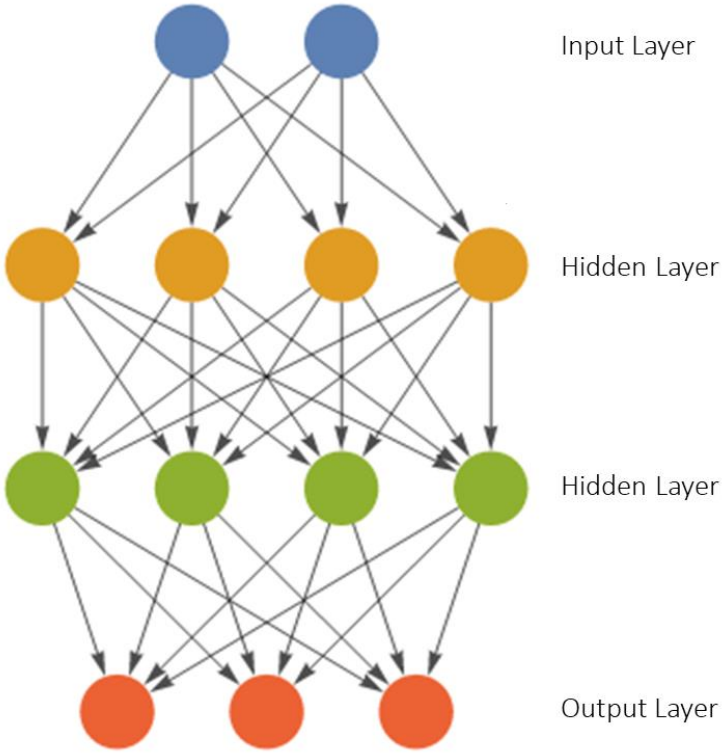


Figure 2.4 FNN Architecture.

This network has three layers: two intermediate layers which are called hidden layers and one output layer. A network with only one (or very few) hidden layer is called a shallow network while a network with more hidden layers is called a deep network. In the above example, the network takes two values as input and returns three values as output. In general, the inputs of a NN be to be defined as numeric values $x_1, x_2, ..x_n$ and the output of an artificial neuron is given by:

$$y = f(w_1x_1 + w_2x_2 + w_nx_n + b) \tag{2.4}$$

Where $w_1, w_2, ..w_n$ are learnable parameters called weights, and b is another learnable parameter called bias. Its value is interpreted as a threshold which fires neurons. The linear transformation $wx+b$ aims at extracting the useful information hidden in the raw data. f is a nonlinear function used to model the relation between the inputs and the outputs. Figure 2.5 is an illustration of the computation made by an artificial neuron.

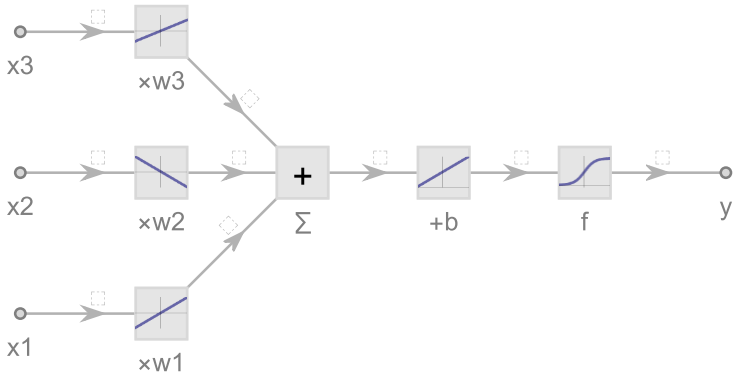


Figure 2.5 A graph representation of a simple neuron. Arrows represent the weights and squares represent operators.

The first part is a linear combination of the features, and then a sigmoid nonlinearity is applied in this example. The presence of this nonlinearity allows neural networks to model nonlinear systems. A classic modern activation function is the Rectified Linear Unit (ReLU). Most modern nonlinearities are variations of the ReLU activation.

When FNN was first introduced, it never really managed to take over classic ML methods. On the one hand, using only shallow FNN limits the model in what it can learn and achieve. On the other hand, using deep FNN models were typically unstable because of two well-known problems: the vanishing gradient and the exploding gradient problems. By 2017, though, it had been shown that using a specific activation function and regularization mechanism allows the FNN to overcome both problems and outperform rival classic machine learning algorithms. In particular, using a self-normalizing NN (SNN) using a Scaled Exponential Linear Unit (SELU) activation function [33]. Unlike batch normalization which requires explicit normalization, SNN's neurons activations automatically converge towards zero mean and unit variance. This is done using the Banach fixed point theorem. The convergence of the SNN allows to train deeper networks, employ stronger regularization and make the learning more robust. Furthermore, SNN do not suffer from high variance and vanishing gradient.

The construction of the SNN is done by two choices: (1) the activation function and (2) the initialization of the weights. The activation function requires to have negative and positive values to control the mean, a saturation region to dampen the variance, a slope larger than one to increase the variance if it is too small and a continuous curve. For the weight initialization, they should be

drawn from Gaussian distribution with mean=0 and variance = 1/n (where n is the number of weights connected to a neuron). Following this approach allows to use FNN networks, while avoiding the vanishing gradient and the exploding gradient problems, and without the necessity of using strong regularization or dropout levels.

Nevertheless, the use of FNN is still marginal overall. Since neural networks are mostly used on structured data (image, sound, text, etc..), other architectures such as CNN and RNN have often been found to outperform FNNs. These architectures still use the concept of layers, but their connectivity is quite different from MLP/FNN.

2.2.1.2 Recurrent Neural Network (RNN)

One big disadvantage of the FNN that it cannot use previous NN states when the current set of data is being processed. Recurrent models, which were invented in the 1990s, on the other hand, have a unique architecture that allows them to memorize short term dependencies via a discrete hidden state. In RNNs [32], the output at time t not only depends on the current inputs and weights W but also on the hidden state which summarizes all previous inputs:

$$\begin{aligned} s_t &= \phi(x_{t-1},...,x_{t-N};W) \\ &= \phi(x_t, s_{t-1};W) \end{aligned}, \tag{2.5}$$

Where x_t is the input at time t, s_{t-1} is the hidden state that summarizes the previous inputs up to the previous time step, and s_t is the current output (or state). An illustration of the RNN cell is shown in Figure 2.6. Studies have shown that simple RNNs can be used effectively up to a small number of time steps [32].

Although a RNN neural model processes a sequence, in practice it can typically only use the last ten steps at most. On the one hand, this tiny memory prevents the model from extracting

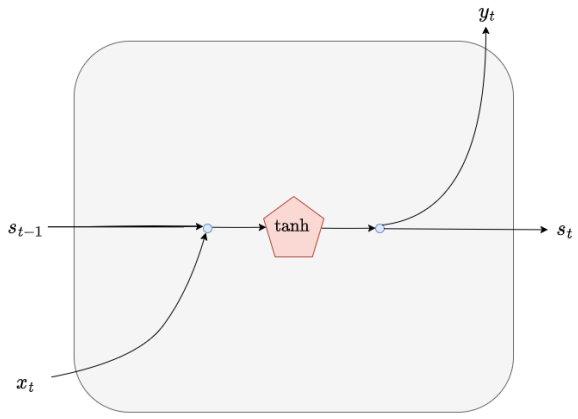


Figure 2.6 An illustration of RNN cell.

information resulting from the interaction of elements far apart. On the other hand, backpropagating further the gradient will become too small, which is known as the vanishing gradient problem where the contribution of the information decays dramatically over time [34].

An RNN can be seen as a module processing sequences in a given direction while keeping and updating a state. This state allows it to remember things from the past of the sequence. The following figure shows a classic representation of a recurrent layer.

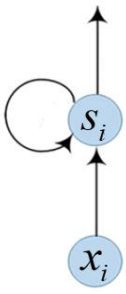


Figure 2.7 A classic representation of a recurrent layer.

Where x_i is the i^{th} element of the sequence and s_i is the state of the network after preprocessing this element. The self-connection means that the state s_i is used to compute the next state s_{i+1}. This network graph can be difficult to understand how information moves through the network. This can be unfolded without having any cycles. Figure 2.8 shows an example.

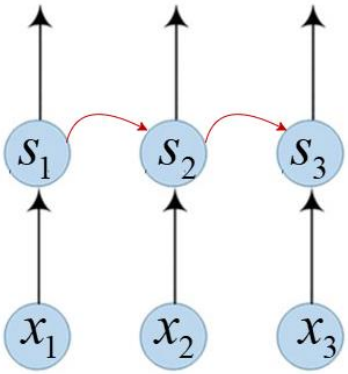


Figure 2.8 An example of unrolled recurrent network with 3 long sequences.

This looks more like a regular fixed-size network with the particularity that weights are shared since the same module is used along the sequence. RNN have been the best method for text and generic sequence processing for a long time. One issue that recurrent neural networks have is that they have trouble remembering long sequence. Another issue is that RNN are fundamentally sequential and thus harder to parallelize in order to speed up the training phase.

2.2.1.3 Gated Recurrent Unit (GRU)

The GRU was proposed in 2014 [34] to allow recurrent units to adaptively capture dependencies of different time scales. Both "vanilla" RNN and GRU have feedback in their structure, and both have the same number of ports (number of inputs and outputs). However, the GRU differs from RNN mainly in two aspects. First, it has gating units that modulate the flow of information inside the unit. Second, it has a free path that allows the parameters to be transmitted to the next computation step without being affected/multiplied by the activation function. These differences have two advantages. First, the gates are more powerful in modelling. They have the ability to decide which values from the past should be transmitted to the new state and which are to be erased. Second, this addition effectively creates shortcut paths that bypass multiple temporal steps. These shortcuts allow the error to be back-propagated easily without too quickly vanishing as a

result of passing through multiple, bounded nonlinearities, thus reducing the difficulty due to vanishing gradients. An illustration of the GRU cell can be found in Figure 2.9

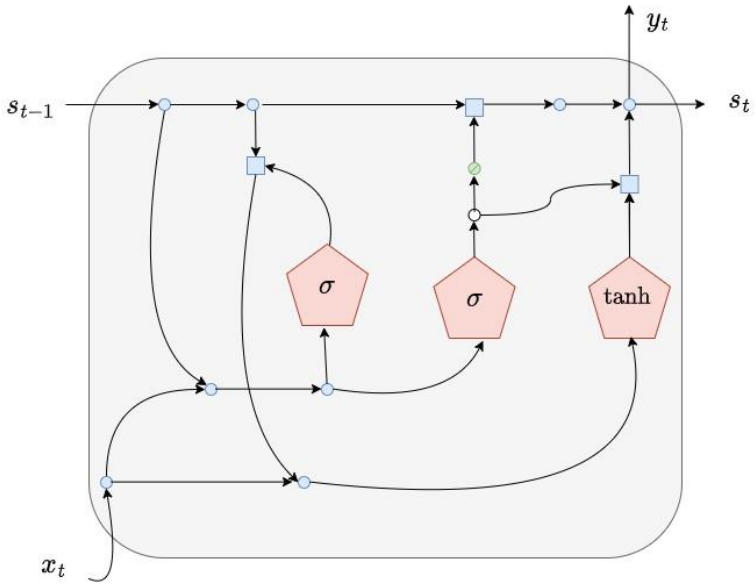


Figure 2.9 GRU cell.

The equations of the GRU process can be expressed as

$$r_t = \sigma(W^r x_t + U^r s_{t-1} + b^r)$$

$$\tilde{s}_t = \tanh(W^{\tilde{s}} x_t + U^{\tilde{s}}(r_t s_{t-1}) + b^{\tilde{s}})$$

$$z_t = \sigma(W^z x_t + U^z s_{t-1} + b^z)$$

$$y_t = s_t = z_t \tilde{s}_t + (1 - z_t) s_{t-1} \qquad (2.6)$$

where W and U are learnable parameters representing the weights and b representing the biases, and $\sigma(.)$ is a non-linear activation function. $r_t, \tilde{s}_t, z_t$ are the output of each gate, and y_t is the final output.

2.2.1.4 Temporal Convolutional Network (TCN)

Temporal Convolutional Network (TCN) [35] is a new family of CNNs that have been introduced in 2016 as a strong alternative for capturing long-term dependencies. The main principles of this neural network are: (1) The architecture can take an input of any sequence length and produce an output of the same length. (2) There is no information leakage from future to past. The first principle can be ensured by using 1D fully convolutional network, where each hidden layer has the same length as the input layer, and zero padding asymmetrically is added to preserve the sequence length. The second principle is achieved by using causal convolution. The causality property is important for real time application, in which there is no access to future samples. In order to ensure a large receptive field, a dilation factor is used for causal convolutions:

$$y_t = \sum_{i=0}^{k-1} f_i x_{t-di} \tag{2.7}$$

where x is the input, y is the output, f are filter coefficients (weights), d is the dilation factor (skipping or downsampling factor), k is the filter size and $t-di$ accounts for the direction in the past. With dilated convolution, the network can look back far in time without increasing too much the number of weights. An illustration of the dilated TCN model is shown in Figure 2.10. However, this can result in sparse hidden connections, which prevents seeing input values between x_{t-di} and $x_{t-d(i-1)}$. To overcome this issue, several stack convolution layers are used with different dilation factors.

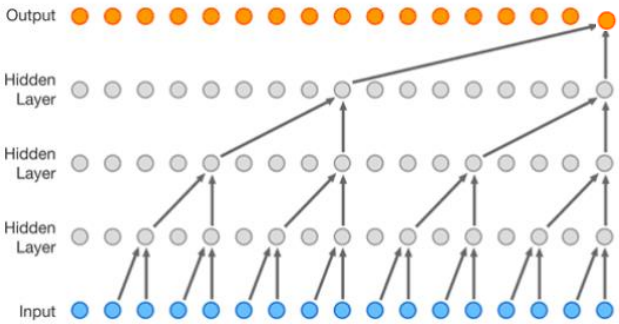


Figure 2.10 Dilated TCN model.

Since TCN do not have recurrent connections, they do not use backpropagation through time and therefore can be trained in parallel. This makes training faster than for recurrent models. Despite bringing some improvements, TCN architecture also has two main disadvantages. First, the design of TCN needs a deep network to accomplish a long effective history size. Second, the receptive field is fixed a priori, which means input values outside the receptive field cannot be considered for the calculations of the output at a particular position.

2.2.2 Evaluation Metrics

Choosing appropriate metrics for evaluating a model is very important. Wrong metrics can be misleading causing to draw wrong conclusion about the model's performance. However, choosing the proper metric is usually challenging. Different tasks have different appropriate metrics. For example, Mean Square Error (MSE) is a popular metric for regression problems, while accuracy is a well-known metric for classification problems. As in this thesis we are dealing with a classification problem, we will discuss the classification metrics in this section.

Accuracy and Error metrics:

The most widely used metrics are Accuracy and Error:

$$Accuracy = \frac{Correct\ Predictions}{Total\ Predictions} \tag{2.8}$$

and the complement of classification accuracy is called classification error:

$$Error = \frac{Incorrect\ Predictions}{Total\ Predictions} \quad (2.9).$$

Although they are widely used, classification accuracy is inappropriate for imbalanced classification, because high accuracy (or low error) can be achievable by a no skill model that only predicts the majority class.

Precision-Recall Metrics:

Precision measures the ratio of positive predictions made which are correct, whereas recall measures the ratio of positive cases which the classifier correctly predicts, over all the positive cases in the data. Higher precision means that an algorithm returns less false positives, and high recall means that an algorithm misses less positive cases (but could still generate a lot of false positives).

$$\textbf{Precision} = \frac{TruePositives}{(TruePositives + FalsePositives)} = \frac{TruePositives}{PredictedPositives} \quad (2.10)$$

$$\textbf{Recall} = \frac{TruePositives}{(TruePositives + FalseNegatives)} = \frac{TruePositives}{Positives} \quad (2.11)$$

Precision and recall can be combined into a single score that seeks to balance both concerns, called the F-score or the F-measure.

$$F-Measure = \frac{2 \times Precision \times Recall}{(Precision + Recall)} \quad (2.12)$$

The F-Measure is a popular metric for imbalanced classification.

Confusion Matrix:

A confusion matrix is a summary of prediction results on a classification problem. The number of correct and incorrect predictions are summarized with count values and broken down by each class. It shows the ways in which the classification model is confused when it makes predictions and gives an insight not only into the errors being made by the classifier but more importantly the types of errors that are being made. Each row corresponds to a predicted class and each column of the matrix corresponds to an actual class. The diagonal of this matrix represents the correct prediction

when $i = j$. The confusion matrix used in this work is as defined in [36], where every element in the matrix is normalized by the sum of elements in each row.

2.2.3 Learning Algorithms

This subsection presents some foundations of optimization used in machine learning. In particular, it discusses gradient descent, stochastic gradient descent, and backpropagation.

2.2.3.1 Gradient Descent

Gradient Descent is an optimization algorithm that finds the set of input variables for a target function that results in a minimum value of the target function, called the minimum of the function. The gradient descent algorithm requires the calculation of the gradient of the target function with respect to the specific values of the input values. The gradient points uphill, therefore the negative of the gradient of each input variable is followed downhill to result in new values for each variable that result in a lower evaluation of the cost function. A step size is used to scale the gradient and control how much to change each input variable with respect to the gradient. This process is repeated until the minimum of the target function is located, a maximum number of candidate solutions are evaluated, or some other stop condition.

2.2.3.2 Stochastic Gradient Descent

Stochastic Gradient Descent is an extension of the gradient descent optimization algorithm for minimizing a loss function of a predictive model on a practical training dataset. The target function is taken as the loss or error function on the dataset, such as mean squared error for regression or cross-entropy for classification [37]. The parameters of the model are taken as the input variables for the target function. The algorithm is referred to as “stochastic” because the gradients of the target function with respect to the input variables are noisy (e.g. a probabilistic or instantaneous approximation). This means that the evaluation of the gradient may have statistical noise that may

obscure the true underlying gradient signal, caused by sparseness of data considered for gradient computation and noise in the training dataset. Stochastic gradient descent is the most efficient (simplest) algorithm discovered for training artificial neural networks, where the weights are the model parameters and the target loss function is the prediction error averaged over a subset (batch) of the entire training dataset. Stochastic gradient descent can be used to train (optimize) many different model types, like linear regression and logistic regression, although more efficient optimization algorithms built on top of stochastic gradient descent have been discovered and are often used instead. Some of these popular extensions to stochastic gradient descent designed to improve the optimization process (same or better loss in fewer iterations) are the Momentum, Root Mean Squared Propagation (RMSProp) and Adaptive Movement Estimation (Adam) algorithms. A challenge when using stochastic gradient descent to train a neural network is how to calculate the gradient for nodes in hidden layers in the network, e.g., nodes one or more layers away from the output layer of the model. This requires a specific technique from calculus called the chain rule and an efficient algorithm that implements the chain rule that can be used to calculate gradients for any parameter in the network. This algorithm is called back-propagation.

2.2.3.3 Back-propagation

Back-propagation is an algorithm for calculating the gradient of a loss function with respect to variables of a model. Back-propagation is used when training neural network models to calculate the gradient for each weight in the network model. The gradient is then used by an optimization algorithm to update the model weights. The algorithm was developed explicitly for calculating the gradients of variables in graph structures working backward from the output of the graph toward the input of the graph, propagating the error in the predicted output that is used to calculate gradient for each variable. The loss function represents the error of the model or error function, the weights are the variables for the function, and the gradients of the error function with regards to the weights are therefore referred to as error gradients. The algorithm involves the recursive application of the chain rule from calculus that is used to calculate the derivative of a sub-function given the derivative of the parent function for which the derivative is known [37]. There are other algorithms for calculating the chain rule, but the back-propagation algorithm is an efficient algorithm for the specific graph structured using a neural network. Although back-propagation was developed to

train neural network models, both the back-propagation algorithm specifically and the chain-rule formula that it implements efficiently can be used more generally to calculate derivatives of functions.

Chapter 3 System Design

This chapter presents the design of the system required for acoustic sources DOA estimation. We first explain how we generate and handle the datasets. Then, we show the steps involved in the audio features extraction process. Next, we discuss the neural networks training.

3.1 Datasets Generation

The dataset generation process consists of two main steps (as shown in Figure 3.1): sample-based and frame-based processing. In the first step, we start with a set of clean single channel recordings. Then, we simulate the room conditions using the image-source method. Next, we convolve the signals from the clean input dataset files with the resulted room impulse responses. The outputs of these convolutions result in a directional multi-channel audio signal. We simulate the background acoustic diffuse noise as measured by the microphone array, depending on the desired SNR.

In the second step, we use STFT to convert the output signal of the above first step to the frequency domain. Then, we calculate the magnitude ratio and phase difference between a reference microphone and the other microphone(s) to generate the features for NN training.

Our simulator combines three public-domain available tools in a unified pipeline and implements the necessary utility functions that allows experimentation for NN research. In particular, we used the Room Impulse Response (RIR) generator from [38], the spatial noise generator from [29], and the STFT function from the nnAudio library [39]. The detail of each step is further explained in the following three subsections.

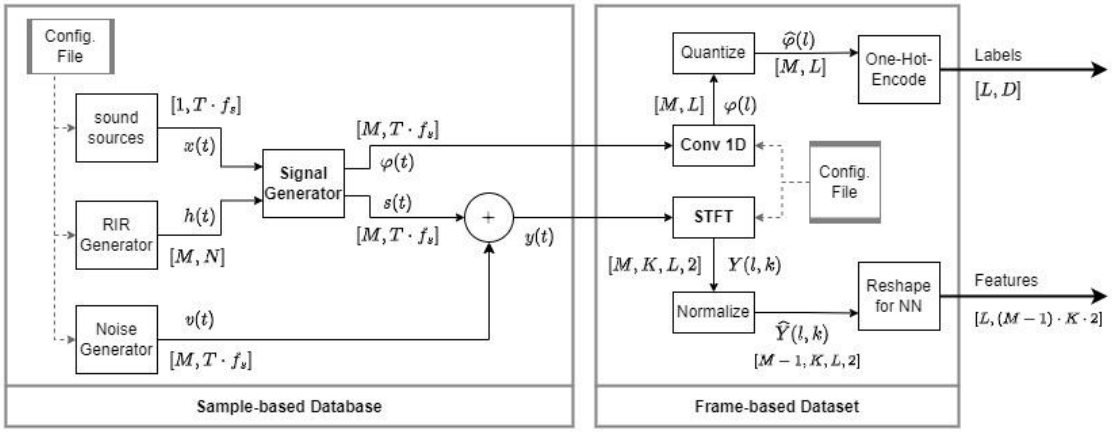


Figure 3.1 Dataset generation: generation of noisy directional raw signals and feature extraction.

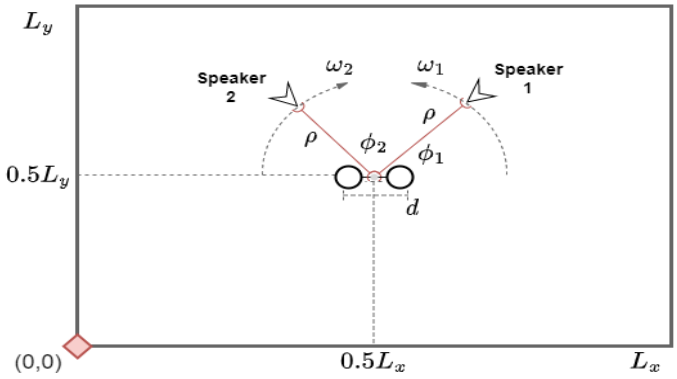


Figure 3.2 An illustration of the experimental setup.

3.1.1 Database

We conducted our proposed experiments for acoustic sources DOA estimation using two datasets for the acoustic sources: (1) White Gaussian Noise (WGN) acoustic sources and (2) speech sources from TIMIT [40]. The WGN dataset was generated using the WGN function from MATLAB. The noise was normalized, the power was set to 0 dB and the audio was sampled at 16 kHz. The TIMIT dataset consists of 630 monaural speech files sampled at 16 kHz, with duration 3-5 seconds each.

We randomly selected 280 files, normalized each in the range (-1, 1) and concatenated them. Then, we pre-processed them to remove silences/pauses, using a threshold.

