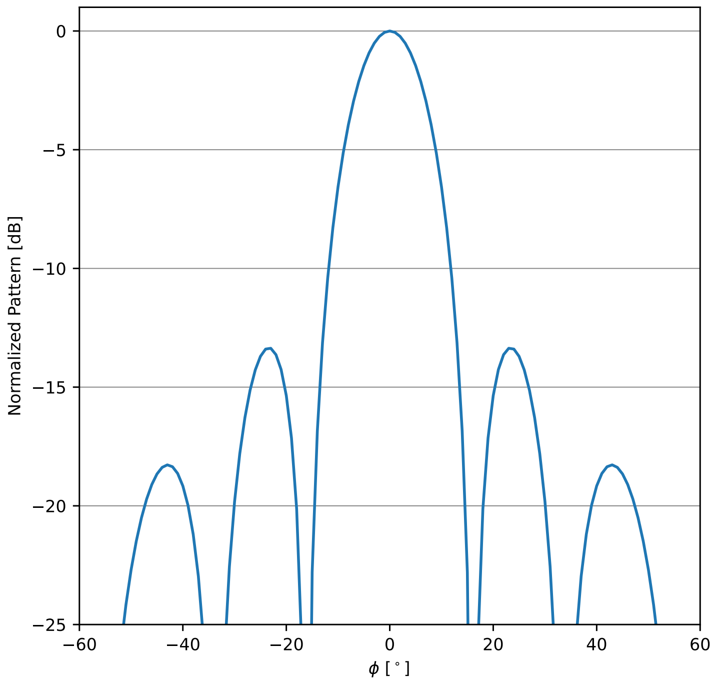
CHALMERS
UNIVERSITY OF TECHNOLOGY



Beamweight Prediction Using Complex Valued Convolutional Neural Networks

A Fast Alternative for Beamweight Handling of Non-ideal Antenna Element Solutions in Antenna Arrays

Master's Thesis in Engineering Mathematics and Computational Science

RUBEN FRILUND
ZACKARIAS OLOFSSON

DEPARTMENT OF MATHEMATICAL SCIENCES
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2024
www.chalmers.se

MASTER'S THESIS 2024

Beamweight Prediction Using Complex Valued Convolutional Neural Networks

A Fast Alternative for Beamweight Handling of Non-ideal Antenna Element Solutions in Antenna Arrays

RUBEN FRILUND
ZACKARIAS OLOFSSON

Department of Mathematical Sciences
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2024

Beamweight Prediction Using Complex Valued Convolutional Neural Networks
A Fast Alternative for Beamweight Handling of Non-ideal Antenna Element Solutions in Antenna Arrays
Ruben Frilund
Zackarias Olofsson

Supervisor: Erik Geterud, Ericsson AB
Examiner: Larisa Beilina, Department of Mathematical Sciences

Master thesis report 2024
Department of Mathematical Sciences
Chalmers University of Technology
SE-412 96 Gothenburg
Sweden
Telephone +46 31 772 1000

Cover: The figure shows a peak normalized radiation pattern of a linear antenna array with 8 antenna elements arranged along a straight line in azimuth.

Typeset in LaTeX
Gothenburg, Sweden 2024

Beamweight Prediction Using Complex Valued Convolutional Neural Networks
A Fast Alternative for Beamweight Handling of Non-ideal Antenna Element Solutions in Antenna Arrays
Ruben Frilund
Zackarias Olofsson
Department of Mathematical Sciences
Chalmers University of Technology

Abstract

The advanced antenna systems used for telecommunication today utilize antenna array structures. These structures consist of grids of antenna elements divided into smaller subarrays of elements where each subarray is dynamically controllable, independently from the others. Each subarray is typically mapped to a baseband port, and the amplitude and phase of the signal on each of this baseband port can be controlled independently, which is called excitation vectors or beamweights.

The main aim of this thesis was to investigate and develop a machine learning model that can predict beamweights for unseen antenna arrays such that the radiated far field of the antenna follows a specific pattern. The model was evaluated and compared to a more common method, stochastic optimization. Furthermore, the model was tested to see how stable the outputs are against noise and if the model is scalable to antenna arrays of different sizes.

The main conclusion from the thesis is that it is possible to train a complex valued convolutional neural network (CV-CNN), a type of machine learning model, to predict the desired beamweights. The results also show that the beamweights generated from the two different methods result in far fields which favour different performance metrics. An important difference in the models is the time it takes to generate beamweights. A pre-trained CV-CNN generates beamweights in a fraction of the time it takes to generate beamweights using the more common method. The conclusion from the comparison is that the method should be chosen case by case.

The stability analysis shows that the model is more sensitive to noise in the phase than noise in the amplitude of the input. The reason for this might come from an approximation made while processing the data but a possible solution to this is presented.

Finally, the results show promise of scalability of the model. The model must however be optimized for each size of antenna array in order to get satisfactory results.

Keywords: Advanced Antenna Systems, Antenna Arrays, Beamforming, Beamweights, Machine Learning, Artificial Neural Networks, Complex Valued Convolutional Neural Networks.

Acknowledgements

This thesis would not be possible without the help and guidance from a number of people. We would like to thank our supervisor Erik Geterud at Ericsson AB for his guidance throughout the project. We would also like to thank Bhushan Billade, Martin Johansson, Henrik Jidhage and Hilding Nilsson at Ericsson AB who shared their expertise and guided us through the thesis. Our thanks also goes to our manager Jean-Marc Pfeiffer who let us conduct the thesis under his section at Ericsson AB. Lastly we would like to thank our examiner Professor Larisa Beilina at Chalmers University of Technology.

Ruben Frilund and Zackarias Olofsson, Gothenburg, May 2024

List of Acronyms

The following acronyms has been used throughout the thesis:

AAS	Advanced Antenna System
AFK	Array Factor Kernel
AI	Artificial Intelligence
ANN	Artificial Neural Network
ASHA	Asynchronous Successive Halving Algorithm
CNN	Convolutional Neural Network
CV-CNN	Complex Valued Convolutional Neural Network
dB	Decibel
DE	Differential Evolution
DPBF	Dual-Polarized Beamforming
GA	Genetic Algorithm
HPBW	Half Power Beam Width
ML	Machine Learning
MIMO	Multiple Inputs, Multiple Outputs
MSE	Mean Squared Error
NSGA-II	Non-dominated Sorting Genetic Algorithm II
SA	Simulated Annealing
SGD	Stochastic Gradient Descent
ReLU	Rectified Linear Unit

Nomenclature

Below is a list of notations used throughout the thesis.

Indices

t	Time step index.
m, n	Indices for numbering of subarrays.
pol	Index for polarization.
i, j	Indices for neurons in a layer.
ℓ	Indices for layers in a neural network.
p, q	Indices for weights in kernels in convolutional layer.

Coordinates

$\{x, y, z\}$	Cartesian coordinates.
$\{\hat{\mathbf{x}}, \hat{\mathbf{y}}, \hat{\mathbf{z}}\}$	Cartesian unit vectors.
$\{r, \theta, \phi\}$	Spherical coordinates.
$\{\hat{\mathbf{r}}, \hat{\theta}, \hat{\phi}\}$	Spherical unit vectors.
$\hat{\mathbf{co}}$	Unit vector for co-polarization.
$\hat{\mathbf{xp}}$	Unit vector for cross-polarization.

