Titre: Title:	Vision-Aided Deep Learning Solutions for Hybrid Beamforming and Beam Tracking
Auteur: Author:	Claudio Alkazzi
Date:	2022
Type:	Mémoire ou thèse / Dissertation or Thesis
Référence: Citation:	Alkazzi, C. (2022). Vision-Aided Deep Learning Solutions for Hybrid Beamforming and Beam Tracking [Master's thesis, Polytechnique Montréal]. PolyPublie. https://publications.polymtl.ca/10697/

POLYTECHNIQUE MONTRÉAL
affiliée à l'Université de Montréal

Vision-aided deep learning solutions for hybrid beamforming and beam tracking

CLAUDIO ALKAZZI
Département de génie électrique

Mémoire présenté en vue de l'obtention du diplôme de *Maîtrise ès sciences appliquées*
Génie électrique

Decembre 2022

POLYTECHNIQUE MONTRÉAL
affiliée à l'Université de Montréal

Ce mémoire intitulé :

Vision-aided deep learning solutions for hybrid beamforming and beam tracking

présenté par **Claudio ALKAZZI**
en vue de l'obtention du diplôme de *Maîtrise ès sciences appliquées*
a été dûment accepté par le jury d'examen constitué de :

Chahé NERGUIZIAN, président
Jean-François FRIGON, membre et directeur de recherche
François LEDUC-PRIMEAU, membre

DEDICATION

To my parents, for their unfailing love,

To my friends, for their immeasurable support,

and To Uncle George, may he rest in peace...

ACKNOWLEDGEMENTS

First and foremost, I would like to thank my director, Dr. Jean-François Frigon, for the opportunity he presented to me to pursue my research in this field that I adore. I am indebted to him for standing by me and making sure that my journey to attain my Master's in a foreign land went as smoothly as possible. I would also like to thank Hamed Hojatian for sharing his technical expertise and for always being there to answer my questions, even on weekend nights! Furthermore, I would like to extend my gratitude to my advisor, Dr. Georges Zakka El Nashef, for facilitating my communication with the Lebanese University.

Finally, I would like to thank Dr. Lama Séoud for inspiring me to incorporate deep learning solutions in my project, and Dr. Lise Safatly for advising me to embark on this journey away from home.

RÉSUMÉ

Dans les systèmes MIMO (Multiple Input Multiple Output) massifs, les méthodes traditionnelles d'estimation du canal deviennent très complexes, notablement dans les architectures hybrides. Pour améliorer la performance de HBF (Hybrid Beamforming), des solutions utilisant l'intelligence artificiel ont été introduits. Cependant, ces solutions nécessitent la connaissance du CSI (Channel State Information) et entraînent les modèles d'une manière supervisée. Ainsi, les solutions optimales doivent être calculées préalablement. Dans ce mémoire, deux solutions utilisant des données visuelles pour réduire la fréquence nécessaire d'estimation du canal sont introduites : HBF basé sur le LiDAR (Light Detection and Ranging) avec entraînement non supervisé, et un suiveur des faisceaux basé sur des séquences d'images.

Concernant la première solution, un LiDAR peut détecter des usagers et des obstacles en 3D dans la zone de couverture de la station. Il est donc utilisé comme entrée pour notre réseau de neurones. En adoptant l'apprentissage profond, un modèle non supervisé peut être créer afin d'effectuer la formation des faisceaux. Ainsi, la transmission du CSI n'est plus nécessaire durant le fonctionnement en ligne. Une fonction de perte pour maximiser le débit total est définie.

Pour la deuxième solution, l'objectif est de suivre les usagers quand ils se déplacent dans la zone de couverture. Un modèle LSTM (Long Short-Term Memory) supervisé est alors utilisée, prenant une séquence d'images et prévoyant le futur faisceau idéal. Le but est de réduire la fréquence nécessaire d'estimation du canal. De plus, il est nécessaire de réduire cette fréquence en utilisant des sources de l'information alternatifs au lieu du CSI pour ne pas augmenter l'utilisation des ressources fréquentielles. Deux modèles sont utilisés, dépendamment du type d'entrée considéré : ConvLSTM pour les images, et SimpleLSTM pour les faisceaux idéaux antérieures. Il est vérifié que l'information visuelle est utile pour suivre les usagers et prévoir les futurs faisceaux.

ABSTRACT

Traditional channel sensing techniques for hybrid beamforming in wireless communications become complex when applied to massive Multiple Input Multiple Output (MIMO) systems due to the increased number of transmitting and receiving antennas. Several deep learning solutions have been proposed to solve this issue. These novel approaches, leveraging artificial intelligence, have the potential to replace the traditional techniques due to the intrinsic statistical nature of wireless channels. However, many of the proposed techniques take the Channel State Information (CSI) as input, and many others use supervised learning to train their models, requiring that optimal solutions that are dependent on the CSI be calculated. Therefore, these solutions do not manage to reduce the overhead introduced with the transmission of the CSI. As a result, we present in this paper two vision-aided solutions to reduce the required frequency of transmission of the CSI: Light Detection and Ranging (LiDAR) enabled unsupervised learning for hybrid beamforming, and time-series beam tracking using images.

LiDAR can detect users and obstacles in the coverage zone of a base station, making it a noteworthy candidate for an input to a deep learning model. We use this fact as the basis for the first part of our work: the implementation of an unsupervised model for hybrid beamforming. The use of unsupervised learning eliminates the need for channel estimation in the online mode, which vastly reduces the communication overhead. Therefore, we implement a LiDAR-based unsupervised deep learning model for hybrid beamforming, using a custom loss function to maximize the sum rate.

For the second solution, we aim to track users as they move in the environment. We use a supervised Long Short-Term Memory (LSTM) model to take a sequence of images as input and output the best future beam. The goal is to reduce the frequency of performing channel estimation by predicting the future beams based on the motion of the user instead of the CSI. We use the ConvLSTM model to allow the processing of images in a neural network, and we build a SimpleLSTM model that takes in the previous beams as input instead of images. The goal is to verify that visual data can assist in beam tracking applications and to provide a simple yet efficient model to perform this task.

TABLE OF CONTENTS

DEDICATION iii

ACKNOWLEDGEMENTS iv

RÉSUMÉ v

ABSTRACT vi

TABLE OF CONTENTS vii

LIST OF TABLES ix

LIST OF FIGURES x

LIST OF SYMBOLS AND ABBREVIATIONS xi

CHAPTER 1 INTRODUCTION 1
1.1 Context 1
1.2 Definitions and core concepts 1
1.2.1 Communication Overhead 1
1.2.2 Massive MIMO 2
1.2.3 Beamforming in Massive MIMO mmWave Systems 3
1.2.4 Deep Learning 5
1.3 Problem and Motivation 5
1.4 Research Objectives 7
1.5 Contributions 8
1.6 Thesis Outline 8

CHAPTER 2 LITERATURE REVIEW 9
2.1 Hybrid Beamforming 9
2.2 Deep Learning 11
2.2.1 Images 11
2.2.2 LiDAR 13
2.2.3 LSTM and Recurrent Networks 16
2.3 ML-Powered Beamforming 18
2.3.1 Unsupervised Learning 18

2.3.2 Images for Beam Selection 22
2.3.3 LiDAR 23
2.3.4 Beam Tracking 25

CHAPTER 3 LIDAR BASED UNSUPERVISED DEEP-LEARNING FOR MASSIVE MIMO HYBRID BEAMFORMING 29
3.1 System Model 29
3.2 Implementation 29
3.3 Results and Discussions 35
3.4 Conclusion 37

CHAPTER 4 BEAM TRACKING USING LSTM 38
4.1 Implementation 38
4.1.1 Implementation of ResNet Beam Selection 39
4.1.2 Implementation of ConvLSTM with images 40
4.1.3 Implementation of Simple LSTM with labels 45
4.1.4 Implementation of ConvLSTM with Images: DeepSense Dataset . . . 46
4.2 Results and Discussions 48
4.3 Conclusion 52

CHAPTER 5 CONCLUSION 53
5.1 Summary of Work 53
5.2 Limitations of the Proposed Methods 54
5.3 Future Research 55

REFERENCES 57

LIST OF TABLES

Table 3.1 System settings for LiDAR based model 36
Table 4.1 Summary of results of beam selection and prediction tasks . . . 49

LIST OF FIGURES

Figure 1.1	Hierarchical Search	4
Figure 1.2	Narrow beam search	7
Figure 2.1	Hybrid beamforming architecture	11
Figure 2.2	Shallow vs. deep neural networks with identity mappings	12
Figure 2.3	ResNet block with 2 layers and a skip connection	13
Figure 2.4	ResNet18 architecture	14
Figure 2.5	PointNet block diagram	15
Figure 2.6	Architecture of LSTM network	17
Figure 2.7	LiDAR input structure for the ViWi and Raymobtime datasets	23
Figure 2.8	Architecture of GRU beam tracking model	26
Figure 2.9	Workflow for beam tracking using upsampled beam pattern images	28
Figure 3.1	Map of Rosslyn, the area where the dataset is simulated	30
Figure 3.2	Sample image of users and obstacles in Raymobtime s008 scenario	31
Figure 3.3	2D representation of 3D LiDAR histogram	32
Figure 3.4	High-level block diagram of LiDAR-based HBF model	33
Figure 3.5	Use of CSI in online and offline modes	35
Figure 3.6	Training curve for unsupervised LiDAR-based HBF	36
Figure 4.1	Transfer learning in ResNet18	39
Figure 4.2	Image data sample from the used scenario	41
Figure 4.3	Sequence-to-Sequence Autoencoder architecture	42
Figure 4.4	Video prediction results of ConvLSTM on MovingMNIST dataset	43
Figure 4.5	Input sequence shape for the LSTM	44
Figure 4.6	Probability distribution of the labels in the DeepSense dataset	47
Figure 4.7	Sample image from the DeepSense dataset taken from scenario 34	47
Figure 4.8	LSTM sliding window for sequence collection	49
Figure 4.9	LSTM sliding window input data leakage	50
Figure 4.10	Training and testing curves for the DeepSense dataset	52

LIST OF SYMBOLS AND ABBREVIATIONS

AP	Analog Precoder
BB	Baseband
CNN	Convolutional Neural Network
CSI	Channel State Information
DFT	Discrete Fourier Transform
DNN	Deep Neural Network
DP	Digital Precoder
FDP	Fully Digital Precoder
FNN	Feed-Forward Neural Network
GRU	Gated Recurrent Unit
HBF	Hybrid Beamforming
LiDAR	Light Detection and Ranging
LOS	Line-Of-Sight
LSTM	Long Short-Term Memory
MIMO	Multiple Input Multiple Output
mmWave	Millimeter Wave
NLOS	Non-Line-Of-Sight
RNN	Recurrent Neural Network
RF	Radio Frequency
RSSI	Received Signal Strength Indicator
SINR	Signal to Interference Noise Ratio
SNR	Signal to Noise Ratio
SVD	Singular Value Decomposition
ULA	Uniform Linear Array
V2I	Vehicle-to-Infrastructure

CHAPTER 1 INTRODUCTION

1.1 Context

As the demand for higher data rates in wireless communications increases, especially with the current surge of Internet of Things, technological advancements in 5G become imperative. For example, virtual reality applications require high data rates and low latency to ensure a smooth experience and combat motion sickness [1], and autonomous vehicles require low latency communication to ensure fast response times and user safety. The required data rates for virtual reality are estimated to be anywhere between 400 Mbps to 1 Gbps. To achieve these stringent requirements, 5G leverages technologies such as millimeter wave frequency bands, beamforming, and massive MIMO. The use of millimeter wave opens up a whole range of frequency resources to handle the increasing number of user equipment and data transmission capabilities [2], whereas massive MIMO incorporates a large number of antennas at the transmitter and receiver to ensure a high reliability, increased spectral efficiency, and a large capacity [3]. The increased number of antennas in massive MIMO provides more flexibility in the formation of beams, allowing the use of beamforming as an active method to direct the radiated power solely to the user, instead of transmitting in all directions [4]. The main focus of this thesis is to use deep learning powered algorithms to perform beamforming using a variety of types of visual data.

1.2 Definitions and core concepts

We define in this section some basic notions that will be used throughout this thesis. Since our work consists in using deep learning methods and visual data to reduce the communication overhead in beamforming, we split our core concepts into three categories: Definition of communication overhead, explanation of beamforming in MIMO systems, and summary of basic deep learning techniques.

1.2.1 Communication Overhead

As a signal propagates in air, it is distorted by the various obstacles present in the environment. Therefore, a knowledge of the channel is required for processing the transmitted signal to adapt it to the channel [5]. In order to estimate the channel, however, we need a set of signals - pilot signals, that are known to both the transmitter and receiver. That way, the receiver can compare the distorted signal to the original, thus estimating the channel using

any estimation technique such as Least Squares or Minimum Mean Squared Error.

Our interest is not in the channel estimation techniques themselves but in the transmission of the pilot signals. These signals do not belong to the core data that need to be sent, so transmitting them wastes communication resources that can otherwise be used to send useful data. Moreover, as the channel changes due to continuous alterations in the environment, we need to estimate the channel periodically, which leads to the need for transmitting pilot signals periodically as well. As a result, it is beneficial to explore different ways to reduce the frequency of transmission of the pilot signals. We will refer to the communication resources used to transmit the pilot signals as *communication overhead*, or simply *overhead*.

1.2.2 Massive MIMO

A key element of 5G communication systems is massive MIMO. MIMO refers to having multiple antennas at the transmitter, and often at the receiver too. By extension, Massive MIMO refers to MIMO systems having large numbers of antennas. The main benefits of massive MIMO are: spatial diversity, spacial multiplexing, and beamforming. We explain beamforming later, so in this section, we highlight how massive MIMO systems can achieve the two former benefits.

Diversity involves sending the same data across multiple resources to increase the reliability of a system. We can send these same data across different frequency resources for frequency diversity or across different time slots for time diversity. By implementing diversity, we ensure that if one propagation medium fails, the signal is still received through the other media. MIMO adds another type of diversity: space diversity. Using multiple antennas that are sufficiently spaced from one another, MIMO systems can create multiple propagation paths between the transmitter and receiver. Therefore, if for example the channel is not in our favor in one of the propagation paths, we can still use the other paths to transmit the signal. As a result, diversity increases the reliability of our system.

Furthermore, if our system already has enough diversity, we can choose to use the remaining resources for sending different messages at the same time. This is the concept of multiplexing, where we send multiple streams of data simultaneously. Therefore, if MIMO allows for space diversity by transmitting the same information across multiple paths, it can achieve spatial multiplexing by sending different data across these same paths. By increasing the number of antennas at the base station, massive MIMO can increase the spatial diversity and the capacity of a system to perform multiplexing. However, the main benefit of massive MIMO comes in the form of beamforming, which is explained in the following section.

1.2.3 Beamforming in Massive MIMO mmWave Systems

Limited by the frequency resources available in the sub-6GHz spectrum, finding available bandwidth to accommodate the ever-increasing number of users becomes difficult. The scarcity of these resources has pushed researchers to look into uncharted frequency bands in the Millimeter Wave (mmWave) spectrum. Migrating to mmWave, we have access to larger bandwidths which allow for gigabit-per-second data rates [6]. Furthermore, the higher frequencies (lower wavelengths) allow the construction of smaller antenna arrays that can fit into more compact appliances, paving the road to even slimmer phones and electronics. On the other hand, the Friis equations [7, 8] dictate that the propagation loss increases dramatically with higher carrier frequencies. To combat this loss, telecommunication companies will be forced to transmit at higher powers, making the migration to mmWave less appealing to them. We need more intelligent ways to compensate for the propagation losses.

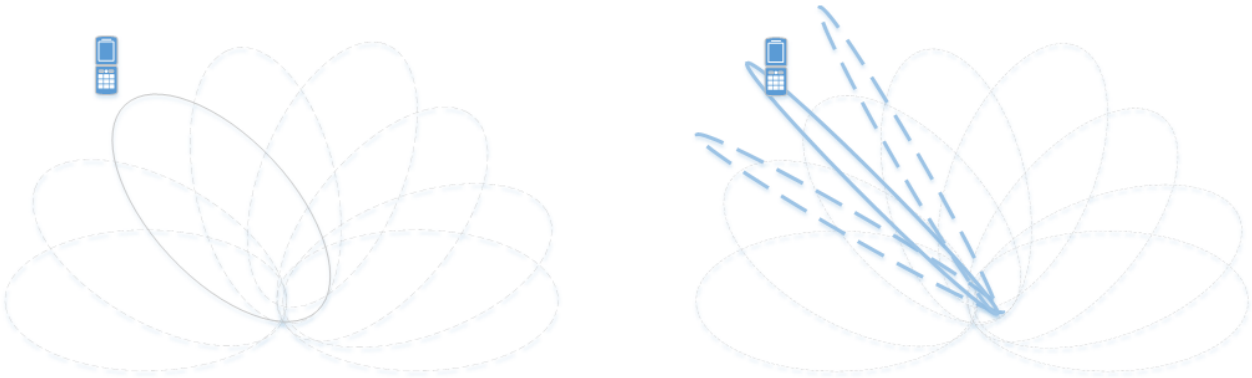
Beamforming is a potential candidate to ensure a sufficient received power while respecting an acceptable transmitting power constraint. Instead of transmitting the power in all directions, we can design directed antenna array beams aimed towards the user [9]. As a result, we transmit the same amount of energy but in a single direction, allowing the signal to be received at longer distances. Implementing beamforming, however, requires a large number of antennas. Fortunately, the use of massive MIMO supplies the large antenna arrays, and mmWave makes it possible to install these arrays in smaller spaces. Since the infrastructure of beamforming and the motivation to use it are both present, it is beneficial to summarize some beamforming concepts, architectures and algorithms in the following.

In its simplest form, beamforming consists in the use of multiple antennas which transmit signals that interact together to create the desired beam. In other words, the input signal is divided across multiple antennas in such a way that the radiating power from each antenna superposes constructively in the direction of the user, and destructively elsewhere. As a result, not only does this increase the received power due to constructive superposition, but also it decreases any potential interference with other users in other locations since the radiated power will not reach them. The question that remains is: In what manner do we divide and pre-condition the signal? One option is to transmit the same signal through all antennas but with a different phase, and another is to completely alter the signal that reaches each antenna. For each option, we have a dedicated beamforming architecture. The three main categories are analog, digital and hybrid beamforming.

Beamforming can be performed either in Radio Frequency (RF) or in the Baseband (BB) [9]. We call beamforming in RF analog beamforming since we manipulate the signal after modulation. Using phase shifters, we can shift the signal reaching each antenna in such a

way to obtain the desired radiation pattern. The advantages of analog beamforming, most notably when using 2-bit phase shifters, are its simplicity and the low cost of phase shifters. At the other extreme we find digital beamforming, where the signal is pre-distorted in the BB before modulation. This yields much more accurate beams, at the expense of costly analog-to-digital and digital-to-analog converters. Viewing analog and digital beamforming as total opposites, where analog provides simplicity and digital assures accuracy, we look for a solution that acts as a middle ground between these two extremes. This compromise is hybrid beamforming. Hybrid beamforming can be seen as a combination of analog and digital beamforming, in the sense that the signal is manipulated in the BB, then it is phase shifted accordingly in the RF. We will go into much more detail explaining hybrid beamforming in Section 2.1, but for the time being, note that hybrid beamforming decreases the costs corresponding to digital beamforming while increasing the accuracy obtained from analog beamforming.

Finally, we go over the main beam management algorithm used in 5G New Radio. One method is to perform a two-stage beams sweep [10] as shown in Figure 1.1. This corresponds to a hierarchical search to find the optimal narrow beam from a set of pre-computed beams. First, we conduct a wide beam sweep to locate the approximate location of the user. We use wide beams first instead of narrow beams directly to reduce the number of candidate beams. After the user is located, we perform another sweep, but this time a sweep of narrow beams which reside inside the chosen wide beam. This step is done to pinpoint the exact location of the user and serve the user equipment with the most refined beam possible.