3.1.2 Signal Generator

The Signal Generator is used to generate multi-channel directional sound sources in a reverberant environment. We used the WGN/TIMIT speech databases to create clean mono audio sources $x(t)$ with a desired total length T s. and a sampling rate f_s Hz. The desired length can be achieved by cycling through the available short .wav files or randomly choosing between them. In theory, the performance of the DOA estimator should not be largely affected by the content of the speech. Nevertheless, we utilized the second option (random selection) in this work. We preprocessed the speech files to remove the silence from them. Therefore, for the datasets files used this work, there is always an active acoustic source.

Each speaker/source with signal $x(t)$ can be static or moving. As illustrated in Figure 3.2, in case of moving sources, we define the angular velocity ω (in rad/s) and the movement trajectory in a configuration file. We assume that each speaker/source is moving at a constant ω angular speed on an arc-like path $r(t)$. The path is defined by the constant radial distance ρ (in meters) between the source and the center of the microphone array, the starting DOA φ_s (in rad) with respect to the microphone array, and the end DOA φ_e (in rad). Once the source position is calculated at every time step in the spherical coordinates, we convert it to Cartesian coordinates as required by the RIR Generator. Note that for a given value of ω, it is possible that a source goes through the whole path in a time shorter than the desired length T. In this case, we support two options: either to keep going back-and-forth along the same path until the desired length is achieved, or to keep selecting new end angles φ_e randomly and keep moving. The first option has the advantage of producing trajectories that contains no DOA discontinuities, while the second option produce more diverse cases (especially when multi-source scenarios are considered). It is important to note that the data collection process is carried out as the source moves, and a constant discretization is performed over time rather than over the source path.

For each source a multi-channel RIR $h(t)$ was generated using the image method [29]. The image method models the propagation of a source in a rectangular, rigid-wall room of dimension D meters. For simplicity, in this section D is assumed to be constant for all the room dimensions (x,y,z). The resulting impulse responses $h(t)$, as given in (3.1), simulate the room reverberation when convolved with any source signal $x(t)$. This is done by modeling all the possible paths that a sound can follow. This is achieved by replacing each reflected path with a virtual source on a line-of-sight with the receiver where (i) its direction is in the direction of the reflection, and (ii) its position has to produce the same level of attenuation. For a given room, each impulse response in a multi-channel RIR depends on the position $r(t)$ of a source and the position of each microphone $\hat{r}_m$. Allowing the source positions to move results in a time-varying RIR expressed as [29]:

$$h(t;r(t),\hat{r}_m)=\sum_{p\in P}\sum_{q\in Q}\left(\beta_{x1}^{|q_x-p_x|}\beta_{x2}^{|q_x|}\beta_{y1}^{|q_y-p_y|}\beta_{y2}^{|q_y|}\beta_{z1}^{|q_z-p_z|}\beta_{z2}^{|q_z|}\frac{\delta\left(t-\tau_{pq}\right)}{4\pi d_{pq}}\right), \tag{3.1}$$

where β is the reflection coefficient of each wall, d_{pq} is the distance between a source image and a microphone, t is the time, τ_{pq} is the time delay of arrival of a reflected unit impulse for a source image $\delta(t)$, P is the set of all (eight) binary triples (p_x, p_y, p_z) (i.e., if any $p_i=1$, then image of the source in that direction i is considered), and Q is a set of all desired triples (q_x, q_y, q_z) with $q_i\in\left[-\left\lfloor\frac{N\,c}{2D\,f_s}\right\rfloor:1:\left\lfloor\frac{N\,c}{2D\,f_s}\right\rfloor\right]$, where c is the speed of sound and N is the length of the RIR impulse response in samples. Note that for a given N, this algorithm computes $8(N/D+1)^3$ different paths [29].

To simulate a scenario with multiple moving-sources, the generated output signal $s_m(t)$ as measured by a microphone m positioned at $\hat{r}_m$ is computed by convolving each anechoic source signal $x_i(t)$ moving along an arc-like path with the corresponding $h(t,r_i(t),\hat{r}_m)$ as follows:

$$s_m(t)=\sum_i x_i(t)*h(t;r_i(t),\hat{r}_m)\quad m=1,\ldots,M \tag{3.2}$$

where M is the number of microphones in a microphone array and "*" represents a linear convolution sum. It is worth noting that while the simulator [25] supports microphones with

different beam-patterns, we decided to work with omnidirectional sensors, for generality and simplicity.

Note that in our work, we combine two sources. However, it's worth noting that the total number of sources that could be detected is constrained by the angular resolution that can be achieved, which in turn depends on the array aperture (i.e., array length, maximum distance between the most distant microphones).

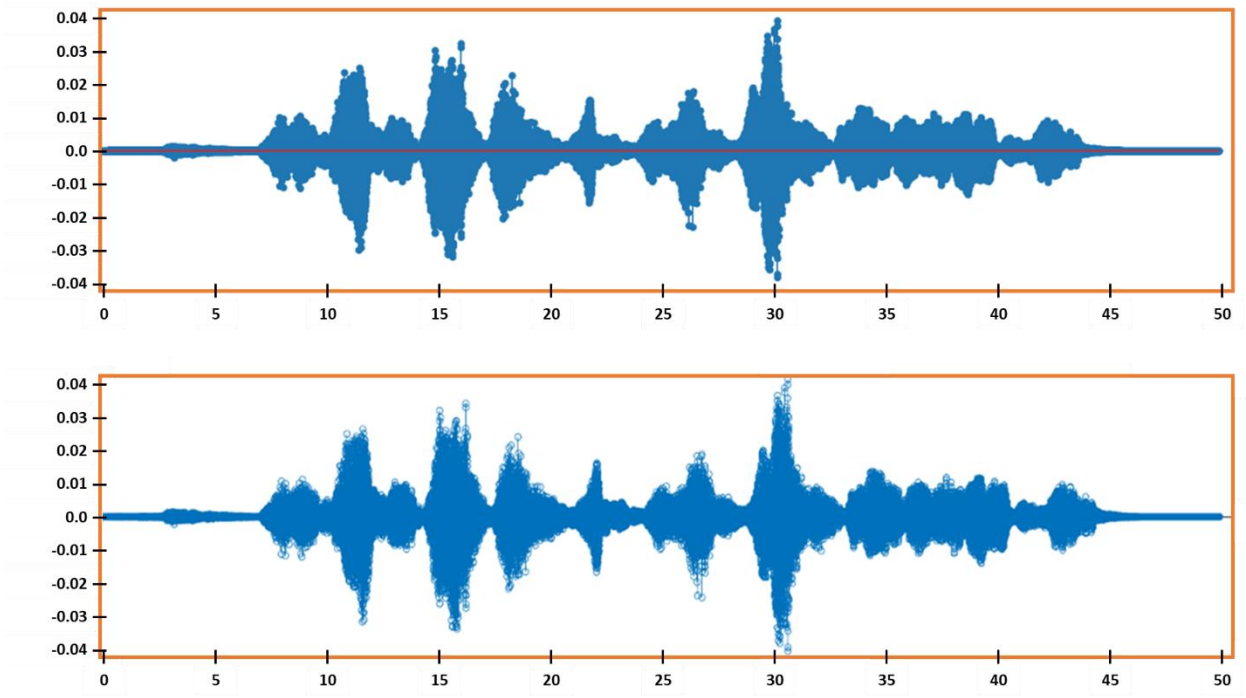


Figure 3.3 A simulated directional signal output (top) in Python vs [15]output (bottom) in MATLAB.

We validated our implementation against [27] by comparing the resulting directional signal, which matched to seven digits. Figure 3.3 shows an example of the simulated output. Furthermore, we tested our signal on a traditional DOA detection algorithm that is known to work for noise-free and reverberation-free conditions. In particular, we implemented the cross power spectrum phase method [2] and tested the implemented algorithm on different angles from 0 to 180 degrees. Since the algorithm assumes that sources are static, we tested our implementation of the signal generator on non-moving sources. The detection error was less than one degree, which indicates that our directional signals were correctly simulated. The results are shown in Table 3.1.

Note that in practice, a 1-degree error is often considered good enough for practical purposes. One reason for this is that the resolution of the human auditory system for localizing sound sources is

coarser than this. Therefore, a 1-degree error is well within the range of what humans can perceive. Additionally, subsequent processing steps such as beamforming or source separation may have their own limitations on resolution, and may not benefit from higher resolution DOA estimates.

Table 3.1 The generated DOAs by signal generator vs. the estimated DOA by the GCC-PHAT algorithm (in degrees).

Generated DOA by signal generator	0	15	30	45	60	75	90	105	120	135	150	165	180
DOA-estimate by algorithm	0.4	14.1	30.3	45.3	59.7	75.4	90.0	104.6	120.3	134.7	149.7	165.8	180.9

3.1.3 Noise Generator

In order to evaluate the robustness against acoustic noise when performing DOA detection with moving sources, a multichannel spatial noise $v(t)$ is added to the generated directional sounds $s_m(t)$ at a desired SNR level (in dB). We call this signal $y_m(t)$. Both the directional sound and the noise are normalized by the RMS value of a common reference sensor instead of normalizing by the maximum of the two sensors, which can cause the reference signal to differ from case to case. Note that we normalize the signals by the reference signal RMS value, and not by the reference power, because the noise signal is later multiplied by a gain or RMS value. The noise signal is also normalized by its RMS value before applying the gain. We chose to work with cylindrically isotropic noise, corresponding to an acoustic scenario where the ceiling and the floor of a room are considered to be covered with an absorbing material. For computational efficiency, the noise generator from [29] creates the signals directly in the frequency domain and then transforms them back to the time-domain. The process combines two ideas: first, the reference microphone receives a noise signal $v_o(t)$ composed of a superposition of 64 uncorrelated waves uniformly distributed on the surface of a cylinder. Second, the noise components received by the other microphones $v_m(t)$ are calculated from $v_o(t)$ in a way that preserves the correct spatial coherence (which is given by a Bessel function in this case of cylindrically isotropic noise). The labels (DOA directions φ) for each time step are based on the true path taken by each speaker. We selected the RIR update period (time step) as $\Delta t = 32/f_s$ seconds, as in [27].

To validate the correctness of our implementation, we compare the spatial coherence of the generated sensor signals with the theoretical spatial coherence. The difference between the two is the result of the spectrum estimation error and the fact that only a finite number of noise sources is used. Figure 3.4 suggests that the number of noise sources that is required, when the microphone separation is 5 cm, should be larger than 20 in order to achieve an acceptable level of the normalized mean square error (NMSE). Note that we chose a 5 cm separation, as this is the distance that we have set between the two microphones in our simulation. However, the number and geometry of sensors used in DOA estimation is not directly related to the number of acoustic sources. Figure 3.5 shows the coherence between two sensors in a cylindrically isotropic noise field.

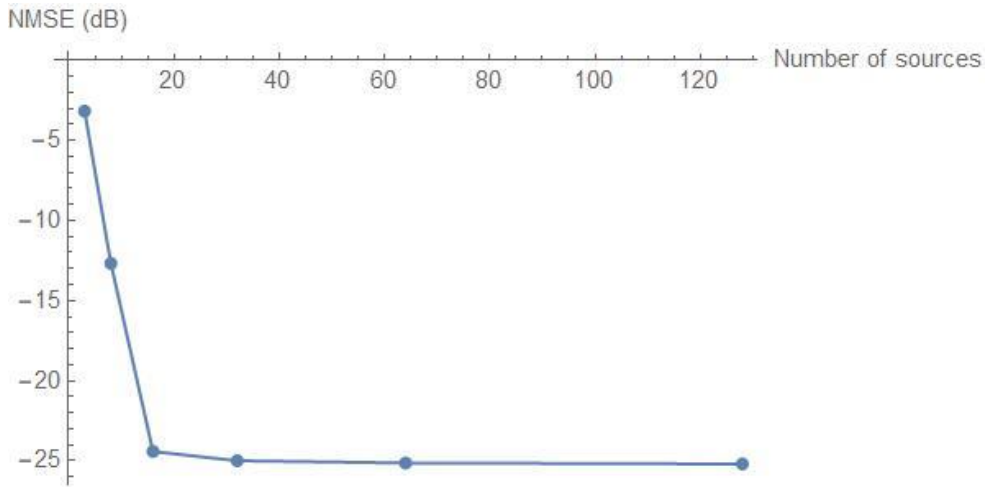


Figure 3.4 Normalized Mean Square Error between the theoretical and spatial coherence of the generated signal of a microphone array with two elements and 5 cm separation.

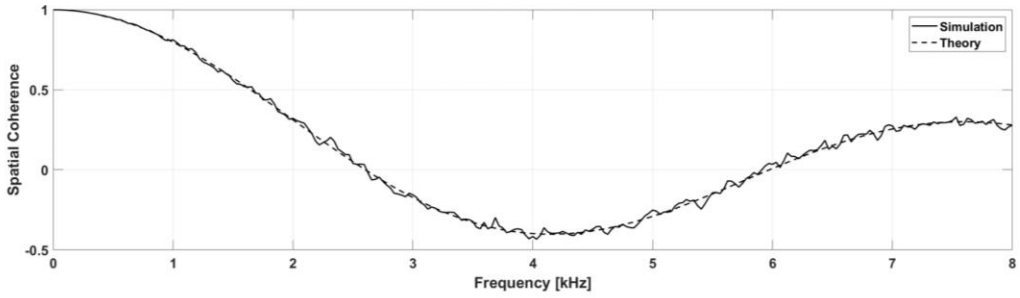


Figure 3.5 Theoretical and simulated spatial coherence between two sensors in a cylindrically isotropic noise field.

3.1.4 STFT

Although it is possible to train the deep learning model on raw signals, it is more efficient to find a feature representation that better summarizes the data. While the use of other transforms could certainly be of interest, in this work we chose to use a common window size to compute STFT components at all frequencies, leading to a constant physical frequency resolution across frequencies, as opposed to having variable window sizes and variable resulting frequency

resolutions across frequencies (as in wavelet transforms or non-uniform filter banks). This was chosen because of two reasons: (1) the use of STFT has been fairly widespread in recent years for NN-based DOA estimation of static acoustic sources, and to simplify comparison we wanted to evaluate the performance of the same features under dynamic/moving sources, and (2) in free field, phase differences between STFTs of different sensors are known to be directly related to the time difference of arrival (TDOA), which is well-known criteria for azimuth angle of arrival estimation. Features in audio classification are very often based on the short-time magnitude and phase spectrum of the signal because the signal's frequency contents change over time. Therefore, the appropriate way is to analyze them in the time-frequency domain. The STFT characterizes the time-frequency distributions by segmenting a non-stationary signal into many frames. However, using the STFT as feature extractor for NN has a few disadvantages. For example, it introduces more hyperparameters that require careful tuning. This includes the number of frequency bins K to be used, the windows size L (in seconds), and the amount of overlap H between consecutive windows (in seconds). One of the main motivations for this work is to systematically study the effect of these parameters on NN performance in order to help NN practitioners design better datasets.

It is worth noting that the extracted features generated by the STFT are signal-level dependent. This can be more challenging for the NN given that the DOA estimation should not depend on the signal level. Therefore, one of the best practices is to normalize the magnitude of the STFT features measured at each microphone relative to one reference microphone. Using such magnitude ratios provides a set of features that are independent of signal level.

In addition to the magnitude ratios, the STFT features include phase component that is more useful for the DOA detection. In particular, the phase difference between each microphone and the reference microphone is then more directly related to the time difference of arrival (TDOA), which is an indicator for the DOA of the received signals. It should be noted that each TDOA or detected DOA corresponds to a single source. If there are multiple sources, there will be multiple distinct TDOAs and DOAs. The ground-truth labels for each STFT frame are calculated from the sample-by-sample DOA information along the trajectory travelled by the speakers. This DOA vector is processed using a 1-D convolution with a kernel size equal to the STFT window length, and a stride corresponding to the overlap factor. Note that the nnAudio library also uses a 1-D convolution operator to calculate the STFT. This makes it easier to align the generated frame labels

with the corresponding feature frames. Once the DOA is generated for each frame, we quantize them to 5 degrees to produce the NN classes and then perform a one-hot encoding step to get the final labels. This last step is the standard way of representing the ground-truth for a multi-class classification. One can interpret the output of this one-hot encoded vector as the probability of detecting a speaker at each DOA. Since the sum of probabilities across different DOA classes should add to one, the NN select the DOA which has the highest probability to represent the true direction of the active speaker.

An extension to the multiclass classification called multi-label classification is considered in this work. This formulation is meant to handle scenarios where more than one active speaker can be active at the same time (as well as the case where no speaker is active). Since more than one class can be assigned the label 1, the total sum of DOA labels does not add to one and therefore can't be interpreted as probabilities. The solution to this issue is to decouple the classes from each other and consider each DOA class as a standalone classification problem. Then, we assign a label to each DOA class representing the number of active speakers expected at this DOA. For example, assume we have three DOAs [0,5,10] degrees. A label vector [1,1,0] means there is one active speaker at 0-degree, one active speaker at 5-degree, and no active speaker at 10-degree. Note that in our work, we limit the maximum number of speakers at each DOA to 1 (i.e., if two speakers are taking at the same DOA, we consider them as one speaker). This simplifies the multi-class multi-label classification problem to a multiple binary-classification problems where each binary classifier is associated with one DOA classe, and the [0,1] represents the two possible labels for each DOA (i.e., [no speaker, active speaker]).

3.2 Neural Network Models

The deep learning neural models were carefully chosen according to their time series modeling capabilities. In particular, feedforward neural networks (FNN) are memoryless and were used as a baseline. Recurrent architectures memorize past events via a feedback mechanism. Two flavors were considered: a vanilla recurrent neural network (RNN) which is good at modeling short-term relations, and a gated recurrent unit (GRU) which also possesses a long-term memory. Finally, temporal convolutional networks (TCN) avoid a feedback mechanism yet possess a long-term memory via an encoder-decoder architecture.

We chose the four models with a comparable network size of approximately 785k parameters (weights, biases). The number of epochs for training was set to be 25. The FNN model consists of four layers, while all the other models consist of two layers. These parameters were adjusted by trial and error.

3.3 Implementation

In this section, we explain our software contribution and design choices.

3.3.1 Software Contribution

In this subsection, we explain our software contribution, and provide a detailed discussion of the software choices we have made.

3.3.1.1 Hyper-parameters tuning

As mentioned earlier, a typical neural network training process involves two key steps: (1) defining the right model architecture, and (2) tuning the different hyper-parameters to achieve best performance. Finding appropriate hyperparameters for those two parts, in order to achieve satisfactory performance, is a challenging task. One reason is that most of the commonly used hyperparameter optimization methods are general purpose and not optimized with domain-specific knowledge to improve their efficiencies in the SSL domain. For example, in the grid search algorithm, the entire parameter space is divided into a grid of points, then each of the points in the grid is evaluated as hyper-parameter. Although the method is simple and easy to implement, it can only be used in the case of low dimensional hyperparameters space, as it is time consuming. Random search is another commonly used approach in which the hyper-parameters are selected randomly independent of other choices. Compared to grid search, this method converges faster. However, it might fall into the problem of making many similar observations that provide redundant information, or never sample parameters close to the best choice. Another challenge is that most available tools and libraries for hyper-parameter optimization suffer from many limitations. For example, the search-space is usually implemented for simple cases, such as a range of continuous values or a discrete set of categorical choices. Configuring a more complicated search space can be tricky and difficult to do. Furthermore, these “Auto-ML” tools are usually based on one-shot search algorithms and don't support human-in-the-loop search space

modification. This makes the algorithm dependent on the quality of the objective function that drives the optimization process and doesn't allow researchers to easily do an interactive exploration of the search space.

Our goal is not limited to finding a good model and a good combination of parameters, but also to better understand the strength and weaknesses of the system and the contribution of each parameter in system performance. To achieve this, we implement an efficient mechanism to support human-in-the-loop interactions for complicated search criteria such as some cases of conditional hyperparameters. Conditional hyperparameters appear when one or more hyperparameters only make sense based on the value of some other parameter. These conditions are often simplified and implemented as a two-level hierarchical space, such that conditional hyperparameters are selected in a level below the hyperparameters they depend on. However, restricting two hyperparameters from simultaneously taking some pairs of values can't be easily defined in a hierarchical form without introducing an arbitrary order between the hyperparameters. We designed our tool such that it is easy for the user to decide on how to define this rule.

To investigate the performance of the hyperparameters for the SSL problem, we followed the principle of exploration and exploitation method. We used domain-experience to define a good starting point. We changed one parameter at a time to study its effect and set the other parameters to a fixed value defined previously. Occasionally, we used random searches on a larger space to explore other parameters, changing multiple values at the same time. If a potential case is found, we interactively refine the search space.

3.3.1.2 A unified framework for DOA-estimation

The process of training NN models is usually challenging, from a software engineering perspective, because it involves lots of book-keeping work. For example, it is important to track the different artifacts such as dataset versions, trained models (possibly at different epochs), hyper-parameters, and performance scores. Furthermore, it is important to track all code used in the process including any simulator used (if synthetic dataset was generated), any pre-/post-processing, any training/tuning/pruning algorithms used to optimize the model, and all random states to ensure reproducibility.

Systematically studying the effect of different parameters for DOA-estimation of moving sources under a wide range of simulated acoustic conditions is even harder because it involves working with multiple components that are often implemented in different languages and requires lots of engineering work to integrate them together. For example, Figure 3.6 shows a set of popular tools needed to build the complete pipeline. As the figure shows, the pipeline involves integrating five libraries: RIR-Generator [27] for simulating the room impulse response, Noise Generator [29] for simulating background noise, NN-Audio [39] for calculating STFT, PyTorch-lightning [42] for training NN-model, and TorchMetrics [43] for performance evaluation. These tools are wrapped together and integrated with our custom hyper-parameter tuning tool, making it easy to mix-and-match different acoustic scenarios, feature extraction parameters, and NN model parameters.

Figure 3.6 Tools Pipeline.

3.3.2 Design Choices

In this subsection, we discuss the justifications for our software design choices. In particular, we highlight the advantages of using the libraries mentioned in section 3.3.1 over other alternatives.

The first component in our design is the RIR generator. We considered two options to generate directional audio signals. On one hand, it is possible to rely on available datasets of RIR for a wide range of microphone array configurations. This option has the advantage of being measured in real environment. However, it lacks flexibility when it comes to the configurations that can be specified. Therefore, we chose to build a simulator that can generate synthetic RIRs. Different algorithms have been proposed to approximate the RIR. We chose the image method described in Section 2.1.2, because it is the most commonly used algorithm.

Among the available implementations of the image method algorithm, we considered three implementations: RIR-Generator [28], gpuRIR [44], and FRA-RIR [45]. The first implementation is the most widely used. The second library has the advantage of using GPU but had many issues when running with PyTorch (which is our framework of choice for deep learning in this work).

The last library replaces the physical simulation in the standard image method by a series of random approximations, which significantly speeds up the simulation process, without the need for GPU, and enables its application in on-the-fly data generation pipelines. This is a very desired property, however, we decided to work with the first library because it is more accurate and a real-time implementation was not part of our objectives.

The downside of using the python implementation of the RIR-Generator is that it is not efficient when used to simulate moving sources. This is because simulating moving sources requires calculating many RIRs, since each RIR is only valid for few samples. Therefore, we switched to a MATLAB implementation for this step, which combines the RIR-generator and Signal Generator in one step, resulting in 70x speed improvement. For static sources, we used the python implementation because it better integrates with the whole pipeline, allowing for more automated tests. Note that we wrapped the Signal Generator provided by [28] to make it easier to use for rapid prototyping. Among the changes are: (1) switching from cartesian to polar coordinates because the latter is directly related to the DOA which is used as labels during NN training, (2) provide utility functions to define a variety of trajectories for multiple sources moving at different speeds in different directions along line-/curve-like paths, and (3) simulate moving sources a in noisy environment at a specific SNR. Note that our code supports two ways to specify the speed of moving sources: an absolute speed (measure as rad./sec.) or normalized speed (measured as rad./window). The latter option is useful to understand the effect of window size of the STFT operating at different speeds.

When it comes to calculating the STFT, we considered four implementations: Scipy [46], TorchAudio [47], nnAudio [40], and Tensorflow [48]. We excluded the first option because it doesn't support GPU. Also, we decided to use PyTorch for our DL research. Therefore, we excluded the Tensorflow implementation. nnAudio provided more flexibility when it comes to the different supported options, as well as the way that it can be used. In particular, with nnAudio, it is possible to attach the STFT to the dataset pre-processing step (similar to all other libraries). It can also be attached as a trainable NN-layer as part of the NN model, allowing for an end-to-end ML pipeline where the gradients for STFT kernels are calculated and the STFT kernels are updated during model training. Another useful feature is supporting the use of a logarithmic scale when determining the spacing between each frequency bin. This flexibility motivated us to use nnAudio over the standard implementation in PyTorch library.