Antennas

A	Amplitude.
Φ	Phase.
P	Power density.
$\mathbf{E}$	Electric field.

E_{co}	Co-polar component of $\mathbf{E}$.
E_{xp}	Cross-polar component of $\mathbf{E}$.
$\mathbf{H}$	Magnetic field.
N	Number of subarray columns.
$\mathbf{G}(\hat{\mathbf{r}})$	Complex far field function.
$\mathbf{G}_n(\theta, \phi)$	Complex far field function for subarray $n = 1, 2, ..., N$.
$\mathbf{G}_A(\theta, \phi)$	Complex far field function for an antenna array.
$\mathbf{G}^{(m,n,pol)}(\theta, \phi)$	Complex far field function for subarray (m, n) and polarization pol.
$G_{co}^{(m,n,pol)}(\theta, \phi)$	Co-polar component of $\mathbf{G}^{(m,n,pol)}(\theta, \phi)$.
$G_{xp}^{(m,n,pol)}(\theta, \phi)$	Cross-polar component of $\mathbf{G}^{(m,n,pol)}(\theta, \phi)$.
λ	Wavelength.
η	Wave impedance.
k	Wavenumber.
$\mathbf{k}$	Wavenumber vector.
F	Radiation pattern.
D	Directivity.
$\mathcal{G}$	Gain.
ϵ_R	Radiation efficiency.
$\mathbf{r}_n$	Phase reference point, for $n = 1, 2, ..., N$.
A_n	Amplitude for subarray $n = 1, 2, ..., N$.
Φ_n	Phase for subarray $n = 1, 2, ..., N$.
$A_{n,noise}$	Amplitude noise.
$\Phi_{n,noise}$	Phase noise.
$R(\phi)$	Reference curve.
$\mathcal{F}$	Set of angles where a reference curve is above -10 dB.
$\mathcal{F}_{side}$	Set of angles where sidelobe suppression is enforced.
G_{ref}	Maximum amplitude of far field function.
G_{side}	Maximum sidelobe level.

Complex numbers

j	The imaginary unit.
z	Complex number.
z^*	Complex conjugate of complex number.

$\mathbb{C}$	Space of complex numbers.
$\mathrm{Re}(z)$	Real part of complex number.
$\mathrm{Im}(z)$	Imaginary part of complex number.
$f(z)$	Complex function.
$f'(z)$	Differentiation of complex function.
$f(z, z^*)$	Complex function in Wirtinger calculus.

Artifial Neural Networks

$V_i^{(0)}$	Input value i to neural network, $i = 1, 2, ..., n_0$.
$V_i^{(\ell)}$	Value at hidden layer ℓ at neuron i, $i = 1, 2, ..., n_\ell$, $\ell = 1, 2, ..., L$.
$V_i^{(L)}$	Output value i from neural network, $i = 1, 2, ..., n_L$.
n_ℓ	Number of neurons in layer ℓ, $\ell = 0, 1, 2, ..., L$.
$w_{i,j}$	Weight from neuron i to neuron j for $i \in \ell - 1$ and $j \in \ell$, $\ell = 1, 2, ..., L$.
b_i^ℓ	Threshold/bias for neuron i in layer ℓ, $i = 1, 2, ..., n_\ell$, $\ell = 1, 2, ..., L$.
g	Activation function.
P, Q	Kernel sizes.
$\mathbf{I}$	Input to convolutional layer.
$\mathbf{K}$	Kernel in convolutional layer.
f_i^ℓ	The local field at neuron i in layer ℓ, $i = 1, 2, ..., n_\ell$, $\ell = 1, 2, ..., L$.
η	Learning rate.
α	Step size.
β_1, β_2	Decay rates.
$\mathcal{L}_T$	Taper loss.
$\mathcal{L}_F$	Far field difference.
$\mathcal{L}_S$	Sidelobe loss.
$\mathcal{L}$	Loss function, total loss.

Contents

List of Figures

List of Tables

1
Introduction

In recent years, the global mobile broadband traffic has increased by $50-70\%$ yearly and this trend seems to continue or even increase in the next coming years [1, p. 23], [2]. To cater for this growth, technological advancements must be made in order for the mobile network to support the needed capacity increase. [1, p. 23]

One of the technologies used to increase the capacity of the mobile network are the Massive Multiple Input Multiple Output (MIMO) systems, a type of advanced antenna system (AAS). The MIMO systems utilizes antenna arrays. These arrays consist of groups of antenna elements arranged in various configurations. Figure 1.1 shows a planar antenna array consisting of 135 dual-polarized antenna element pairs divided into subarrays. These subarrays are the smallest dynamically controllable parts of the antenna array. They are each controlled by two radio chains, one for each polarization. The amplitude and phase of the signal going to each of these subarrays can be independently controlled in baseband software using beamweights. [1, Technology Primer p. 13-23].

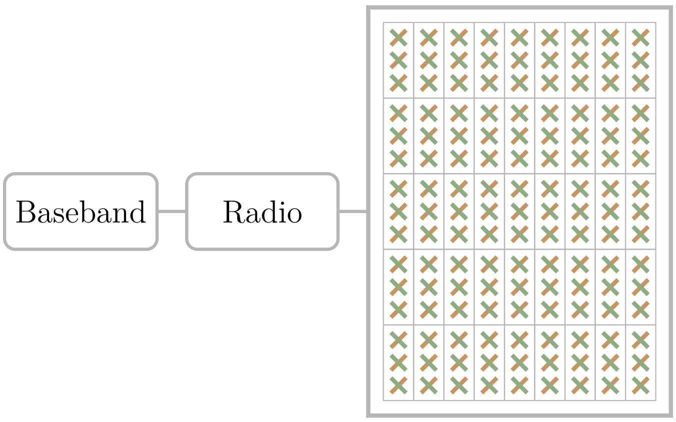


Figure 1.1: An example antenna array together with its radio and baseband unit: Each cross in the picture represents a dual-polarized antenna element pair and three of these elements enclosed in a rectangle represents a subarray. The color of the crosses represents the polarization of the element.

An important technique enabled by AAS systems is beamforming, which allows for better network coverage, capacity, and user throughput [1, p. 15]. Beamforming is achieved by weighting the radiation contribution from the subarrays using beamweights [1, Technology Primer, p. 17].

A common set of methods for determining the beamweights are stochastic optimization methods. An example of a stochastic algorithm is the genetic algorithm. This can be used to fit the radiation to a specific desired pattern, maximize the power throughput or to minimize the sidelobe levels, to name a few examples. In [3], the NSGA-II algorithm was used to minimize the peak side lobe level while simultaneously maximizing the directivity in order to improve the radiation pattern. An improved version of the algorithm, INSGA-II, was used in [4] for a similar purpose. Stochastic optimization algorithms are also used at Ericsson AB for their beamweight optimization.