(a) Wide beam sweep

(b) Narrow beam sweep

Figure 1.1 Hierarchical Search

1.2.4 Deep Learning

The only way for a computer to perform a certain task is if it has been given a specific algorithm to follow to complete said task. This started to change with the introduction of machine learning, where a computer can learn how to deal with a certain problem simply by looking at a given dataset and learning particularities in the data. Deep learning is a subset of machine learning where we train neural networks to approximate a usually non-linear function by a composition of several linear functions [11]. We will go into more detail about the structure of specific neural networks that we will be using, but in general, the two simplest forms of neural networks are the Feed-Forward Neural Network (FNN) and the Convolutional Neural Network (CNN). Staying faithful to the definition of a neural network, a FNN is a collection of fully-connected, linear layers where each node is a matrix multiplication. This makes FNNs perfect for approximating complicated non-linear functions, but we can see how the network becomes incredibly complex when the number of nodes and the size of the input increase. By increasing the number of training parameters, the model requires much more data, increasing the margin of error when these data are not available. Moreover, in image-based applications, using a FNN requires flattening the image into one dimension, loosing all information regarding spacial proximity between pixels. As a result, a FNN would perform poorly in image recognition and classification tasks. Here is where the CNN comes in: Using convolution products as our linear operators, a CNN can dramatically decrease the number of parameters to be trained, thus becoming perfect for higher dimensional inputs and deeper networks.

Furthermore, deeper neural networks suffer from the problem of vanishing and exploding gradients [12]. We also note that we can use deep learning in time-series problems by employing recurrent neural networks. We will go into more detail concerning architectures that solve the vanishing and exploding gradient problems, as well as specific recurrent neural networks for the processing of sequential data in Chapter 2. These will be pertinent in Chapter 4, where we use a complex neural network to extract information from images and need to process time-series data.

1.3 Problem and Motivation

Any classical beamforming algorithm, be it for analog, digital, or hybrid beamforming, will introduce some inefficiency into our system. This inefficiency can be in the form of a communication overhead for channel estimation or a delay due to the process of beam selection. In terms of hybrid beamforming, we require the channel state information to compute the

optimal hybrid precoders following an optimization problem discussed in Section 2.3.1. The need for the channel matrix introduces a communication overhead associated with the transmission of the pilot signals. Further, an optimal exhaustive optimization solution algorithm needs to check all precoders to find the best one. On the other hand, for purely analog beamforming, we mention above the hierarchical search used for finding the optimal beam. While being an improvement on the exhaustive narrow beam search (Figure 1.2), it is still a brute force algorithm, and a small delay is introduced until the best beam is found.

Both these issues are aggravated with the incorporation of massive MIMO. The increased number of antennas entails a larger channel matrix. Consequently, channel estimation becomes a much more daunting task, as the matrix inversions and arithmetic operations become more complex. This is evident in particular in the Least Squares and Minimum Mean Squared Error algorithms [5, 13]. Also, the large number of antennas allows the creation of more beams. This is advantageous since we can obtain narrower and more accurate beams, but it also increases their cardinality, making both the exhaustive and hierarchical searches more costly.

The worst part is that, while the classical methods are inefficient, they reap accurate results, making them attractive despite their complexity. In the case of hybrid beamforming, searching through all the possible analog precoders in the codebook guarantees finding the best one. This is further strengthened by using optimal yet inefficient methods for the calculation of the digital precoders; these methods include matrix inversions which we discussed to be costly with larger numbers of antennas. A similar argument can be made for analog beam selection, where the hierarchical search is sure to choose the best candidate beam. All these reasons make the use of these inefficient methods tempting, despite the high costs associated with them. As a result, any new method that promises to reduce the complexity of either of these tasks should also be able to compete with the optimal performance guaranteed by the classical methods.

Another problem that can vastly improve the performance of wireless communication systems if solved is the frequency of transmission of pilot signals. Due to changes in the environment, especially in areas of high mobility, the channel matrix will change often. Therefore, another area we can focus on alongside reducing the complexity of beamforming is reducing the necessary frequency of calculation of optimal beams. The result is that we not only obtain a more efficient algorithm for beamforming but also reduce the necessary frequency of employing it.

To summarize, the problematic lies upon three fronts: The overhead issues with traditional channel estimation, the inefficiency of the hierarchical search algorithm, and the frequency of transmission of pilot signals. These three problems result in the need to reserve more

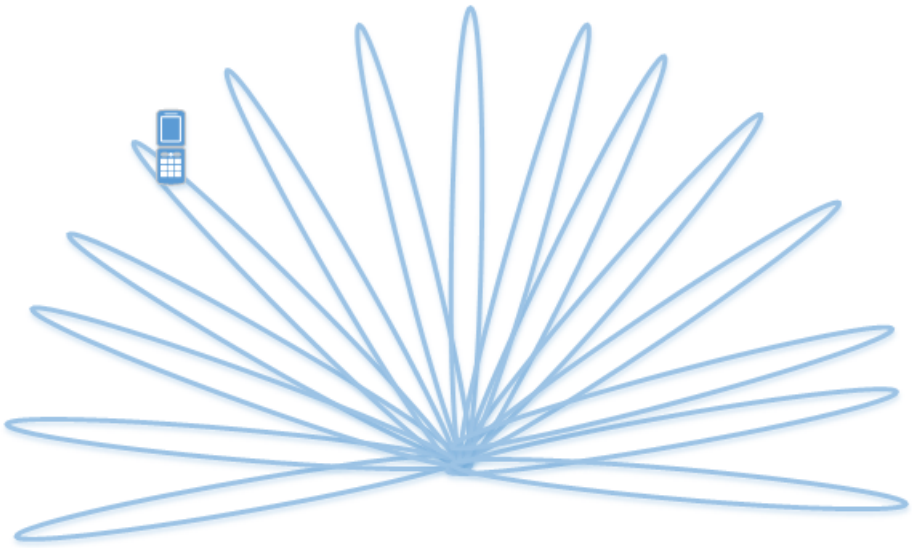


Figure 1.2 Narrow beam search

communication resources for maintaining a good connection and in a larger latency while searching for the beams. All these issues go against the core goals of 5G, and we need to look into methods to solve these issues. Therefore, other techniques must be found to speed up this process, in order to reap the benefits of beamforming without sacrificing the latency of the communication.

1.4 Research Objectives

We aim to integrate deep learning methods with alternative data sources to simplify the initial access process, thus reducing the communication overhead and decreasing the complexity of beam selection.

The problems outlined in the problematic section can both be addressed by integrating deep learning in wireless communications. By using alternative data sources that are readily available without the use of communication resources, we can speed up the beam selection process by reducing the overhead associated with frequent channel estimation.

Therefore, the aim of this research is to explore novel deep learning powered methods to:

1. Reduce the required frequency of transmission of pilot signals for channel estimation,
2. Use environmental information such as images (photos or videos) and LiDAR captures to replace the transmission of pilot signals, and
3. Leverage recursive neural networks for beam tracking.

1.5 Contributions

Our contributions include using deep learning solutions to better understand the channel conditions in a given environment and to reduce the communication overhead corresponding to the initial access. To reduce this overhead, we work on two fronts: Using LiDAR and unsupervised learning to reduce the complexity of computing the hybrid precoders in the present instant, and using recursive neural networks with supervised learning to reduce the latency by computing the optimal beams for the future instants. Thus, we first design a neural network that treats LiDAR data in the form of a 3D histogram and outputs the best hybrid precoders. Then, we turn our attention to a time-series based neural network to perform beam tracking. To achieve this beam tracking, we use images as input to track the user through time in the environment.

1.6 Thesis Outline

The thesis is organized as follows. In Chapter 2, we go over the major previous contributions in the fields of beamforming and machine learning in wireless communications. We also provide insight to various existing and specific deep learning models that will be used in our simulations. Chapters 3 and 4 contain details about our experiments, from the ideation and implementation to a discussion about the results. Chapter 3 is concerned with our first contribution: hybrid beamforming using LiDAR data. We describe the model and share our results, achievements, and shortcomings. Chapter 4 is concerned with beam tracking using time-series data taken from images. In this chapter, we enumerate the various steps we take to reach our final model, as well as the different approaches we take. Finally, Chapter 5 summarizes the thesis and provides insights into possible future work.

CHAPTER 2 LITERATURE REVIEW

Before tackling the contributions we put forth in the previous chapter, it is imperative that we go over previous work done in the field. We do so to understand what has been done before, what is yet to be implemented, and how our work fits in the grand scheme of things. We also need to expand upon some of the core concepts we explain above, for these will be the grounds on which we build our solutions. The literature review will contain notions about hybrid beamforming and deep learning, as well as a summary of the work done in the domain of vision-aided wireless communications.

This chapter is organized into three main parts. The first part is concerned with hybrid beamforming: From the definition to the general form of its implementation. In the second part, we go over important deep learning models that we will use in our study. This section will not provide an explanation of deep learning, but it rather highlights the architecture of the specific models we either use or are inspired from. Finally, the last part shows applications of machine learning in wireless communications. We describe how deep learning has been implemented with various types of inputs, whether visual or not, in order to understand how we can improve upon them or tackle uncharted territories.

2.1 Hybrid Beamforming

In a normal wireless communication scenario, the input signal will be distorted by the channel throughout its propagation in the environment and by the noise at the receiver. As a result, denoting by $\mathbf{x}$ and $\mathbf{y}$ the input and output signal vectors of a massive MIMO system respectively, $\mathbf{H}$ the channel matrix, and $\boldsymbol{\eta}$ the additive white Gaussian noise, the received signal can be expressed as

$$\mathbf{y} = \mathbf{H}\mathbf{x} + \boldsymbol{\eta} \tag{2.1}$$

By looking at the above expression, we can attempt to eliminate the effects on the channel by counteracting the influence of $\mathbf{H}$ on $\mathbf{x}$ by adding some pre-distortion and post-distortion operators at the transmitter and receiver respectively. This is the basic idea of beamforming, where we align our signal with the channel to transmit the power in the direction where it is most useful. One method to do so is with Singular Value Decomposition (SVD) based beamforming [14]. Here, we decompose the channel matrix $\mathbf{H}$ into left and right matrices $\mathbf{U}$ and $\mathbf{V}$ using

$$\mathbf{H} = \mathbf{U}\Sigma\mathbf{V}^H \tag{2.2}$$

where Σ is a diagonal matrix containing the singular values of $\mathbf{H}$. We can now express $\mathbf{y}$ as

$$\mathbf{y} = (\mathbf{U}\Sigma\mathbf{V}^H)\mathbf{x} + \boldsymbol{\eta} \tag{2.3}$$

So, if we transmit $\mathbf{Vx}$ instead of $\mathbf{x}$ (pre-processing), and if we receive $\mathbf{U}^H\mathbf{y}$ instead of $\mathbf{y}$ (post-processing) [15], we obtain

$$\mathbf{U}^H\mathbf{y} = \mathbf{U}^H\mathbf{H}(\mathbf{Vx}) + \mathbf{U}^H\boldsymbol{\eta} \tag{2.4}$$

$$\mathbf{U}^H\mathbf{y} = \mathbf{U}^H\mathbf{U}\Sigma\mathbf{V}^H(\mathbf{Vx}) + \mathbf{U}^H\boldsymbol{\eta} \tag{2.5}$$

$$\mathbf{U}^H\mathbf{y} = \Sigma\mathbf{x} + \mathbf{U}^H\boldsymbol{\eta} \tag{2.6}$$

Since $\mathbf{U}$ and $\mathbf{V}$ and orthogonal and unitary matrices (as a result of SVD), $\mathbf{U}^H\boldsymbol{\eta}$ is still a white Gaussian noise. Furthermore, we see that by transmitting and receiving in the direction of the eigenvectors, we can leverage the fact that Σ is a diagonal matrix and parallelize the detection of the received vector using

$$\tilde{y}_i = \sigma_i x_i + \tilde{\eta}_i \tag{2.7}$$

where $\tilde{y}_i = \mathbf{U}_i^H\mathbf{y}$ and $\tilde{\eta}_i = \mathbf{U}_i^H\boldsymbol{\eta}$ are elements of the post-processed received signal, $\mathbf{U}_i$ is the i^{th} column of $\mathbf{U}$, and σ_i is the i^{th} entry in the diagonal matrix of singular values Σ of $\mathbf{H}$.

This is the most general idea of beamforming, but it is computing and resource intensive, specifically in massive MIMO systems with large number of antennas. Specific architectures and algorithms expand upon the above notions to achieve specific tasks. One of these architectures is hybrid beamforming shown in Figure 2.1. As mentioned in Section 1, hybrid beamforming is a combination of analog and digital beamforming, where an analog precoder is used to tweak the signal in the RF band, and a digital precoder is used in the BB. Let $\mathbf{F}_{RF}$ and $\mathbf{F}_{BB}$ be the Analog Precoder (AP) and Digital Precoder (DP) respectively, and let $\mathbf{W}_{RF}$ and $\mathbf{W}_{BB}$ be the analog and digital combining matrices respectively. Given the input signal $\mathbf{x}$, we transmit $\mathbf{F}_{RF}\mathbf{F}_{BB}\mathbf{x}$ [17]. Using the same logic for the combination matrices, the final received signal becomes

$$\mathbf{y} = \mathbf{W}_{BB}^H\mathbf{W}_{RF}^H\mathbf{H}\mathbf{F}_{RF}\mathbf{F}_{BB}\mathbf{x} + \mathbf{W}_{BB}^H\mathbf{W}_{RF}^H\boldsymbol{\eta} \tag{2.8}$$

We can make some comparisons with (2.4). $\mathbf{W}_{BB}^H\mathbf{W}_{RF}^H$ is analogous with $\mathbf{U}^H$, and $\mathbf{F}_{RF}\mathbf{F}_{BB}$ is analogous with $\mathbf{V}$, so it can be seen how using these analog and digital precoders and combiners can be considered beamforming. The goal in the following sections and in the first

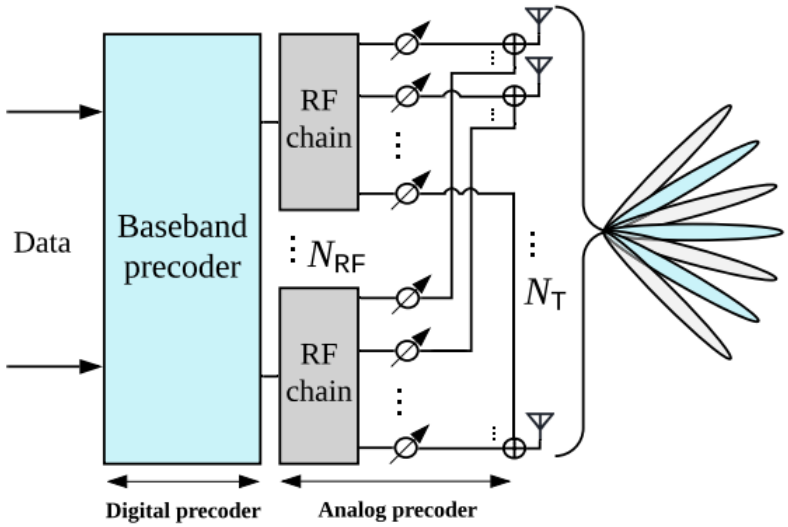


Figure 2.1 Hybrid beamforming architecture [16] ©2021 IEEE

part of our research is to find deep learning powered methods to calculate these precoders and combiners to achieve a more efficient beamforming algorithm.

2.2 Deep Learning

As we will be using deep learning in our simulations, a quick review of pertinent models is helpful. We go over image classification and object detection models used on both images and LiDAR captures. Finally, we give a brief introduction to Long Short-Term Memory (LSTM), as it will be useful when treating time-series data.

2.2.1 Images

Image recognition and classification are some of the simplest deep learning tasks to implement due to the high availability of pre-trained models and online tutorials. Even simple CNNs can be employed with supervised learning to perform simple tasks with images as inputs. However, if we are to attain the full potential of images in wireless communications, a field which is not a particularly interesting field in the computer science world, we, as electrical engineers, need to introduce these notions into our domain. Residual networks [18] in particular are of use in our applications for two reasons: the existence of pre-trained models on exhaustive datasets such as the COCO dataset [19] and their ability to train deeper models.

Why would we be interested in pre-trained models, especially when they are trained on

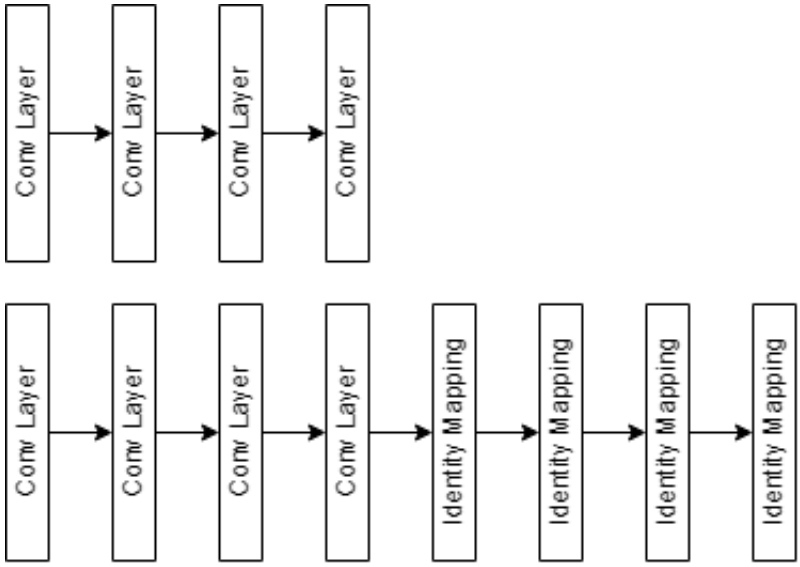


Figure 2.2 Shallow (top) vs. deep (bottom) neural networks with identity mappings

datasets that have nothing to do with our applications? The answer lies in transfer learning. According to [20], transfer learning is "the ability of a learning algorithm to exploit commonalities between different learning tasks to share statistical strength and transfer knowledge across tasks". That is, if a model is trained with a large exhaustive dataset of images for example, the learned parameters can be "transferred" to another machine learning task that may use images but to achieve some other objective. This fact, alongside the effectiveness of ResNet18 for example in image classification, is why we are interested in residual networks.

The motivation to use residual networks comes from the difficulty of stacking extra layers in deeper neural networks. In addition to the vanishing/exploding gradients problem [12], we encounter the problem of degradation. To explain degradation, imagine we have a shallow network with m layers. Now take a new deeper network with n layers, where $n > m$. We would expect that the deep network would perform at least as well as the shallow one since the deep one can mimic the m layers of the shallow network and set the remaining layers to identity mappings, as seen in Figure 2.2. However, what really happens is a saturation of learning in the deeper model. Contrary to the intuitive explanation, degradation could not be caused by overfitting since it can also be observed in the training phase, a phase where we do not expect to find symptoms of overfitting. Also, a mere reorganization of the network can solve this problem, as seen in [21]. By simply introducing *information highways* (or *skip connections*), [21] is able to train very deep neural networks with minimal degradation problems. Inspired by this approach, [18] *constructs* a large network by concatenating small blocks of residual layers and adding an identity mapping.

More rigorously, consider $\mathcal{H}(\mathbf{x})$ as a subset of mappings of a neural network. [18] argues that if, for example, a certain problem has an identity mapping as the optimal solution, then a deep and a shallow concatenation of identity mappings should yield no training error difference. However, due to the degradation problem, that is not the case, as driving the weights of a neural network to the identity mapping seems to be difficult. Following the assumption that it is easier for a model to drive layers to zero, the authors aim to train a residual function $\mathcal{F}(\mathbf{x}) := \mathcal{H}(\mathbf{x}) - \mathbf{x}$. The final weight then becomes $\mathcal{F}(\mathbf{x}) + \mathbf{x}$. Figure 2.3 visualizes this operation. As a result, if the solution is indeed the identity mapping, then $\mathcal{F}(\mathbf{x})$ can simply be driven to zero. The authors verify that the identity mapping is a good initialization, as their trained residuals are usually relatively small.