For NN training, we decided to use PyTorch-Lightning [42] over the popular Keras alternative for many reasons. PyTorch-Lightning is a high-level python framework built on top of PyTorch with the intention to scale and speed up the research process by eliminating low-level code while keeping the ML pipeline readable, logical, and easy to execute. Unlike the Keras library, PyTorch-Lightning fits our purposes better because it does not hide away details of network and optimization design but rather provides a standardized way to organize native PyTorch code. Furthermore, the fact that PyTroch-Lightning is using PyTorch makes it easier to develop and debug, thanks to PyTorch dynamic computational graph generation. While TensorFlow 2.0 has an eager execution mode that makes the static graph generation similar to dynamic computations, PyTorch is more popular choice in academia because it resembles more the "pythonic" way of writing software.

Regarding model evaluation, we used TorchMetrics [43] because it is well integrated with the PyTorch-Lightning framework. However, computing the evaluation metrics for multi-class multi-label classification problems is not straightforward using this library. Therefore, we provided a wrapper to simplify this process.

Chapter 4 DOA Estimation using Feedforward Neural Networks Trained with Speech Audio Signals

In this chapter we investigate the effect of changing the STFT parameters, the room reverberation time (RT), and the signal to noise ratio (SNR), on both static sources and moving sources at different angular speeds. All tests were done on a multi-class multi-label DOA detection setup and on FNN model with simple datasets.

4.1 Experiment Setup

In our experimental setup, we consider a room of size $(L_x, L_y, L_z) =$ (5m, 4m, 6m), with equal reflectivity coefficient (β) from all walls, ceiling, and floor. This value of β is calculated from the RT60 reverberation time as in [27]. A microphone array with $M = 2$ microphones separated by d = 0.05m is placed at different positions in the room: at the center, at edges or at a corner as shown in Figure.4.1. The vertical position of the microphone array is chosen to be 1.5m.

We allowed two active sources to move for 50 s at a radial distance 0.5 m from the center of the microphone array at different angular speeds. The speech source was generated by concatenating around 180 random speech files from the TIMIT database. All pauses or silence segments were removed, and all files were sampled at 16 kHz. The power of the human voice is concentrated below 4kHz, and the phase difference information used for acoustic DOA estimation is mostly reliable below 5kHz. Therefore, a sampling frequency of 16 kHz is considered sufficient to discretize the directional audio and get accurate estimation of amplitude and phase. The resulting signal is then divided into files of length 50 s each and used to generate different scenarios. Regarding the RIR parameters, we defined a filter length 1024 samples and set the reflection order to -1 (i.e., the maximum order supported in [27] will then be used). The outputs of the sample-based stage have the following shape: [2 channels (microphones), $T \times f_s = 50 \times 16000 = 80000$ samples].

For the frame-based stage, L frames were generated based on the window size $W \times f_s$ (in samples), the padding $P \times f_s = W/2 \times f_s$ (in samples), and the hop length $H \times f_s$ (in samples), as shown in (4.1):

$$L = \left\lfloor \frac{T \cdot f_s + 2 \cdot P \cdot f_s - (W \cdot f_s - 1) - 1}{H \cdot f_s} + 1 \right\rfloor \tag{4.1}$$

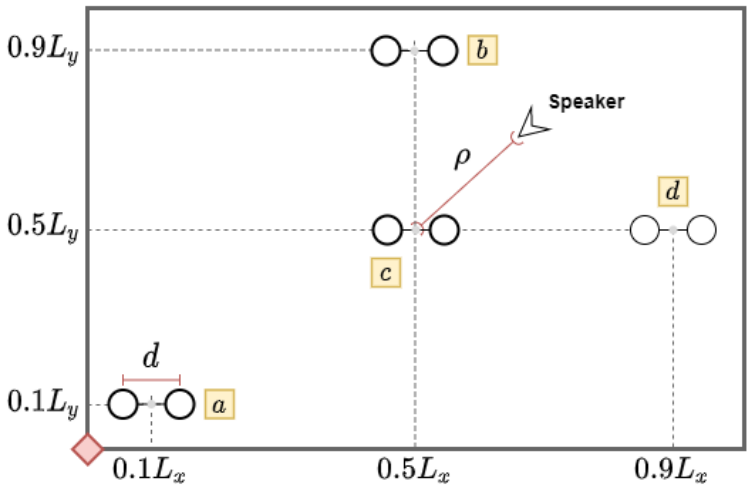


Figure 4.1 Microphone-array position in the room.

We kept the resulting STFT output frequencies from 50 Hz to 6000 Hz and divided them into K frequency bins distributed uniformly. Since the STFT outputs produce a complex ratio between the frequency content of the two microphones, we squeezed the first dimension and reshaped the features as follows: $[L, 2KL]$, where 2 represents the magnitude ratios and phase differences. We discretized the DOA range [0, 180] degrees into 13 classes with 15° resolution. We varied the SNR level, reverberation time, angular speed, number of frequency bins, and hop length (window shift) as given in Table 4.1. As a result, for each case under investigation, we used two different datasets: one for static sources and one for moving sources. Since static sources contain only one class for each file, a larger dataset is needed to cover all classes. The moving sources contain all classes in each file (since the moving sources travel along an arc). The number of features samples used for the moving cases is 135,135 frames, while 1,230,710 frames were used in static conditions. The moving sources dataset combines two sources, with one source moving clockwise and its reverse which is moving counterclockwise. The static sources files combine two sources speaking at two different DOAs (at least 60° apart). All DOAs in the moving sources dataset were quantized to generate the labels. This has the advantage of producing a data augmentation effect compared to the static dataset.

Table 4.1 Experimental conditions.

Parameter	**Simulated data**
Sampling rate	16k Hz
Speech signal	Random segment from TIMIT
Room size	(5,4,6) m
Number of microphones	2
Considered classes (DOAs)	[0,180] with 15-degree increment
Array positions in the room	3 different positions: [center, edge, corner] of the room.
Noise field	Cylindrically isotropic (2D diffuse)
SNR	[∞,15,10, 5,0] dB
Source array distance	0.5 m
Distance between mics	0.05 m
RT60	[0, 0.2, 0.4, 0.6, 0.8] s
Number of frequency bins (over 50-6000 Hz range)	[10 - 20]
Window size	[0.1] s
Hop length	[25, 50, 75] %
speed (ω)	[0 – 75] degrees/s
Length per feature file	50 s
Number of training data	135,135 frames for moving dataset 1,230,710 frames for static dataset
Learning rate	0.005
Batch size	512
Optimization method	Adam

4.2 Evaluation Metrics

We chose a self-normalized FNN model, shown in Figure 4.2, consisting of 4 layers with 500 neurons.The inputs (shown as circles) are flattened along the time axis if a sequence of inputs is provided. We tested the effect of changing different hyperparameters of room characteristics (reverbration time, SNR and angular speed) and some STFT parameters (hop length and frequency bins). The performance was evaluated in terms of precision and recall. Precision measures the ratio of positive predictions made which are correct, whereas recall measures the ratio of the positive cases in the data which are correctly detected. Also, the multi-label confusion matrix was calculated under a one-vs-rest approach. As a results, 13 confusion matrices were produced for each confusion between every two classes.

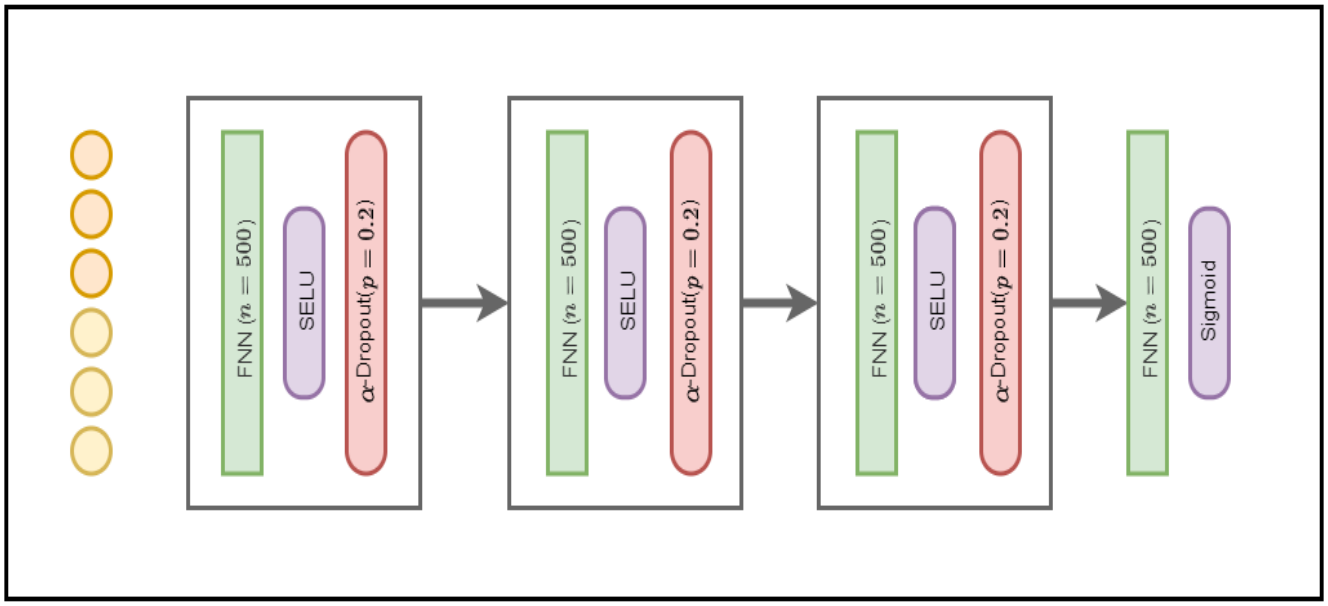


Figure 4.2 An overview of the proposed FNN architecture, where circles represent the input features.

4.3 Analysis

4.3.1 STFT Parameters

Table 4.2 and Table 4.3 present the effect of changing the STFT parameters (frequency bins and hop length) in noise-free scenarios with a reverberation RT = 0.2 s.

Since for FNN networks the DOA angle predictions are performed as a memoryless system (no trajectory tracking performed), the hop length is not expected to be a factor affecting the metrics, but including it allows for a sanity check. We consider different possible values [10,15, 18, 20] for the number of frequency bins K in Table 4.2, with a constant window size of 0.1 s and a hop (shift) length of 0.05 s (i.e., 50% overlap). In Table 4.3 we consider different hop lengths [0.025s 0.05s, 0.075s], with a constant value of $K = 15$ frequency bins.

In Table 4.2 and Table 4.3, when testing acoustic scenarios with moving sources, we can very clearly see the beneficial effect of including acoustic scenarios with moving sources in the training dataset. Indeed, when the training dataset only includes static sources and the testing dataset includes moving sources, the recall performance drops dramatically.

We observe in Table 4.2 that the effect of the number of frequency bins K in static sources is not significant. However, the effect for high-speed sources is more significant. Note that the column "# of detected classes" shows the number of classes that the neural network successfully identified out of all the classes. For example, when the number of frequency bins are lower (e.g., [10, 15]), the FNN is not able to detect some DOAs in the lateral regions (e.g., DOAs 0, 165, 180 degrees) as shown in Figure 4.4. However, going to higher number of frequency bins (18, 20), the precision increased by 0.08. We also observe in Table 4.3 that the effect of changing the hop length was not significant, as expected.

Table 4.2 Effect of changing number of frequency bins and training data (with ω =75 degrees/sec.)

Testing data	Training data	Freq. bins	Avg. precision	Avg. recall	# of detected classes
static	static	10	0.92	0.91	13
static	static	15	0.93	0.91	13
static	static	18	0.92	0.91	13
static	static	20	0.94	0.91	13
moving	static	10	0.80	0.50	13
moving	static	15	0.80	0.53	13
moving	static	18	0.80	0.52	13
moving	static	20	0.80	0.53	13
moving	moving	10	0.81	0.75	11
moving	moving	15	0.82	0.78	11
moving	moving	18	0.89	0.87	12
moving	moving	20	0.90	0.87	12

Table 4.3 Effect of changing hop length in noise-free room with RT=0.2 s and different angular speeds.

speed ω [deg/s]	Hop length [%]	Avg. precision	Avg. recall
5	25	0.98	0.98
5	50	0.99	0.98
5	75	0.98	0.97
10	25	0.99	0.95
10	50	0.99	0.97
10	75	0.98	0.96
15	25	0.98	0.94
15	50	0.98	0.94
15	75	0.97	0.94
30	25	0.98	0.95
30	50	0.98	0.94
30	75	0.98	0.96
50	25	0.96	0.95
50	50	0.97	0.95
50	75	0.96	0.94
75	25	0.97	0.93
75	50	0.97	0.95
75	75	0.94	0.90

4.3.2 Reverberation Time

The effect of varying the RT levels with $K = 15$ frequency bins, a window size of 0.1 s, and a hop length of 0.05 s under noise-free scenarios is shown in Table 4.4. We notice that the effect of changing RT is more obvious in the case of moving sources at high angular speed than it is for low-speed sources. For example, going from low RT (i.e., 0 and 0.2 s) to high RT (i.e., 0.6s and 0.8 s), the recall dropped by 0.12 and 0.07 for high and low angular speed scenarios, respectively. Also, we observe that the presence of some reverberation (e.g, RT = 0.2s) in the training dataset

can help in achieving better detection for the DOA. Figures 4.5 and 4.6 show an example of the performance when RT=0.2, 0.8s respectively at angular speed $\overline{\omega}$ =7.5 degree/window.

Table 4.4 Effect of changing RT at different angular speeds.

speed ω [deg/s]	RT in training and testing [s]	Avg. precision	Avg. recall
5	0.0	0.98	0.97
5	0.2	0.99	0.98
5	0.4	0.97	0.97
5	0.6	0.96	0.95
5	0.8	0.94	0.90
10	0.0	0.98	0.97
10	0.2	0.97	0.98
10	0.4	0.96	0.96
10	0.6	0.95	0.95
10	0.8	0.93	0.89
15	0.0	0.97	0.95
15	0.2	0.97	0.97
15	0.4	0.96	0.93
15	0.6	0.94	0.90
15	0.8	0.92	0.87
30	0.0	0.96	0.94
30	0.2	0.98	0.94
30	0.4	0.95	0.91
30	0.6	0.94	0.87
30	0.8	0.91	0.86
50	0.0	0.96	0.95
50	0.2	0.95	0.94
50	0.4	0.94	0.91
50	0.6	0.93	0.88
50	0.8	0.90	0.86
75	0.0	0.95	0.89
75	0.2	0.95	0.93
75	0.4	0.93	0.91
75	0.6	0.91	0.85
75	0.8	0.90	0.81

4.3.3 SNR and Normalized Angular Velocity

Figure 4.3 presents the effect of changing the noise levels (SNR), given the following parameters: $K = 15$ frequency bins, hop length of 0.05 s and reverberation RT = 0.2 s. The training dataset consists of different SNR (∞, 15dB, 10dB, 5dB, and 0 dB) while the testing dataset is done on a single SNR for each test, as shown in the figure. From Figure 4.3, for a given SNR, we see that the precision and recall decay slowly at low angular speeds. However, when the sources angular speed is further increased, the performance drops sharply. On the other hand, for a given angular speed, we notice that decreasing the SNR greatly affects the performance. For example, at low angular speed, the precision dropped by 0.12 and the recall dropped by 0.25, while at high angular speed the recall dropped by 0.11 from its value for the noise-free case.

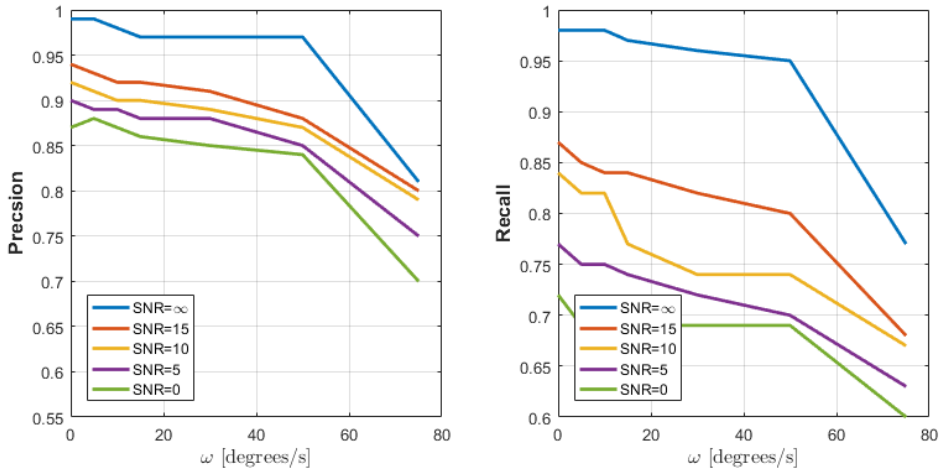


Figure 4.3 Effect of changing angular speed and SNR.

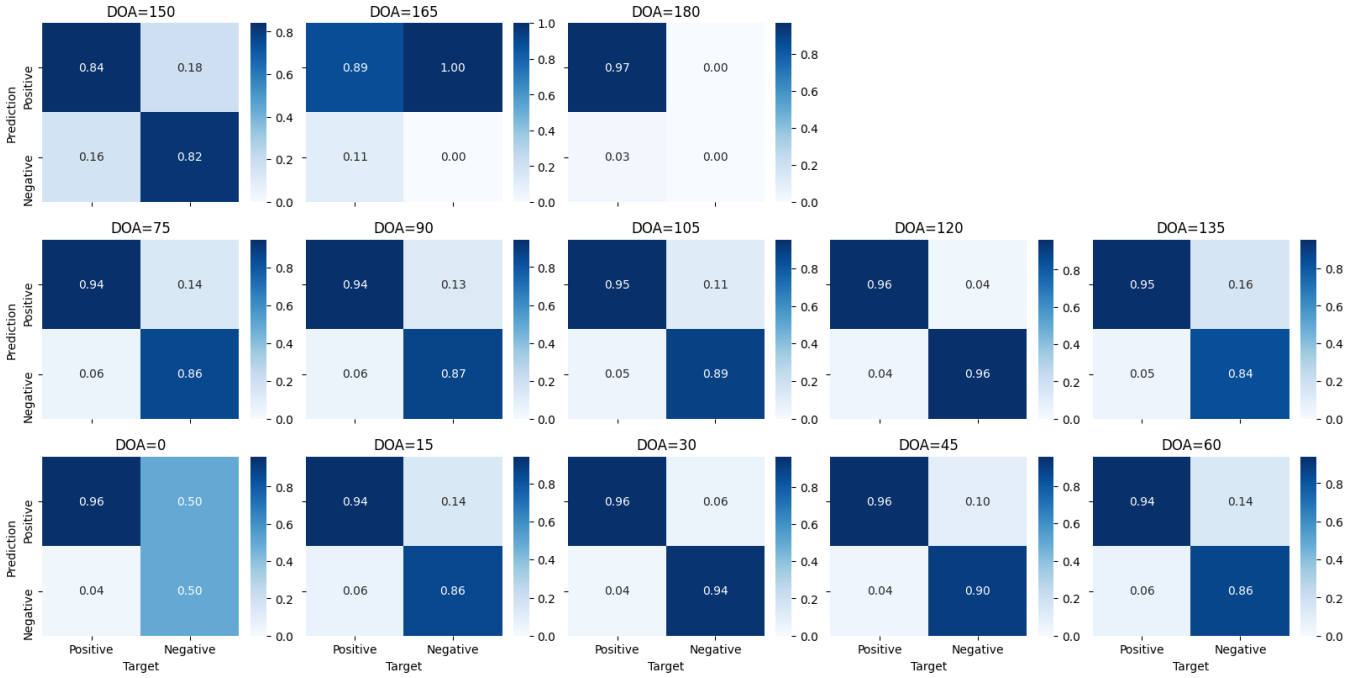


Figure 4.4 ***Performance of an FNN model when the number of frequency bins = 10 at w =75 degree/s.***

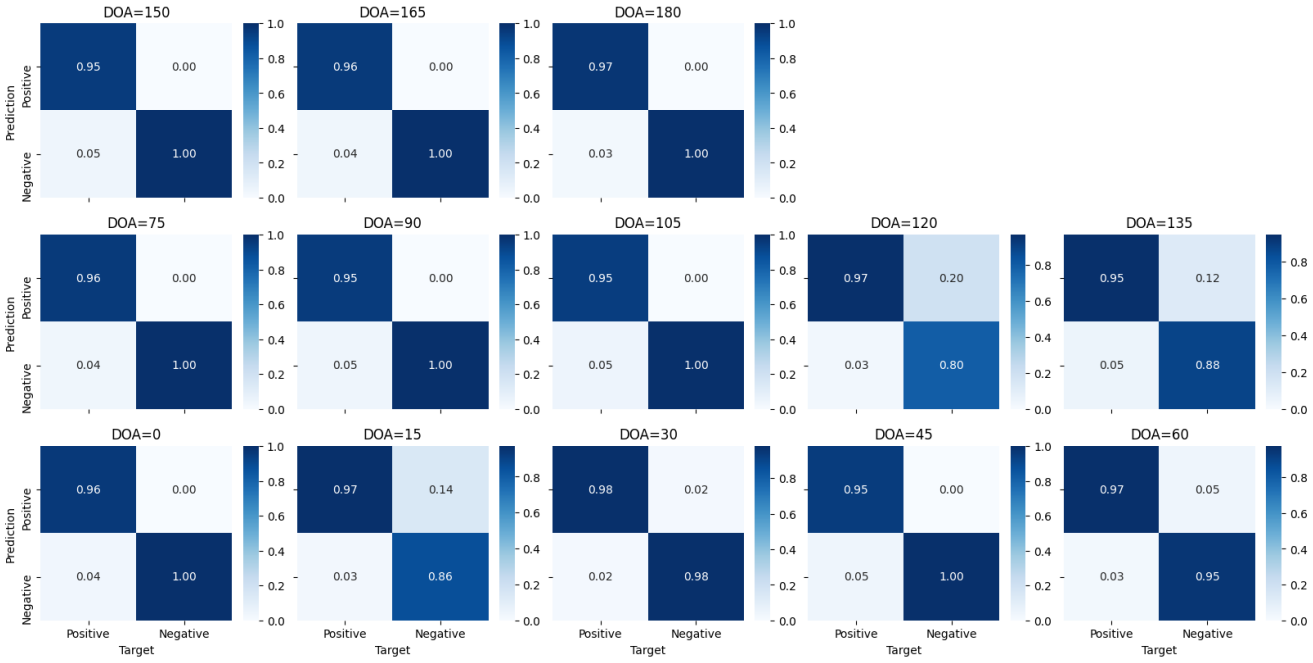


Figure 4.5 ***Performance of an FNN model when the RT = 0.2s at w =75 degrees/s.***

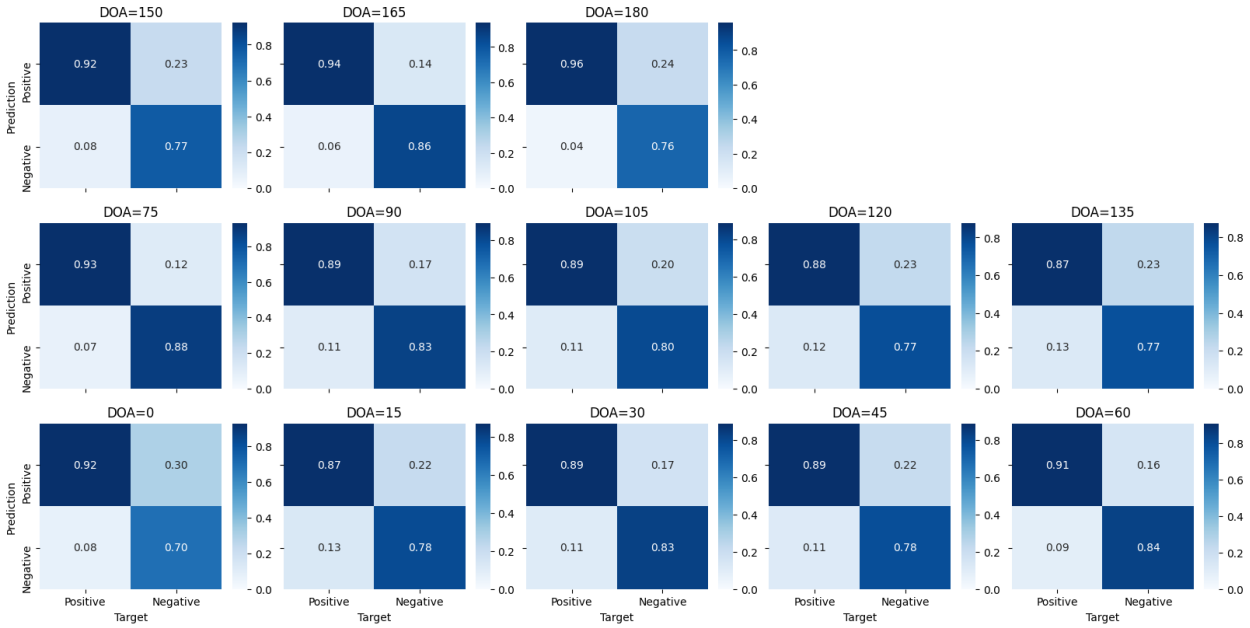


Figure 4.6 ***Performance of an FNN model when the RT = 0.8s at w =75 degree/s.***

Chapter 5 DOA Estimation using Different Deep Learning Models Trained with Noise

In this chapter we investigate the effect of changing the window size, sequence length, SNR and source angular speed. All tests were done on a multi-class multi-label DOA detection setup and using FNN, RNN, GRU and TCN models.