In recent years, other machine learning (ML) models such as neural networks have been used to generate beamweights. In [5] a convolutional neural network (CNN) was used for 2D beamforming of antenna arrays. The study showed that the CNN was able to estimate nearly optimal Wiener array weights. In [6] a complex valued CNN was used for a similar purpose, also showing promising results. However, both of these studies trained and evaluated their models on simulated data. The purpose of this thesis is to implement, train and evaluate similar ML models on measurements of Ericsson AB AAS products.

1.1 Aim

The main objective of this thesis is to implement a predictive ML model which generates beamweights for any given antenna array such that the resulting far field resembles a predetermined shape. The model should be based and evaluated on measurements of Ericsson AAS products.

The current implementation at Ericsson is a stochastic optimization algorithm and part of the goal is to compare the performance of the two approaches.

1.2 Research Questions

To reach the goal of the thesis some research questions are to be answered. They are:

- Is it possible to train a ML model to predict beamweights that generate any given Gaussian reference beam?
- How well does the model perform on the data provided by Ericsson?
- How does the predictive model perform in relation to a stochastic optimization algorithm used at Ericsson?
- Is the model scalable? How does the model perform on different sizes of antenna arrays?
- How stable are the outputs of the model?

1.3 Limitations

The scope of this thesis will be limited and therefore only consider the radiation of the antenna in the far field as this is where all user equipment, such as mobile phones are. The antennas will also be assumed to be in free space. Only measurements of antennas from Ericsson and Ericsson vendors in the sub 6 GHz range will be considered. The measurements provided are assumed to be correct. The thesis will also be limited to only consider a linear antenna array, i.e. one row of subarrays. The resulting far field will only be evaluated in a cut in azimuth. Lastly, only one loss function will be considered.

1.4 Thesis Outline

The thesis consists of six chapters. After the introduction in chapter 1, is the necessary theory explained in chapter 2. This chapter is divided into four main sections. The first section contains the relevant antenna and electromagnetic field theory. The second explains some necessary complex calculus. The third covers artificial neural networks and the fourth covers some important optimization concepts.

The methodology is covered in chapter 3 and it includes the modeling of the project, the data handling and the tests made to evaluate the final model.

Chapter 4 shows the results found during the tests defined in the method chapter. The results presented are then discussed and analysed in chapter 5.

Finally, some conclusions from the analysis are presented in chapter 6 together with some suggested further advancements.

2
Theory

The following chapter will introduce relevant theory and concepts to the thesis. First, the fundamentals of the antenna theory needed for the model will be described. A brief overview of complex calculus is introduced followed by relevant concepts of artificial neural networks. Furthermore, a description of how artificial neural networks and complex calculus can be combined is included. Lastly the chapter contains a brief introduction to optimization.

2.1 Antenna Theory

This thesis revolves around determining optimal beamweights for different antenna arrays in different scenarios. The aim of this section is to explain these concepts in detail and to give the necessary antenna and electromagnetic field theory needed to complete the thesis task.

2.1.1 Coordinate Systems

Throughout this report two different coordinate systems will be used: Cartesian coordinates and spherical coordinates. A visual representation of both coordinate systems can be seen in Figure 2.1. These coordinate systems can be related through the equations

$$\begin{aligned} x &= r\sin\theta\cos\phi, \\ y &= r\sin\theta\sin\phi, \\ z &= r\cos\theta, \end{aligned}$$

as stated in [7, p. 35]. In general the coordinate system will be placed such that the antenna elements lies in the y-z-plane with rows parallel to the y-axis, columns parallel to the z-axis and the lower left subarray centered in the origin. For the purpose of this report, the angle θ will be referred to as angle in elevation and the angle ϕ will be referred to as angle in azimuth. These angles will both have a respective unit vector pointing in their direction of increase. These are defined to be

$$\begin{aligned} \hat{\theta}(\theta,\phi) &= \hat{\mathbf{x}}\cos\theta\cos\phi + \hat{\mathbf{y}}\cos\theta\sin\phi - \hat{\mathbf{z}}\sin\theta, \\ \hat{\phi}(\phi) &= -\hat{\mathbf{x}}\sin\phi + \hat{\mathbf{y}}\cos\phi. \end{aligned}$$

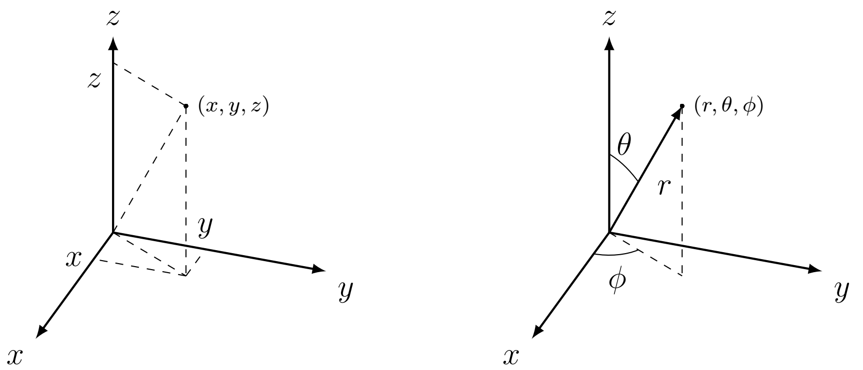


Figure 2.1: The coordinate systems used in the report: Left shows Cartesian coordinates and right shows spherical coordinates. Source: [8], CC BY-SA 4.0

2.1.2 Decibel

Decibel (dB) is a logarithmic measure, in this case, of power. For antennas the dB value is often presented as a relative measure of either amplitude or power density. The calculations for the relative measures dB are

$$
\begin{aligned}
|\frac{A}{A_{ref}}|_{\mathrm{dB}} &= 20\log_{10}|\frac{A}{A_{ref}}|\mathrm{dB}, \\
|\frac{P}{P_{ref}}|_{\mathrm{dB}} &= 10\log_{10}|\frac{P}{P_{ref}}|\mathrm{dB}.
\end{aligned}
$$

Where A is the amplitude, P the power density, A_{ref} and P_{ref} is the reference value in amplitude and power density respectively [7, p. 9-10].

2.1.3 Frequency and Bandwidth

The frequency is a measure of occasions of something per time unit, the unit of frequency is Hertz (Hz) which is equal to 1/second. Frequency in the sense of antennas is how many times the electromagnetic wave oscillates per second. The wave is travelling with constant speed, the speed of light, $c = 2.998 \cdot 10^8$ meters per second in free space and $c = f \cdot \lambda$. Where λ is the length of one oscillation in meters and f the frequency in Hertz. The frequency is then inversely proportional to the wave length with a constant, c, the speed of light in free space [9].

Bandwidth is the range of frequencies which an antenna is able to transmit. An antenna radiating between 2.5 and 2.6 GHz has a bandwidth of 0.1 GHz or 100 MHz. Normally antennas are divided in categories depending on which frequencies they transmit. In telecommunication there are low-, mid- and high band. Low band is below 1 GHz, mid band is 1 to 6 GHz and high band over 24 Ghz. High band is sometimes referred to as millimeter wave (mmWave) [10].