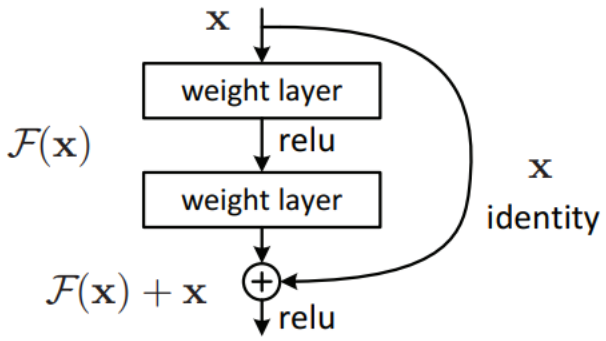


Figure 2.3 ResNet block with 2 layers and a skip connection (link)

Figure 2.3 shows a block inside the ResNet network. We can add these blocks one after the other to increase the depth of the model. Existing models include ResNet18, ResNet34, ResNet50, and ResNet101, where the number indicated the depth of the model. To obtain the ResNet18 model, we concatenate residual blocks. The final network becomes as shown in Figure 2.4.

2.2.2 LiDAR

At first glance, treating 3D LiDAR information should not be any different than 2D data; we can simply replace 2D-CNNs by 3D-CNNs. One example of that is VoxNet [23]. VoxNet takes as input a point cloud, and it extracts an occupancy grid by discretizing the space into a histogram. The dimensions of the occupancy grid are predefined, so the input size will always be constant; thus, we can use this grid as input to the neural network. We can then use any 3D-CNN with final fully connected layers as an architecture for our model. This allows the customization of the model for different applications. However, the use of 3D

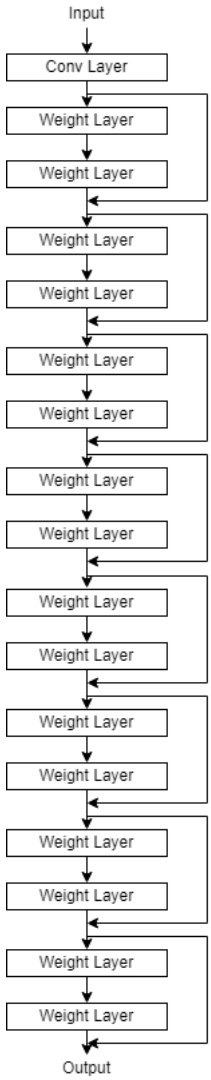


Figure 2.4 ResNet18 architecture

convolution products dramatically increases the complexity of the model [24]. In our case, this increased complexity is not warranted since most of the space is in reality empty. This is evident when looking at the nature of the environment, where most of the coverage area consists of an empty sky. As a result, the 3D convolutions will mostly be done on sparse tensors.

The above issues motivate the creation of a new type of model for LiDAR data that works directly on the point cloud, without the need of occupancy grids or voxelization. The state-of-the-art, which also responds to our needs, is PointNet [22]. PointNet takes as input the x, y, and z coordinates of every point in the point cloud, meaning that the input is $\{P_i | i = 1, ..., n\}$, where each $P_i \in \mathbb{R}^3$ is the set of (x, y, z) coordinates of the point. On

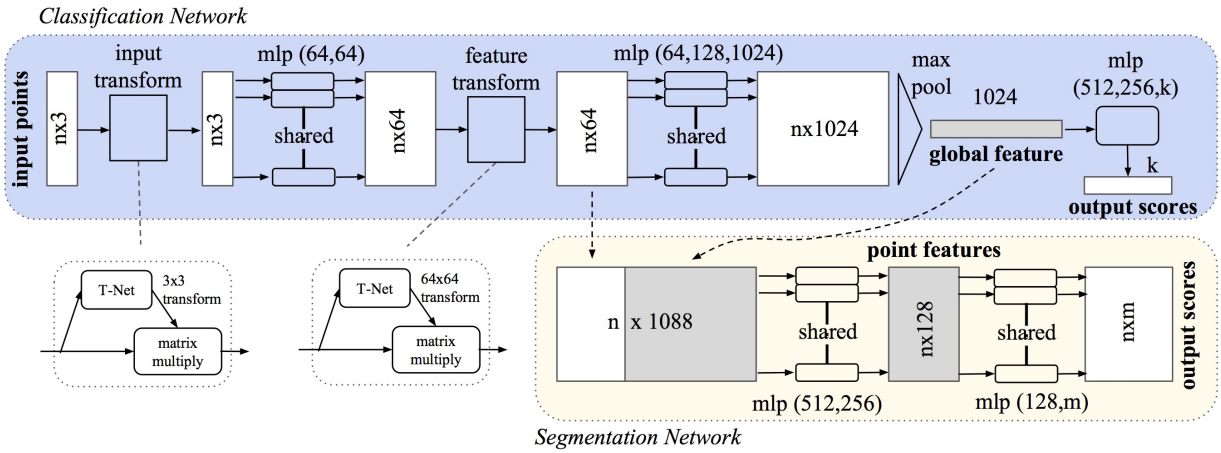


Figure 2.5 PointNet block diagram. [22]

the other hand, seeing that a point cloud is just a point *set* in $\mathbb{R}^n$, it must, by definition, follow these three properties: **unordered**, **invariance under rigid transformations**, and **interaction among points**. The last property will be satisfied when using a CNN, for the convolution product innately takes the product of a point with its neighbors, ensuring that we highlight the relations between nearby points. This leaves us with the first two problems that need to be addressed. Figure 2.5 shows a block diagram of PointNet.

First, we need to address the invariance to input order. To satisfy order invariance, the function f we are learning should be symmetric. The authors realize that we can approximate a general function f by a "symmetric function on transformed elements". Formally:

$$f(\{x_1, ..., x_n\}) \approx \gamma \circ g(h(x_1), ..., h(x_n)), \tag{2.9}$$

where h and γ are multi-layer perceptrons, and g is a symmetric function. As a result, since g is symmetric, f will definitely be symmetric. Therefore, as long as we subject our network to a final symmetric function, we are certain that our network is order invariant. The authors found that max-pooling is the best symmetric function in most applications.

Furthermore, to ensure rotation invariance, we can simply apply a transformation matrix to our input point cloud. This transformation matrix, denoted by $\mathbf{A}$, can be predicted by a smaller neural network (T-net). The T-net will learn to rotate the input in such a way that, no matter how we initially rotate the object to be classified, the model will always be able to correctly classify the object without taking into account said rotation. The only complicated part is training the T-nets, for they are neural networks after all. Since they are part of the entire model (See Figure 2.5), one might assume that no extra processing should

be done, and that they can be trained end-to-end along with the rest of the network. This, in essence, is correct, but the authors find that it is difficult to learn these T-nets without further constraints. As a result, the authors recommend that the transformation networks be unitary since it is found that we may obtain more stable results. To achieve this, we add a regularization term $\mathcal{L}_{reg}$ to the initially required loss function $\mathcal{L}_0$. Knowing that a square matrix is orthogonal if $\mathbf{A}\mathbf{A}^T = \mathbf{I}$, we can define:

$$\mathcal{L}_{reg} = \left|\left|\mathbf{I} - \mathbf{A}\mathbf{A}^T\right|\right|_F^2 \tag{2.10}$$

where $||.||_F$ denotes the Forbenius norm. Lastly, the final loss function of any model using PointNet as its base becomes:

$$\mathcal{L} = \mathcal{L}_0 + \mathcal{L}_{req} \tag{2.11}$$

Combining all the above additions, we successfully guarantee that PointNet will follow the three required properties mentioned above. The symmetric function ensures that the points are unordered, the T-Nets make the objects represented by the point clouds invariant under rigid transformations, and the use of CNN takes the interaction between neighboring points into consideration.

2.2.3 LSTM and Recurrent Networks

As mentioned before, we can use recurrent neural networks to treat time-series data. However, these networks, in their basic form, suffer from the vanishing gradient problem. With larger input sequences, recurrent networks tend to *forget* important information gathered from older elements of the sequence while the newer data is being fed. Long Short-Term Memory (LSTM) [25] intends to solve this problem. The structure of the LSTM is shown in Figure 2.6. We see that the output of the cell depends on three pieces of information: the input of the cell at a given step, the cell state at the same step, and the hidden state taken from the previous step. The cell state denotes the state of the long term memory at the current step, and the hidden state is the output of the cell at the previous state. These values are tweaked using *gates*. The forget gate dictates how much of the long term memory - the cell state, should be forgotten given the hidden state. The input gate dictates how important the new set of input information is and how much it should affect the current cell state. Finally, the output gate updates the cell state and hidden state according to the outputs of the previous gates. These gates constitute the main trainable parameters of an LSTM cell. Note that this is a simplified explanation considering only the relevant information for our project. For a more sophisticated definition of LSTM, please consult [25, 26].

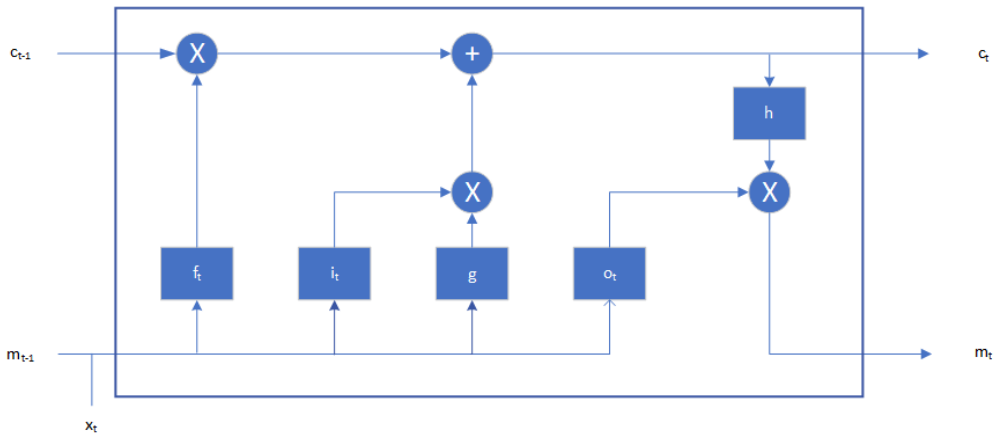


Figure 2.6 Architecture of LSTM network

The implementation of the LSTM found in [25] is complicated and outdated, so we find a more efficient one in [26], described by the equations below:

$$\begin{aligned} i_t &= \sigma(W_{ix}x_t + W_{im}m_{t-1} + W_{ic}c_{t-1} + b_i) \\ f_t &= \sigma(W_{fx}x_t + W_{mf}m_{t-1} + W_{cf}c_{t-1} + b_f) \\ c_t &= f_t \odot c_{t-1} + i_t \odot g(W_{cx}x_t + W_{cm}m_{t-1} + b_c) \\ o_t &= \sigma(W_{ox}x_t + W_{om}m_{t-1} + W_{oc}c_t + b_o) \\ m_t &= o_t \odot h(c_t) \end{aligned} \tag{2.12}$$

where W_{rs} is the weight matrix from gate r to input s, b is a bias, i is the input gate, f is the forget gate, and o is the output gate. c and m are the cell and output activation vectors respectively, and x_t is the input to the LSTM at instant t. g and h are non-linear activation functions, σ is the sigmoid activation function, and $\odot$ denotes the Hadamard product.

We are also interested in feeding the LSTM with a sequence of images. Seeing that images are 2-dimensional, we cannot use a normal LSTM without flattening the image first. The large input size of a flattened image, alongside the spatial information lost between neighboring pixels after the flattening operation, render a normal LSTM obsolete when dealing with images. We need a specific type of LSTM that deals specifically with images. We find that ConvLSTM [27] is the model we need. Using convolution products, similar to how CNNs made it possible to use neural networks on images, ConvLSTM makes it possible to use recurrent neural networks on images. With minor notation changes from (2.12) due to the inputs, cell outputs, etc. using 3D tensors instead of vectors, and by replacing matrix multiplications with convolution products, we describe ConvLSTM in (2.13).

$$
\begin{aligned}
i_t &= \sigma(W_{ix} * \mathcal{X}_t + W_{im} * \mathcal{M}_{t-1} + W_{ic} * \mathcal{C}_{t-1} + b_i) \\
f_t &= \sigma(W_{fx} * \mathcal{X}_t + W_{mf} * \mathcal{M}_{t-1} + W_{cf} * \mathcal{C}_{t-1} + b_f) \\
\mathcal{C}_t &= f_t \odot \mathcal{C}_{t-1} + i_t \odot g(W_{cx} * \mathcal{X}_t + W_{cm} * \mathcal{M}_{t-1} + b_c) \\
o_t &= \sigma(W_{ox} * \mathcal{X}_t + W_{om} * \mathcal{M}_{t-1} + W_{oc} * \mathcal{C}_t + b_o) \\
\mathcal{M}_t &= o_t \odot h(\mathcal{C}_t)
\end{aligned}
\tag{2.13}
$$

2.3 ML-Powered Beamforming

As previously discussed, classical beamforming techniques are costly and introduce a communication signalling overhead. The difficulty arises from the intrinsically statistical nature wireless communications: from the channel model to the distribution of the users, and all the errors that may occur in between. This statistical nature, however, can be a blessing if we are able to integrate machine learning based algorithms in our solutions. Besides, machine learning, and subsequently deep learning, excels in wireless communication problems that are statistical in nature [28]. Since the Channel State Information (CSI) is required for the classical techniques, the first idea that comes to mind is using the CSI as an input to the neural network. [29] replaces the exhaustive search with a 1D CNN that predicts the transmit and receive beams in a hybrid architecture using the vectorized CSI as input. Furthermore, due to the sparsity of mmWave channels, the CSI can be inferred from compressive measurements, reducing the channel estimation overhead [30].

While this solution drastically reduces the complexity of the system by eliminating the exhaustive search and addresses the problem of channel estimation overhead, it can still be improved upon by reducing the need of the CSI and by using alternate sources of data that do not use communication resources to obtain. We explore examples of such data sources in the following sections.

2.3.1 Unsupervised Learning

The use of unsupervised learning is interesting since we do not need to calculate optimal results to be used as labels during training. We can simply deploy the system, and the model will learn by itself how to find the best results by minimizing a defined loss function. An example of using unsupervised learning is shown in [16], where the authors use Received Signal Strength Indicator (RSSI) to find the best hybrid precoders. The benefit of using RSSI as an input is that it can be directly and easily measured from the received signal

without any extra transmission of information. This makes it a perfect candidate for low overhead channel estimation and beam selection. We use the example of RSSI to formulate a distinction between supervised and unsupervised learning shown in [16, 31].

In [31], the authors introduce a CNN that calculates the weights of a hybrid beamformer. This method falls under the category of supervised learning since the optimal solution is required during the training of the model. In other words, in addition to the complexity of training the neural network, additional computation should be expended to calculate the optimal hybrid precoder; the calculation we are aiming to avoid in the first place.

To fully demonstrate the consequences of using supervised learning, we summarize the process of calculating the optimal hybrid precoders. For the sake of simplicity, for all optimization problems mentioned in this solution, we do not include the constraints since they are not crucial for the understanding of the principal concept - the drawbacks of calculating labels in supervised learning. Please note that these constraints do in fact exist and are necessary if one requires to implement this method. At any rate, the optimal solution in the case of hybrid beamforming is the pair of analog and digital precoders $\mathbf{A}^\star$ and $\mathbf{W}^\star$ that maximize the sum rate. First, we find the optimal Fully Digital Precoder (FDP) $\mathbf{u}_m = \mathbf{A}\mathbf{w}_m$ for user m by solving the following optimization problem

$$\{\mathbf{u}_m^\star\} = \max_{\{\mathbf{u}_m\}} \sum_{\forall m} \left(1 + \frac{\left|\mathbf{h}_m^H \mathbf{u}_m\right|^2}{\sum_{j \neq m} \left|\mathbf{h}_m^H \mathbf{u}_j\right|^2 + \sigma^2}\right) \tag{2.14}$$

We then reconstruct the hybrid precoders by approximating the obtained $\mathbf{u}_m^\star$ using minimum mean squared error as shown below

$$\mathbf{A}^\star, \{\mathbf{w}_m^\star\} = \min_{\mathbf{A}, \{\mathbf{w}_m\}} \sum_{\forall m} \left|\left|\mathbf{u}_m^\star - \mathbf{A}\mathbf{w}_m\right|\right|^2 \tag{2.15}$$

We see that, to calculate the label of each data point in the dataset, we first need to solve an optimization problem to find the optimal FDP, then we need to solve another double optimization problem to find both $\mathbf{A}^\star$ and $\{\mathbf{w}_m^\star\}$. This makes the deployment of this method in the real world very difficult since, every time the channel conditions change, for example at the installation of every new base station, we need to retrain the model with a new set of data; ergo, we need to resolve the above optimization problems. This is not feasible, and we need an alternative which employs self-supervised learning, a type of learning where we do not need to explicitly provide the optimal labels. Due to the prevalence of the term "unsupervised learning" in the literature when "self-supervised learning" is concerned, especially in our domain of wireless communications, we shall use these two terms interchangeably.

Luckily, the authors of [16] manage to use RSSI in an unsupervised model. The difference is that, while the channel information is still needed, it is only needed in the training phase for the calculation of the loss function, and not for the calculation of labels. In other words, the prediction of the precoders does not use full channel information in the online phase; during the training phase, once the precoders are predicted, the channel information is used only to compute the loss function given the output of the neural network, and it is never used in the online phase during which the model is fully deployed. This independence from labels and full channel state information makes this solution robust against changes in the environment and allows plug-and-play deployment in any new setting. If any changes to the channel conditions occur, the model can simply be retrained without the need of computing optimal solutions normally required for supervised learning.

As for the neural network, the authors use a simple CNN that takes the RSSI values as input and outputs the hybrid precoder. The prediction of the AP is a classification problem since we are choosing the best precoder from a codebook, while that of the DP is a simple regression model. Therefore, the final fully connected layer for the AP is subjected to a softmax function to output the probabilities of each precoder to be the optimal solution. We also differentiate between two models: HBF-Net and AFP-Net. HBF-Net simply outputs in parallel the AP and DP directly, while AFP-Net attempt to align the analog and digital precoders. Since HBF-Net predicts the AP and DP independently, its implementation is easy and computationally simple, but there is no guarantee that the analog and digital precoders are aligned. AFP-Net ensures that this alignment is satisfied by calculating the FDP and projecting it on the AP by using

$$\mathbf{W} = \mathbf{A}^{\dagger}\mathbf{U} \tag{2.16}$$

where $\mathbf{U}$ is the FDP, $\mathbf{A}^{\dagger}$ is the pseudo-inverse of the AP, and $\mathbf{W}$ is the resulting DP.