5.1 Experiment Setup

We consider a room of size $(L_x, L_y, L_z) =$ (5m, 4m, 6m), with equal reflectivity coefficient (β) from all walls, ceiling, and floor. This value of β is calculated from the RT60 reverberation time as in [27]. A microphone array with $M = 2$ microphones separated by $d = 0.05$ m is placed at the center of the room as shown in Figure 5.1. By avoiding a location close to a wall (which could lead to strong short echoes) and remaining far from the sources (which increases importance of late echoes/reverberation and reduces importance of direct path propagation), we reduce the effect of specific microphone locations on the results. The vertical position of the microphone array is chosen to be 1.5 m.

We allowed two active sources to move for $T = 50$ s at a radial distance $\rho = 0.5$ m from the center of the microphone array at different angular speeds. All the acoustic scenarios were generated with files sampled at $f_s = 16$ kHz, with a length of 50s for each scenario.

Regarding the RIR parameters, we defined a filter length $N = 1024$ samples and set the reflection order to -1 (i.e., the maximum order supported in [27] will then be used). The outputs of the sample-based stage have the following shape:

[2 channels (microphones), $T \times f_s = 50 \times 16000 = 80000$ samples].

For the frame-based stage, L frames were generated based on the window size $W \times f_s$ (in samples), the padding $P \times f_s = W/2 \times f_s$ (in samples), and the hop length $H \times f_s$ (in samples), as shown in [27].

$$L = \left\lfloor \frac{T \cdot f_s + 2 \cdot P \cdot f_s - (W \cdot f_s - 1) - 1}{H \cdot f_s} + 1 \right\rfloor \tag{5.1}$$

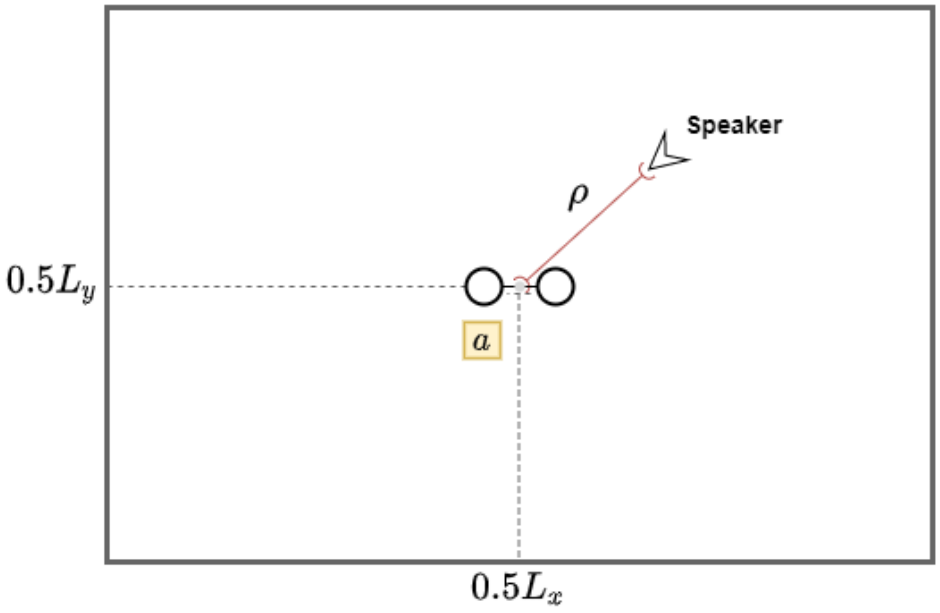


Figure 5.1 Microphone-array position in the room.

We kept the resulting STFT output frequencies from 50 Hz to 6000 Hz and divided them into K multiple bins distributed uniformly. Since the STFT outputs produce one microphone ratio, we squeezed the first dimension and reshaped the features as follows: $[L, 2KL]$, where 2 represents the magnitude ratios and phase differences.

In the previous chapter we discretized the DOA range [0, 180] degrees into 13 classes with 15° resolution and we studied the effect of changing the reverberation times (RT), number of frequency bins, and hop length, using a feedforward NN architecture. In this chapter, in addition to measure the impact of the key window size hyperparameter, we also consider recurrent and TCN neural network architectures, and we improve the angular azimuth resolution to 5°, which resulted in 37 DOA classes for the same range of [0, 180] degrees. We set the value of RT to 0.2s and vary the SNR level to include [5, 10, 15] dB as given in Table 5.1.

Note that for each case under investigation, we used different training datasets with increasing size as shown in Table 5.2. For angular speeds below 15 degrees/seconds, the initial position was chosen such that the distance between the sources in the multi-sources' scenarios are varied between [20-180]

Table 5.1 Experimental conditions.

Parameter	**Simulated data**
Sampling rate	16k Hz
Speech signal	Random segment from TIMIT
Room size	(5,4,6) m
Number of microphones	2
Considered classes (DOAs)	[0,180] with 5-degree increment
Array positions in the room	At the center of the room.
Noise field	Cylindrically isotropic (2D diffuse)
SNR	[15,10, 5]
Source array distance	0.5 m
Distance between mics	0.05 m
RT60	0.2s
Number of frequency bins (over 50-6000 Hz range)	15
Window size	[0.01 - 1] s
Hop length	50 %
Angular speed (ω)	[0 – 140] degrees/window
Length per feature file	50 s
Learning rate	0.0001
Batch size	512
Optimization method	Adam

Table 5.2 Number of files in each dataset.

Dataset id	Total # of files	Source 1 angular speed (degrees/window)	Source 2 angular speed (degrees/window)					
			0	**5**	**15**	**45**	**95**	**140**
0	2292	0	2292	0	0	0	0	0
1	2996	5	1740	1229	0	0	0	0
2	3630	15	1168	1280	1182	0	0	0
3	4844	45	1146	1264	1264	1170	0	0
4	5886	95	1146	1264	1264	1264	948	0
5	7372	140	1146	1264	1264	1264	1264	1170

degrees with 15 degrees increment. For higher angular speeds, the distance was selected from [20, 45, 60, 95, 120] degrees to limit the dataset size to include all speeds up to the desired value. The two sources were moving either in the same direction or in opposite direction of each other.

5.2 Evaluation Metrics

We chose a self-normalized FNN as well as RNN, GRU and TCN models, with a comparable network size of approximatly 785k parameters. The number of epochs for training was set to be 25. The FNN model consists of four layers. Both RNN and GRU networks consist of two layers with 500 and 300 neurons respectively, as shown in Figure 5.2. Both networks process the data sequentially in a many-to-one fashion, where only the output of the last step is used. The TCN network consists of two residual blocks, as shown in Figure 5.3, with a total of four dilated causal convolution layers each with 500 filters, a dilation factor $d = 1$, a filter size $k = 3$ and a stride $s = 1$. The input shape of the TCN is similar to the RNN. However, the time information is processed by the 2D convolutional layer.

All networks are optimized using the Adam optimizer [24] and are trained for 25 epochs on a single GPU (Nvidia RTX 3080 Ti). The dropout used in all models is a rate of 0.2. A ReLU activation function is used in RNN, GRU and TCN. However, in FNN architecture, the SELU activation function is used.

We tested the effect of the NN model type (architecture), with different window sizes, sequence lengths which refers to the number of previous frames that is provided as an input to the neural network, source angular speeds and SNR levels. The performance was evaluated in terms of precision and F-score. Precision measures the ratio of positive predictions made which are correct, whereas F-score combines the precision and recall of the model (where recall is the ratio of ground truth positive events which are correctly predicted by a model). F-score is defined as the harmonic mean of the model's precision and recall. Also, the multi-label confusion matrix was calculated under a one-vs-rest approach.

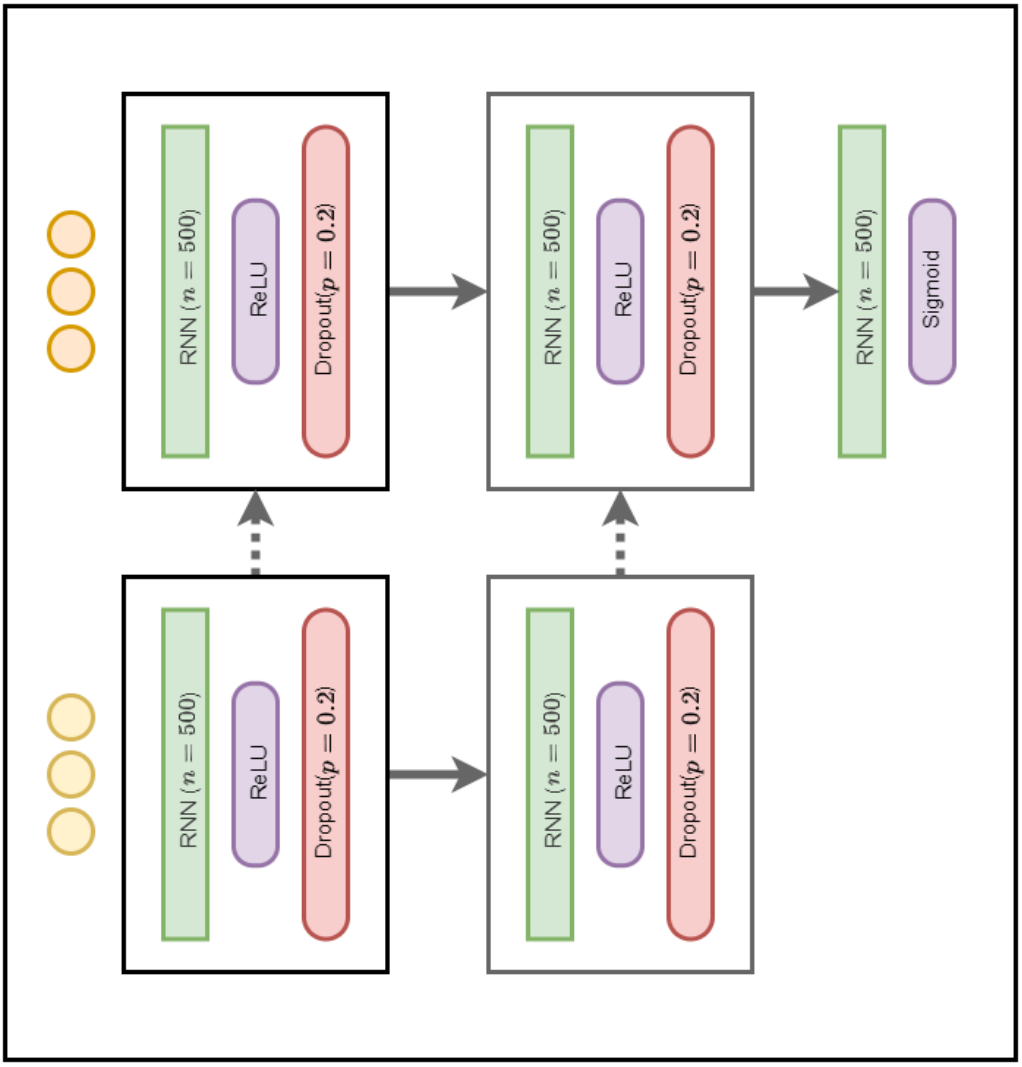


Figure 5.2 An overview of the proposed RNN architecture. Continuous lines depict connections between layers from input (left) to the output (right). Dashed lines depict times where the output of a previous step (bottom) is used in a next step (top).

.

5.3 Analysis

In this subsection, we investigate the effect of changing the window size, sequence length, SNR and angular speed.

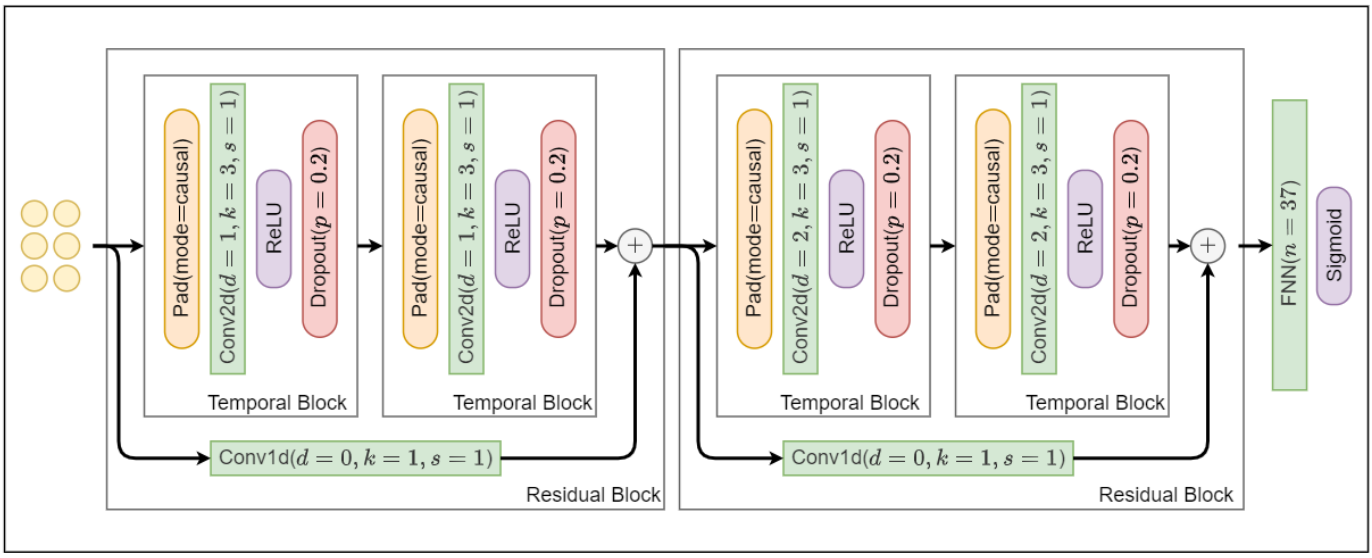


Figure 5.3 An overview of the proposed TCN architecture.

5.3.1 Window Size and Sequence Length

Figures 5.4 - 5.7 present the effect of changing the window size for different sequence lengths, given the following parameters: $K = 15$ frequency bins, overlap of 50%, reverberation RT = 0.2 and SNR = 15dB. These figures show the precision and F-score results. We consider different sequence lengths [1, 4, 9, 19]. For a given history of T_s seconds, the sequence length (L_s) in frames is calculated as follows:

$$L_s = \frac{1+(T_s - W)}{H} \tag{5.2}$$

Where *W*, *H* are window size and hop length in seconds, respectively. Note that the FNN model does not have a memory. Therefore, to study the effect of the sequence length, all past frames were concatenated and then fed to the model.

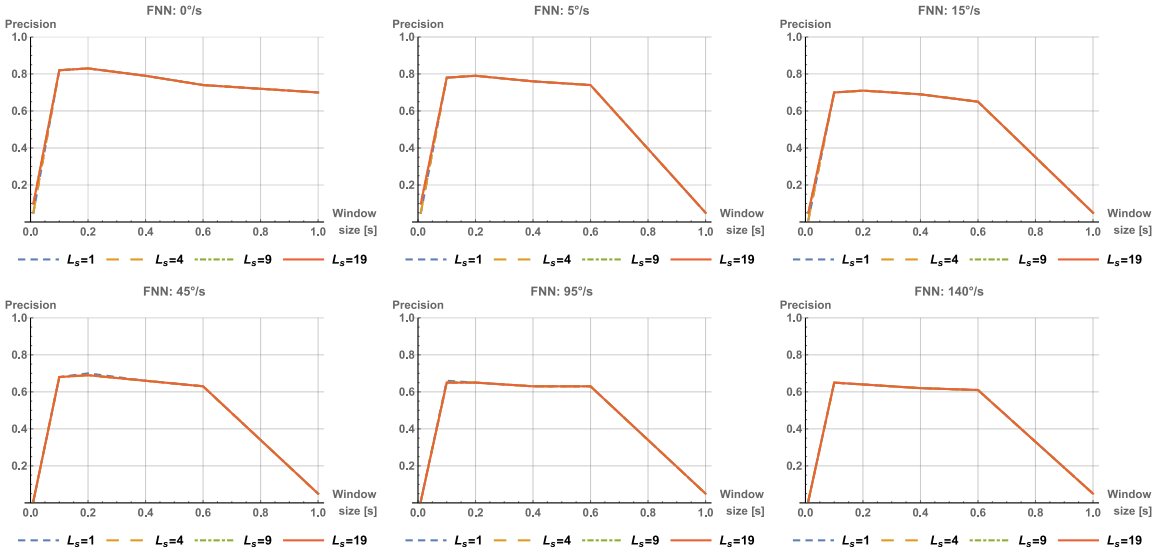
From Figure 5.4, we observe that different sequence lengths have no significant effect on FNN model. In other words, the model does not benefit from using past frames. We notice that FNN was able detect DOAs only at very low angular speeds (i.e., $0°/s - 5°/s$). However, it was barely detecting sources at medium-speeds (i.e., $15°/s - 45°/s$) and failed to have any detection at high-speeds (i.e., $95°/s - 140°/s$).

Additional figures at the end of this chapter show examples of DOA predictions obtained using the different NN models. For the FNN models, Figures 5.10 and 5.11 show DOA predictions at angular speed =0, 45 degree/s respectively, with SNR= 15dB, window size=0.2s and sequence length =1.

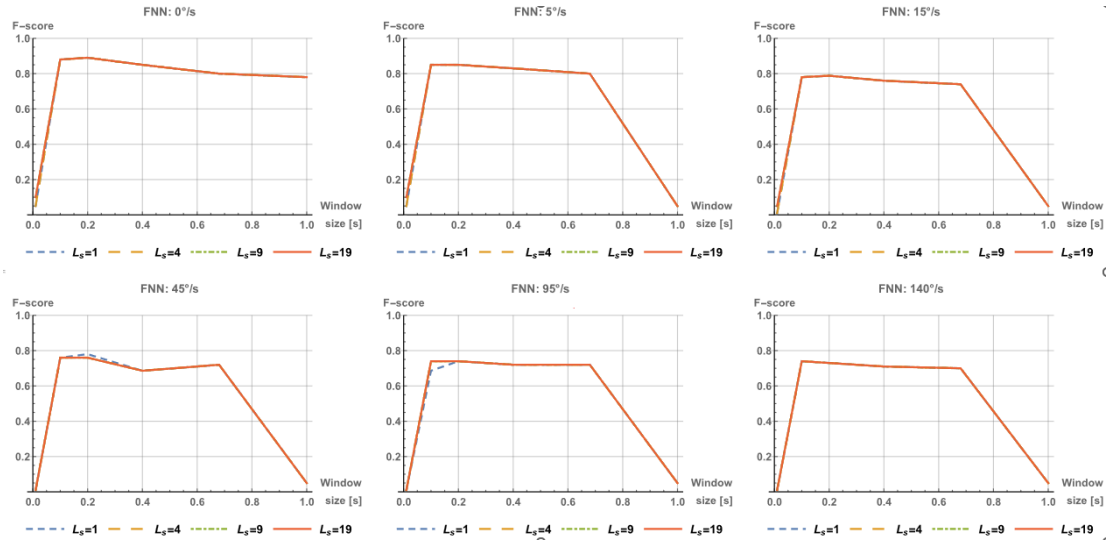
Figures 5.5 - 5.7 present the performance of RNN, GRU and TCN models which are equipped with memory of previous states. We notice that a larger sequence length significantly improves the performance for all three models and the best sequence lengths are within [9, 19] frames. For example, when the sequence length is increased from 1 to 19 frames for low angular speed scenarios, the best achieved precision for RNN, GRU, and TCN was increased by 12%, 14%, and 8%, respectively, and by 30%, 17%, and 16% respectively for high-speed scenarios.

In static scenarios, most of the improvement was achieved by increasing the sequence length from 1 frame to 4 frames. Any further increase had negligible effect. For low angular speed scenarios, TCN was different from the recurrent models (RNN & GRU) in two ways: the total improvement is lower despite having comparable results, and most improvement can be obtained using only the 4 past frames. This means TCN is more efficient at learning from recent history. On the contrary, recurrent models mostly benefit from a sequence length of 9 past frames. However, the performance drops at 19 frames. We believe that this degradation in performance at high sequence lengths is because the total history in seconds becomes too long for NNs to remember, and the observed signal is highly non-stationary within this sequence.

Regarding the effect of the window size, we first analyze the performance on static cases. While RNN, GRU and TCN performed well when using a small sequence length, there was no benefit from increasing the window size even if it covers the same effective history (in seconds).

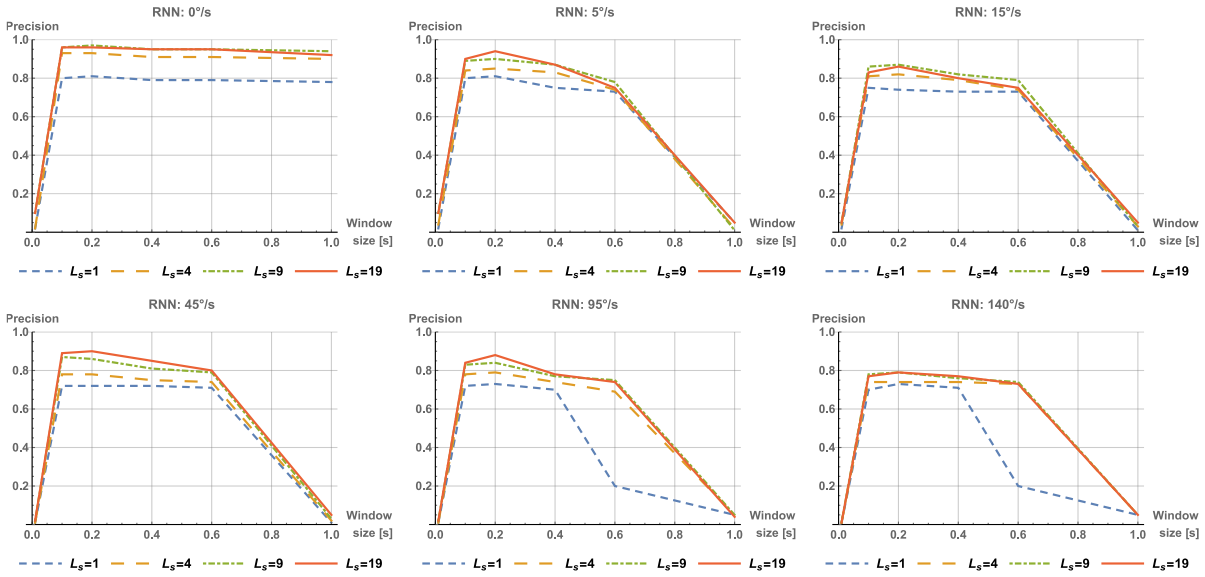


(a) Precision

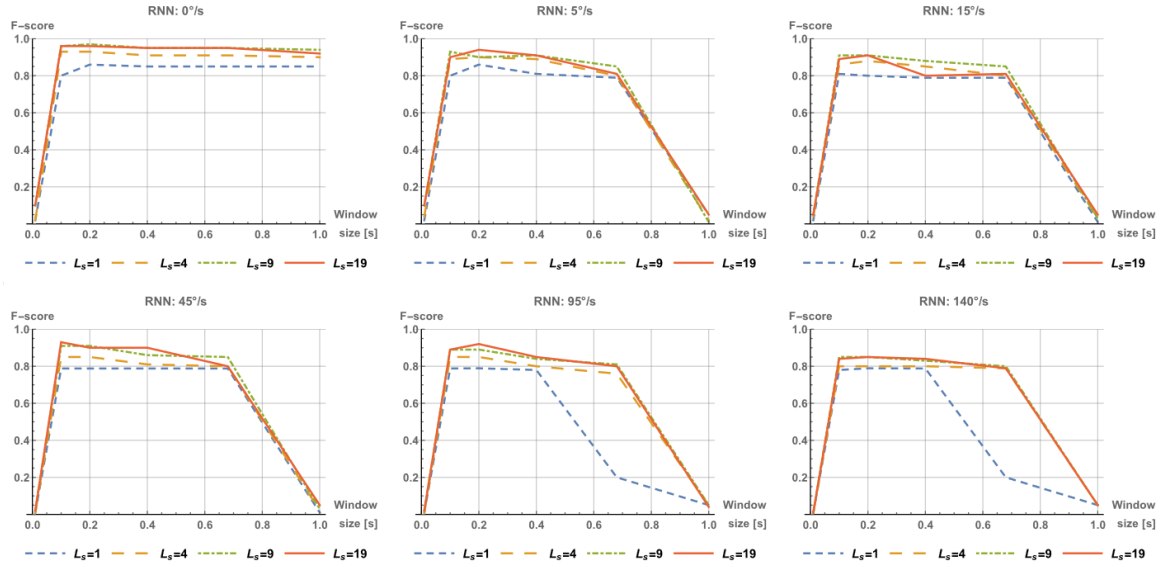


(b) F-score

Figure 5.4 Effect of changing the sequence length for different window sizes at multiple angular speeds, using FNN model.