2.1.4 Field Regions

The electro-magnetic radiation of an antenna can be divided into different regions depending on how far from the antenna a measurement is done. For the purpose of this thesis, only the so called far field is considered and in this region the radiation pattern, see section 2.1.6, does not change shape when moving further from the antenna. Only considering the far field is reasonable since most antennas operate in the far field [11], including the antennas examined in this report. To be in the far field the following condition must be met

$$r \geq \frac{2D^2}{\lambda},$$

where r is the distance from the antenna, D is the maximum diameter of the antenna and λ is one wavelength [7, p. 33-34]. An important aspect of the far field is that it shares properties of a plane wave [12, p. 605]. Therefore, at an observation point in the far field we can always assume the radiation field to be a propagating plane wave locally. In the same way, incident fields on a receiving antenna can most often be considered as plane waves [7, p. 23].

2.1.5 Plane Waves

A plane wave propagating in the positive z-direction in free space, is described by its time-harmonic electric, $\mathbf{E}$, and magnetic, $\mathbf{H}$, fields by

$$\mathbf{E} = \mathbf{E}_t e^{-jkz} = \left[E_{x,t}\hat{\mathbf{x}} + E_{y,t}\hat{\mathbf{y}}\right] e^{-jkz}, \tag{2.1}$$

$$\mathbf{H} = \frac{1}{\eta}\left(\hat{\mathbf{z}} \times \mathbf{E}_t\right) e^{-jkz} = \frac{1}{\eta}\left[-E_{y,t}\hat{\mathbf{x}} + E_{x,t}\hat{\mathbf{y}}\right] e^{-jkz}. \tag{2.2}$$

Where $\mathbf{E}_t$ is the E-field vector without the propagation factor e^{-jkz}, $k = 2\pi/\lambda$ is the wavenumber in free space, η is the wave impedance [7, p. 23] and j is the imaginary unit, such that $j^2 = -1$.

More generally, a plane wave propagating in the direction of the unit vector $\hat{\mathbf{r}}$, is described by its electric, $\mathbf{E}$, and magnetic, $\mathbf{H}$, fields by

$$\mathbf{E} = \mathbf{E}_t e^{-j\mathbf{k}\cdot\mathbf{r}},$$

$$\mathbf{H} = \frac{1}{\eta}\left(\hat{\mathbf{r}} \times \mathbf{E}_t\right) e^{-j\mathbf{k}\cdot\mathbf{r}}.$$

Where $\mathbf{k} = k\hat{\mathbf{r}}$ is the wavenumber vector and $\mathbf{r} = x\hat{\mathbf{x}} + y\hat{\mathbf{y}} + z\hat{\mathbf{z}}$ is a radius vector from the origin. An important aspect of plane waves are that $\mathbf{E}$ and $\mathbf{H}$ are normal to both each other and the direction of propagation, $\hat{\mathbf{r}}$ [7, p. 23], [12, p. 356, 361–363].

2.1.5.1 Polarization

The electric and magnetic fields propagating in the positive z-direction in equations (2.1) and (2.2) have both x- and y-components. This electromagnetic wave is then said to be polarized. This property is often used in antenna systems as it makes it possible to send two different signals on the different polarizations, E_x and E_y, simultaneously in the same frequency. Generally, any pair of orthogonal polarizations can be used to reuse the frequency by transmitting on both polarizations and thus doubling the capacity. In these cases one of the polarizations is considered to be desired and the other not. The desired polarization is often called co-polar and the orthogonal undesired polarization is called cross-polar. Using these definitions we can describe the E-field in terms of the desired co-polar component, E_{co}, and a undesired cross-polar component, E_{xp}. Both of these are parallel to a respective unit vector, $\hat{\mathbf{co}}$ and $\hat{\mathbf{xp}}$, which are normal to both each other and the direction of propagation. The E-field can then be written as

$$\mathbf{E} = (E_{co}\hat{\mathbf{co}} + E_{xp}\hat{\mathbf{xp}})\, e^{-j\mathbf{k}\cdot\mathbf{r}},$$

where $\mathbf{r}$ is a radius vector from the origin [7, p. 23-24]. Another use of the polarizations is a technique called dual-polarized beamforming (DPBF) [13] which makes it possible to generate broader beams, see section 2.1.6, from antenna arrays with less power lost [14].

In words, the polarization of a plane wave describes how the electric field intensity vector at a fixed point in space varies with time. If an electric field is confined to a plane in the direction of its propagation, $\hat{\mathbf{r}}$, then the wave is said to be linearly polarized. An example of this is an electric field propagating in the z-direction that is fixed in the x-direction [12, p. 364]. Linearly polarized waves are often referred to as being horizontal or vertical in reference to the ground [7, p. 24]. It is common for AAS and user equipment products to use dual-polarized antenna setups with two linearly polarized elements [1, Technology Primer, p. 32], such as the one in Figure 1.1 with $\pm 45°$ polarizations.

2.1.6 Radiation Pattern

The radiation pattern, $F(\theta, \phi)$, of an antenna describes the antennas relative far field power at a fixed distance from the antenna as a function of the direction away from the antenna, in this case using spherical coordinates. In general, a radiation pattern is three-dimensional, varying with both θ and ϕ, but is often presented in a combination of two dimensional plots, one for the radiation in azimuth and one in elevation [12, p. 607-608].

A radiation pattern that is equal in all directions is often called isotropic, or that of a point source. These patterns are often used as a reference point in comparison with practical antennas [7, p. 10].

The radiation pattern of a practical antenna generally has a major maximum and several minor maxima, as in Figure 2.2. The region containing the major maximum of such a pattern is often denoted the main beam and is defined to be the region of maximum radiation between the first null points around it. The similar regions around the minor maxima are often called sidelobes. When comparing various antenna patterns, there are a three characteristic parameters that are of importance, directivity, main beamwidth and sidelobe levels [12, p. 608-609]. These will be the topic of the next sections as they are crucial for the understanding of this thesis.

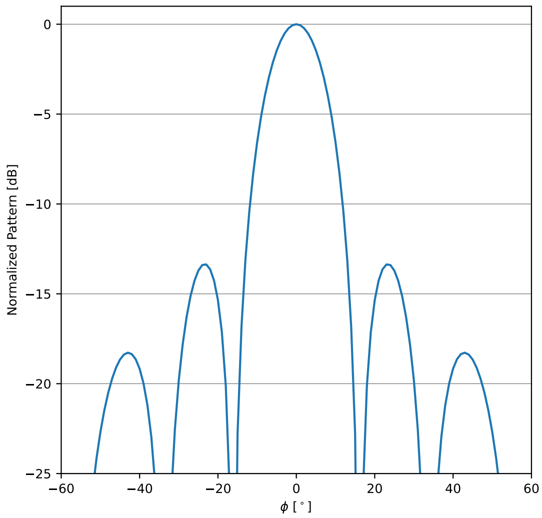


Figure 2.2: Example radiation pattern: The figure shows a peak normalized radiation pattern of a linear antenna array with 8 antenna elements arranged along a straight line in azimuth.