Now, when using any of the above methods, be it HBF-Net or AFP-Net, we need a custom loss function for the unsupervised learning task. The authors resort to maximizing the sum rate of all the active users. The optimization problem can thus be expressed as

$$\max_{\mathbf{A},\mathbf{w}_u} R(\mathbf{A},\mathbf{W}) \tag{2.17}$$

$$\text{s.t. } \mathbf{A} \in \mathcal{A} \tag{2.18}$$

$$\sum_{\forall u} \mathbf{w}_u^H \mathbf{A}^H \mathbf{A} \mathbf{w}_u < P_{max}, \tag{2.19}$$

$$\text{where } R(\mathbf{A},\mathbf{W}) = \sum_{\forall u} log_2\Big(1 + SINR(\mathbf{A},\mathbf{w}_u)\Big) \tag{2.20}$$

$$\text{and } SINR(\mathbf{A},\mathbf{w}_u) = \frac{|\mathbf{h}_u^H \mathbf{A} \mathbf{w}_u|^2}{\sum_{j\neq u} |\mathbf{h}_u^H \mathbf{A} \mathbf{w}_j|^2 + \sigma^2}. \tag{2.21}$$

$\mathbf{w}_u$ is the DP for user u, and P_{max} is a power constraint to be respected. Keeping the above in mind, the formulation of the loss functions for the two models becomes evident. Knowing that the loss function is minimized through training, we can maximize the sum rate by minimizing the negative sum rate. As a result, the loss function for HBF-Net is defined as

$$\mathcal{L}_{HBF} = -\sum_{l=1}^{L} p_{a_{(l)}} R(\mathbf{A}_{(l)}, \overline{\mathbf{W}}) \tag{2.22}$$

where L is the codebook length, p_a is the output of the softmax function, $\overline{\mathbf{W}}$ is the predicted DP, and $\mathbf{A}_{(l)}$ is the l^{th} precoder of $\mathcal{A}$. We can easily see that this loss function not only maximizes the probability of choosing the correct AP but also minimizes the probability of choosing any incorrect ones. Moreover, since HBF-Net jointly predicts the AP and DP, the loss function simultaneously finds the DP $\overline{\mathbf{W}}$ which maximizes the sum rate. As for AFP-Net, since the FDP is calculated first, we need a separate loss function for the FDP task, and another for the AP task. The FDP is maximized using the following loss function

$$\mathcal{L}_{FDP} = -R(\overline{\mathbf{U}}) \tag{2.23}$$

$$\text{where } R(\mathbf{U}) = \sum_{\forall u} log_2\Big(1 + SINR(\mathbf{u}_u)\Big) \tag{2.24}$$

where $\overline{\mathbf{U}}$ is the output of the FDP task, and $\mathbf{u}_u$ is the FDP for user u. The DP $\tilde{\mathbf{W}}$ is then calculated using (2.16), and the choice of the AP is optimized in a similar fashion as in (2.22), namely

$$\mathcal{L}_{AP} = -\sum_{l=1}^{L} p_{a_{(l)}} R(\mathbf{A}_{(l)}, \tilde{\mathbf{W}}) \tag{2.25}$$

Finally, the final loss function to be minimized becomes

$$\mathcal{L}_{AFP} = \mathcal{L}_{FDP} + \mathcal{L}_{AP} \tag{2.26}$$

2.3.2 Images for Beam Selection

The simplest task that can be thought of when using images in wireless communications is blockage prediction, for we can easily detect the presence of a user or obstacle in the field of view. As seen in Section 1.2.3, mmWave suffers from low penetration ability, so in addition to the use of beamforming, we can predict whether the user is blocked from the line of sight of the base station and mitigate the resulting problems using proactive handoff [32]. Furthermore, using the images, we can infer the location of the user and select the best beam to serve it. These two tasks are tackled in [32].

For blockage prediction, ResNet-18 is used to detect the presence of the user. While this is simple, in the case where a user is not detected, we cannot know whether this is due to the user being blocked or not existing in the first place. As a result, the visual information alone is not a sufficient predictor of blockage; the sub-6 GHz channels are needed to differentiate between blocked and absent users. Taking all this into consideration, let $G_\Theta(X, \mathbf{h}_u^{sub-6})$ be a prediction function denoting a neural network with a set of parameters Θ, and let $b_u \in \{-1.0, 1\}$ represent an absent, unblocked and blocked link status respectively. G_Θ should be trained to solve the following problem:

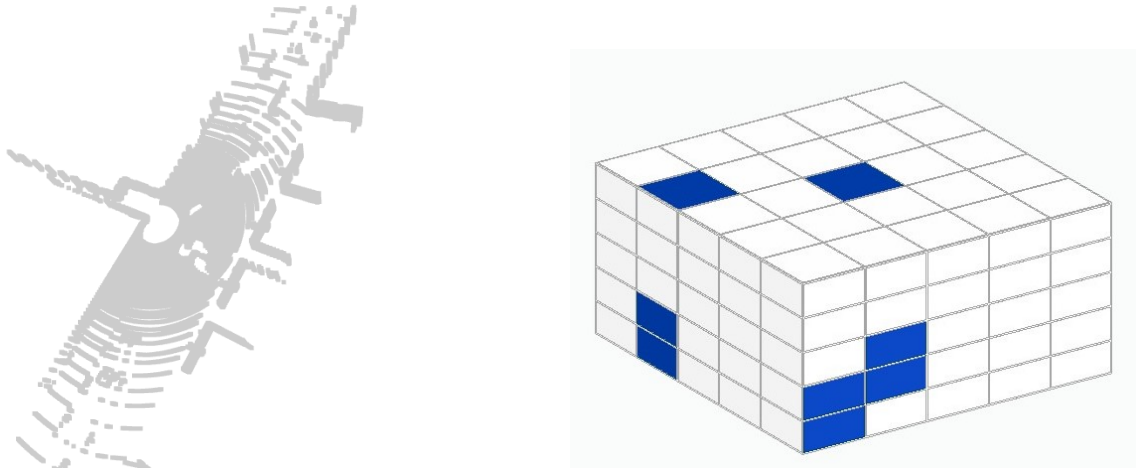
$$\max_{G_\Theta(X, \mathbf{h}_u^{sub-6})} \prod_{u=1}^{U} \mathbb{P}(\hat{b}_u = b_u | (X, \mathbf{h}_u^{sub-6}) \tag{2.27}$$

where $\hat{b}_u$ is the predicted link status, $\mathbb{P}(\hat{b}_u = b_u | (X, \mathbf{h}_u^{sub-6})$ is the probability of guessing the link status correctly, and U is the number of user positions.

On the other hand, for the beam selection task, it is sufficient to train yet another ResNet-18 neural network, denoted by $f_\Theta(X)$, to find the best beamforming vector $\mathbf{f}^\star$ from the input image $X \in \mathbb{R}^{H \times W \times C}$, where $\mathbf{f}^\star \in \mathcal{F}$ is the optimal beam from codebook $\mathcal{F}$ calculated from (2.28), and H, W, C are the height, width, and channels of the image X. Since we are using supervised learning, the labels should be calculated a priori. Thus, the optimal beamforming vector is calculated to maximize the received Signal to Noise Ratio (SNR) given the mmWave channel $\mathbf{h}_k^{mmW}$. $\mathbf{f}^\star$ is then expressed as:

$$\mathbf{f}^\star = \arg\max_{\mathbf{f} \in \mathcal{F}} \frac{1}{K} \sum_{k=1}^{K} \mathbb{E}\left[\left| \left| (\mathbf{h}_k^{mmW})^T \mathbf{f} \right| \right|_2^2 \right] \tag{2.28}$$

where K is the number of subcarriers.



(a) Raw point cloud from the ViWi dataset

(b) 3D occupancy grid

Figure 2.7 LiDAR input structure for the ViWi and Raymobtime datasets

2.3.3 LiDAR

As opposed to images, LiDAR generates 3D point clouds using laser scans. The extra depth dimension could prove useful in obtaining more accurate results when performing beam selection and precoder calculation. To achieve this objective, [33] uses LiDAR sensors mounted on vehicles in a Vehicle-to-Infrastructure (V2I) mmWave scenario. The LiDAR scans are converted to a 3D histogram, which is then used as the input to a neural network. The neural network then outputs the best beam pairs.

The authors resort to using a 3D histogram due to the high dimensionality of the point cloud $\mathcal{C}$ obtained from the LiDAR scan (e.g. Figure 2.7a). Let $P_v = (x_v, y_v, z_v)$ be the position of the vehicle. The base station transmits to the vehicle its absolute position $P_b = (x_b, y_b, z_b)$ as well as cuboid $Z = (x_z^b, y_z^b, x_z^e, y_z^e, h)$, which is the coverage zone of the base station where (x_z^b, y_z^b) and (x_z^e, y_z^e) denote the rectangular base, and h denotes the height of the coverage zone. Using the absolute positions of Z and $\mathcal{C}$, we split the coverage area into a histogram $\mathcal{G}$, where each element of $\mathcal{G}$ is a bin containing the number of points of $\mathcal{C}$ situated within it, as seen in Figure 2.7b. Finally, $\mathcal{G}$ is used as input to a neural network employing supervised learning to predict the top-K beam pairs. For K=30, the authors achieve a classification accuracy of around 98% for Line-Of-Sight (LOS) and 70% for Non-Line-Of-Sight (NLOS), noting that the number of classes to choose from is 240. This neural network used by the authors has an approximate 10^5 number of parameters.

However, a 70% accuracy for NLOS, especially if we still have to search from 30 beam pairs to find the best one, is not sufficient. Also, there is room for improvement in the complexity of the neural network. [34] tackles these problems. The authors mention that, concerning [33], even though the approach is distributed, the model still has to be trained on a centralized dataset. This would not be a problem were it not for the possible change in the channel conditions. Training in the offline phase does not affect the distributive nature of the method, but if the scattering environment changes, the site-specific neural network should be retrained with an up-to-date dataset, making the distribution of this dataset to the users costly. As a result, [34] proposed a federated learning approach to reduce the overhead of transmitting full LiDAR point clouds.

The solution consists of three phases. In the first phase, data acquisition, vehicles collect a local dataset containing LiDAR scans and the corresponding optimal beams. This method employs supervised learning, as the optimal beams are given. Then, in the federated learning phase, a single neural network is collaboratively trained using the global dataset collected from a subset of the vehicles in the coverage zone of the base station. The authors argue that this is a middle-ground between centralized learning, where the full LiDAR point cloud should be periodically transmitted, and per-vehicle learning, where a separate neural network is trained at each vehicle. The former method introduces a huge communication overhead, while the latter introduces a risk of bias due to the specific characteristics of the vehicle collecting the data - making generalization to most other vehicles difficult. In federated learning, each vehicle trains a set of local parameters $\boldsymbol{\theta}_v$, which are periodically synchronized with the other vehicles to obtain a total set of parameters $\boldsymbol{\theta}$. Finally, in the inference phase, each vehicle predicts the best beam pairs using its LiDAR data and the trained neural network.

The bottleneck that will make or break the proposed solution is phase two, namely the transmission of the local parameters. Since this transmission introduces a communication overhead, it is imperative that the total number of trainable parameters $|\boldsymbol{\theta}|$ be as small as possible. Therefore, the authors propose to use only a two dimensional representation of the point cloud, thus sacrificing the height data for reduced neural network complexity and communication overhead. This is done by representing the coverage area as a 2D grid (similar to the 3D histogram in [33]) of hit-or-miss cloud points. We obtain a grid containing 1 where a cloud point is present, -1 where the vehicle is located, -2 for the base station, and 0 otherwise. While one might think that this simplification of the data structure would be detrimental to the performance of the model, empirical data collected through experiments by the authors suggest otherwise. On the contrary, this simplification outperforms methods using the 3D histogram as input.

Concerning the results, this method achieves a top-10 accuracy of 91.17%, with a total number of trainable parameters $\boldsymbol{\theta} = 7462$. All the problems with [33] are solved: we get an improved accuracy while the final beam search overhead is reduced from 30 to 10. These results are obtained while respecting the communication overhead constraint, where the number of trainable parameters is reduced from 10^5 (though this value was not as important before as it is in this solution) to 7462. The questions that remain are: Can we employ a completely centralized solution using LiDAR data collected only from the base station to completely eliminate the communication overhead associated with the transmission of the trainable parameters? And can we employ unsupervised learning to completely eliminate the overhead associated with the exhaustive beam search required for the ground truth optimal beams?

Luckily, for the first question, [35] proves that distributive learning outperforms centralized learning, so the benefits that come with training only with data collected at the base station are not worth the significant reduction of accuracy that could be otherwise obtained when using data collected from all users. However, the use of unsupervised learning with LiDAR data for beam selection is still, to our knowledge, a topic that has not been tackled.

2.3.4 Beam Tracking

All of the above applications deal with either beam selection or hybrid precoder calculation in real time. In other words, given an input image or LiDAR scan, we find the corresponding optimal beam or AP/DP precoder pair for the same input. While this dramatically reduces both the communication and computation overheads, what if we can compute our output even before acquiring the image or LiDAR input? What if we can use previously collected data about the optimal beams or precoders, or even previous images or LiDAR scans, to compute future optimal beams ahead of time? This is the goal of our research in Chapter 4, but before we can tackle our solution, we need to review previous work in order to understand the necessary concepts, check weak points that can be improved, and justify the use of the input and models that we decide to use.

The article that is most pertinent to our work is [36]. This paper uses a Gated Recurrent Unit (GRU) with previous beams as input to track the user and predict the future beam. The goal is to justify the necessity of visual data for beam tracking, arguing that previous beam data are not enough. To test their hypothesis, the authors employ a Q-layer recurrent network with a final softmax layer for the classification task, with cross entropy loss as the loss function and top-1 accuracy as the evaluation metric. A GRU can be regarded as a LSTM with only two gates: an *update* gate and a *reset* gate [37]. GRUs are less complex

than LSTMs and can still combat the short memory problem of traditional Recurrent Neural Network (RNN)s, making them a great choice where increased complexity is an issue. The exact architecture and benefits of a Q-layer GRU are outside the scope of our research, so for the sake of simplicity, we accept that a Q-layer GRU is a specific architecture that is widely used in many time-series applications.

The authors provide the beam embeddings as input to the model. The beam embeddings are simply transformations from $\mathbb{N}$ to $\mathbb{R}^D$, where D is the size of the embedding vector. This is needed since the beam indices are simply non-negative integers, so the authors find it necessary to convert them into real-numbered vectors. The final architecture of the model is shown in Figure 2.8. Using these settings, they obtain an accuracy of 85% for 1 future beam, 60% for 3 beams, and 50% for 5 future beams. Looking solely at these results, the authors conclude that the past beams alone are not sufficient for the prediction of multiple future beams. While they do not give a definitive answer, they recommend that visual data is necessary for the prediction of more than one future beam. Therefore, it is worth validating their claim on two fronts: Using a different model to check the performance without visual data, and including images to compare the results.

Other interesting applications of time-series based deep learning in wireless communications include [38–41]. [38] is quite similar to the previous case, but instead of predicting the next beam, the model predicts which base station should serve the user at the next instant. This is called "Proactive Handoff". Typically, handoff is done after a user approaches the boundary of the coverage zone of the base station. At that point, some communication overhead will be introduced for the user to be switched to the next base station. As one can imagine, this process takes time. Thus, knowing beforehand which base station will serve the user, can

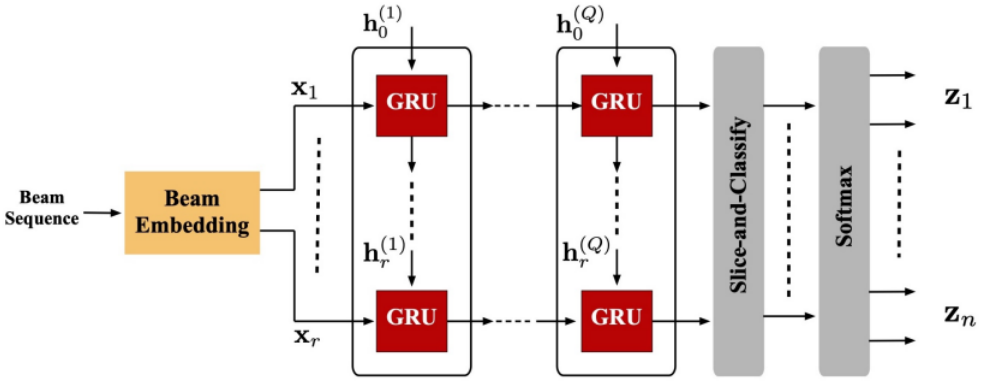


Figure 2.8 Architecture of GRU beam tracking model [36]

allow this preparation to be done ahead of time. The proposed GRU-based model takes as input the beam sequence and outputs, with an accuracy of 95%, the probability of each base station being the best candidate based on the existence of obstacles or the region of coverage of each. Looking at these probabilities, the base station that has the highest one will be chosen as the next serving base station. Going even further, [39] performs both tasks stated above (beam tracking and proactive handoff) for drones with flying reconfigurable intelligent surfaces. The method is quite similar to the one shown in [36] and Figure 2.8, but in addition to the beam sequence, the GRU takes as input the location sequence and the index of the used reflective surface. Using all this information, the model outputs the best future beam index, as well as the most probable reflective surface to be used. The prediction of the best reflective surface is particularly important since most sources say that these surfaces will play an integral part in 6G [42, 43], which is why we include it here for general information.

On the other hand, another approach for beam tracking can be to optimize the beam sweeping pattern explained in Section 1.2.3. We discussed that the hierarchical search can be a tedious task, so performing the sweep in a more intelligent manner can reduce the number of iterations necessary to find the best beam. [40] does just that. The goal of the proposed model is to output the best sweeping order to reduce the search overhead. This, on its own, is not very interesting, but combined with [41], we obtain a novel research field: Using images with RNNs for beam tracking. [41] aims to optimize the beam sweeping algorithm using a clever variation on LSTMs on images. The images are not actual captures taken by a camera but a representation of the power distribution among all the possible beams. Recalling again the hierarchical search, the proposed algorithm performs the wide beam sweep only and represents the beam power associated to each beam on an image. Then, using a CNN, the image is upsampled to predict the beam powers of the narrow beams. This way, the authors reduce the search overhead by only searching through the wide beams and locally estimating which narrow beams would be used based on the power distribution. Finally, a convolutional LSTM takes this image as input to predict the best beam at t+1. The entire process is shown in Figure 2.9. As a result of this whole approach, the authors reduce the overhead on two fronts. First, it is easier to search through wide beams than narrow ones due to their smaller number. Second, only one step of beam sweeping is required, eliminating the need of a hierarchical search. Finally, due to the LSTM, the model can predict a narrow beam one step into the future, further reducing the overhead associated with the collection of the wide beams for input.

Going through all these applications may seem to be a waste of time, but we argue that learning about these various approaches is essential. Not only do we broaden our scope regarding research that is not entirely linked to our work, but we understand the placement

of our project among all other solutions put forth by these esteemed authors. Finally, these proposed ideas act as a foundation for any future work that can be done in the field. The most notable example is [41]. This innovative solution inspired us to use an LSTM on a sequence of actual images taken from a camera to perform beam tracking. The list goes on, but it is perfectly evident while reading our proposed solutions at which points we use the methods shown in this chapter as a launching pad for our own ideas.

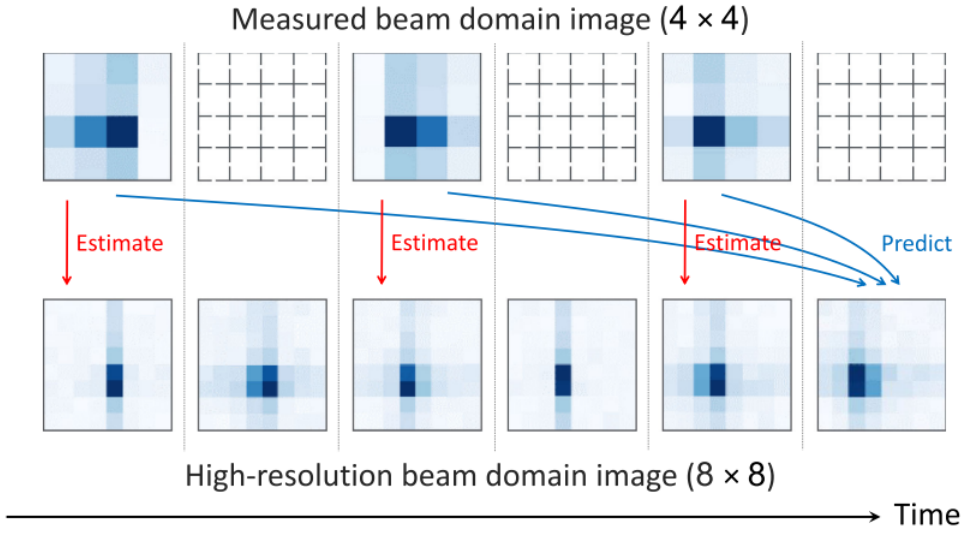


Figure 2.9 Workflow for beam tracking using upsampled beam pattern images ©2021 IEEE

CHAPTER 3 LIDAR BASED UNSUPERVISED DEEP-LEARNING FOR MASSIVE MIMO HYBRID BEAMFORMING

Following the work presented in [16] that uses RSSI with unsupervised learning to predict the hybrid precoder, we believe it is beneficial to investigate replacing RSSI with visual data. We assume that the visual data will provide a wider understanding of the environment, while hopefully eliminating the need for the synchronization signals. This will not just reduce the overhead associated with channel estimation in the online phase but will eliminate it completely. We resort to using LiDAR captures to leverage their depth dimension for the selection of more accurate beams. In all our simulations, we use PyTorch [44] as our machine learning framework.