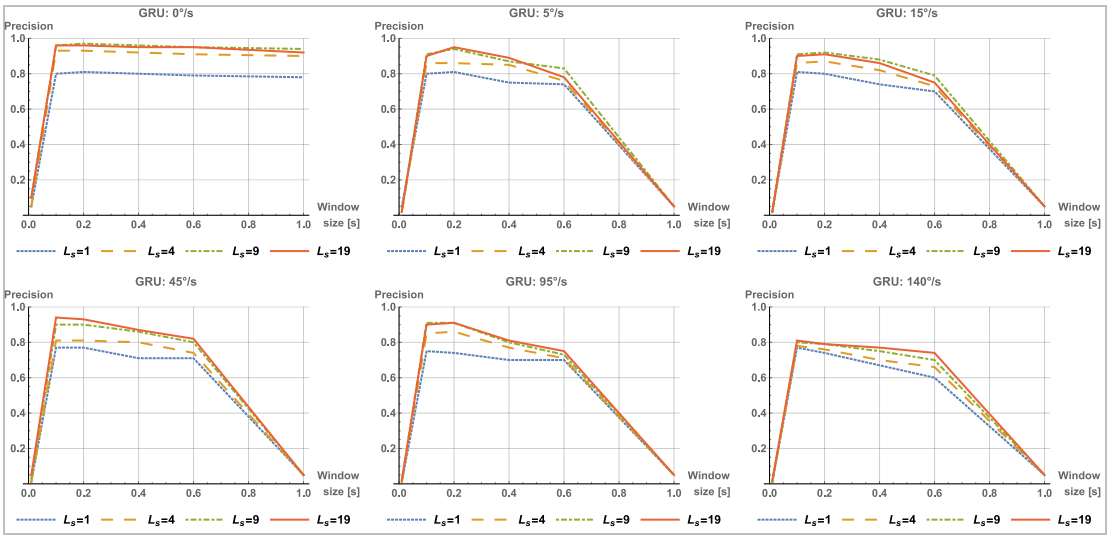


(a) Precision

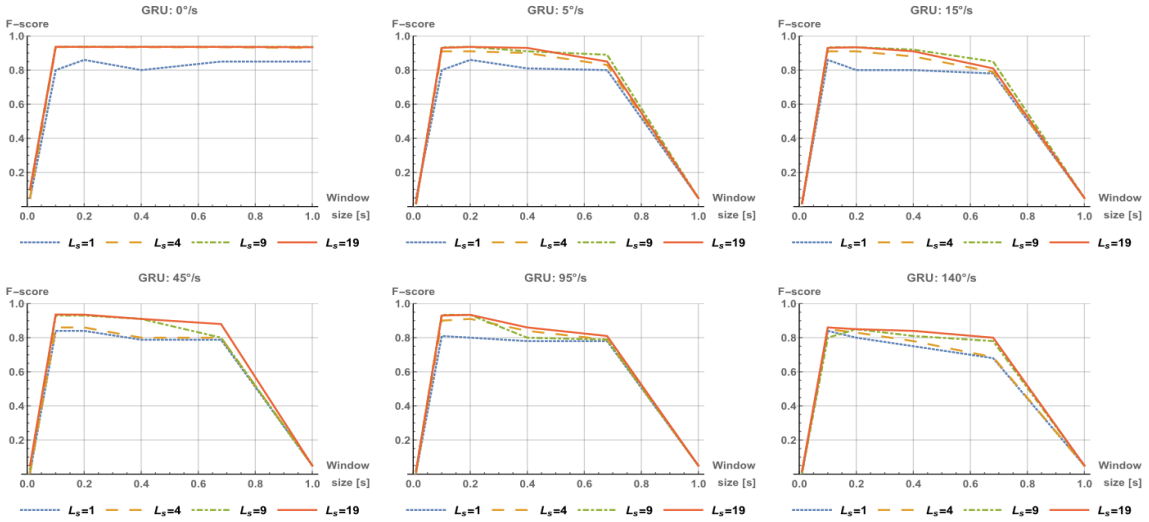


(b) F-score

Figure 5.5 Effect of changing the sequence length for different window sizes at multiple angular speeds, using RNN model.



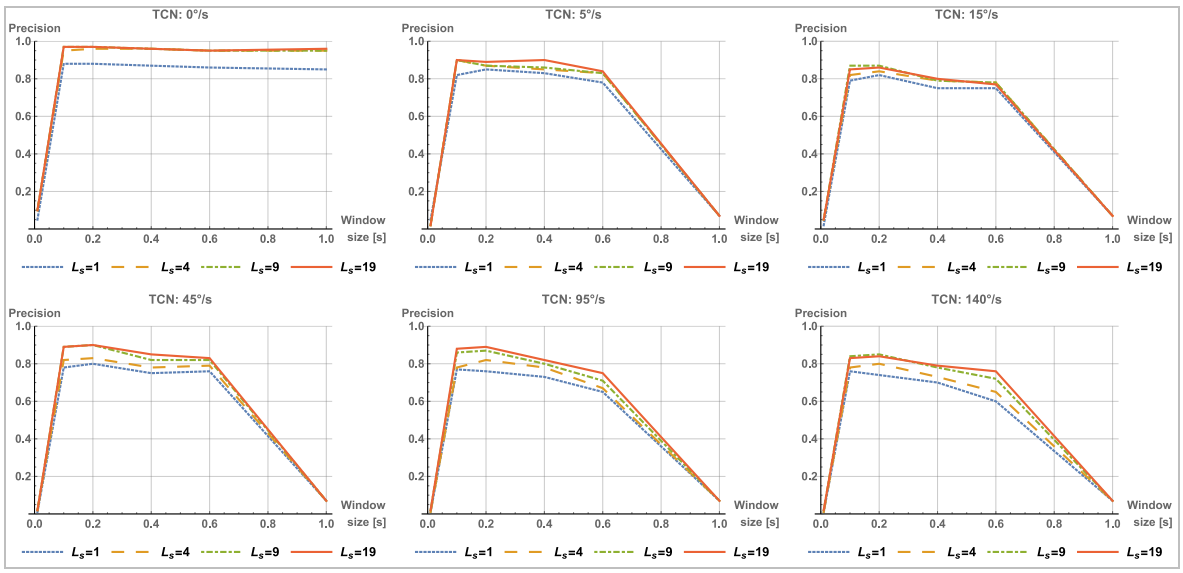
(a) Precision



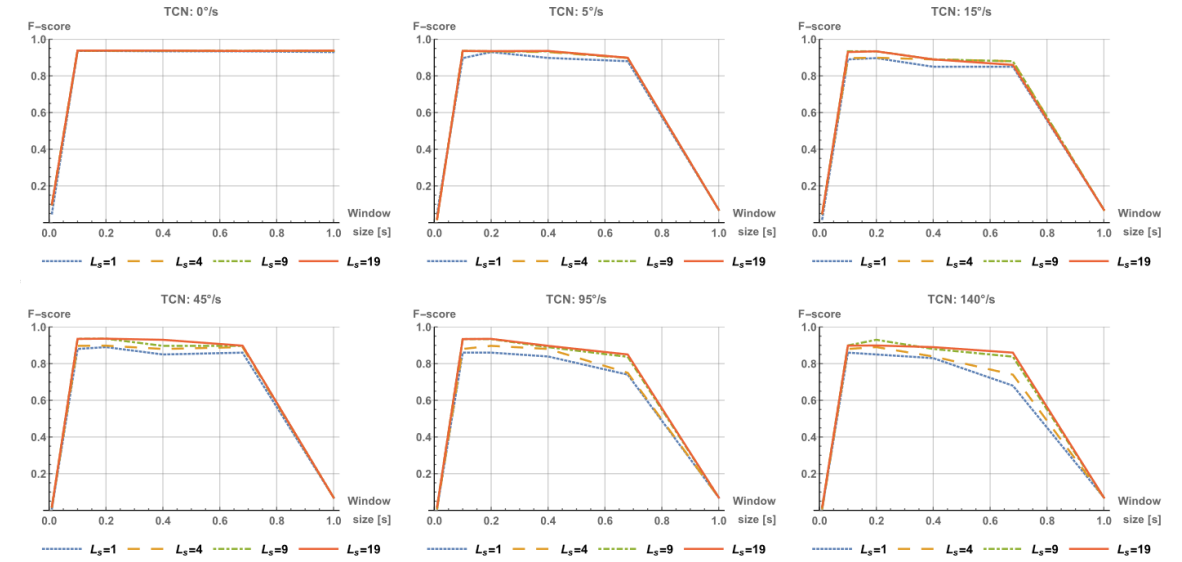
(b) F-score

Figure 5.6 Effect of changing the sequence length for different window sizes at multiple angular speeds, using GRU model.

This can be explained as follows: the sequence length and window size determine the effective memory of the recurrent models, and it can be useful to know the previous location of a source to determine the next location. However, if a source is static, there was no added value in knowing the previous location or having a larger window size.



(a) Precsion



(b) F-score

Figure 5.7 Effect of changing the sequence length for different window sizes at multiple angular speeds, using TCN model.

While the best values for sequence length and window size in GRU and RNN models are similar, we can see from Figure 5.8 that the GRU model outperforms the RNN model by around 5% in precision. This suggests that the DOA detection does not require a long history, which is the main difference between the RNN and the GRU architectures. By looking at Figure 5.8, we notice that the performance of TCN and GRU models are quite similar. However, TCN slightly outperforms

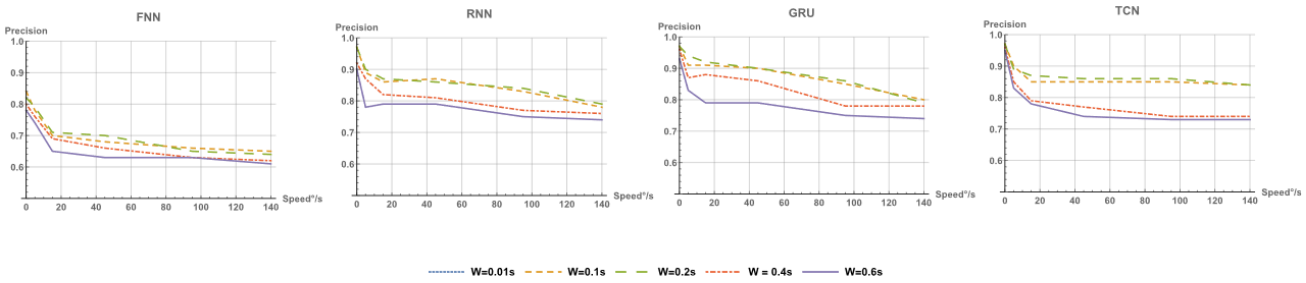
GRU by around 4% in precision at high angular speed. This is possibly because the TCN was more efficient in extracting features from short sequence lengths.

From Figures 5.5-5.7, we see two differences between TCN and recurrent models: (1) beyond window size = 0.4s, the performance of TCN in moving sources drops faster. We believe this is because TCN is not recurrent and doesn't have hidden state that can serve as extra input, so a larger sequence length will not be able to compensate the low-quality features (due to non-stationarity) generated by a larger window size; (2) the gap between performance of sequence length 1 and 19 frames is smaller in TCN but the improvement is consistent regardless of window size or the sources angular speed.

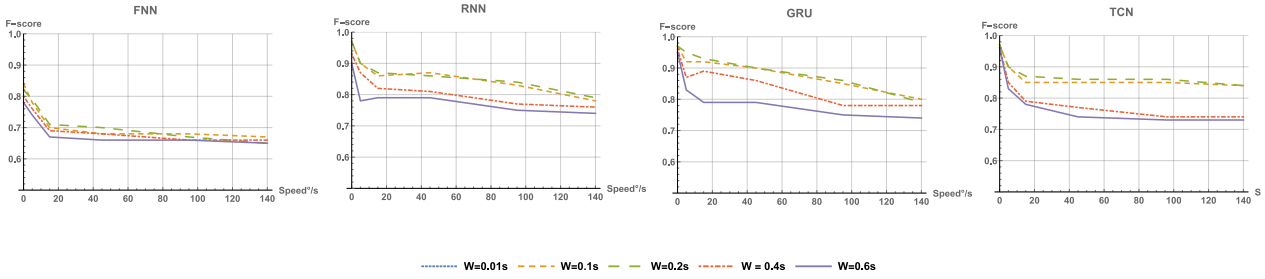
As mentioned when previously discussing FNN models results, additional figures are provided at the end of this chapter showing examples of DOA predictions obtained using the different NN models. Figures 5.12 – 5.15 show DOA predictions from RNN models at different sequence lengths, Figures 5.16 - 5.18 show DOA predictions from GRU models at different SNRs, and Figures 5.19 - 5.21 show DOA predictions from TCN models at different window sizes.

5.3.2 Window Size and Angular Speeds

When comparing the performance of different window sizes at different source angular speeds using the four models (FNN, RNN, GRU, TCN), we notice that TCN performance is slightly better than GRU and (in decreasing order of performance) RNN and FNN. In particular, the TCN model was able to maintain its performance at high angular speed despite being slightly worse at low speed. This is possibly from the TCN lack of feedback which makes RNN training harder and unstable.



(a) Precision



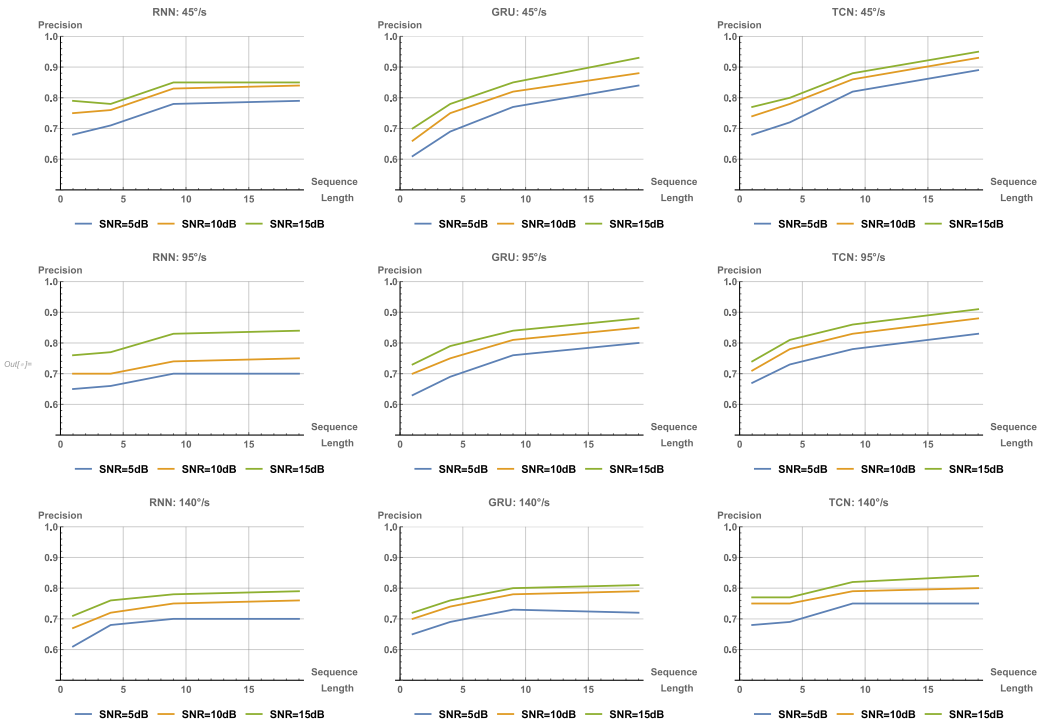
(b) F-score

Figure 5.8 Effect of changing window size at different angular speeds using sequence length of 9 frames, for different NN architectures.

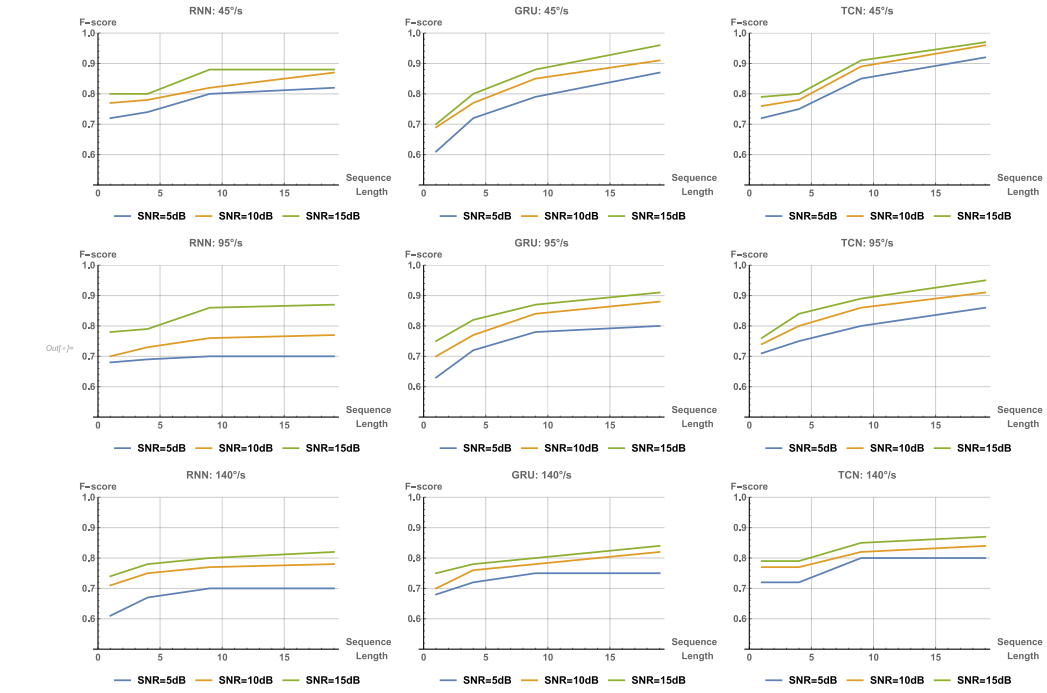
Figure 5.8 shows that best performance is achieved at a window size in the interval [0.1s, 0.2s]. With a window size of 0.2 seconds, it is possible to achieve precision above 85% using RNN, GRU, and TCN for angular speeds up to 60°/s, 95°/s, and 130°/s respectively. However, the performance collapses with moving sources when FNN is used.

5.3.3 SNR

Figure 5.9 presents the effect of changing the sequence length at different SNRs, given the following parameters: $K = 15$ frequency bins, overlap of 50%, reverberation $RT = 0.2\ s$ and $W = 0.2\ s$. We can very clearly notice a drop in the performance for all models when the SNR drops from 15dB to 5dB. Also, we noticed that using a longer sequence length improves the performance but does not improve the model robustness against noise.



(a) Precision



(b) F-score

Figure 5.9 Effect of changing sequence length with different SNRs on different models. Data parameters were: window size=0.2s, frequency bins = 15, hop length =50%, SNR=15 dB, RT=0.2s, two microphones separated by 0.05m.

We measure the robustness by keeping the sequence length fixed at certain value and reducing the SNR. For angular speeds below $95°/s$ and an SNR = 5dB, only the TCN was able to maintain a precision above 80% using a sequence length of at least 9 frames. For lower noise levels (e.g., SNR above 10dB), GRU matches the performance of TCN and outperforms RNN. The degradation in the RNN model performance can be due to the feedback connection propagating some noise signal from the noisy features. TCN on the other hand uses convolution which can smooth/filter out some of the noise in the signal.

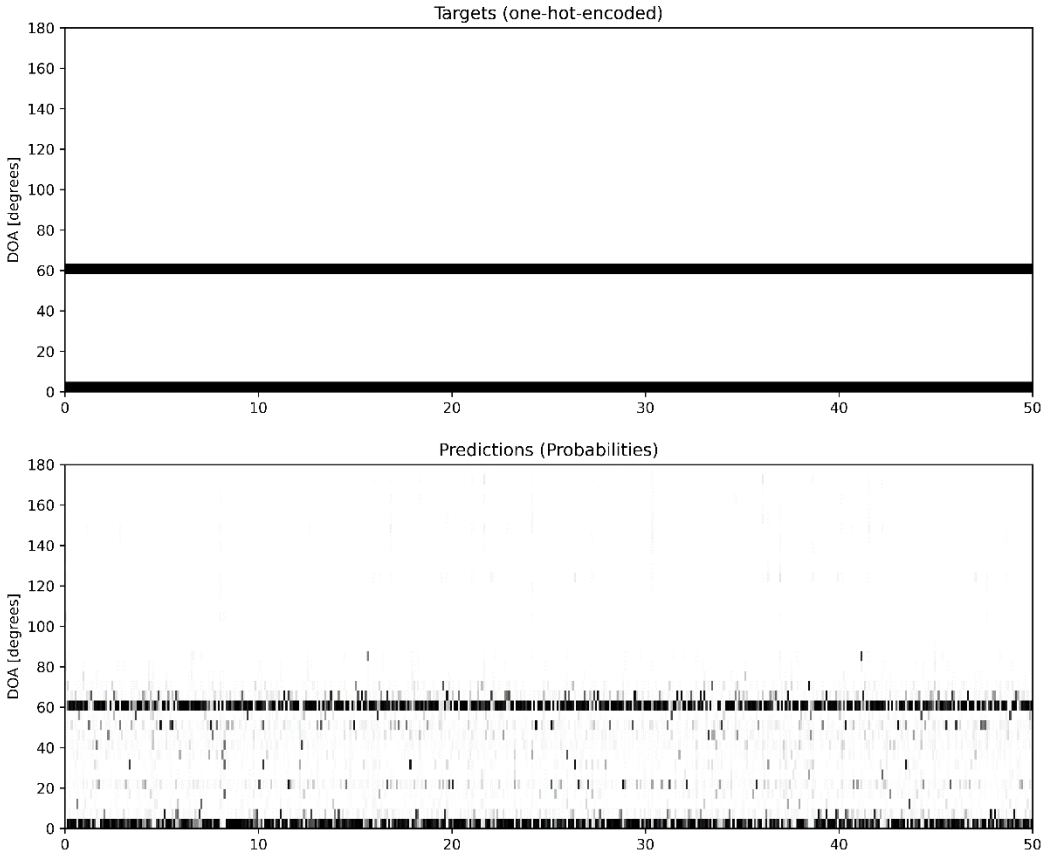


Figure 5.10 A prediction plot of FNN model when the angular speed = 0 degree/s at SNR=15dB and window size = 0.2s.