2.1.6.1 Directivity and Gain

Directivity is a measure of how focused the radiation pattern is towards a point, a narrower radiation pattern gives higher directivity and a wider radiation pattern gives lower directivity. The directivity is calculated by

$$D = \frac{4\pi}{\int_0^{2\pi}\int_0^{\pi}|F(\theta,\phi)|^2 sin(\theta)d\theta d\phi}.$$

Where θ is the angle in elevation, ϕ is the angle in azimuth and $F(\theta, \phi)$ is the antenna's normalized radiation pattern as a function in spherical coordinates. The radiation pattern in $F(\theta, \phi)$ is normalized such that it's peak value in magnitude is equal to 1 [15].

The gain of an antenna describes the radiated power in the peak direction to that of an isotropic source. The gain, $\mathcal{G}$, of an antenna is related to it's directivity through

$$\mathcal{G} = \epsilon_R D$$

where ϵ_R is the radiation efficiency of the antenna [16]. The antenna radiation efficiency can in turn be expressed as the ratio of the radiated power to the input power of the antenna, or

$$\epsilon_R = \frac{P_{radiated}}{P_{input}},$$

as stated in [12, p. 612].

2.1.6.2 Beamwidth

As described above, the width of the main beam, or beamwidth, is of much importance as it describes the directivity of the main radiation region. The beamwidth is often measured in the angle where the radiated power is halved compared to the direction where the maximum power is radiated. This is called half power beamwidth (HPBW) and is measured in angle between the points where the relative gain is -3 dB (half power).[12, p. 610].

2.1.6.3 Sidelobe Levels

The sidelobes are power radiated in another direction than the main beam. These are usually smaller than the main lobe and unwanted. Because of this they should have as low levels as possible. In general, the levels of the sidelobes closest to the main beam are the highest. Therefore, one usually refers the sidelobe level to the level of the sidelobe that is highest and closest to the main beam. In practice the location of the sidelobe might also be of interest [12, p. 610]. The sidelobes may interfere with other antennas such as satellites. Because of this there are requirements on sidelobe suppression [17].

2.1.6.4 Radiation Pattern of Antenna Arrays

The radiation pattern of an antenna array is highly dependent on the number of antenna elements in the array. It affects the shape but also the gain of the antenna. A uniform linear antenna array consists of equidistant elements with identical radiation patterns that are spread out in a line. For such an antenna the gain of the antenna array is proportional to the number of elements in the array. If the elements of the linear array is spread out in azimuth, then the main beamwidth in azimuth is proportional to the inverse of the number of elements in the array. The beamwidth in elevation is however unchanged, but the gain is still higher. For a linear array with elements spread out in elevation the opposite properties are true [1, Technology Primer, p. 17-18].

A uniform planar array consists of a grid of elements instead of a line. For such an antenna the gain scales with both the number of elements in elevation and in azimuth but the beamwidth in elevation and azimuth is still only proportional to the inverse number of elements in elevation and azimuth respectively [1, Technology Primer, p. 19].

2.1.7 Far Field Function

If the radiated field of an antenna is observed in the far field then the field variations with distance r, and direction $\hat{\mathbf{r}} = \mathbf{r}/r$ from the antenna become separable [7, p. 33]. The electric field can then be expressed as

$$\mathbf{E}(\mathbf{R}) = \frac{1}{r} e^{-jkr} \mathbf{G}(\hat{\mathbf{r}}), \tag{2.3}$$

where $1/r$ is called the divergence factor, e^{-jkr} is the phase factor and $\mathbf{G}(\hat{\mathbf{r}})$ is the so called complex far field function. The radiation field is often expressed in spherical coordinates, in that case the direction vector is expressed as

$$\hat{\mathbf{r}} = \sin\theta\cos\phi\hat{\mathbf{x}} + \sin\theta\sin\phi\hat{\mathbf{y}} + \cos\theta\hat{\mathbf{z}},$$

and the far field is expressed as

$$\mathbf{E}(r,\theta,\phi) = \frac{1}{r}e^{-jkr}\mathbf{G}(\theta,\phi), \tag{2.4}$$

as stated in [7, p. 35]. Since the far field is locally a plane wave we can divide it into co- and cross-polar components as done in section 2.1.5.1, [7, p. 38-39].

An important note is that e^{-jkr} in equation (2.3) and (2.4) is only valid in free space [7, p. 33]. For the purpose of this report however, only antennas in free space will be considered.

The phase of the radiation field in (2.4) is determined by both the exponent of the phase factor as well as the phase of the far field function $\mathbf{G}(\theta,\phi)$. The phase of the radiation field at a given point in space must be equal regardless of where the origin is located under the assumption that the antenna is located in the same point in space. Therefore, the far field function must have a phase reference point, usually the origin. If the phase reference point is moved from the origin to an arbitrary point $\mathbf{r}_0 = x_0\hat{\mathbf{x}} + y_0\hat{\mathbf{y}} + z_0\hat{\mathbf{z}}$ then the far field function $\mathbf{G}'(\theta,\phi)$ with $\mathbf{r}_0$ as phase reference point is given by

$$\mathbf{G}'(\theta,\phi) = \mathbf{G}(\theta,\phi)e^{-jk\mathbf{r}_0\cdot\hat{\mathbf{r}}},$$

using the Fraunhofer approximations, where $\mathbf{G}(\theta,\phi)$ is the far field function with reference in the origin [7, p. 37].

If instead the antenna is moved from the origin to a point $\mathbf{r}_A$, then the far field function of the antenna at the new location becomes

$$\mathbf{G}_A(\theta,\phi) = \mathbf{G}(\theta,\phi)e^{jk\mathbf{r}_A\cdot\hat{\mathbf{r}}}, \tag{2.5}$$

where both $\mathbf{G}_A(\theta,\phi)$ and $\mathbf{G}(\theta,\phi)$ has the origin as the phase reference point [7, p. 38].

The complex far field function can be related to the radiation pattern by

$$F(\theta,\phi) = \left|\mathbf{G}(\theta,\phi)\right|,$$

as stated in [7, p.41].

2.1.7.1 Beamweights and Far Field Function of a Planar Antenna Array

An antenna array with elements arranged in a planar two-dimensional grid is often referred to as a planar array. Now consider a planar array with N elements oriented in the same direction that are located at their individual phase reference points, $\mathbf{r}_n$ for $n = 1, 2, \ldots, N$. Then the radiated far field of element n is given by

$$\mathbf{E}(r, \theta, \phi) = \frac{1}{r} e^{-jkr} \mathbf{G}_n(\theta, \phi) e^{jk\mathbf{r}_n \cdot \hat{\mathbf{r}}},$$

using equation (2.4) and (2.5) where $\mathbf{G}_n(\theta, \phi)$ is the far field function of element n with phase reference point $\mathbf{r}_n$. If the fields radiated by all elements are superimposed then the far field function of the entire array becomes

$$\mathbf{G}_A(\theta, \phi) = \sum_{n=1}^{N} A_n e^{j\Phi_n} \mathbf{G}_n(\theta, \phi) e^{jk\mathbf{r}_n \cdot \hat{\mathbf{r}}}, \tag{2.6}$$

where A_n is the amplitude and Φ_n is the phase of the current or voltage excitation of element n [7, p. 335, 356–357]. In this report the factors $A_n e^{j\Phi_n}$, $n = 1, \ldots, N$ will be referred to as beamweights, as done in [1].