3.1 System Model

Choosing the hybrid beamforming architecture, we use the system model illustrated in Figure 2.1. Consider M single-antenna users served by one base station (BS) equipped with N_T antennas and N_{RF} Radio Frequency (RF) chains, with $M < N_{RF} < N_T$. As seen in [31], the input signal x_j corresponding to user m first passes through the digital precoder (DP) $\mathbf{w}_m \in \mathbb{C}^{N_{RF} \times 1}$. Then, the analog precoder (AP) $\mathbf{A} \in \mathbb{C}^{N_T \times N_{RF}}$ designs the output beams and maps the output of the DP from the RF chains to the N_T antennas. Thus, as shown in (3.1), the received signal at user m can be expressed as:

$$y_m = \mathbf{h}_m^H \sum_{\forall j} \mathbf{A}\mathbf{w}_j x_j + \eta_m \tag{3.1}$$

Where $\mathbf{h}_m$ is the channel vector for user m, and η_m is an Additive White Gaussian Noise at the receiver. Furthermore, to further decrease the complexity of the system, we assume 2-bit phase shifters in the AP where each codeword $\mathbf{a}^{(i)} \in \{1+j, 1-j, -1+j, -1-j\}^{N_T \times 1}$.

3.2 Implementation

The first step is to choose the dataset for the training and testing of the neural network. Two possible choices are the ViWi and Raymobtime datasets [45, 46] since they both contain ray tracing and LiDAR information. ViWi contains LiDAR scans in the form of point clouds, as well as the channel matrices affecting the signal from the base station to the user equipment. It would be tempting to use this dataset due to the availability of the raw point cloud and,

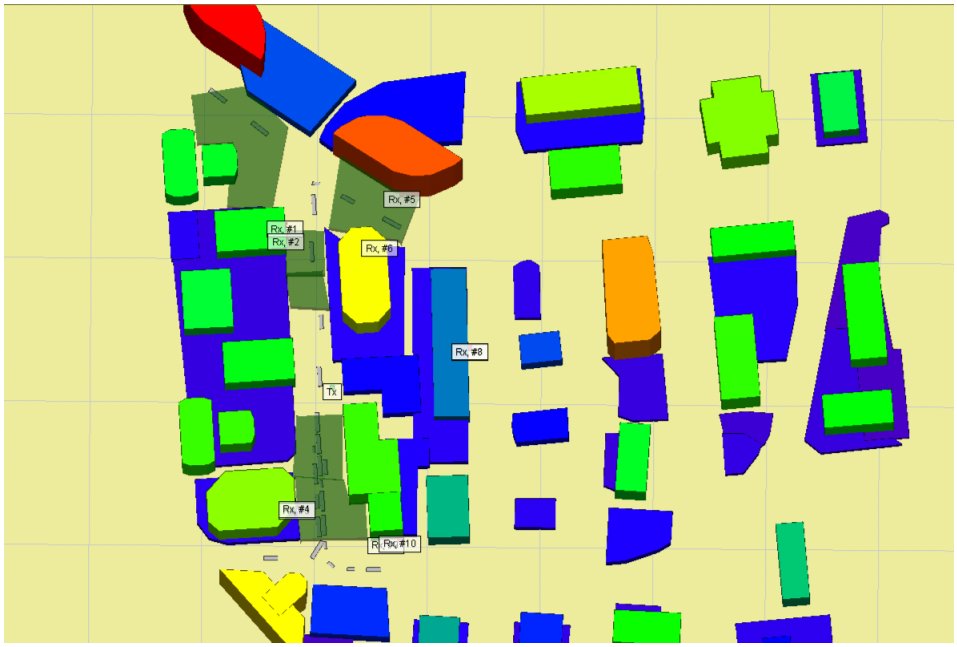


Figure 3.1 Map of Rosslyn, the area where the dataset is simulated (link)

more importantly, the complete and accurate channel matrices, but Raymobtime proves to be the simpler option due to the out-of-the-box preprocessing of the LiDAR point clouds into 3D histograms. However, instead of the complete channel matrices, only the ray-tracing information is given (angles of arrival and departure, complex gain, ...). Taking all of the above into consideration, we choose to use the Raymobtime dataset. To further corroborate our choice, we circle back to the fact that a lot of the research papers done on LiDAR data in wireless communications use models that employ 3D histograms and occupancy grids, as seen in Section 2.3.3. Using the same form of data, we can continue the work done by these authors instead of starting from scratch. The final reason we avoid using ViWi is that, even though using the raw point cloud allows us to use the state-of-the-art PointNet, adding rotation invariance and order invariance will increase the complexity of our model since we have to train transformation networks and add symmetric functions as seen in Section 2.2.2. While the goal is to use the best model possible, we need to explore the limitations of our method on the least complex model and dataset possible. We note that it is still possible to convert the point clouds from the ViWi dataset into 3D histograms, but since this is already done in Raymobtime, we use the latter dataset.

In particular, we use scenario s008 from Raymobtime, which simulates mobile users (cars) travelling through Rosslyn, Virginia. Figure 3.1 shows a map of the simulated environment, and Figure 3.2 shows an image of the neighborhood of concern at a certain instant. The

Figure 3.2 Sample image of users and obstacles in Raymobtime s008 scenario

mobile users are equipped with LiDAR sensors and provide us with the point clouds. Having access to LiDAR scans from multiple users instead of the base station allows us to use distributive learning which, as mentioned in Section 2.3.3, outperforms centralized learning.

As for the data preparation, we need to convert the ray-tracing information into valid channel matrices. We can use a variety of channel models to do so, such as the models described in [47]. We choose to use a model that takes into account the angles of arrival and departure, as well as the complex gain since this information is provided in the ray-tracing information in Raymobtime. That way, we ensure that we are using all the data that is available to us. As a result, we use the prominent Saleh-Valenzuela channel model [48, 49]. Through this model, we obtain the channel matrices using (3.3). Let $\mathbf{a}_t$ and $\mathbf{a}_r$ be the transmit and receive array responses respectively. According to [50], for a Uniform Linear Array (ULA), the array response is given by

$$\mathbf{a}^{ULA}(\phi) = \left[1,\ e^{jKdsin(\phi)},\ ...,\ e^{jKd(N-1)sin(\phi)}\right]^T \tag{3.2}$$

where ϕ is the azimuth angle of arrival or departure, $K = 2\pi/\lambda$, λ is the carrier wavelength, d is the antenna spacing, N is the number of antenna elements, and $[.]^T$ is the array transpose. All array response vectors are normalized. Denoting by L_k the number of paths, $\alpha_{k,l}$ the complex gain of path l, and $\phi^t_{k.l}$ and $\phi^r_{k.l}$ the azimuth angles of departure and arrival, the channel matrix $\mathbf{h}_k$ for user k can be expressed as

$$\mathbf{h}_k = \sum_{l=1}^{L_k} \alpha_{k,l} \mathbf{a}_r(\phi_{k,l}^r) \mathbf{a}_t^H(\phi_{k,l}^t) \tag{3.3}$$

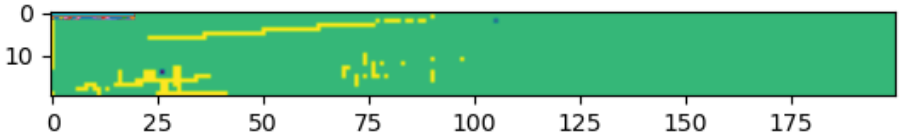


Figure 3.3 2D representation of 3D LiDAR histogram

Now that we have the channel data, we need to perform a bit of preprocessing to clean the data and turn it into a valid input to our neural network. First, the channel data contains some NULL values, so we eliminate all channel (and corresponding LiDAR) entries where at least one element is NULL. This leaves us with 11194 samples. Then for the LiDAR data, inspired by [34], we convert the initial 3D histogram into a 2D hit-or-miss projection (Figure 3.3), where pixels resembling the locations of objects are given a value of 1. We set the value of the pixel corresponding to the vehicle initiating the LiDAR scan to -1. Similarly, we set a value of -2 for the base station, and 0 where no cloud point is present. We also verify the claim in [34] that a 2D representation would be sufficient to encompass the information about the environment. We do so by inspecting several data points in the dataset. The original 3D histogram (hit-or-miss) provided by Raymobtime has a shape of $\{0,1\}^{N_X \times N_Y \times N_Z}$, where N_X, N_Y, N_Z are the number of points in the X, Y, and Z directions respectively. In particular, we have N_X=20, N_Y=200, and N_Z=10. By slicing the histogram with a plane of equation $z = 0, 1, 2, ..., N_Z - 1$, we observe that for a majority of the data points we consider, if an obstacle appears by slicing with any of the above planes, it will almost always be registered at $z = 1$. As a result, we choose to slice the 3D histogram at $z = 1$, obtaining a 2D histogram of size 20×200.

Finally, what remains is treating the large codebook size. Using 2-bit phase shifters, we obtain four possible ways to shift the signal. As a result, each AP codeword $\mathbf{a}^{(i)} \in \{1+j, 1-j, -1+j, -1-j\}^{N_T \times 1}$, we obtain a total number of permutations of $4^{N_T N_{RF}}$. This becomes problematic with increasing N_T and N_{RF}, and we can see why this is an issue in massive MIMO. As a result, we decide to limit the size of the codebook to a subset of the codewords yielding the best sum rates. This is a reasonable solution since most of the analog beams will probably never be used (such as beams pointing to the sky). A detailed explanation of the codebook reduction algorithm can be found in [16], but to summarize, we iteratively remove APs from the initial codebook using a genetic algorithm as long as we maintain a threshold average sum rate calculated in (2.20).

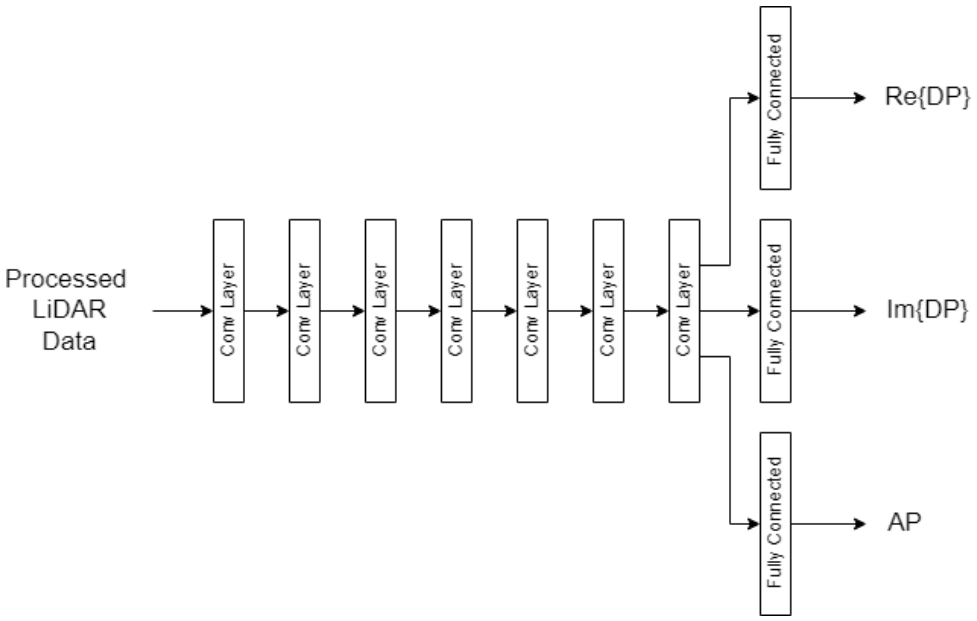


Figure 3.4 High-level block diagram of LiDAR-based HBF model

For the neural network, our design, shown in Figure 3.4 is inspired by HBF-Net from [16] and Lidar2D from [34]. We use 7 2D convolutional layers to *understand* the LiDAR input, and we add 3 separate linear layers to jointly predict the real and imaginary parts of the digital precoder, as well as the index of the best analog precoder in the codebook. After each layer, we use the ReLU (Rectified Linear Unit) activation function defined in (3.4), and we use batch normalization to prevent overfitting and facilitate training. For the final output layer, the outputs associated with the digital precoder are taken without any additional activation function, while that of the analog precoder is subjected to a softmax function defined in (3.5). We can see that the softmax activation function gives a probability distribution for all the possible outputs. This is useful since for the classification task, we are aiming to maximize the probability of choosing the optimal AP codeword while minimizing that of choosing APs with a low sum rate.

$$ReLU(x) = \max(0, x) \tag{3.4}$$

$$Softmax(x)_i = \frac{e^{x_i}}{\sum_{\forall j} e^{x_j}} \tag{3.5}$$

Expanding on the idea of using 3 separate linear layers for the prediction of the analog and digital precoders, we mention that our model is an example of a multitasking neural network. In essence, this means that we use the same neural network to perform multiple tasks, namely outputting the real and imaginary parts of the DP (regression tasks) and the index of the

best codeword in the codebook (classification task). We resort to using multitasking to concurrently achieve the different requirements without having to design a separate neural network for each. The 7 2D convolutional layers which, as we mention above, are used to *understand* the LiDAR input, act as the "representation learning" part of the network, where we attempt to extract features from the LiDAR scan to be used in the prediction of the hybrid precoders. Extracting these features separately for each task will prove to be redundant. Therefore, we use our multitasking model to share these features among the different tasks.

With the outputs now fully representing analog and digital precoders, what remains is defining a loss function to perform unsupervised learning and maximize the obtained sum rate. To do so, we use the loss function proposed in (2.22), as our final task of predicting hybrid precoders is similar to that found in [16]. We rewrite the loss function here for convenience:

$$\mathcal{L}_{HBF} = -\sum_{l=1}^{L} p_{a_{(l)}} R(\mathbf{A}_{(l)}, \overline{\mathbf{W}})$$

We recall that this loss function will train the neural network to find in parallel the best digital precoder from the regression task and the best analog precoder from the classification task. We can optimize these two precoders simultaneously since they both affect the final loss function being minimized. Seeing that $p_{a_{(l)}}$ is the l^{th} output of the Softmax layer in the AP classification task, corresponding to the probability that the l^{th} entry in the codebook is the best AP, we are both maximizing the chance of choosing the correct codeword while minimizing that of choosing completely wrong ones. A sub-optimal codeword that still ensures a sufficient sum rate will have a lower yet significant probability to be chosen. This ensures that even when the model is mistaken, the beam index it chooses will still be acceptable. Further, seeing that the loss function is also a function of the predicted DP $\overline{\mathbf{W}}$, we are making sure that we are concurrently training the regression task as well. As a result, we should theoretically obtain the best AP and DP pair. On the other hand, we also recall that there is no guarantee that these precoders will be aligned. We still resort to using this architecture, however, due to its simplicity and the absence of pseudoinverse calculations.

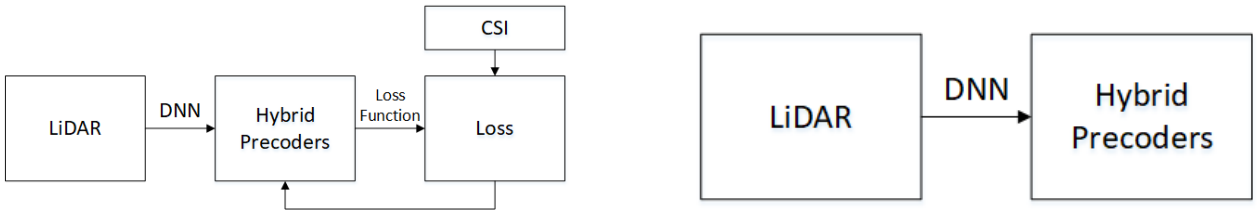
On the other hand, we recall that the expression of the sum rate R is also a function of the channel matrix, so the loss function will be a function of the channel matrix. If that is the case, then how are we decreasing the communication overhead in our proposed solution, seeing that we still need the channel state information? We need the channel information only during the training phase to calculate the loss function; besides, we need to train the neural network on a specific channel model and environment. However, in the evaluation

phase, as well as during normal deployment, we hide the channel information completely to simulate the model not having access to it. As a result, the model will output the best precoders solely based on the LiDAR scans. In our simulation, we use the channel information in the evaluation phase only to calculate the loss for testing purposes. This entire process is summarized in Figure 3.5.

Finally, we tackle a crucial point in this method: respecting the power constraints. Failing to respect these constraints may lead to driving the transmitted power arbitrarily high while maximizing the sum rate. We should make sure that the total transmitted power P does not exceed a maximum value of P_{max}. Taking $P_{max} = 1$ unit (this could represent any value for the power constraint), we must make sure that our precoders are normalized so they neither increase nor decrease the total transmitted power. The APs are already normalized during the creation of the codebook, so there is nothing that needs to be done there, but seeing that the DP is the result of a regression task, there is no guarantee that it will be normalized. The solution is simple yet critical: Normalizing the DP before calculating the loss function. If we forget to normalize our precoders, it is feasible that the model drive the DP towards infinity in the worst case to drive the sum rate towards infinity as well. The normalization makes sure that the model maximizes the sum rate by correctly dividing the available power among the antennas and transmitting along the channel instead of just increasing the power arbitrarily.

3.3 Results and Discussions

We simulate the above implementation using the settings shown in Table 3.1. Setting the number of transmitting antennas N_T to 24, the number of RF chains to half of that, i.e. 12, and assuming a maximum of 10 active users at a time, we generate the channels using the ray tracing data following the extended Saleh-Valenzuela channel model, and we start the codebook reduction algorithm. We obtain 11194 samples after preprocessing and a final



(a) Offline Mode (b) Online Mode

Figure 3.5 Use of CSI in online and offline modes

Table 3.1 System settings for LiDAR based model

N_T	24
N_{RF}	12
num_users (M)	10
codebook length	604
num_epochs	20
batch size (train)	16
batch size (validation)	1

codebook size of 604. We train the model for 20 epochs. During each epoch of training, we calculate the average loss over the entire training set. Then, after each epoch, we run the model on the validation set to check how it does on data it has not been trained on. We calculate the average loss on the validation set, but we do not perform backpropagation since we do not want to leak data from the validation set (data that should remain unknown during training) to our training set. After doing this process for all epochs, we plot the training and evaluation curves in Figure 3.6.