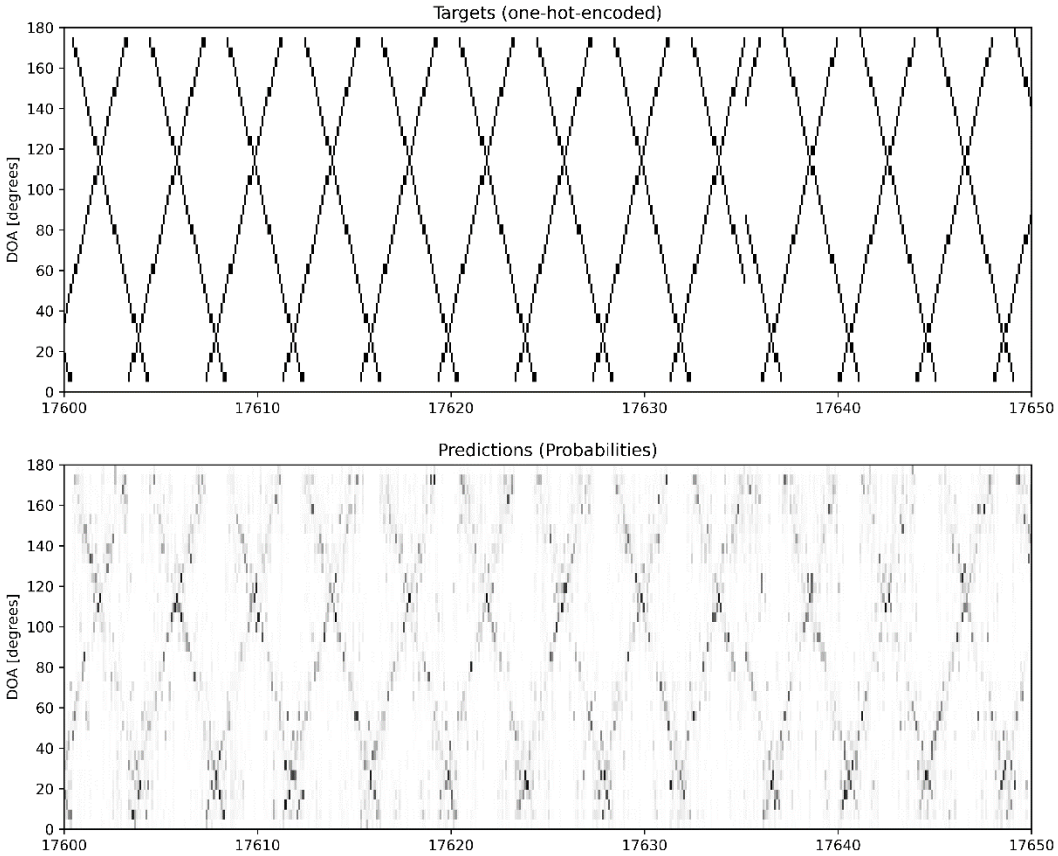


Figure 5.11 A prediction plot of FNN model at SNR=15dB and window size = 0.2s.

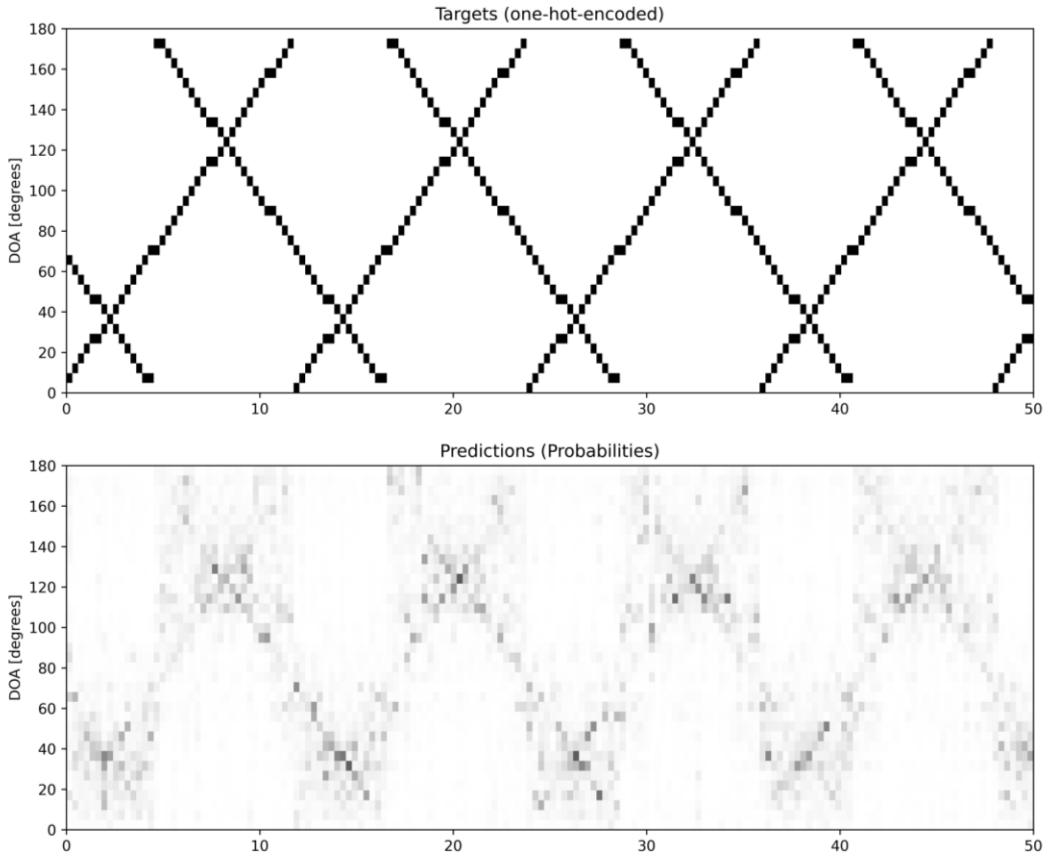


Figure 5.12 A prediction plot of RNN model of two sources moving opposite to each other, the sequence length =1, SNR=15dB and window size = 0.2s.

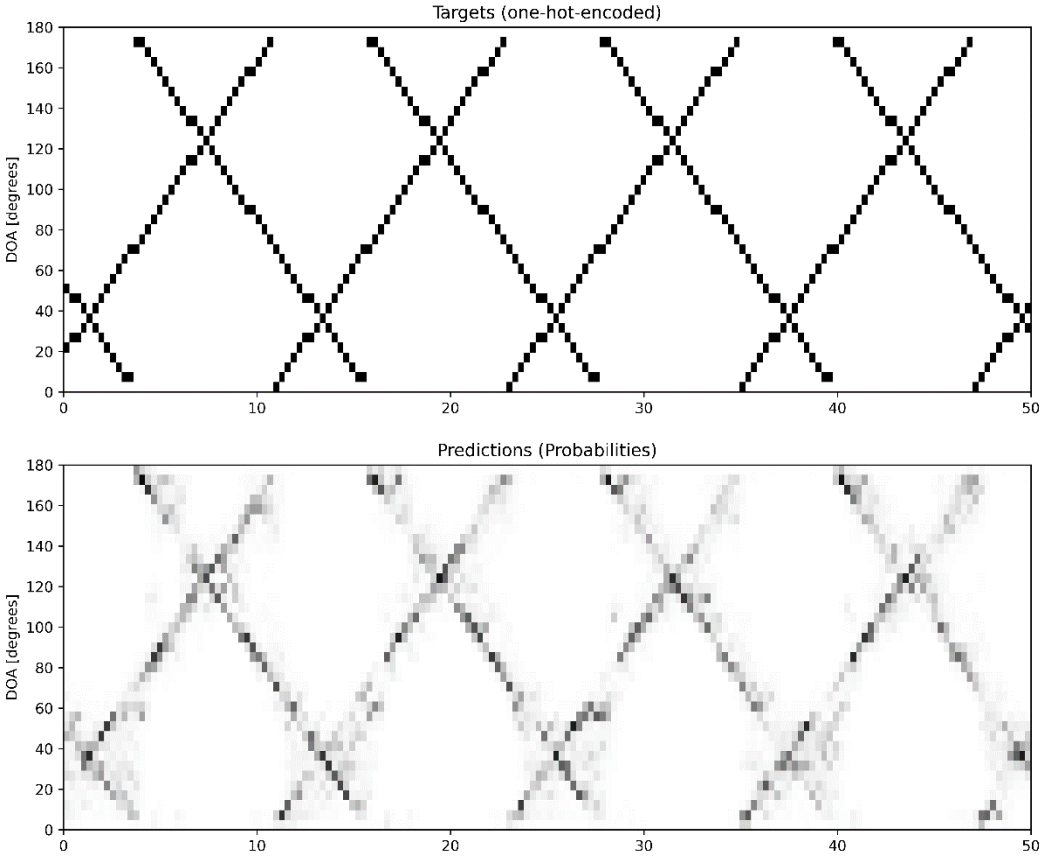


Figure 5.13 A prediction plot of RNN model of two sources moving opposite to each other, the sequence length =4, SNR=15dB and window size = 0.2s.

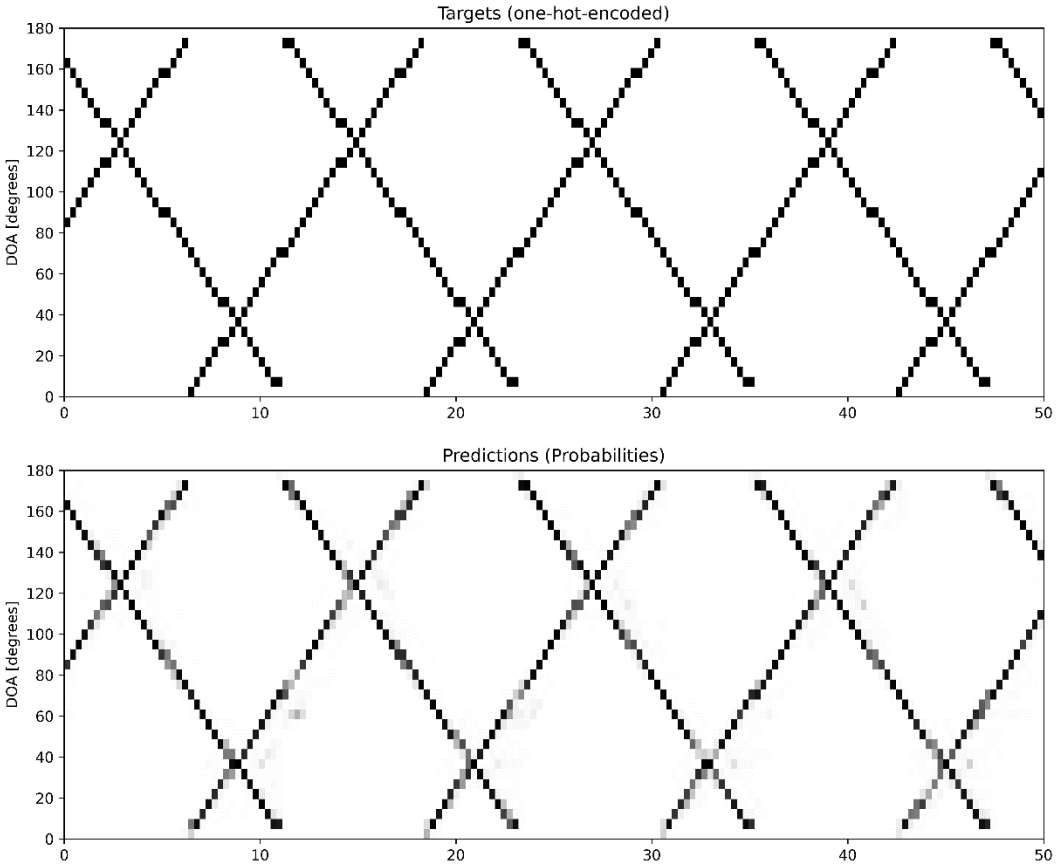


Figure 5.14 A prediction plot of RNN model of two sources moving opposite to each other, the sequence length =9, SNR=15dB and window size = 0.2s.

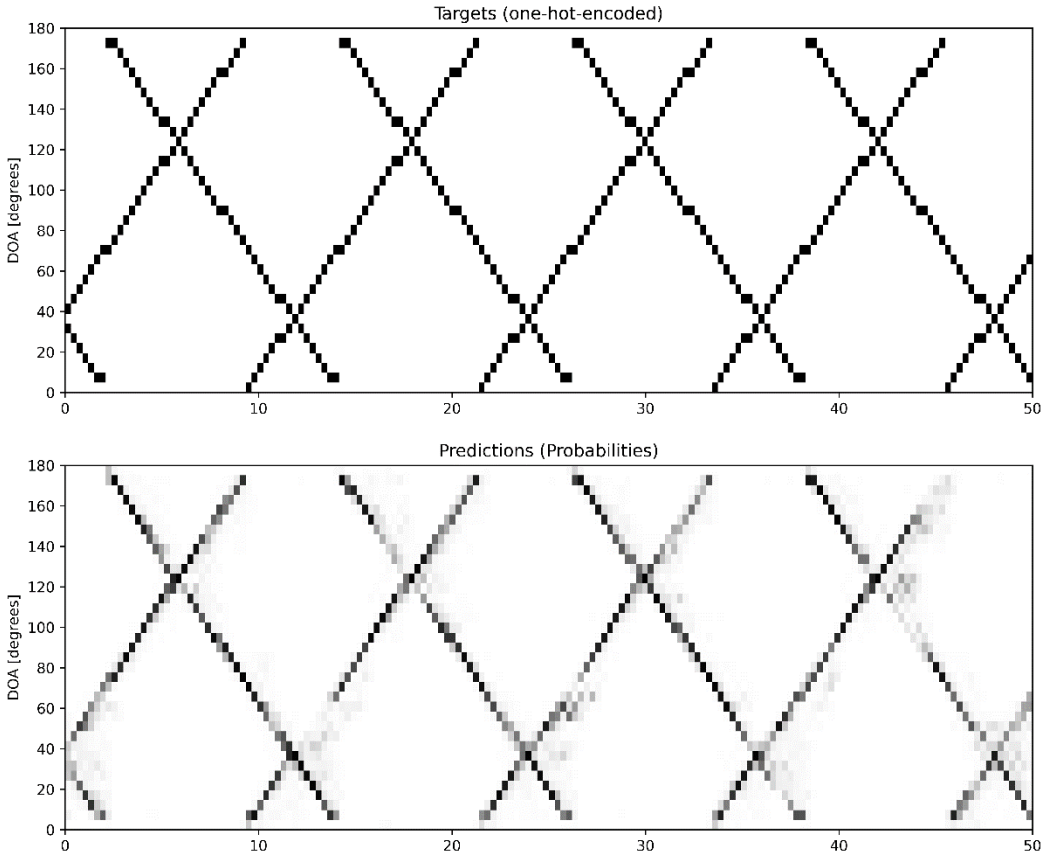


Figure 5.15 A prediction plot of RNN model of two sources moving opposite to each other, the sequence length =19, SNR=15dB and window size = 0.2s.

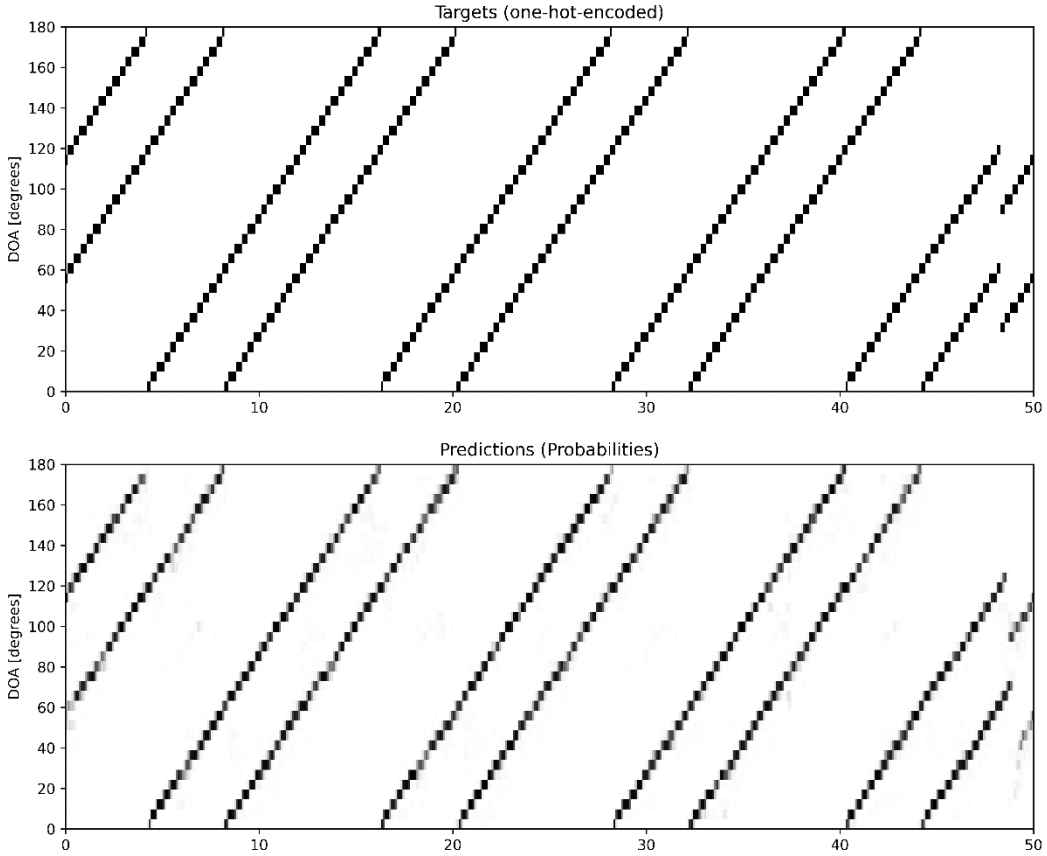


Figure 5.16 A prediction plot of GRU model of two sources moving in the same direction, the sequence length =9, SNR =15dB and window size = 0.2s.

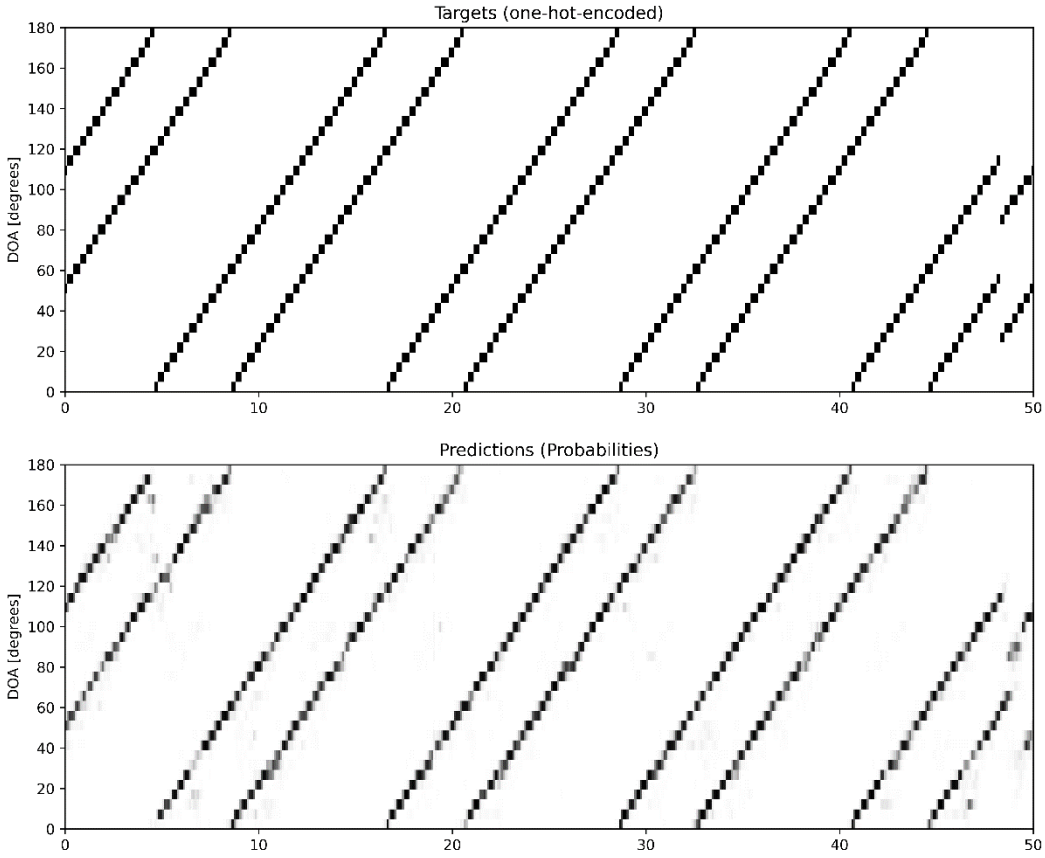


Figure 5.17 A prediction plot of GRU model of two sources moving in the same direction, the sequence length =9, SNR=10dB and window size = 0.2s.

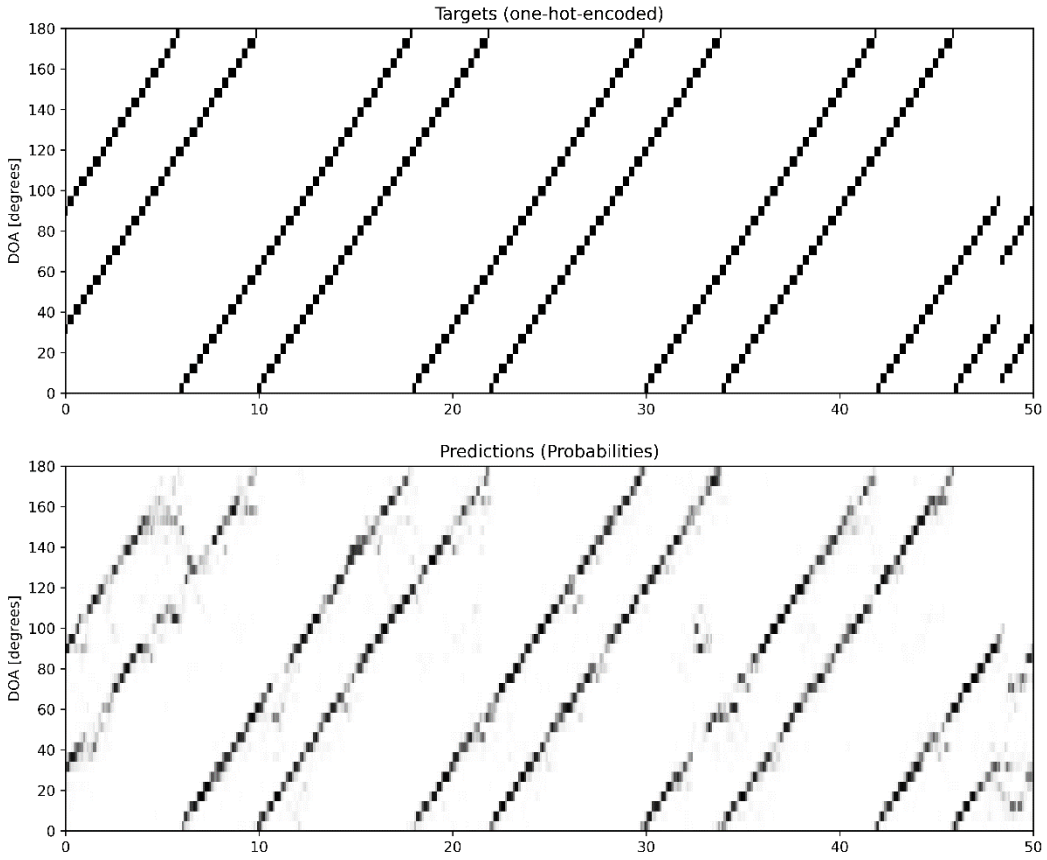


Figure 5.18 A prediction plot of GRU model of two sources moving in the same direction, the sequence length =9, SNR=5dB and window size = 0.2s.