2.2 Complex Calculus

A complex number is on the form $a + j \cdot b$ where j is defined as $j^2 = -1$ and $a, b \in \mathbb{R}$. The space of complex numbers is defined as $\mathbb{C} = \{(a, b) : a, b \in \mathbb{R}\}$ where a and b are pairs of real numbers. Two different complex numbers $a + j \cdot b$ and $x + j \cdot y$ has addition and multiplication as

$$\begin{aligned}(a + jb) + (x + jy) &= (a + x) + j(b + y),\\ (a + jb) \cdot (x + jy) &= (xa - yb) + j(xb + ya).\end{aligned}$$

From this it follows that complex numbers are a field under these operations and that $\mathbb{R} \subset \mathbb{C}$. Define $\mathrm{Re}(a + jb) = a$ and $\mathrm{Im}(a + jb) = b$, the real and imaginary part of a complex number [18, p. 1-3].

2.2.1 Differentiation

A complex function is defined as $f : G \to \mathbb{C}$ where G is a subset to $\mathbb{C}$. As for the complex numbers a complex function can be represented as two real valued functions by $f(z) = f(x, y) = u(x, y) + jv(x, y)$ where (x, y) is a pair defining a complex number. The definition of the derivative of a complex function $f(z)$ at z_0 is given by

$$f'(z_0) = \lim_{h \to 0} \frac{f(z_0 - h) - f(z_0)}{h},$$

if f is differentiable in all points in a open disc around z_0 the function is called holomorphic. If a complex function $f(x, y)$ is differentiable it must satisfy the Cauchy-Riemann equations

$$\frac{\partial u}{\partial x} = \frac{\partial v}{\partial y},$$
$$\frac{\partial u}{\partial y} = -\frac{\partial v}{\partial x},$$

where $f(x, y) = u(x, y) + jv(x, y)$ [18, p. 19-24]. However, many commonly used functions does not satisfy the Cauchy-Riemann equation so the theory of Wirtinger calculus is used. In Wirtinger calculus a complex function is rewritten to a function with two variables, which is always holomorphic. It is rewritten as $f(z, z^*)$ where z^* is the complex conjugate of z and the real and imaginary part of z is

$$\mathrm{Re}(z) = \frac{z - z^*}{2},$$
$$\mathrm{Im}(z) = \frac{z - z^*}{2j}.$$

This results in the derivatives

$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial z} + \frac{\partial f}{\partial z^*},$$
$$\frac{\partial f}{\partial y} = j\left(\frac{\partial f}{\partial z} - \frac{\partial f}{\partial z^*}\right),$$

which results in

$$\frac{\partial f}{\partial z} = \frac{1}{2}\left(\frac{\partial f}{\partial x} - j\frac{\partial f}{\partial y}\right), \tag{2.7}$$
$$\frac{\partial f}{\partial z^*} = \frac{1}{2}\left(\frac{\partial f}{\partial x} + j\frac{\partial f}{\partial y}\right), \tag{2.8}$$

as stated in [19].

2.3 Optimization

Optimization is a field in mathematics which aims at finding an optimal value to a specified goal within constraints. Often this goal is to find the minimum or maximum of a function, called objective function, within a space of interest. Different types of optimization exists, one is classical optimization which works well for some problems. Another type of optimization is stochastic optimization which incorporates stochasticity into the optimization which can be of help when working with highly nonlinear objective functions [20, p. 1-5]. The objective function can be multi dimensional and optimize towards multiple values at one. This is known as multi objective optimization [21].

2.3.1 Stochastic Optimization

Stochastic optimization is an optimization technique often inspired by biology, the inspiration comes from different adaptations in nature. The adaptation to an environment can be seen as optimization with the goal of surviving. A stochastic optimization algorithm inspired by this is genetic algorithms (GAs) as the inspiration for it is evolution of species. A GA is based on a set of chromosomes, such a set is called a population and the chromosomes are built by genes. The population is randomly initialized. The chromosomes are a representation of the objective function and their values are known as fitness. The chromosomes are manipulated to find better and better solutions. The manipulation is made by mutation and crossover, mutation is randomly changing genes and crossover is randomly switching genes between different chromosomes. The algorithm tends toward better values as the probability for crossover is higher for better chromosomes [20, p. 35-44].

Another stochastic optimization algorithm is differential evolution (DE) which also uses a randomly initialized population. The population in DE consists of vectors that are a representation of the objective function with a connected fitness. To update the population for each individual in the population two other random individuals are chosen and a vector difference between them is computed. If the new vector has better fitness than the old individual, the old individual is replaced [22, p. 189-191].

A third stochastic optimization algorithm is simulated annealing (SA) which is inspired by thermodynamics. The standard SA does not include population as the others, only a state. In each iteration a random neighbour to the current state is chosen, if the new state has better fitness than the current, move to the new state. If the new state has worse fitness it can be accepted by an acceptance probability given by the current temperature. The temperature is decreasing in each step in order to decrease the acceptance probability with time [23, p. 1-39].

2.3.2 Multi Objective Optimization

When optimizing towards multiple objectives there could be many optimal solutions, some perform well on one objective and bad on another and vice versa. The set of optimal solutions can be found using Pareto fronts. To define Pareto fronts we must first define, as in [21, Definition 5]:

Definition 2.3.1 (Pareto Dominance)**.** Given two points in the objective space, $\mathbf{x}, \mathbf{y} \in \mathbb{R}^m$, then the point $\mathbf{x}$ is said to *Pareto dominate* the point $\mathbf{y}$ if and only if

$$\forall i \in \{1, \ldots, m\} : x_i \leq y_i \text{ and } \exists l \in \{1, \ldots, m\} : x_l < y_l.$$

Where the objective space is the set of possible solutions to objective function. A Pareto front can then be defined, as in [21, Definition 8]:

Definition 2.3.2 (Pareto front)**.** Denote the set of attainable objective vectors with $Y = \mathbf{F}(X)$. The set of all non-Pareto dominated objective vectors in Y is called

Pareto Front.

Here X is the search space, the set of feasible points, and Y is the objective space. The Pareto front is then the set of optimal solutions to the objective function [21].

One stochastic optimization algorithm to find the Pareto front is the NSGA-II, Non-dominated Sorting Genetic Algorithm. NSGA-II uses a randomly initialized population. Then until a termination criterion stops the algorithm, the population generates a child population. The previous population is then known as the parent population. The individuals, in both populations, which has fewest Pareto dominating individuals becomes the new population. When the termination criterion is fulfilled the current population has a subset of individuals considered the Pareto front [21].

2.4 Artificial Neural Networks

An Artificial Neural Network (ANN) is inspired by the neural networks in a mammal brain. These networks are built on connected neurons which perform computations in order to process data. The artificial neural networks are simplified both in the connections and the neurons, however the underlying concept is the same. A neuron in an ANN receives a number of weighted inputs. It has an activation function and a threshold, also known as bias, as can be seen in Figure 2.3. The output of the neuron is the activation function computed on the sum of the weighted inputs and a threshold connected to the neuron. [24, p. 1-12].