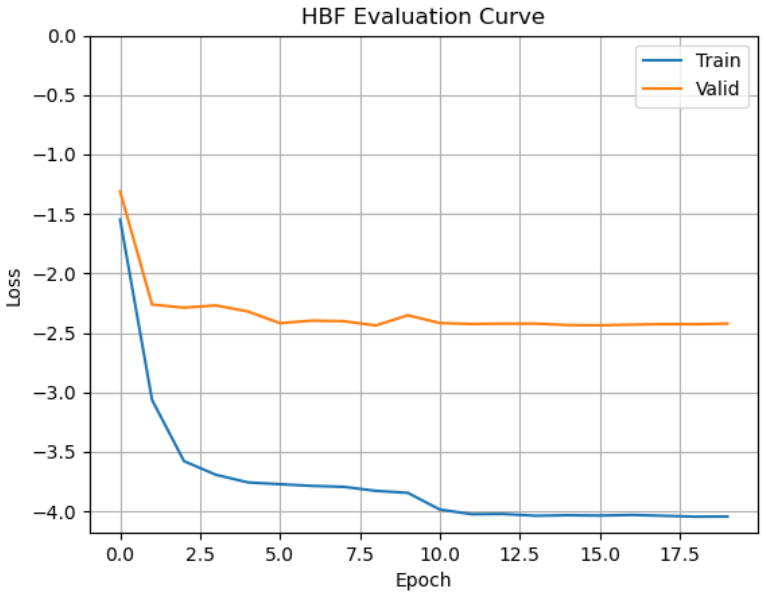


Figure 3.6 Training curve for unsupervised LiDAR-based Hybrid Beamforming (HBF)

We see an exponential decrease in the loss in both the training and evaluation curves. This behavior is indicative of a model that is training correctly, which is a good sign. This validates the fact that LiDAR can be used as an appropriate data source for our method, and it proves that our proposed neural network can correctly model a hybrid beamformer. Furthermore,

we see that the training loss caps out at around -4.0. This on its own does not mean much; however, paired with a final validation loss of -2.5, we see that we have a problem of lack of generalization. The vast gap between the training and validation curves shows that the model is fine-tuned on the training set, and that it is not generalized enough to treat information that it has not seen before. In our opinion, this behavior is not indicative of an error in our method, but of an insufficient number of samples for training the model using unsupervised learning.

3.4 Conclusion

To summarize the entire idea in one sentence, we take a LiDAR scan in the form of a 3D occupancy grid, slice it to obtain a 2D histogram, and use a CNN to estimate the optimal hybrid precoders. We find that the 2D histogram is sufficient to describe the environment, and we conclude that it is possible to train such a model in an unsupervised fashion. We encounter a problem of lack of generalization, however, as the gap between the training and validation curves is large. This is indicative that we need a larger dataset that encompasses more cases.

CHAPTER 4 BEAM TRACKING USING LSTM

Using LiDAR to reduce the overhead for beam selection proved to be an effective manner of increasing the efficiency of massive MIMO systems, as we have seen that we do not need to obtain the CSI in the online mode. However, traversing the neural network to obtain the best beam still takes a set amount of time. Thus, while we addressed the issue of the overhead, it does not follow that we impacted the latency in a significant way. Therefore, since we did not address how the previous method affects latency, we now turn our attention to reducing the latency of communication by means of beam prediction with recursive neural networks.

Up until now, the beam selection process has been done in real time; that is, we predict the beam for the current instant. What if, however, we can predict what the next beam is using previously acquired data? For example, if a vehicle is moving at a constant velocity, we need not continuously calculate the associated beam, but we can, based on its perceived movement, predict its position after a certain amount of time and precompute the beam before it even reaches that position. This is the concept of negative latency, a hot topic with virtual reality and augmented reality right around the corner [51]. Negative latency refers to preparing any required data beforehand so that when these data are actually needed, they are already loaded and no extra latency is introduced.

We aim to leverage recurrent neural networks to achieve a negative latency in massive MIMO systems. To do so, we conduct different experiments to conclude what the best method to reach this goal is. Namely, we investigate if we need to provide images as input to the RNN, or if the previous beams alone are enough. That is, does the increased complexity of a convolutional RNN using images provide sufficiently better results to warrant its use over a much simpler RNN using only the previous beams?

In the following sections, due to the prevalence of both in the literature, we use the terms *beam prediction* and *beam tracking* interchangeably.

4.1 Implementation

We explore different approaches of using recurrent networks for beam tracking. Using images as our visual data, we first learn how to predict the best beam from a codebook. We start with a non-time-series scenario where, for each input image, we select the best beam index. This task will act as an initial test to the capability of learning beamforming from images. After that, we delve into the use of images for beam prediction/tracking. Finally, we explore

a simpler approach, discarding visual data to verify its necessity.

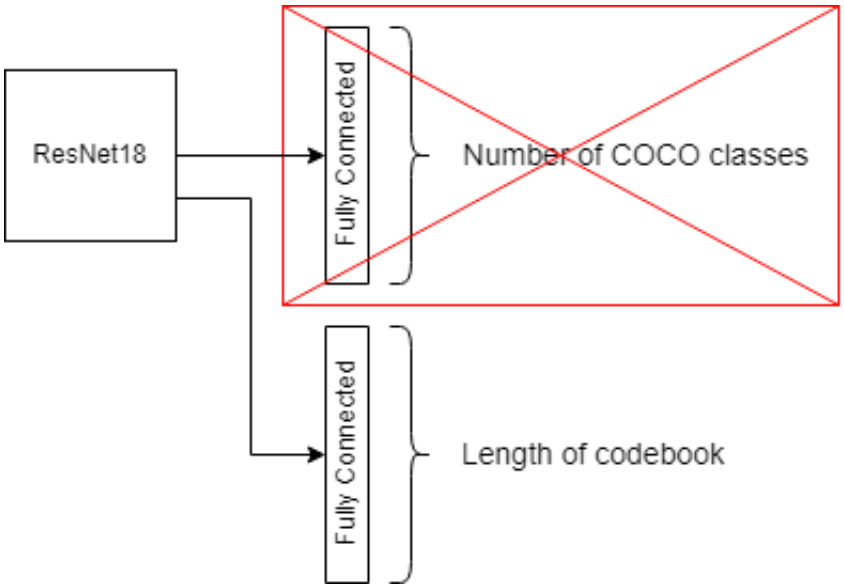


Figure 4.1 Transfer learning in ResNet18

For all these applications, we employ supervised learning. Therefore, we need to calculate the optimal beam indices before training any model. Unlike in the previous chapter, we now use digital beamforming with N transmitting antennas. To simplify the DP calculation process, we precompute a codebook of digital precoders. The choice of the codebook is insignificant, so we lay our eyes on Discrete Fourier Transform (DFT) beamforming [52], for our goal is to demonstrate vision-aided beam prediction and not compare codebook types. As a result, we use the DFT-codebook implementation provided by [32]. Given a DFT-codebook $\mathcal{F}$, the optimal beams are calculated following the optimization problem below:

$$\mathbf{f}^{\star} = \max_{\mathbf{f} \in \mathcal{F}} \log_2 \left(1 + \frac{|\mathbf{hf}|}{\sigma^2}\right) \tag{4.1}$$

where $\mathbf{f}$ is a codeword in $\mathcal{F}$, $\mathbf{h}$ is the channel vector (vector since we will be dealing with single user cases), and σ^2 is the noise power. We can see that we are choosing the best beamforming vector to maximize the achieved rate.

4.1.1 Implementation of ResNet Beam Selection

As an initial step, we need to validate the performance of beam selection using images as input before we can use these images to predict future beams or not. This gives us an opportunity to use transfer learning on a pre-trained ResNet model and to reproduce the

results shown in Section 2.3.2. To achieve this, we load a pre-trained ResNet18 model from torchvision [53] and change the final fully connected layer to have an output size equivalent to the length of our codebook, allowing us to change the classification task from its original common object classification to our required beam selection. We show the replacement of the final fully connected layer in Figure 4.1. Now that the size of the output is the same as the length of the codebook, we can use a Softmax function to calculate the probability of each codeword being the best beam. In order to test our model, we use a simple scenario, "The Colocated-Camera Scenario with Direct View (Single User)", from the ViWi dataset containing a single user in an empty street. The base station is equipped with three colocated cameras providing a view of the entire street. As the user travels through the street, one of the three cameras picks up the car, and based on the information found in the image (location of the user, obstacles and reflectors ...), we can use a neural network to find the best beam. A sample image of this dataset is shown in Figure 4.2. We choose such a simple yet theoretical case to provide a proof of concept that our method will work.

This scenario contains 5000 samples divided among the three cameras located at the base station. We split these samples into training and testing sets. Then we simply train our new pre-trained model, using Cross Entropy Loss as the loss function for supervised learning. The final top-k accuracies in the validation phase are: top-1 = 77%, top-2 = 93%, and top-3 = 97.5%. We see that with a relatively small dataset, the results look promising, so we are now hopeful in our quest to use a time-series of images for future beam prediction instead of only performing beam selection at the current instant.

4.1.2 Implementation of ConvLSTM with images

Looking back at Section 2.2.3, we see that LSTMs are perfect for time-series data. We also see that ConvLSTM has been introduced for time-series inputs in the form of images, where normal matrix multiplications are replaced by convolution products defined in (4.2). However, it can easily be seen that ConvLSTM is computationally heavy, and due to the complexity of the neural network, we also predict it will be data hungry, especially in our application where we are trying to predict optimal beams. Taking that into consideration, we see that there are many possible points of failure, so we deem it necessary to divide this task and test whether ConvLSTM can be useful for us in itself before we test it on our dataset. Therefore, we attempt to use ConvLSTM in a Sequence-to-Sequence Autoencoder[1] to predict future frames from the MovingMNIST [54] dataset. This task has been done many times before, but reproducing the results will help us build the basis of our own model.

[1] `https://github.com/holmdk/Video-Prediction-using-PyTorch`

(a) Camera 1

(b) Camera 2

(c) Camera 3

Figure 4.2 Image data sample from the used scenario

$$x[m,n] * h[m,n] = \sum_j \sum_i x[i,j]h[m-i,n-j] \tag{4.2}$$

A Sequence-to-Sequence Autoencoder [55, 56], or seq2seq for short, encodes a sequence or time-series of variable length into a constant length vector, then it decodes this obtained vector into a transformed variable length sequence or time-series which serves as the output of the model. Seq2seq autoencoders are useful in applications such as text translation where it is required to transform a sentence from English (input sequence), for example, to French. The length of the input sentence is variable for obvious reasons, and the length of the output sentence in French is variable and not necessarily equal to that of the input English sentence. This highlights the importance of seq2seq's ability to handle variable length inputs and outputs by encoding and decoding a fixed length representation vector. This behavior is also useful for the current frame prediction task where we can also specify input and output sequences of variable lengths.

The general architecture of the ConvLSTM based seq2seq is shown in Figure 4.3. Both the encoder and decoder parts have K ConvLSTM cells following the equations in (2.13). To strike a balance between depth and complexity, we choose $K = 2$ throughout all our

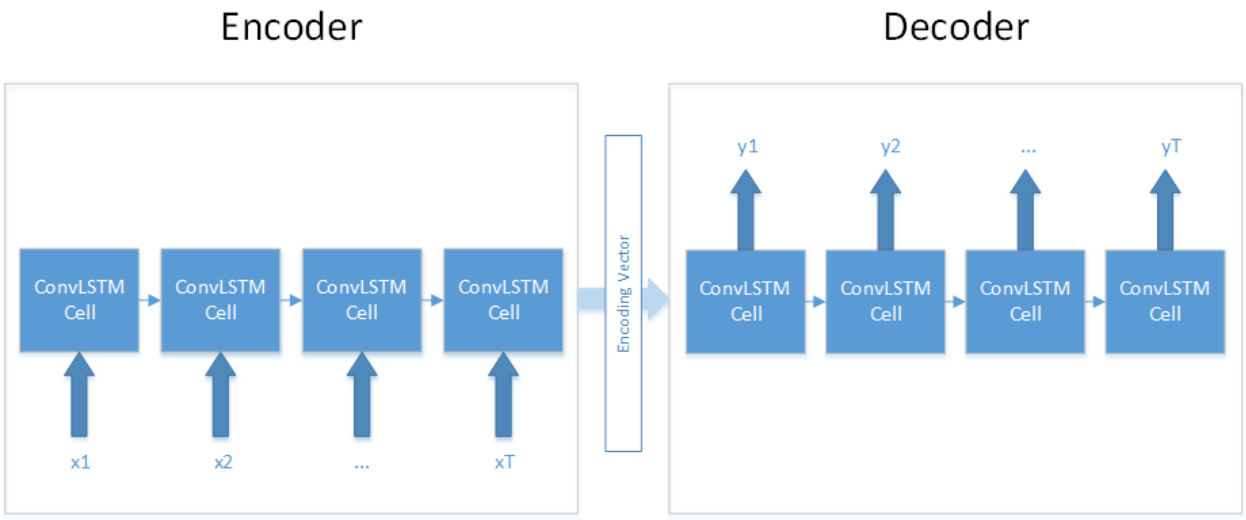
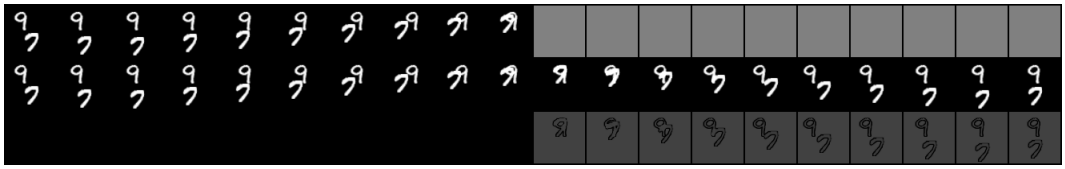


Figure 4.3 Sequence-to-Sequence Autoencoder architecture

simulations that use ConvLSTM. To reap the benefits of seq2seq, namely its ability to treat variable length sequences, we choose to deploy an architecture which, for both the encoder and decoder, the input of every ConvLSTM cell is the hidden state of the previous one. This applies to all ConvLSTM cells except the first one, where the input is the collection of images in the time-series fed sequentially. To put it simply, this translates to concatenating ConvLSTM cells, where the output of the former is the input to the latter. Finally, the output of the seq2seq autoencoder is the hidden state of the final decoding ConvLSTM cell. Seeing that this raw output has a number of channels equal to the dimension of the hidden states, we need a final 3D convolutional layer to decode this output into 1 channel (we are predicting monochromatic images. Finally, we apply a Sigmoid function to obtain the values of the pixels[2].

To test the effectiveness of this model, we run it on the MovingMNIST dataset. This dataset contains sequences of images, each containing two handwritten digits that are moving randomly throughout the image. Our goal is to perform frame prediction; predicting a series of n frames given m input frames. In other words, given the motion of the digits in 10 given frames, we want to predict their positions in the next 10 frames. We show the results in Figure 4.4. The first row contains the predicted images, the second the true values, and the third the difference between the predicted and true values. The first 10 images on the left in the top two rows are the input images, and the 10 images on the right are the outputs. We see that in the final epoch (epoch 235), we have a combination of both decent and un-

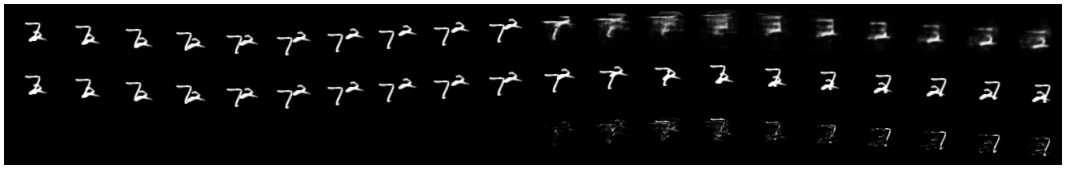
[2]Seq2seq is explained in great detail in [55,56]. That being said, this link contains a simplified description of this network architecture:



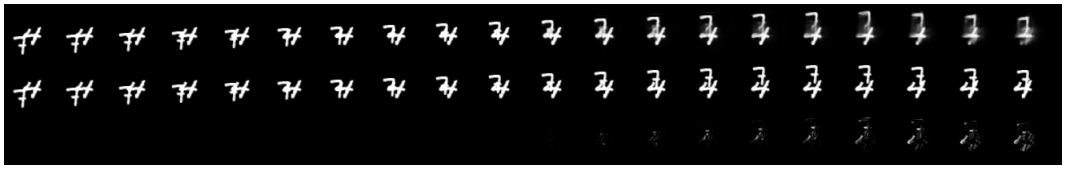
(a) Start

(b) Epoch 0

(c) Epoch 10



(d) Epoch 235 (Bad Result)



(e) Epoch 235 (Good Result)

Figure 4.4 Video prediction results of ConvLSTM on MovingMNIST dataset

satisfactory results. The variable quality of the results, as seen in Figures 4.4d and 4.4e, and the complexity of the method make us doubtful about the effectiveness of ConvLSTM based seq2seq autoencoders for our application, but looking at the difficulty of the frame prediction task for this dataset, and knowing that the motion of the digits in the MovingMNIST dataset is completely random, we remain hopeful that we can leverage this model to perform beam prediction for at least one frame after the sequence of images. Having 1 future output instead of 10, alongside the innate simplicity of predicting one beam index instead of 10 full images, should decrease the complexity of the system to obtain more reliable results.

Armed with the knowledge we gained during the implementation of ConvLSTM on MovingMNIST, we tackle beam prediction using images. Up until this point, we prepared a DFT codebook and calculated the optimal beams for each data point in the dataset, we performed beam selection in real-time given one image, we came up with the hypothesis that we can use multiple images from previous instances $t-1$, $t-2$, ..., $t-m$ to predict the best beams for instances t, $t+1$, ..., $t+n-1$, and we tested the possibility of using LSTMs on images in the form of a ConvLSTM based seq2seq autoencoder. Now, we use the same dataset and scenario as in Section 4.1.1 to train the seq2seq model explained above. We divide our 5000 samples into sequences of 5 images each, resulting in 1000 samples. From these samples, we use 750 for training and 250 for testing. For each sequence of 5 images, we use the first 4 as input to the model, and we train it to predict the best beam for the fifth image as seen in Figure 4.5. We train the model using supervised learning, where the predicted beam is compared to the optimal beam calculated beforehand. After training for 10 epochs, we obtain the following top-k accuracies: top-1 = 31%, top-2 = 60.24%, and top-3 = 74.7%.

We see that these results are not as good as expected, and what is worse is that there are many possible reasons for this failure. One cause might be the small size of the dataset. This is a valid concern since we are using only 750 samples for training and 4 images per sequence as input, but comparing with the total randomness of the MovingMNIST dataset and the obtained performance on it, we expected to obtain at least somewhat better results. The one sliver of hope is that the model is performing better than random choice (randomly choosing any entry in the codebook as the predicted beam), for the top1 accuracy is much larger than 6%. Another reason might be the complexity of the ConvLSTM cells. Since the seq2seq model we are using is quite computationally demanding and data hungry, there should exist, hypothetically, a simpler model that can learn from the small dataset we have, especially since the scenario is simple with only one user and one base station. Taking all this into account, we are not able to pinpoint which of these two possibilities is the true perpetrator. Luckily, we may not need to find out.

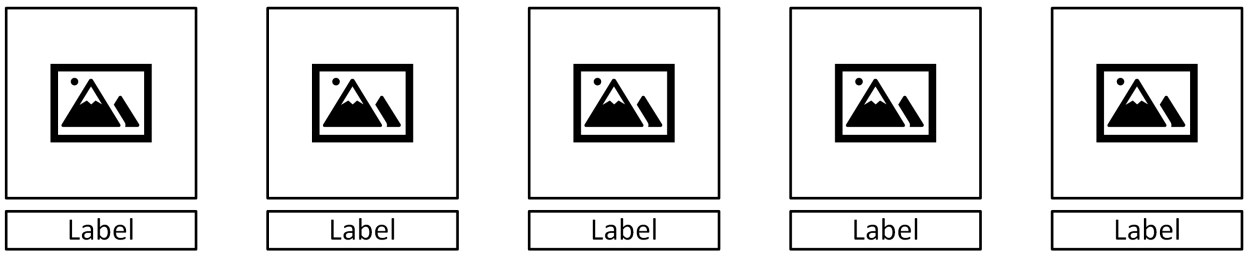


Figure 4.5 Input sequence shape for the LSTM

While working on this task and realizing the obstructions that lie ahead, it dawns on us that we can vastly decrease the complexity of the system if we use the beam indices of the previous instances instead of the images themselves. In other words, we can attempt to use the method discussed in 4.1.1 to find the best beam for the first four images in the sequence of data, and use these beam indices as input to a much simpler LSTM to predict the future beams. We tackle this in the following section.