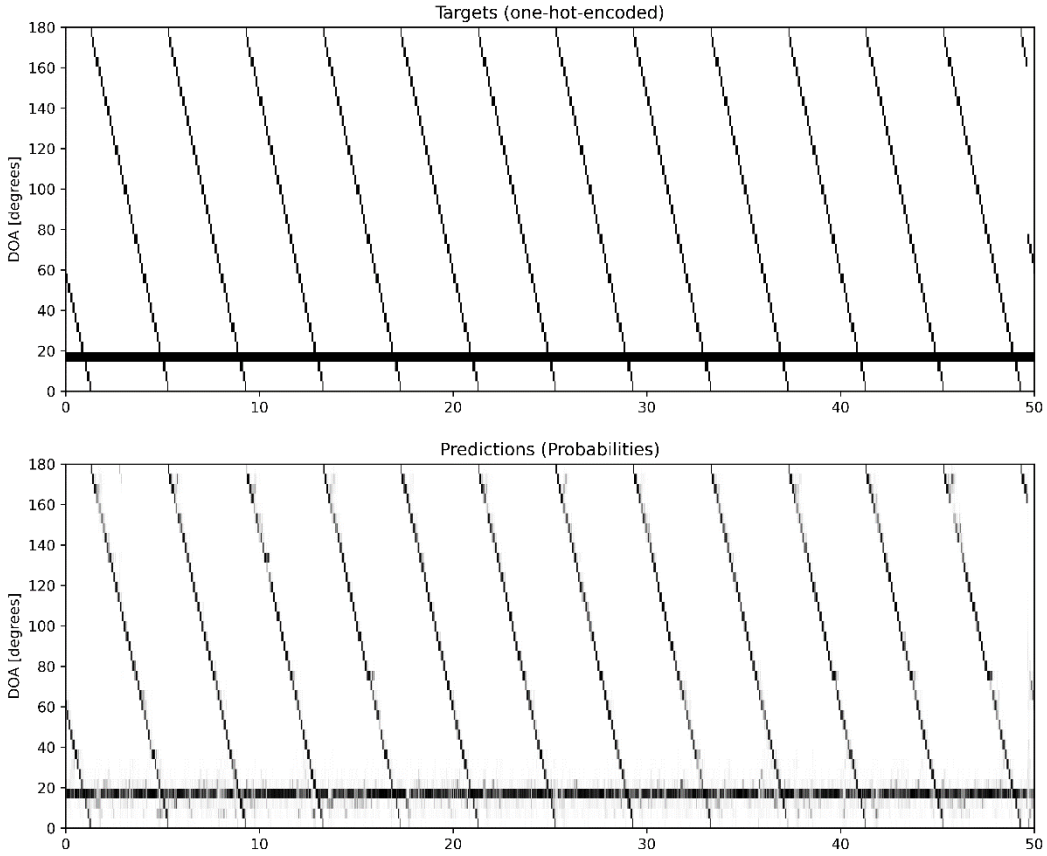


Figure 5.19 A prediction plot of TCN model for two sources: one is moving and the other is static, the sequence length =9, SNR=5dB and window size = 0.1s.

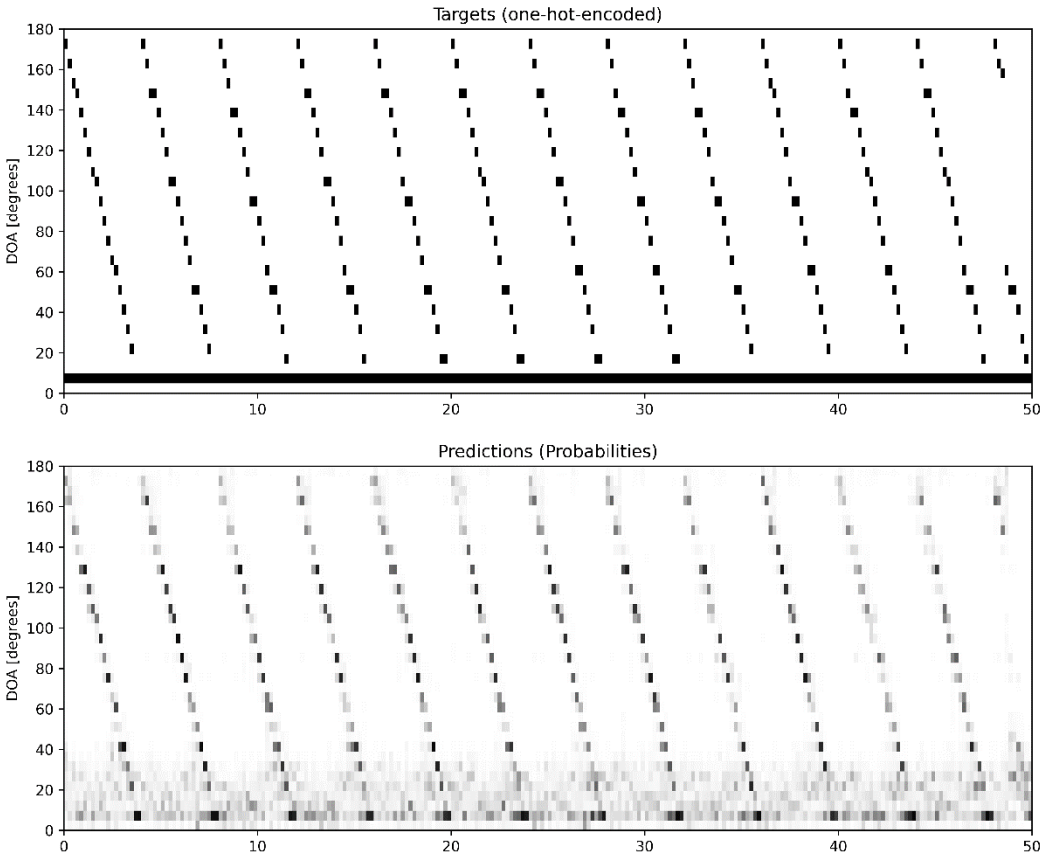


Figure 5.20 A prediction plot of TCN model of two sources moving in the same direction, the sequence length =9, SNR=5dB and window size = 0.4s.

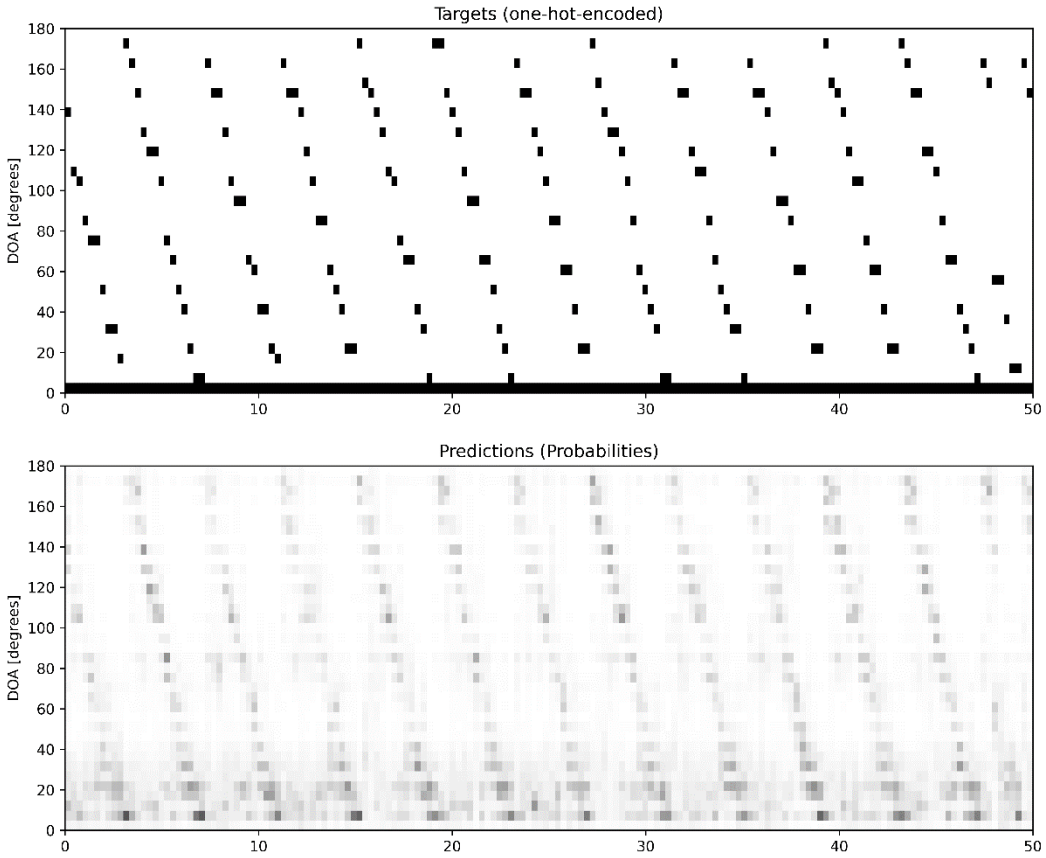


Figure 5.21 A prediction plot of TCN model of two sources moving in the same direction, the sequence length =9, SNR=5dB and window size = 0.6s.

Chapter 6 Effect of Using Noise-only Dataset versus Speech Audio Dataset

In this chapter we investigate the effect of using WGN as a dataset versus using speech audio as a dataset. The reason we do these tests, is that we expect that the spectral content of the acoustic sources used should not have a very strong effect on the DOA detection performance, because the features that are provided to the NN models are based on spectral magnitude ratios and phase differences (i.e. they are normalized to a reference microphone), and the tests are performed at fairly high SNR (i.e., the frequency dependent SNR is fairly high at all frequencies). We also investigate how using the TIMIT dataset with and without removing silence periods affects performance. However, since the silence periods in the TIMIT dataset are short, we also considered adding synthetic pause periods in which the only sounds present are the background noise. The purpose is to investigate whether the model is robust to signal segments without target source signals, i.e., where no detection of that target should occur. All tests were done on a multi-class multi-label DOA detection setup and using FNN, RNN, GRU and TCN models, as in Chapter 5.

6.1 Experiment Setup

We used the same setup as in Chapter 5 to create the scenarios to generate the two datasets. However, a first dataset was generated using the TIMIT database without removing the silence, a second dataset was generated with added pauses, a third one was generated using TIMIT without silence, and finally a fourth one was generated using WGN signals.

In the previous two chapters we studied the effect of changing the reverberation time (RT), number of frequency bins, hop length, window size and sequence length using the FNN, RNN, GRU and TCN architectures. In this chapter, we test the impact of training NNs with two different types of sound sources and observe the resulting testing performance for multi source DOA estimation. The resolution is set to 5°, RT to 0.2s, SNR level to 15 dB, and sequence length to 9. Note that we use the same evaluation metrics and same scenarios to generate the datasets as those in Chapter 5.

6.2 Analysis

Figure 6.1 presents the effect of changing the training dataset source type, for different angular speeds, given the following parameters: sequence length = 9, $K = 15$ frequency bins, overlap of 50%, reverberation RT = 0.2 and SNR = 15dB. The figure shows the precision and F-score results.

From Figure 6.1, we observe that training and testing with flat spectral noise, rather than speech data, produces better performance of 11%, 9%, 7% and 4% using FNN, RNN, GRU and TCN models, respectively, in the DOA estimation precision.

We believe that this is due to two differences between speech and WGN. First, the auto power spectral density (auto-PSD) for TIMIT database is different speech-type auto-PSDs, while the auto-PSDs in WGN is flat. For a given global SNR, the resulting frequency-dependent SNR at the different frequency bins used in our features will vary based on the auto-PSD, and thus the quality of the features used in the models can vary. Second, unlike WGN which always has the same statistics, speech has non-stationary levels and frequency content. This variation in statistics make it more challenging for the NN to find a fixed solution (with constant weights) that can provide a good performance under these different conditions.

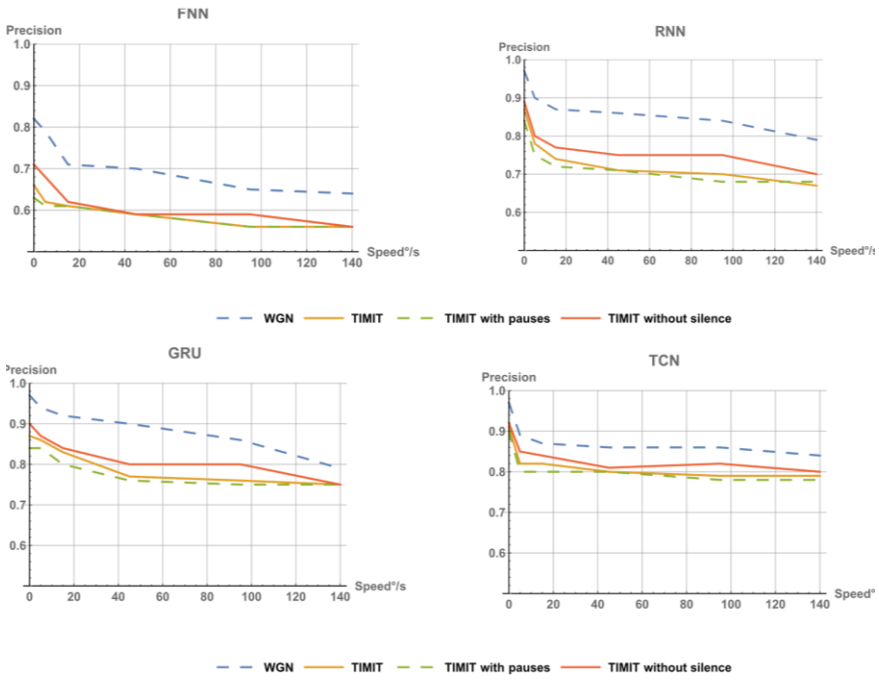
Compared to cases when silence is included in the dataset, Figure 6.1 shows that training and testing on WGN dataset resulted in the best performance, followed by training and testing on the TIMIT without silence, followed by training and testing on the TIMIT dataset with silence. The worst performance was observed when training and testing on the TIMIT dataset with additional synthetic pauses. We believe that this is due to the transitions between silence and speech, which can be more challenging for the neural network to predict. In particular, for a given sequence length, the presence of silence in some frames reduces the effective sequence length because fewer frames have useful information about the DOA of the active speakers. By removing the silence periods, the dataset provides a complete sequence with clearer signal for the model to learn from, which can improve its performance.

We can also notice from the same figure that while all the models have the same trend, TCN is more robust, and the performance wasn't affected as much compared to other models. In FNN, the difference is not very noticeable because it is always poor. One explanation for why TCN is robust is that TCN architecture is inherently sparse when it comes to processing the time-dimension. This means that even without silence, the features generated by TCN don't necessarily depend on the whole sequence length. Therefore, the presence of silence can have less negative effect on TCN performance.

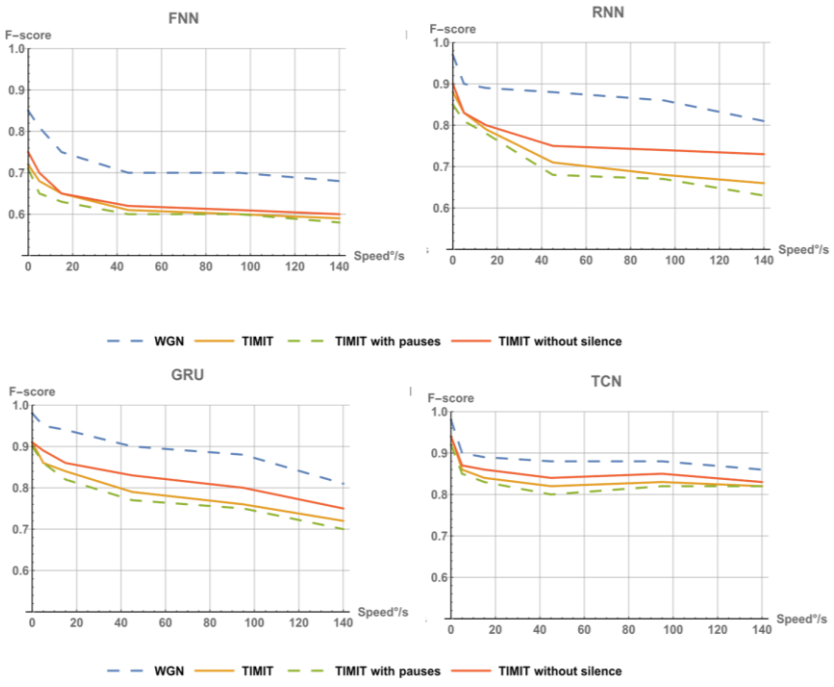
Also, we tried training on the WGN dataset and testing on the TIMIT dataset, training on the TIMIT dataset and testing on the WGN dataset. Figure 6.2(a) illustrates the result of a DOA

prediction plot example obtained when the TIMIT database is used for both training and testing dataset. It presents good prediction for segments when there are two speakers moving near and opposite to each other. However, the prediction is weaker when the speakers move further from that point and lose some segments when there is only one speaker moving. Figure 6.2(b) presents the results obtained when both training and testing were done on WGN. It presents the best result among the four cases. It presents strong predictions for most segments, except some weak predictions in some segments when there is one speaker, in which case it still outperforms all other three cases.

Figure 6.2(c) shows a prediction plot example when the network is trained using WGN and tested on synthetic speech from the TIMIT database. This does not perform very well (the weakest prediction of all four cases in Figure 6.2 a-d, and most false detections). Figure 6. 2(d) shows a prediction plot example resulting from using TIMIT for training dataset and WGN for testing dataset. Even though both Figure 6.2 (c) and (d) presents similar quality of prediction, using the TIMIT for training and WGN for testing provides a bit better prediction results than using the WGN for training and TIMIT for testing, particularly less false detections and a bit stronger predictions.

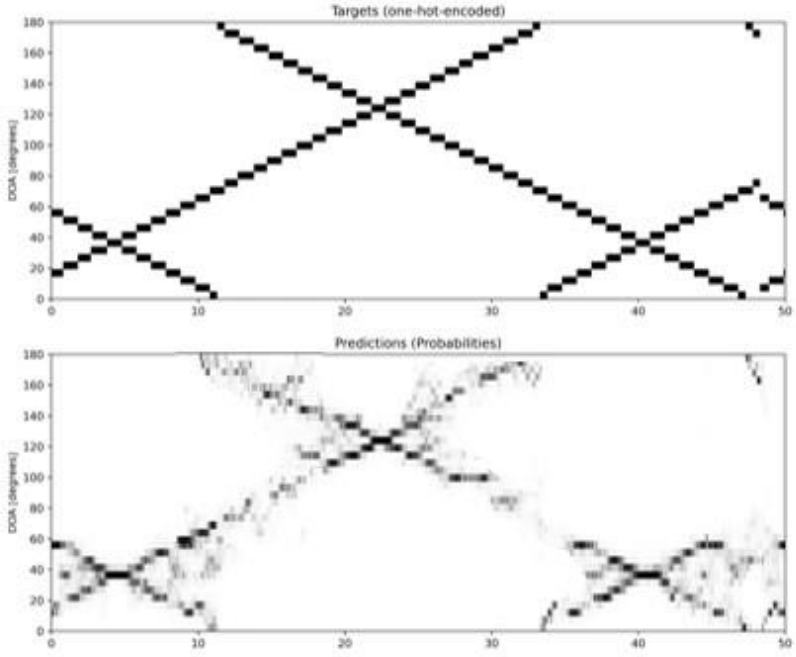


(a) Precision

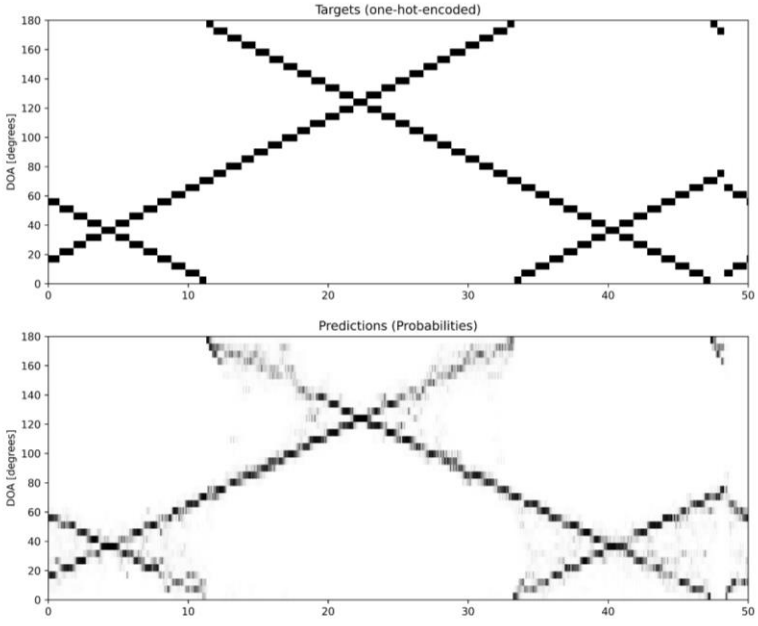


(b) F-score

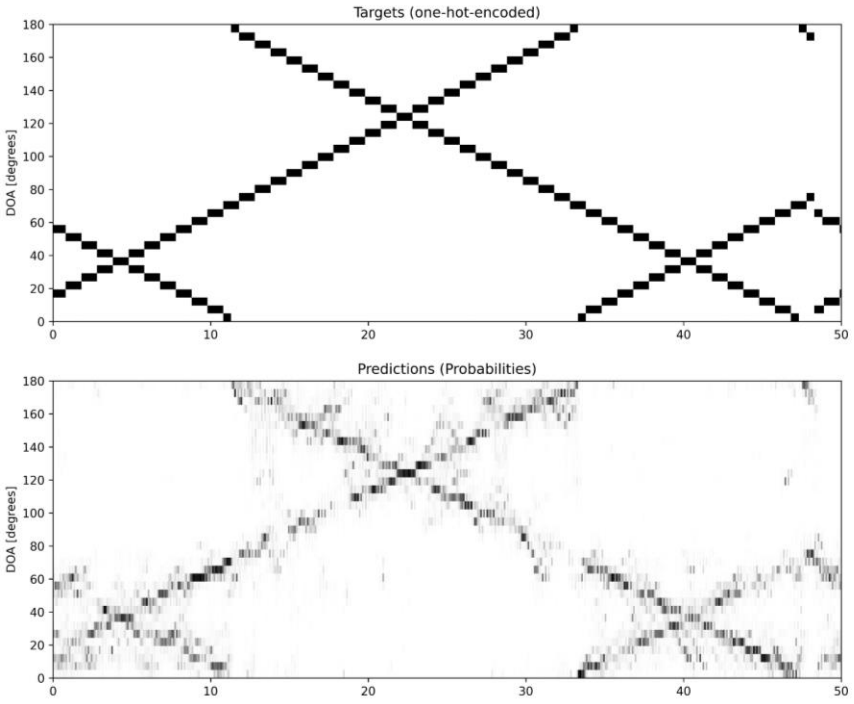
Figure 6.1 Comparing the performance when using speech and white noise for the training data at window size=0.2s, sequence length = 9, SNR=15dB, frequency bins = 15, overlap= 50% and for different NN architectures.


Targets (one-hot-encoded)
DOA [degrees]
Predictions (Probabilities)
DOA [degrees]

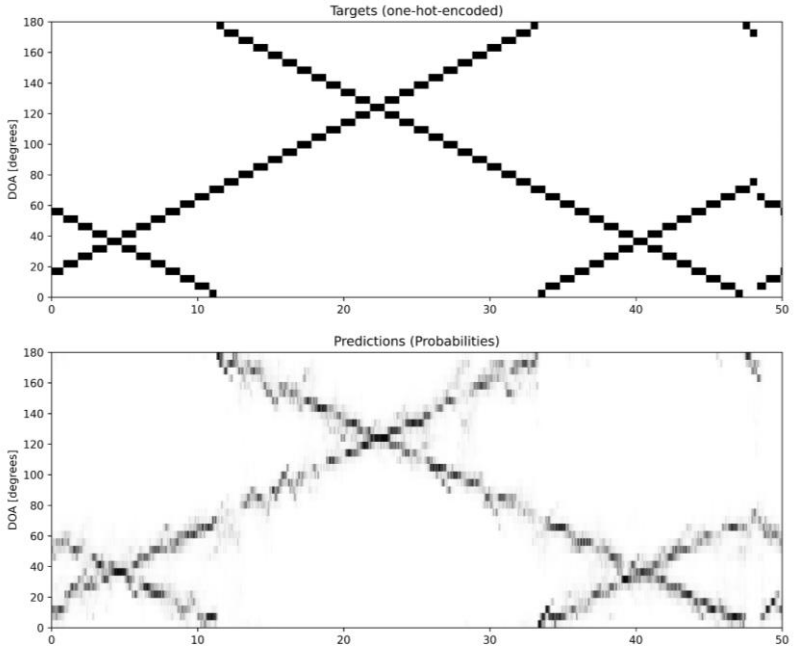
(a)


Targets (one-hot-encoded)
DOA [degrees]
Predictions (Probabilities)
DOA [degrees]

(b)



(c)



(d)

Figure 6.2 A prediction plot of TCN model of two sources moving opposite to each other direction, sequence length =9, SNR=15dB and window size = 0.2s. (a) using TIMIT for training and testing dataset, (b) using WGN for training and testing dataset, (c) using WGN for training dataset and TIMIT for testing dataset, (d) using TIMIT database for the training dataset and WGN for testing dataset.