2.4.1 Artificial Neurons

An ANN with these type of neurons has an input layer where the data is parsed, the inputs are then connected to a number of neurons. These neurons are called a layer. The neurons in the first layer are then connected with the neurons in the second layer and so on. The last layer is known as the output layer and the layers in between are known as hidden layers [24, p. 75–95]. An example of a neural network with this structure can be seen in Figure 2.4, however the number of hidden layers is not limited to three.

2.4.2 Feed-Forward Neural Network

A feed-forward neural network is an ANN with m layers and n_ℓ neurons in layer ℓ where neurons in layer ℓ are connected to one or more neurons in layer $\ell + 1$ for all m layers. If each neuron in layer ℓ is connected to each neuron in layer $\ell + 1$, the feed forward neural network is called fully connected [24, p. 114–140]. An example of a feed-forward fully connected neural network is shown in Figure 2.4 where each neuron in a layer is connected to every neuron in the following layer. The input layer denoted x_i can be considered as $V_i^{(0)}$ and the output layer y_i as $V_i^{(L)}$ where

$L = m + 1$ and m is the number of hidden layers as these nodes can be represented in the same way as hidden neurons.

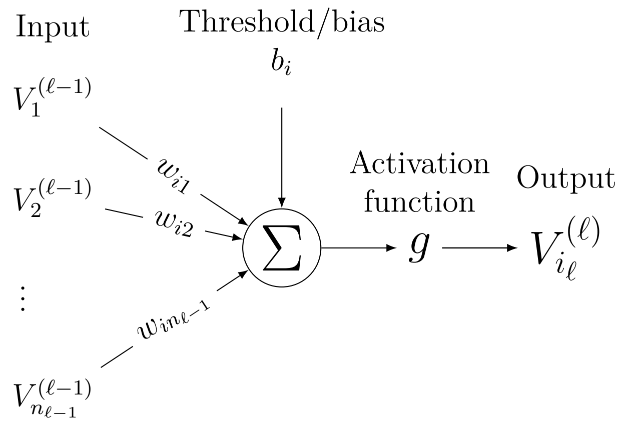


Figure 2.3: A single neuron: The figure shows a single neuron in an ANN with $n_{\ell-1}$ inputs and their respective weights together with the threshold and activation function. This is neuron i in layer (ℓ).

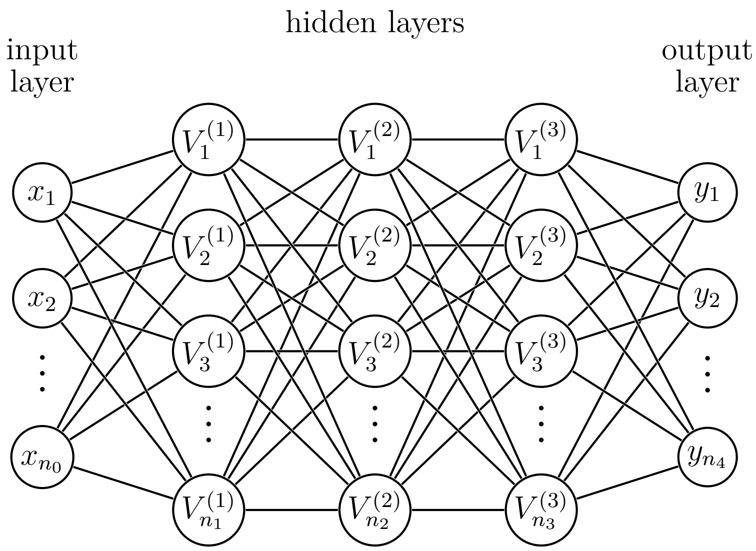


Figure 2.4: Fully connected feed forward neural network: The figure shows a fully connected neural network with n_0 inputs, three layers of n_1, n_2 and n_3 hidden neurons respectively and an output layer with n_4 outputs. Adapted from [25], CC BY-SA 4.0.

2.4.3 Convolutional Neural Network

A Convolutional Neural Network (CNN) differs from a fully connected neural network as it has fewer connections between nodes. This reduces the number of computations in training and reduces the risk of overfitting. CNNs are commonly used when the input data is an image and is popular for image classification and pattern recognition. A CNN has the following structure, one or more convolutional layers, one or more pooling layers and a fully connected feed forward neural network [24, p. 141–156].

2.4.3.1 Convolutional Layers

In the convolutional layer the input connects to a hidden layer by a kernel, the kernel is of fixed size $m \times n$ and relates each $m \times n$ part of the input to a single value in the hidden layer. The layers in the hidden layer are known as feature maps. If there is more than one feature map there is one kernel for each, however the size of the kernel is the same. In Figure 2.5 a convolutional layer is visualized with a 3×3 kernel to one feature map. In the figure the kernel is applied to each 3×3 squares moving one square at the time. It could be valuable to move more than one square at the time, the number of squares moved is called stride. Stride $[2, 2]$ is two steps horizontally and two step vertically. This is sometimes referred to as stride two, when the stride is the same in both directions. Padding could also be added to the input. It is an extra row or column of zeros added to the beginning or end of the input. The weights in the convolutional layer are the values in the kernel. The case in Figure 2.5 therefore has 9 weights, and one threshold. Assuming equal stride s horizontally and vertically and kernel size $P \times Q$ the computation for the feature map is

$$V_{ij} = g\left(\sum_{p=1}^{P}\sum_{q=1}^{Q} w_{pq}x_{p+s(i-1),q+s(j-1)} - b\right), \tag{2.9}$$

where b is the threshold, x the two dimensional input and g is an activation function [24, p. 141–156].

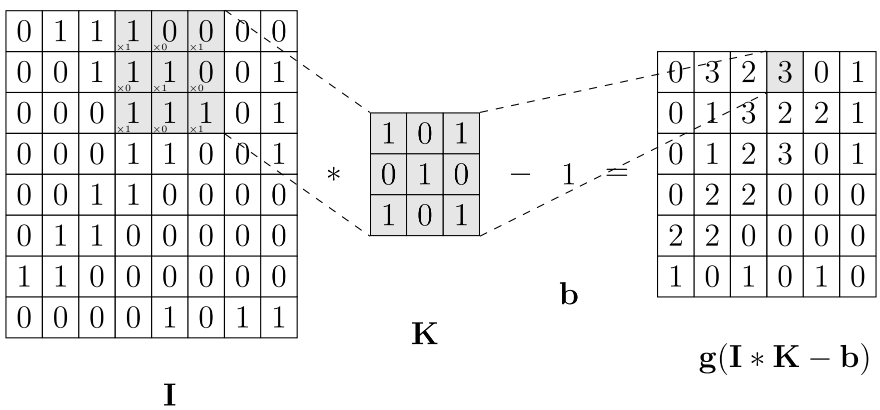


Figure 2.5: Convolutional layer in a convolutional neural network: The figure shows an example of a CNN with input size 8×8 and kernel size 3×3 and threshold 1. The activation function g in this layer is the ReLU function covered in section 2.4.4. This convolutional layer has zero padding and stride one. It visualizes how a convolutional layer takes an input on which the kernel is applied and how a single feature map is computed from that. Adapted from [26], MIT-license.