4.1.3 Implementation of Simple LSTM with labels

For one final experiment, we test the hypothesis presented in the previous section by using the simplest LSTM possible. We aim to put forth a variation on the work done in [36, 38, 39]. Looking at [36] in particular, we see that the authors use a beam embedding layer to convert the input from a non-negative integer to a real-valued vector. We attempt to eliminate the embedding layer and replace the GRU used in this paper with a slightly more complex model. For that, we use the built-in LSTM module in PyTorch, which is a mere implementation of the equations shown in (2.12). Our argument is that we can slightly reduce the total complexity of the model by using the raw beam indices as input while compensating the lack of depth by replacing the GRU with an LSTM. We add a final linear layer to convert the output to the required size - the size of the codebook. Similarly to the previous section, we group the 5000 samples into sequences of 5 images each, and we split the resulting 1000 sequences into 750 training and 250 testing data points. We also note that for each image, we have already calculated the optimal beam; thus, supervised learning can be used. Even in the online mode, we assume that we have the previous labels using optimal results. The difference is that, seeing we already have the optimal beams, we use these beam indices as input to the LSTM instead of the images. We obtain the following top-k accuracies: top1 = 57%, top2 = 75%, top3 = 86%. These results are very promising; with a larger dataset, it is definitely possible to reach higher accuracies and even predict beams for more than one instant to the future.

How is a top-1 accuracy of 57% considered promising when [36] achieves accuracies as high as 86%? First, we are using a different scenario from the ViWi dataset with a smaller sample size and different channel conditions, so it is unfair to compare accuracies when multiple factors such as channel conditions and dataset size could influence the results. Furthermore, it should be noted that the training of our model is lightning fast; training for 20 epochs takes less than 5 minutes. This shows that our initial goal of reducing the complexity of the model is achieved. What remains is to ameliorate the outputs of the model so that it can compete with the state-of-the-art. This can be done by looking into newer datasets and adding some depth to the neural network while remaining less heavy than the existing models.

4.1.4 Implementation of ConvLSTM with Images: DeepSense Dataset

If most of the problems encountered above can be attributed to the dataset, they could be solved by using the recently developed DeepSense [57] dataset. This dataset contains a plethora of scenarios taken from different streets, where different sensors are placed to collect alternative visual information. These sensors include cameras, LiDAR, and RADAR. To demonstrate the effectiveness of this new dataset, we use the image data to perform the same task described in the previous sections: beam tracking using images.

Before we run any simulations, we need to pay attention to some differences in the data structure of this new dataset. First, the data comes in sequences of five samples, so we need not perform any preprocessing to collect these sequences. Each sequence of five images has a corresponding label for the best beam index, meaning we do not need to split our sequence into four inputs and one output. This allows us to use even more input into our LSTM, resulting in better user tracking. Finally, the codebook length is now 64 instead of 16, which is closer to real world applications. We note that the labels are integer values belonging to the interval $[1, 64]$, but if we are to use Cross Entropy Loss as our loss function, we need to decrement all the labels by 1 to obtain values in the interval $[0, C-1]$, where C is the number of classes equal to the length of the codebook. Taking all these points into consideration, we use scenarios 32, 33, and 34 to obtain an aggregate data size of 11143 sequences, which is a significant increase from the 1000 sequences we previously had. We run our code with minimal changes to check the initial results, and we obtain a dismal top1 accuracy of 7%.

Why do we end up at this exact value of 7% throughout our multiple trials? To answer this, we manually inspect the output of the neural network. We see that the model outputs the same beam index of 21 for all inputs. While normally this is indicative of a model that fails to learn, we persist in our inspection and plot the probability distribution in Figure 4.6. Incidentally, beam index 22 (corresponding to label 21 after decrementation) is prevalent in 7% of all cases. This means that the model learns to output the beam having the maximum likelihood of occurrence. This is unacceptable, and we suspect the large complexity of the system to be the main culprit. We therefore move to the hyper-parameter tuning phase.

The main parameters we can tune are the image size, the learning rate, and the number of hidden layers of the ConvLSTM cells. We suspect that the image size could be the main problem for two reasons. First, a large image size increases the complexity of the network due to the necessity of a larger decoding fully connected layer, as well as the larger number of multiplications per convolution. Second, as illustrated in figure 4.7, the images contain significant patches of empty space. Reducing the resolution, thus, decreases the number of pixels corresponding to empty space, making the final data structure denser. Therefore,

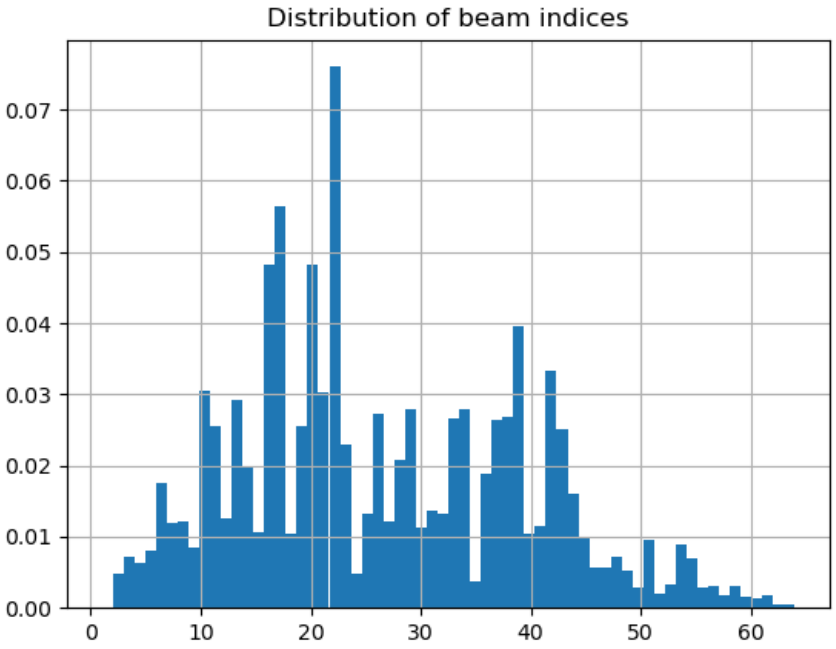


Figure 4.6 Probability distribution of the labels in the DeepSense dataset

Figure 4.7 Sample image from the DeepSense dataset taken from scenario 34

we define the hyper-parameter r controlling the image size, where the image size becomes $r/1.76 \times r$. The value of 1.76 ensures that we maintain the aspect ratio of the original image. Surely enough, the top1 accuracy skyrockets from 7% to 41.25% at epoch 10, with top2 and top3 accuracies of 63.9% and 75.18% respectively. We also verify that the model does not output the same results for all inputs, meaning that the neural network trains correctly. This verifies our initial hunch that the large complexity is detrimental to the model's performance. We aim to further improve our results by tuning the remaining two hyper-parameters.

The next parameter we tune is the learning rate. The above experiment had a learning rate of 0.001. The learning rate that gives us the best results, after multiple trials, is 0.0001. giving us a measly gain of 1-2%. The biggest difference is the slower convergence, which is to be expected with the decrease of the learning rate. The best accuracies we obtain are at epoch 11, namely top1=42.76%, top2=66.21%, top3=77.88%. The final parameter to be examined is the number of hidden layers per ConvLSTM cell, denoted by n_f. Increasing n_f increases the number of trainable parameters, resulting in a better ability to represent data at the cost of higher complexity. Therefore, we try $n_f = 64, 32, 16$ to find the best trade-off between generalization and complexity. Surprisingly, we observe minimal changes between our trials. All signs point towards a problem of overfitting. We provide further proof to this claim in Section 4.2, but for now, we attempt to solve this issue using data augmentation.

Data augmentation is a process in which we increase the size of our dataset by applying certain transforms on the input images. The simplest solution we have is applying a set of random transforms on all the images belonging to the training set. These transforms could range from anything between rotation, cropping, padding, blurring, etc. We achieve this by defining a new data loader that takes a training and testing transform as input during initialization. Both these transforms contain the necessary operations to preprocess the image for use in the neural network, but the training transform also includes a random data augmentation transform. This way, we ensure the proper functioning of our system without performing any data augmentation on the testing set. We successfully address the problem overfitting (also discussed in more detail in Section 4.2), but we see that the performance slightly deteriorates to the following: top1=37.56%, top2=59.75%, top3=70.88%, but is maintained even after many epochs (which means that the overfitting problem is solved).

4.2 Results and Discussions

To train the various RNNs discussed above, we use backpropagation through time [58]. Backpropagation through time is the equivalent of normal backpropagation but in recurrent networks, where we need to train the parameters of the network not only through its layers

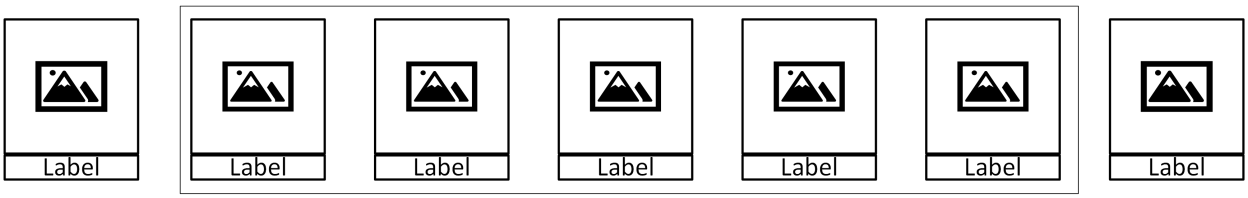


Figure 4.8 LSTM sliding window for sequence collection

but also through time. Much like normal backpropagation, the gradients are calculated based on the error between the desired output and the result of the model. Luckily, this process is automated in PyTorch whenever any type of RNN is employed, so we need not delve deep into the fundamentals of backpropagation.

We show in Table 4.1 a comparison between the obtained results from the above section. We notice a significant decrease in accuracy between the beam selection and beam tracking (beam prediction) tasks, highlighting the increased difficulty of using time-series data to predict future beams with no input from the corresponding instants. Furthermore, in the beam tracking case, we predict only one beam into the future, so we can expect the performance to be even worse when predicting multiple beams. However, this comparison may not be totally fair due to the large difference in the dataset sizes used for training the beam selection and beam tracking models. Even though we are using the same dataset, grouping the data points into sequences of five images/labels reduces the net dataset size by five times, giving the model less room to train. One possible solution to this problem is using a sliding window approach while collecting the input sequences (Figure 4.8), but we quickly dismiss this idea due to our suspicion of possible data leakage when using images that would not be available to the model during deployment for training.

Table 4.1 Summary of results of beam selection and prediction tasks

Codebook Length	Task	Top-1	Top-2	Top-3
16	ResNet Beam Selection (Section 4.1.1)	77%	93%	97.5%
	ConvLSTM (Section 4.1.2)	31%	60.24%	74.7%
	SimpleLSTM (Section 4.1.3)	57.4%	75.1%	86%
64	DeepSense GPS Input	28.61%	57.49%	70.22%
	DeepSense ConvLSTM (Section 4.1.4)	42.76%	66.21%	77.88%

To explain further, consider a sequence of 5 images, where the first 4 entries are the input of the LSTM and the last element is the target (label). We suppose that the label is for the future instant, so in a causal system, we will not have access to that data. As a result, it

makes sense not to include the fifth image/beam in the input of the LSTM. However, suppose that we are using the sliding window approach, now the fifth element of the sequence used before will become the fourth element in the next sequence. We visualize this process in Figure 4.9. This, on its own, is not a great issue, but the model must not have any idea whatsoever about any data that is used for the calculation of the loss, i.e. the labels. If we are in the ConvLSTM case, we must ensure that the images corresponding to the labels are never seen by the model. As a result, we are still plagued by this 5× decrease in the dataset size. This also limits our control over the size of the input sequence. We need that the sequence be as long as possible to collect as much information as possible about the mobility of the users, but we cannot increase the size too much since our final dataset size will decrease even further. For example, in our dataset of 5000 single images, taking sequences of 5 leaves us with 1000 samples, but taking sequences of 10 leaves us with only 500 samples. Thus, we see that we have a trade-off between how much information we can include about the velocity of the users and the final size and generalization capabilities of the dataset.

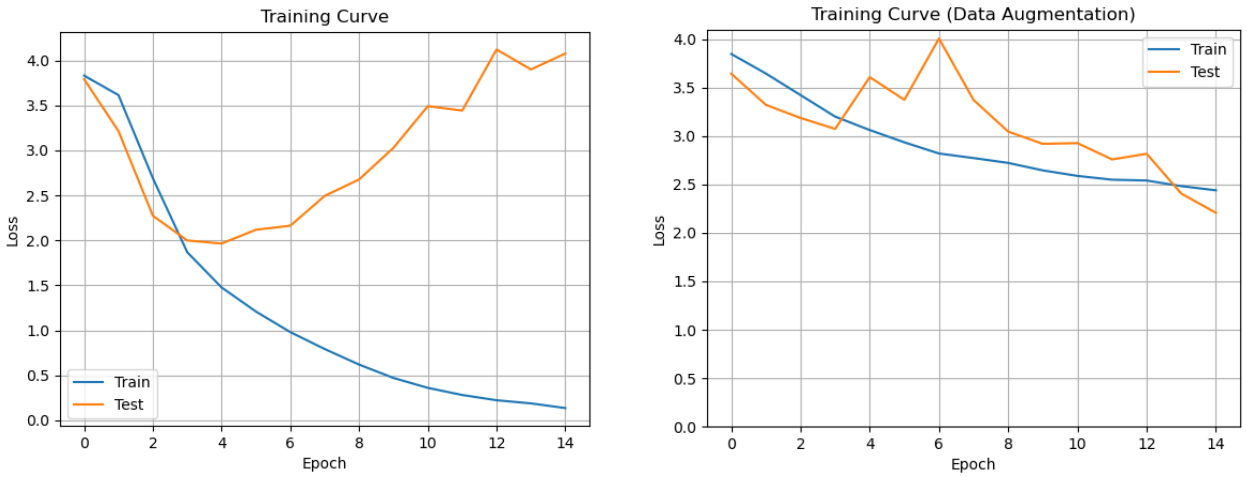
Figure 4.9 LSTM sliding window input data leakage

On the other hand, for the beam prediction task using DeepSense, we compare our results with a baseline[3] that uses the position data only. We see that our results outperform the baseline by a decent margin. However, comparing with our previous simulations with the ViWi dataset, we obtain slightly better results than with ConvLSTM, and worse results than with SimpleLSTM. The numbers may be misleading, as the codebook length in the ViWi case is 16, as opposed to 64 in DeepSense. This means that the space of possible labels has quadrupled, so it is no surprised to see the accuracies drop down. That being said, the values obtained with the DeepSense dataset are arguably better than those with ViWi.

[3]Results taken from https://github.com/DeepSense6G/Multi-Modal-Beam-Prediction-Challenge-2022-Baseline. Here, the results of the proposed baseline for the dataset and task we use are shown.

On the other hand, we mention in Section 4.1.4 that we encounter a problem of overfitting. Here, we provide the evidence to our claim. Before adding data augmentation, and given a complex enough model, the performance of the system evolves as follows. In the first few epochs, the validation accuracy is very low since the model is not yet trained. After we converge to the optimal solution, the model is at its best performance where we obtain the highest accuracies. However, towards the final epochs, the validation performance begins to deteriorate. These three phases point towards an overfitting problem, so it is worthwhile to investigate. To prove once and for all that this phenomenon is present, we plot the average training loss per epoch against the validation loss, and we get the graph shown in Figure 4.10a. We see a similar behavior as in Figure 3.6, but here the effect is even more pronounced. At first, the training and testing curves decrease together until they reach a certain point where they begin to separate. The training curve continues to descend, while the testing curve diverges. This shows that the model is capable of extracting information from the dataset but reaches a plateau after a few epochs in the testing phase. As a result, we should normally stop the training at the point where the two curves diverge. However, we are not satisfied with our results, even though they beat the baseline, since the training curve shows so much promise that it would be a shame to stop here. We attempt to rectify this situation by using data augmentation, and as we see in Figure 4.10b, we are successful; the training and testing curves do not diverge. However, the final achieved loss (and subsequently top-k accuracy), is lower than what we achieved without data augmentation. We leave this issue for future work.

The million-dollar question now is: How could we improve the performance of beam prediction? Unfortunately, the results of our research do not provide a definitive answer, but we do stumble upon some weak points during the ideation and implementation of this project. Unlike the project presented in Chapter 3, it is unclear whether a better dataset alone is sufficient to reach acceptable results. Nevertheless, we do not like to put the blame solely on factors which are out of our control. Therefore, we rephrase the initial question: Assuming that we have a large, general and clean dataset, what changes can we make to improve our results? For starters, we unintentionally prove that the previous beams alone are not enough to perform beam prediction multiple instances into the future. This is counter-intuitive since the results with SimpleLSTM way outperform those of ConvLSTM, but keep in mind the increased complexity of using images instead of beam indices as input. [36] backs our claim, showing diminishing returns when the number of predicted beams into the future increases. As a result, more research must be conducted on creating simpler LSTM algorithms which take images as input.



(a) Without data augmentation

(b) With data augmentation

Figure 4.10 Training and testing curves for the DeepSense dataset

4.3 Conclusion

For the treatment of time-series data, we experimented with different models and scenarios to validate different hypotheses. First, we attempted a simple beam selection algorithm taking an image as input, and we see that we can use our findings as a basis for beam prediction. We used ConvLSTM, first on a verified dataset to test its potential then on our own dataset to evaluate its performance for our task. Finally, we simplified the problem to the extent that visual data may no longer be necessary. We list below the conclusions we draw.

Concerning SimpleLSTM, we obtain promising results, but our accuracies are outshined by the state-of-the-art. This is problematic, but we recall that our main objective is to provide a simpler variation to the existing methods. In doing so, we are successful since we remove the embedding layer at the cost of a slightly more complex LSTM (instead of the used GRU). It is true that more work can be done to improve our results, but at the end of the day, we will always be presented with a trade-off: A trade-off between a less complicated model with worse results and a more complicated one with better results. This is up for debate depending on the most critical aspects in a particular project.

On the other hand, we present proof that no matter how much we improve SimpleLSTM, we will reach a plateau of diminishing returns, for it is difficult to predict more than one beam into the future with only the beam indices as input. As a result, we find that more work needs to be done to improve the performance and decrease the complexity of convolution-based recurrent neural networks.

CHAPTER 5 CONCLUSION

We share some final thoughts concerning our work. Although the results obtained in both chapters are not entirely satisfactory to the point where they could be implemented in newer wireless communication systems, they open the doors for some interesting further investigations. We believe that the field of vision-aided wireless communications will be the focal point of future research as well as a driving force of fifth and sixth generation (5G and 6G) systems. Therefore, for the sake of future scientists looking to improve upon our work, we provide a summary of everything we have done for this thesis, we share the limitations and obstacles we faced, and we look into the key areas for future research.

5.1 Summary of Work

Our work is divided into two main tasks: Hybrid beamforming using LiDAR data, and beam prediction using images. In the first task, we use a single LiDAR scan taken at the present instant to compute the best analog and digital precoders for the same instant in real time. For the second task, we use a collection of previous data to predict beforehand the best analog precoder for a future instant. For each of these tasks, we encounter various challenges and learn valuable lessons.