Chapter 7 Effect of Changing Microphones Configuration

In this chapter we investigate the effect of using 2 microphones versus 4 microphones for the DOA detection task. All tests were done on a multi-class multi-label DOA detection setup and on FNN, RNN, GRU and TCN models.

7.1 Experiment Setup

We used the same setup as in Chapter 5 to create the scenarios to generate the datasets, except that we used 4 microphones in the generation of the second dataset. In chapters 4 and 5 we studied the effect of changing the reverberation times (RT), number of frequency bins, hop length, window size and sequence length using the FNN, RNN, GRU and TCN NN architectures. In this chapter, we test the impact of using 2 microphones and 4 microphones on the acoustic DOA detection performance. The resolution is set to 5°, RT to 0.2s, SNR level to 15 dB, and sequence length to 9. We used WGN as the type of data for the datasets in this chapter. We used the same evaluation metrics and same scenarios to generate the training datasets as those used in Chapter 5.

7.2 Analysis

There are two ways of designing microphone arrays: the first one is non-uniform spaced arrays (also called nested arrays), and the second one is uniform spaced arrays. Traditionally non-uniform spacing has been popular as a way to achieve near-constant performance characteristics across frequency. For example, acoustic DOA detection performance can be described in terms of the ability to create beampatterns directed towards different DOA directions (in our case, the NN takes care of doing this implicitly). However, as frequency decreases, the main lobes in the beampatterns become larger and larger (less angular resolution). This can be compensated by using a pair of microphones with a larger distance, which improves angular resolution. However, in [41] it is reported that the difference between uniform and non-uniform is not very large; non-uniform array can be better at some angles, while uniform arrays can be better at some other angles.

As a scientific method, we choose to compare the results between 2-sensors and 4-sensors linear arrays, by changing only one factor at a time, so we chose to keep uniform spacing between the sensors, and the same spacing as with 2 microphones (5 cm) between all the microphones, as shown in Figure 7.1.

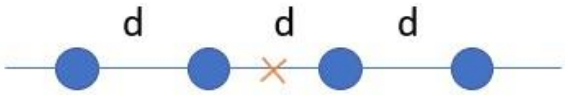
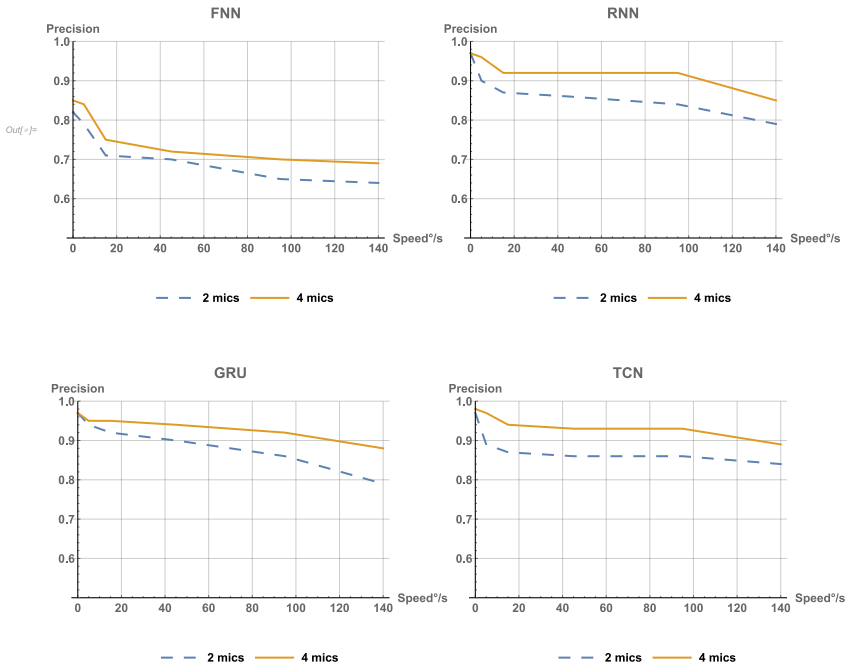


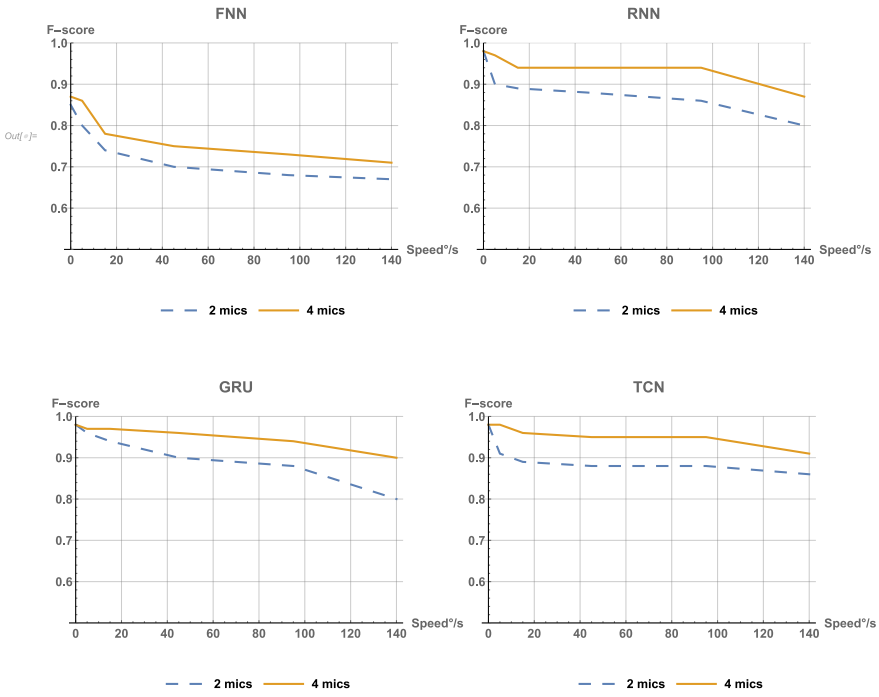
Figure 7.1 Microphone array configuration used, with uniform distance.

Figure 7.2 presents the effect of changing the dataset (2 microphones vs. 4 microphones) for different angular speeds, given the following parameters: sequence length = 9, $K = 15$ frequency bins, overlap of 50%, reverberation RT = 0.2 and SNR = 15dB. The figure shows the precision and F-score results. From Figure 7.2, we observe that using 4 microphones, rather than 2 microphones, produces better performance. For example, the improvement was around 5% in all angular speeds using FNN, 6% and 8% in moderate and high speeds in RNN and GRU, respectively, and 7% and 5% in moderate and high speeds in TCN, respectively.

We believe that this is due to two reasons. First, the microphone array becomes 3 times longer (longer "aperture") and as a result the angular resolution becomes 3 times better. Second, having access to 4 signals instead of 2 signals allows to generate more features and improves the capability to "average out" some noise (directional or not) by averaging the features (implicitly done by the NN models). Figure 7.3 shows some DOA prediction plots for a scenario using 2 microphones and 4 microphones, where it can be observed again that the 4-microphone scenarios performs better, as expected.

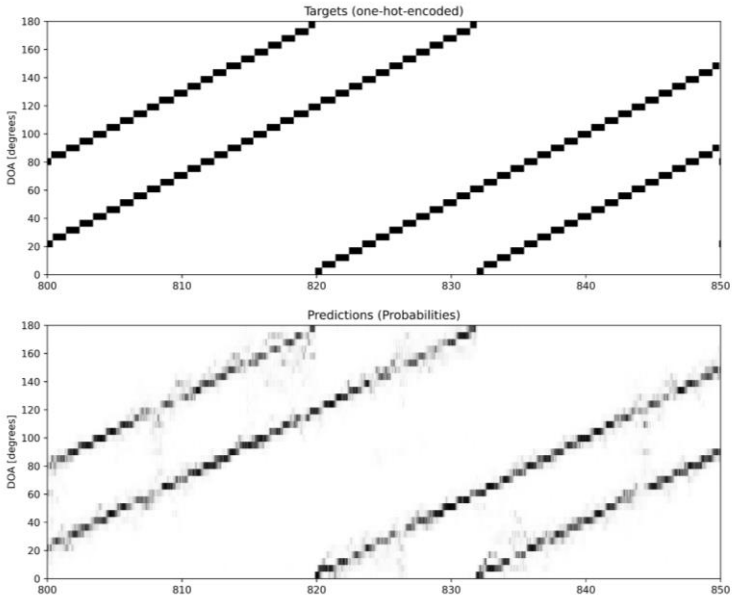


(a) Precsion

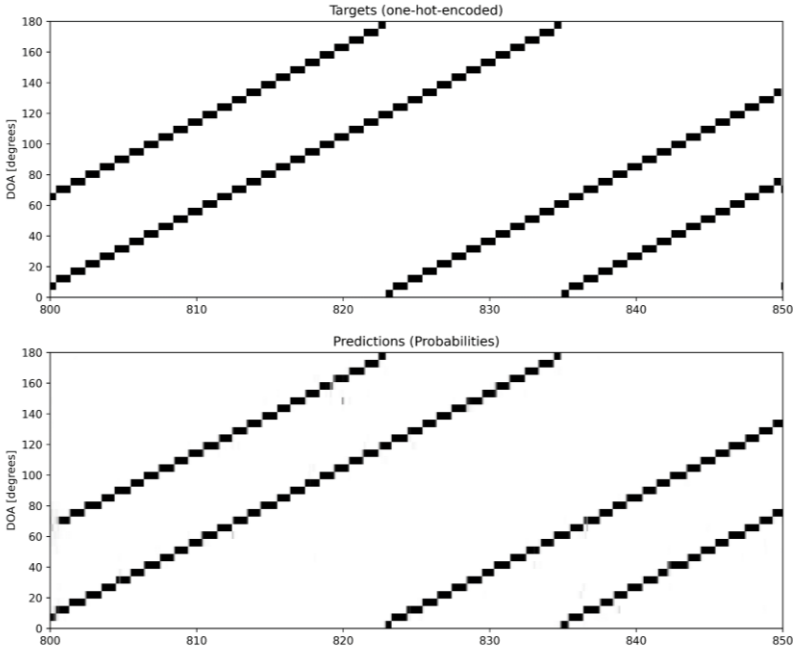


(b) F-score

Figure 7.2 Comparing the performance when using 2 microphones and 4 microphones, for WGN sources, window size=0.2s, sequence length = 9, SNR=15dB, frequency bins = 15, overlap= 50% and for different NN architectures.



(a)



(b)

Figure 7.3 A prediction plot of GRU model for two sources moving in opposite directions to each other, sequence length =9, SNR=15dB and window size = 0.2s. (a) using 2microphones, (b) using 4 microphones.

Chapter 8 Conclusion and Future work

8.1 Conclusion

The main purpose of this work was to develop an acoustic source DOA detection system (or SSL system) that works with moving sources and to comprehensively study the effect of different parameters such as reverebration time, number of frequency bins, hop lengths, window sizes, sequence length, SNR, sources angular speeds, with different deep learning NN architectures (in particular: FNN, RNN, GRU, TCN architectures).

In Chapter 1, a brief review of the different acoustic source DOA estimation methods using deep learning techniques was provided. The motivation and approach for our work were also defined in this chapter. Also, a brief background of related topics in both signal processing and deep learning was introduced in chapter2.

In Chapter 3, the system design was explained. First, with a description of the dataset generation in the time domain and then conversion to frequency domain and feature extraction. Then, the details of the neural network architectures that were implemented in this work.

In Chapter 4, the effect of different hyperparameters and acoustic conditions (RT, SNR, STFT parameters, training on static scenarios vs. training on moving scenarios) have been studied using a FNN neural netowrk. We observed that training with the presence of some reverberation can helps the NN model performance at testing. Also, we showed that the NN performance can be significantly affected by the choice of hyperparameters used in the feature extraction step. In addition, we observed that training with moving sources can greatly improve the performance for moving sources DOA detection at testing.

In Chapter 5, a comprehensive study was conducted on the effect of window sizes, sequence lengths, SNR, sources angular speeds, on the four deep learning NN architectures considered. We found that the TCN and GRU architectures have comparable (best) performance. However, TCN is better at utilizing short sequences and maintaining good performance at high sources angular speeds. On the other hand, GRU outperforms TCN at lower speeds. However, it is slightly more sensitive to noise and requires a longer sequence length for best results.

In Chapter 6, further tests were done regarding the training data used to train neural networks for DOA estimation. We compared speech and WGN data. Our results indicate that using WGN for both training and testing performed best, yielding an average improvement of 11%, 9%, 7% and 4% using FNN, RNN, GRU and TCN, respectively, compared to training and testing on speech data.

Finally, in Chapter 7, we compared the performance of NN-based DOA detection systems using 2 microphones or 4 microphones, in uniform linear microphone arrays. We showed that increasing the number of microphones improves the DOA performance, as expected.

8.2 Future Work

As for future work, various experiments and tests can be done. For instance, one potential improvement we didn't consider is tracking the detected moving sources. A traditional way for doing this is by using a Kalman filter, modeling certain aspects of the moving source's dynamics, and integration of such a filter with a NN-based acoustic DOA detection system could be performed. Or, a fully NN-based tracking system could be designed to model the moving source's dynamics.

Other investigations could also be added to the current experiments. For example, the effect of combining both speech and WGN sources (or other types of sources) in the training data and if this can improve the performance at testing.

References

[1] A. Schmidt, "Multiple emitter location and signal parameter estimation", *IEEE Transactions on Antennas and Propagation,* vol. 34, pp. 276–280, March 1986.

[2] M. Omologo and P. Svaizer, "Use of the crosspower-spectrum phase in acoustic event location", *IEEE Transactions on Speech and Audio Processing,* vol. 5, pp. 288–292, May 1997

[3] S. Adavanne, A. Politis, J. Nikunen, and T. Virtanen, "Sound event localization and detection of overlapping sources using convolutional recurrent neural networks," *IEEE Journal of Selected Topics in Signal Processing,* vol. 13, pp. 34–48, Jan. 2019.

[4] S. Chakrabarty and E.A.P. Habets, "Multi-speaker DOA estimation using deep convolutional networks trained with noise signals," *IEEE Journal of Selected Topics in Signal Processing,* vol. 13, pp. 8–21, Jan 2019.

[5] S. Chakrabarty and E.A.P. Habets, "Multi-scale aggregation of phase information for complexity reduction of CNN-based DOA estimation," in *Proc. 27th European Signal Processing Conference (EUSIPCO),* A Coruña, Spain, Sept. 2-6, 2019, pp. 1-5.

[6] R. Takeda and K. Komatani, "Sound source localization based on deep neural networks with directional activate function exploiting phase information," in *Proc. 2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP),* Shanghai, China, March 20-25, 2016, pp. 405–409

[7] K. Guirguis, C. Schorn, A. Guntoro, S. Abdulatif, and B. Yang, "SELDTCN: Sound event localization & detection via temporal convolutional networks," in *Proc. Eur. Signal Process. Conf. (EUSIPCO),* 2021, pp. 16–20.

[8] S. Jung, J. Park, and S. Lee, "Polyphonic Sound Event Detection Using Convolutional Bidirectional LSTM and Synthetic Data-based Transfer Learning," in *Proc. 2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP),* May 2019, pp. 885–889.

[9] E. Cakir, G. Parascandolo, T. Heittola, H. Huttunen, and T. Virtanen, "Convolutional Recurrent Neural Networks for Polyphonic Sound Event Detection," *IEEE/ACM Transactions on Audio, Speech, and Language Processing,* vol. 25, Feb. 2017.

[10] P. Zinemanas, P. Cancela, and M. Rocamora, "End-to-end Convolutional Neural Networks for Sound Event Detection in Urban Environments," in *Proc. 24th Conference of Open Innovations Association (FRUCT),* pp. 533-539 April 2019.

[11] S. Adavanne, A. Politis, and T. Virtanen, "Localization, detection and tracking of multiple moving sound sources with a convolutional recurrent neural network," in *Proceedings of the Detection and Classification of Acoustic Scenes and Events Workshop* (DCASE Workshop), New York, NY, 2019.

[12] D. Diaz-Guerra, A. Miguel, and J. R. Beltran, "Robust sound source tracking using SRP-PHAT and 3D convolutional neural networks," *IEEE/ACM Trans. Audio. Speech. Lang. Process.*, 29, 300–311, 2021.

[13] Z. Huang, J. Xu, and J. Pan, "A regression approach to speech source localization exploiting deep neural network," in *Proc. 2018 IEEE Fourth International Conference on Multimedia Big Data (BigMM),* Xi'an, China, September 13-16, 2018, pp. 1–6.

[14] N. Ma, T. May, and G.J. Brown, "Exploiting deep neural networks and head movements for robust binaural localization of multiple sources in reverberant environments," *IEEE/ACM Transactions on Audio, Speech, and Language Processing,* vol. 25, pp. 2444–2453, Dec. 2017.

[15] N. Ma, J.A. Gonzalez, and G.J. Brown, “Robust binaural localization of a target sound source by combining spectral source models and deep neural networks,” *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, vol. 26, pp. 2122–2131, Nov. 2018.

[16] C. Pang, H. Liu, and X. Li, “Multitask learning of time-frequency CNN for sound source localization,” *IEEE Access*, vol. 7, pp. 40725–40737, March 2019.

[17] L. Perotin, R. Serizel, E. Vincent, and A. Guérin, “CRNN-based multiple DOA estimation using acoustic intensity features for ambisonics recordings,” *IEEE Journal of Selected Topics in Signal Processing*, vol. 13, pp. 22–33, Jan. 2019.

[18] P.-A., Grumiaux, S., Kitic, L., Girin, and A. Guerin, “Improved feature extraction for CRNN-based multiple sound source localization,” in *Proceedings of the 2021 EUSIPCO Conference*, August 23–27, Dublin, Ireland, 2021 (virtual conference).

[19] S. Sakavičius and A. Serackis, “Estimation of sound source direction of arrival map using convolutional neural network and cross-correlation in frequency bands”, in *Proc. 2019 Open Conference of Electrical, Electronic and Information Sciences (eStream)*, Vilnius, Lithuania, April 25, 2019, pp. 1–6.

[20] P. Vecchiotti, N. Ma, S. Squartini, and G.J. Brown, “End-to-end binaural sound localisation from the raw waveform,” in *Proc. 2019 IEEE International Conference on Acoustics, Speech and Signal Processing* (ICASSP), Brighton, UK, May 12-17, 2019, pp. 451–455.

[21] X. Xiao, S. Zhao, X. Zhong, D. Jones, E. Chng, and H. Li, “A learning based approach to direction of arrival estimation in noisy and reverberant environments,” in *Proc. 2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 2814–2818, Aug. 2015

[22] H. Hammer, S.E. Chazan, J. Goldberger, and S. Gannot, “Dynamically localizing multiple speakers based on the time-frequency domain,” *EURASIP Journal on Audio, Speech, and Music Processing*, vol. 2021, pp.1–10, April 2021

[23] J. S. Wilson, Ed., *Sensor Technology Handbook*. Boston, MA: Newnes, 2005.

[24] M. S. Brandstein and D. B. Ward, Eds., *Microphone Arrays: Signal Processing Techniques and Applications*. New York, NY, USA: Springer, 2001

[25] J. Loar, *The Sound System Design Primer*. Routledge, 2019.

[26] J.B. Allen and D.A. Berkley, “Image method for efficiently simulating small-room acoustics,” *Journal of the Acoustical Society of America*, vol. 65, pp. 943–950, Apr. 1979.

[27] E. Habets, “Room impulse response generator,” 2010. [Online]. Available: http://home.tiscali.nl/ehabets/rir generator.html

[28] M. Vorländer, *Auralization: Fundamentals of acoustics, modelling, simulation, algorithms and acoustic virtual reality*. Springer Science & Business Media, 2007.

[29] E.A.P. Habets and S. Gannot, “Generating sensor signals in isotropic noise fields,” *Journal of the Acoustical Society of America*, vol. 122, pp. 3464–3470, June 2007.

[30] J. S. Bendat and A. G. Piersol, *Random Data: Analysis and Measurement Procedures*. Wiley, 2010.

[31] E. Sejdić, Igor Djurović, and J. Jiang. "Time–frequency feature representation using energy concentration: An overview of recent advances." *Digital Signal Processing*, 1.19 (2009), pp.153-183.

[32] Y. Bengio, I. J. Goodfellow, and A. Courville. *Deep learning*, MIT Press, 2016. http://www.deeplearningbook.org

[33] G. Klambauer, T. Unterthiner, A. Mayr, and S. Hochreiter, “Self-normalizing neural networks,” in *Proc. 31st International Conference on Neural Information Processing Systems*, Long Beach, CA, USA, Dec. 4 – 9, 2017, pp. 972–981.

[34] J.Chung, C. Gulcehre, K. Cho, and Y. Bengio, “Empirical evaluation of gated recurrent neural networks on sequence modeling,” arXiv Preprint, arXiv:1412.3555, 2014.[Online]. Available: https://arxiv.org/abs/1412.3555

[35] C. Lea, M.D. Flynn, R. Vidal, A. Reiter, and G.D. Hager,“Temporal convolutional networks for action segmentation and detection”, in *Proc. IEEE Conference on Computer Vision and Pattern Recognition*, Honolulu, HI, USA, 22–25 July 2017, pp. 156–165.

[36] M. Grandini, E. Bagli, and G. Visani, “Metrics for Multi-Class Classification: an Overview,” *ArXiv200805756 Cs Stat*, Aug. 2020, Accessed: Dec. 03, 2021. [Online]. Available: http://arxiv.org/abs/2008.05756

[37] E. Bernard. *Introduction to Machine Learning*, Wolfram Media, Inc., 2021. http://www.wolfram.com/language/introduction-machine-learning

[38] E.A.P. Habets, *Room Impulse Response Generator*, Tech. Rep, Technische Universiteit Eindhoven, 2006.

[39] K. W. Cheuk, H. Anderson, K. Agres, and D. Herremans, “Nnaudio: an on-the-fly GPU audio to spectrogram conversion toolbox using 1D convolutional neural networks”, *IEEE Access*, vol. 8, pp. 161981–162003, Aug. 2020.

[40] V. Zue, S. Seneff, and J. Glass, “Speech database development at MIT: Timit and beyond,” *Speech communication*, vol. 9, pp. 351–356, April 1990.

[41] J. Rosado-Sanz, N. Rey-Maestre, D. Mata-Moya, M. P. Jarabo-Amores, M. Rosa-Zurera and J. L. Bárcena-Humanes, "Advantages of non-uniform linear arrays based on COTS elements in passive radar applications," *22nd International Microwave and Radar Conference (MIKON)*, 2018, pp. 199-203, doi: 10.23919/MIKON.2018.8405177.

[42] W. Falcon, "Pytorch lightning." https://github.com/PyTorchLightning/pytorch-lightning 3.6 (2019).

[43] N. S. Detlefsen et al. "TorchMetrics-Measuring Reproducibility in PyTorch." *Journal of Open Source Software* 7.70 (2022): 4101.

[44] D. Diaz-Guerra, A. Miguel, and J. R. Beltran, “gpuRIR: A python library for room impulse response simulation with gpu acceleration,” *Multimedia Tools and Applications*, pp. 1–19, 2020.

[45] Y. Luo, and Y. Jianwei, "FRA-RIR: Fast Random Approximation of the Image-source Method." *arXiv preprint arXiv:2208.04101* (2022).

[46] P. Virtanen et al., SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. Nature Methods, 17(3), 261-272, 2020

[47] Y.-Y. Yang et al., TorchAudio: Building Blocks for Audio and Speech Processing. preprint arXiv:2110.15018 (2021)

[48] M. Abadi et al. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org