For hybrid beamforming, when choosing between using raw point clouds or 3D histograms, we ultimately choose histograms due to the simplicity of the resulting model. We then learn that a projection of the 3D histogram on a 2D plane is sufficient to train a neural network to find the hybrid precoders. This is due to the dimensionality reduction of the input, meaning that we have less parameters to train, so we need fewer samples in our dataset. The model consists of a Convolutional Neural Network to convert the 2D projection of the LiDAR scan into embedded features, and some fully connected layers that output the real and imaginary parts of the Digital Precoder, as well as the probabilities of each codeword in the codebook to be the best Analog Precoder. We note that this final codebook is a reduction of the initial collection of all possible codewords obtained via a genetic algorithm which iteratively removes codewords that result in a low sum rate. We train this model in an unsupervised (semi-supervised) fashion to maximize the sum rate achieved by all users. We use unsupervised learning to ensure plug-and-play functionality in any environment, without the need for the expensive computation of optimal labels. Furthermore, to successfully train the model to maximize the sum rate - an objective function not previously defined in PyTorch, we are forced to define our own loss function. Our loss function ensures that we simultaneously

optimize both the regression and classification tasks by including terms corresponding to both the AP and DP. Minimizing this loss function results in achieving all our goals. We maximize the probability to find the best AP, minimize that of choosing completely wrong ones, and emphasize the choice of APs that still result in a sufficient sum rate when the model is wrong in its prediction. Minimizing the loss function also optimizes the regression task, which is responsible for the calculation of the DP. Further, we know that the model can *cheat* and maximize the sum rate by increasing the transmit power (calculating a DP which increases the power), so we make sure to normalize the DP before computing the loss function. This way, the model is forced to intelligently divide the power among the antennas instead of simply increasing the transmit power. After performing all of the above, we obtain our hybrid precoders.

We also research the prediction of future beams (analog precoding only) based on the current and past states of the user and the environment. This way, we can calculate the beam that would hypothetically serve a mobile user in a particular location before the user reaches said location. That way, we decrease the latency associated with searching for the optimal beam. We experiment with various models to find which architecture provides the best trade-off between simplicity and accuracy. First, we use ConvLSTM, a LSTM with convolution products instead of matrix multiplications, to process images and output the best beam index in a codebook. To improve the results, we come up with the idea of exchanging previous images with previous beam indices, and using a simple LSTM similar to those used in natural language processing for example. We notice a jump in the performance of this new model in both the speed of training and computational complexity, and accuracy of the results. We discuss the trade-off between complexity and accuracy regarding the chosen RNN.

5.2 Limitations of the Proposed Methods

A lot of the difficulties we encountered during our work can be traced back to the small datasets that we used. This is evident in the case of unsupervised learning where the gap between the training and validation curves is large. Also, regarding beam tracking, when we use the DeepSense dataset, we obtain better results using the same ConvLSTM model. It is easy to blame poor results in deep learning on the dataset, but a closer look at the training curve in the hybrid beamforming task, as well as the top-k accuracies in the beam prediction task proves that that is indeed the case. In the first task, the exponential shape of the training and validation curves is indicative of successful training. Furthermore, the final achieved loss in the training phase is decent, meaning that the difference we see between the two curves in Figure 3.6 is solely due to overfitting - a result of a poor and non-generalized

dataset. Moreover, in the second task, even when ConvLSTM, for example, performs poorly, especially in the top-1 accuracy category, the accuracy obtained is much higher than what we expect to get if we are choosing the beam index randomly. Seeing that the top-k accuracies are computed on the validation set - a set that is never seen beforehand by the neural network, we conclude that a better dataset will most definitely give rise to better results.

A minor issue we encountered is the variable number of points in the LiDAR point clouds provided by ViWi. Neural networks should have a fixed shape and size, and that depends on the input and output sizes. Seeing that the input size in this case would be variable, some work should be done to convert all the point clouds into a fixed size. We attempt to crop the point clouds to a certain number of points, but we quickly realize the drawback of our method since by randomly removing points, we may risk removing crucial information. We also try other methods such as sampling and aggregation of similar points, all to no avail.

5.3 Future Research

First and foremost, we believe that the most important step before any other significant work can be done on vision-aided wireless communications is the creation of better datasets. In the computer science domain, generalized, labeled, simple, and exhaustive datasets exist for a plethora of fields, be it medicine, economics, fashion, or entertainment. This, unfortunately, is not available in our field of electrical engineering due to the lack of specialists (compared to computer science) that understand both electrical engineering needs, and the labeling, collection, and simulation of datasets. This leaves only a handful of capable researchers with a gold mine of exploration and dataset creation. One example of a recent, promising dataset is DeepSense [57]. Similar to ViWi, DeepSense contains images and LiDAR point clouds, as well as GPS and RADAR data. Furthermore, DeepSense contains more data and a larger variety of scenarios, which become helpful when treating time-series data and improving the generalization capabilities of deep learning models.

We showcase the power of the DeepSense dataset firsthand in Section 4.1.4, where we notice a significant performance boost with the larger sample size. However, there is much room for improvement, even in our own work. First, there is the unsolved issue of overfitting where despite our best efforts, we fail to eliminate the biases existent in our data. Further, we use only a fraction of the available data. We can extend our work to using GPS, RADAR, or LiDAR, and we can tackle different tasks such as blockage prediction and unsupervised beamforming. As in any field using artificial intelligence, we are limited only by the availability of data. With DeepSense, we can explore beyond the current horizons and discover new frontiers in vision-aided wireless communications.

On the other hand, while a decent amount of research exists on using particular vision based sensors, not much exists regarding the combination of multiple streams of information. Extracting information from both LiDAR and images at the same time, for example, may provide even better results. Moreover, in the work we have done concerning beam prediction, we are only able to predict the best beam for one future instant. It would be useful to look into ways to predict further into the future to decrease the overhead associated with the channel estimation required for the optimal labels. This could be done by both defining a loss function that takes into account multiple outputs sequentially, as well as simplifying the Recurrent Neural Network structure for easier training.

Furthermore, keeping the issue of the datasets aside, a potential liability may be the inability to identify particular users using either LiDAR scans or images. To explain, consider an image resembling that of Figure 3.2. If we take the cars to be the users in the environment, both our LiDAR and image based models are perfectly capable of detecting all the users. However, it would be beneficial to have an understanding of the particular users. That way, we can transmit specific streams of data requested by each user, instead of transmitting the aggregation of all the data for all users. We imagine that we can divide and tailor the data streams to each user instead of transmitting the same data in all resulting radiation lobes and rely on the end user's ability to extract its own data from the aggregated stream.

Finally, while 3D histograms have worked amazingly in the past, we think it is time to move on to using raw point clouds. The reasons to do so are many. First, the LiDAR scan provided by the sensor will be a raw point cloud, so converting it into an occupancy grid will require some preprocessing, hence computational complexity and increased overhead and latency. Second, we still believe in the power of the state-of-the-art PointNet, which takes the point cloud as input directly. As mentioned in Section 5.2, one of the gravest issues we encountered is the variable size of the point clouds. If a solution can be found, the benefits of using PointNet will be innumerable.

REFERENCES

[1] S. Mangiante, G. Klas, A. Navon, G. Zhuang, J. Ran, and M. Silva, "VR is on the Edge: How to Deliver 360° Videos in Mobile Networks," in *VR/AR Network '17: Proceedings of the Workshop on Virtual Reality and Augmented Reality Network*, 08 2017, pp. 30–35.

[2] Nokia. (2021) 5G spectrum bands explained — low, mid and high band. [Online]. Available: https://www.nokia.com/networks/insights/spectrum-bands-5g-world/

[3] E. G. Larsson, O. Edfors, F. Tufvesson, and T. L. Marzetta, "Massive MIMO for next generation wireless systems," *IEEE Communications Magazine*, vol. 52, no. 2, pp. 186–195, 2014.

[4] E. Ali, M. Ismail, R. Nordin, and N. F. Abdulah, "Beamforming techniques for massive MIMO systems in 5G: overview, classification, and trends for future research," *Frontiers of Information Technology & Electronic Engineering*, vol. 18, no. 6, pp. 753–772, 2017. [Online]. Available: https://doi.org/10.1631/FITEE.1601817

[5] S. Coleri, M. Ergen, A. Puri, and A. Bahai, "Channel estimation techniques based on pilot arrangement in OFDM systems," *IEEE Transactions on Broadcasting*, vol. 48, no. 3, pp. 223–229, 2002.

[6] N. Gonzalez-Prelcic, A. Ali, V. Va, and R. W. Heath, "Millimeter-Wave Communication with Out-of-Band Information," *IEEE Communications Magazine*, vol. 55, no. 12, pp. 140–146, 2017.

[7] H. Friis, "A Note on a Simple Transmission Formula," *Proceedings of the IRE*, vol. 34, no. 5, pp. 254–256, 1946.

[8] T. S. Rappaport, *Wireless Communications: Principles and Practice*. Prentice Hall, 2002, ch. 3, pp. 70–73.

[9] T. Bai, A. Alkhateeb, and R. W. Heath, "Coverage and capacity of millimeter-wave cellular networks," *IEEE Communications Magazine*, vol. 52, no. 9, pp. 70–77, 2014.

[10] F. W. Vook, A. Ghosh, E. Diarte, and M. Murphy, "5G New Radio: Overview and Performance," in *2018 52nd Asilomar Conference on Signals, Systems, and Computers*, 2018, pp. 1247–1251.

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

[12] R. Pascanu, T. Mikolov, and Y. Bengio, "On the difficulty of training Recurrent Neural Networks," 2012. [Online]. Available: https://arxiv.org/abs/1211.5063

[13] J. Ma, H. Yu, and S. Liu, "The MMSE Channel Estimation Based on DFT for OFDM System," in *2009 5th International Conference on Wireless Communications, Networking and Mobile Computing*, 2009, pp. 1–4.

[14] F. Kettlun, F. Rosas, and C. Oberli, "A Low-Complexity Channel Training Method for Efficient SVD Beamforming over MIMO Channels," *EURASIP J. Wirel. Commun. Netw.*, vol. 2021, no. 1, jul 2021. [Online]. Available: https://doi.org/10.1186/s13638-021-02026-x

[15] D. Tse and P. Viswanath, *Fundamentals of Wireless Communication*. Cambridge University Press, 2005.

[16] H. Hojatian, J. Nadal, J.-F. Frigon, and F. Leduc-Primeau, "Unsupervised Deep Learning for Massive MIMO Hybrid Beamforming," *IEEE Transactions on Wireless Communications*, vol. 20, no. 11, pp. 7086–7099, 2021.

[17] S. Jain, A. Markan, and C. Markan, "Performance Evaluation of a Millimeter Wave MIMO Hybrid Beamforming System," in *2020 IEEE Latin-American Conference on Communications (LATINCOM)*, 2020, pp. 1–5.

[18] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 770–778.

[19] T.-Y. Lin, M. Maire, S. Belongie, L. Bourdev, R. Girshick, J. Hays, P. Perona, D. Ramanan, C. L. Zitnick, and P. Dollár, "Microsoft COCO: Common Objects in Context," 2014. [Online]. Available: https://arxiv.org/abs/1405.0312

[20] Y. Bengio, A. Courville, and P. Vincent, "Representation Learning: A Review and New Perspectives," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 35, no. 8, pp. 1798–1828, 2013.

[21] R. K. Srivastava, K. Greff, and J. Schmidhuber, "Highway Networks," 2015. [Online]. Available: https://arxiv.org/abs/1505.00387

[22] R. Q. Charles, H. Su, M. Kaichun, and L. J. Guibas, "PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation," in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017, pp. 77–85.

[23] D. Maturana and S. Scherer, "VoxNet: A 3D Convolutional Neural Network for real-time object recognition," in *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2015, pp. 922–928.

[24] Y. Yan, Y. Mao, and B. Li, "SECOND: Sparsely Embedded Convolutional Detection," *Sensors (Basel, Switzerland)*, vol. 18, 2018.

[25] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[26] H. Sak, A. Senior, and F. Beaufays, "Long Short-Term Memory Based Recurrent Neural Network Architectures for Large Vocabulary Speech Recognition," 2014. [Online]. Available: https://arxiv.org/abs/1402.1128

[27] X. Shi, Z. Chen, H. Wang, D.-Y. Yeung, W.-k. Wong, and W.-c. Woo, "Convolutional LSTM Network: A Machine Learning Approach for Precipitation Nowcasting," 2015. [Online]. Available: https://arxiv.org/abs/1506.04214

[28] T. O'Shea and J. Hoydis, "An Introduction to Deep Learning for the Physical Layer," *IEEE Transactions on Cognitive Communications and Networking*, vol. 3, no. 4, pp. 563–575, 2017.

[29] X. Li and A. Alkhateeb, "Deep Learning for Direct Hybrid Precoding in Millimeter Wave Massive MIMO Systems," in *2019 53rd Asilomar Conference on Signals, Systems, and Computers*, 2019, pp. 800–805.

[30] R. W. Heath, N. González-Prelcic, S. Rangan, W. Roh, and A. M. Sayeed, "An Overview of Signal Processing Techniques for Millimeter Wave MIMO Systems," *IEEE Journal of Selected Topics in Signal Processing*, vol. 10, no. 3, pp. 436–453, 2016.

[31] H. Hojatian, V. N. Ha, J. Nadal, J.-F. Frigon, and F. Leduc-Primeau, "RSSI-Based Hybrid Beamforming Design with Deep Learning," in *ICC 2020 - 2020 IEEE International Conference on Communications (ICC)*, 2020, pp. 1–6.

[32] M. Alrabeiah, A. Hredzak, and A. Alkhateeb, "Millimeter Wave Base Stations with Cameras: Vision Aided Beam and Blockage Prediction," *arXiv e-prints*, p. arXiv:1911.06255, Nov 2019.

[33] A. Klautau, N. González-Prelcic, and R. W. Heath, "LIDAR Data for Deep Learning-Based mmWave Beam-Selection," *IEEE Wireless Communications Letters*, vol. 8, no. 3, pp. 909–912, 2019.

[34] M. B. Mashhadi, M. Jankowski, T.-Y. Tung, S. Kobus, and D. Gündüz, "Federated mmWave Beam Selection Utilizing LIDAR Data," *IEEE Wireless Communications Letters*, vol. 10, no. 10, pp. 2269–2273, 2021.

[35] M. Dias, A. Klautau, N. González-Prelcic, and R. W. Heath, "Position and LIDAR-Aided mmWave Beam Selection using Deep Learning," in *2019 IEEE 20th International Workshop on Signal Processing Advances in Wireless Communications (SPAWC)*, 2019, pp. 1–5.

[36] M. Alrabeiah, J. Booth, A. Hredzak, and A. Alkhateeb, "ViWi Vision-Aided mmWave Beam Tracking: Dataset, Task, and Baseline Solutions," 2020. [Online]. Available: https://arxiv.org/abs/2002.02445

[37] R. Dey and F. M. Salem, "Gate-variants of Gated Recurrent Unit (GRU) neural networks," in *2017 IEEE 60th International Midwest Symposium on Circuits and Systems (MWSCAS)*, 2017, pp. 1597–1600.

[38] A. Alkhateeb, I. Beltagy, and S. Alex, "Machine Learning for Reliable mmWave Systems: Blockage Prediction and Proactive Handoff," in *2018 IEEE Global Conference on Signal and Information Processing (GlobalSIP)*, 2018, pp. 1055–1059.

[39] N. Abuzainab, M. Alrabeiah, A. Alkhateeb, and Y. E. Sagduyu, "Deep Learning for THz Drones with Flying Intelligent Surfaces: Beam and Handoff Prediction," in *2021 IEEE International Conference on Communications Workshops (ICC Workshops)*, 2021, pp. 1–6.

[40] A. Mazin, M. Elkourdi, and R. D. Gitlin, "Accelerating Beam Sweeping in mmWave Standalone 5G New Radios Using Recurrent Neural Networks," in *2018 IEEE 88th Vehicular Technology Conference (VTC-Fall)*, 2018, pp. 1–4.

[41] H. Echigo, Y. Cao, M. Bouazizi, and T. Ohtsuki, "A Deep Learning-Based Low Overhead Beam Selection in mmWave Communications," *IEEE Transactions on Vehicular Technology*, vol. 70, no. 1, pp. 682–691, 2021.

[42] J. Tang, M. Cui, S. Xu, L. Dai, F. Yang, and M. Li, "Transmissive RIS for 6G Communications: Design, Prototyping, and Experimental Demonstrations," 2022. [Online]. Available: https://arxiv.org/abs/2206.15133

[43] Z. Zhang, L. Dai, X. Chen, C. Liu, F. Yang, R. Schober, and H. V. Poor, "Active RIS vs. Passive RIS: Which Will Prevail in 6G?" 2021. [Online]. Available: https://arxiv.org/abs/2103.15154

[44] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "PyTorch: An Imperative Style, High-Performance Deep Learning Library," in *Advances in Neural Information Processing Systems 32*. Curran Associates, Inc., 2019, pp. 8024–8035. [Online]. Available: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf

[45] M. Alrabeiah, A. Hredzak, Z. Liu, and A. Alkhateeb, "ViWi: A Deep Learning Dataset Framework for Vision-Aided Wireless Communications," in *submitted to IEEE Vehicular Technology Conference*, Nov. 2019.

[46] A. Klautau, P. Batista, N. González-Prelcic, Y. Wang, and R. W. Heath, "5G MIMO Data for Machine Learning: Application to Beam-Selection Using Deep Learning," in *2018 Information Theory and Applications Workshop (ITA)*, 2018, pp. 1–9.

[47] K. V. S. Hari, *Channel Models for Wireless Communication Systems*. New York, NY: Springer New York, 2011, pp. 47–64. [Online]. Available: https://doi.org/10.1007/978-1-4419-6111-2_3

[48] W. H. Tranter, D. P. Taylor, R. E. Ziemer, N. F. Maxemchuk, and J. W. Mark, *A Statistical Model for Indoor Multipath Propagation*. Wiley-IEEE Press, 2007, pp. 127–136.

[49] C. Liu, E. Skafidas, T. Pollock, and R. Evans, "Angle of Arrival Extended S-V Model for the 60 Ghz Wireless Desktop Channel," in *2006 IEEE 17th International Symposium on Personal, Indoor and Mobile Radio Communications*, 2006, pp. 1–6.

[50] O. E. Ayach, R. W. Heath, S. Abu-Surra, S. Rajagopal, and Z. Pi, "The capacity optimality of beam steering in large millimeter wave MIMO systems," in *2012 IEEE 13th International Workshop on Signal Processing Advances in Wireless Communications (SPAWC)*, 2012, pp. 100–104.

[51] P. Seeling and F. H. Fitzek, "Anticipatory Networking: Negative Latency for Ubiquitous Computing," in *2021 IEEE 18th Annual Consumer Communications & Networking Conference (CCNC)*, 2021, pp. 1–4.

[52] D. Yang, L.-L. Yang, and L. Hanzo, "DFT-Based Beamforming Weight-Vector Codebook Design for Spatially Correlated Channels in the Unitary Precoding Aided Multiuser Downlink," in *2010 IEEE International Conference on Communications*, 2010, pp. 1–5.

[53] D. Falbel, *torchvision: Models, Datasets and Transformations for Images*, 2022, https://torchvision.mlverse.org, https://github.com/mlverse/torchvision.

[54] N. Srivastava, E. Mansimov, and R. Salakhutdinov, "Unsupervised Learning of Video Representations using LSTMs," 2015. [Online]. Available: https://arxiv.org/abs/1502.04681

[55] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to Sequence Learning with Neural Networks," 2014. [Online]. Available: https://arxiv.org/abs/1409.3215

[56] K. Cho, B. van Merrienboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation," 2014. [Online]. Available: https://arxiv.org/abs/1406.1078

[57] A. Alkhateeb, G. Charan, T. Osman, A. Hredzak, and N. Srinivas, "DeepSense 6G: Large-Scale Real-World Multi-Modal Sensing and Communication Datasets," *to be available on arXiv*, 2022. [Online]. Available: https://www.DeepSense6G.net

[58] G. Bird and M. E. Polivoda, "Backpropagation Through Time For Networks With Long-Term Dependencies," 2021. [Online]. Available: https://arxiv.org/abs/2103.15589