2.4.3.2 Pooling Layers

Following the convolutional layer is the pooling layer, the pooling layer compresses values neighbouring each other in the feature map. A pooling layer do not have any weights or thresholds. A 2×2 pooling layer takes 4 values in a square and compresses them to one value in the output layer from the feature maps. The pooling layer uses stride and padding in the same way as the convolution layers. A common set of pooling functions are the max-pooling units. These takes the maximum of the values in the pooling unit [24, p. 141–156]. Figure 2.6 shows an example of a max-pooling layer. The figure shows that the information is compressed as the number of features are reduced from 36 to 9.

Feature map

1	4	3	4	1	2
1	2	4	3	3	2
1	2	3	4	1	2
1	3	3	1	1	1
3	3	1	1	0	0
2	1	2	0	2	1

Max-pooling

4	4	3
2	4	2
3	2	2

Figure 2.6: A max-pooling layer: The figure shows a single feature map with a max-pooling unit applied to it. The max-pooling unit is of size 2×2 with stride $[2, 2]$ and no padding.

Another set of common pooling functions are the L_2-pooling units, these return the root mean square of the values. After the last pooling layer the convolutional neural network connects to a fully connected neural network, as the one in Figure 2.4, where the output of the pooling layer is connected to the nodes in the first hidden layer in the fully connected network [24, p. 141–156].

2.4.4 Activation Functions

The activation function in the neuron is a function of the sum of inputs and a threshold. Lets denote the inputs as $V_k^{(\ell-1)}$, the weights associated with them as w_{jk} and the threshold b_j. The inputs can be either the inputs to the ANN or outputs from a previous layer. The output from a single neuron given an activation function $g(f_j^{(\ell)})$ is then

$$V_i^{(\ell)} = g\left(\sum_k w_{ik} V_k^{(\ell-1)} - b_i\right), \tag{2.10}$$

where $V_i^{(\ell)}$ is neuron i in layer ℓ. The equation $f_i^{(\ell)} = \sum_k w_{ik} V_k^{(\ell-1)} - b_i$ is known as the local field of the neuron $V_j^{(\ell)}$. If this is a hidden layer the output becomes input to the next layer. If it is the output layer this is the output of the network [24, p. 75–95].

The activation function is dependent on the kind of problem the ANN is used for, some common activation functions is the rectified linear unit (ReLU) function and the hyperbolic tangent function, tanh. The ReLU function is given by

$$g(f) = max\{0, f\}, \tag{2.11}$$

also ReLU is inspired by neurons in the brain [24, p. 1-12, 96–113].

2.4.5 Loss Functions

A loss function in an ANN is the function which measures how well the network performs. The loss function takes the output of the model and a target as input and return a measure of how close the output is to the target. The lower the loss function the better the performance of the network. When training a network the loss function is minimized using an optimizer [24, p. 27-30, 114–140]. The loss function can also be referred to as the energy function.

2.4.6 Optimizers

The training of a network is updating the trainable parameters, the weights and thresholds, for each neuron. The updating is done with an optimizer applied to the loss function. There are different optimizers which have unique updating rules. Two optimizers relevant for this thesis are stochastic gradient descent (SGD) and Adam optimizer (Adam) which both are described below.

2.4.6.1 Stochastic Gradient Descent

One optimizer is stochastic gradient descent which initializes the weights and thresholds randomly. During training the weights and thresholds are updated by derivatives of the loss function. The weights and thresholds are updated each iteration from a random subset of inputs, therefore the name stochastic gradient descent. The updates are computed by $w_{mn}^{(\ell)} = w_{mn}^{(\ell)} + \delta w_{mn}^{(\ell)}$ and $b_m^{(\ell)} = b_m^{(\ell)} + \delta b_m^{(\ell)}$ with

$$\delta w_{mn}^{(\ell)} = \eta \frac{\partial \mathcal{L}}{\partial w_{mn}^{(\ell)}}, \tag{2.12}$$

$$\delta b_m^{(\ell)} = -\eta \frac{\partial \mathcal{L}}{\partial b_m^{(\ell)}}, \tag{2.13}$$

where η is the learning rate, a hyperparameter chosen by the user and $\mathcal{L}$ is the loss function defined by the user to fit the specific case. The partial derivative of the loss function is calculated for the output of the network as the goal is to minimize the difference between the output and the target. Since the weights and thresholds in layer l is dependent on each neurons weights and thresholds in layer $l + 1$ which is dependent of layer $l + 2$ and so on until the output layer, these derivatives are heavily dependent on the chain rule for derivatives. One way to reduce the number of updates to compute is the concept of batching. A batch is a collection of inputs and the loss function for the batch is the sum of the loss functions for outputs of each input in the batch. Then the updating of the parameters is analogous to function (2.12)-(2.13). The term epoch is used when all training data is fed to the network and trained on, so 10 epochs is training the network on all data 10 times [24, p. 96-102].

2.4.6.2 Adam Optimizer

The Adam optimizer is another algorithm using stochastic gradients to minimize the loss function. The algorithm for updating one parameter is outlined in algorithm 1 which requires a stepsize, two decay rates, the loss function and the current value of the parameter to update. Usual values for the stepsize is $\alpha = 0.001$ and decay rates are $\beta_1 = 0.9, \beta_2 = 0.999$. The current value of the parameter is as previously $w_{mn}^{(\ell)}$ for the weights and $b_m^{(\ell)}$ for the thresholds [27].

Algorithm 1 Adam optimizer algorithm: A stochastic gradient algorithm for minimizing the loss function by updating the parameters in the neural network. The following algorithm is outlined for updating a single parameter and need to be applied for each trainable parameter in the network. The value of ϵ is assumed to be 10^{-8} [27].

Require: α: Learning rate
Require: β_1, β_2: Decay rates for the moments
Require: $\mathcal{L}$: The loss function
Require: p_0: Either a weight or a threshold
  $m_0 \leftarrow 0$
  $v_0 \leftarrow 0$
  $t \leftarrow 0$
  while p_t not converged **do**
    $t \leftarrow t + 1$
    $g_t \leftarrow \frac{\partial}{\partial p}\mathcal{L}(p_{t-1})$
    $m_t \leftarrow \beta_1 m_{t-1} + (1 - \beta_1) g_t$
    $v_t \leftarrow \beta_2 v_t + (1 - \beta_2) g_t^2$
    $\hat{m}_t \leftarrow m_t/(1 - \beta_1^t)$
    $\hat{v}_t \leftarrow v_t/(1 - \beta_2^t)$
    $p_t \leftarrow p_{t-1} - \alpha \hat{m}_t/(\sqrt{\hat{v}_t} + \epsilon)$
  end while
  return p_t

The Adam optimizer algorithm is related to the stochastic gradient descent as the step for calculating m_t has the component $(1 - \beta_1)g_t$ which is closely related to equation (2.12) if η was set to $1 - \beta_1$. The components m_t and v_t are known as the mean and uncentered variance of the gradient which are biased towards the initialization at 0. The bias is corrected by calculating $\hat{m}_t$ and $\hat{v}_t$ which are bias-corrected estimates of the moments. The term g_t^2 is the squared derivative of the loss function, in case of it being a vector the square is element wise [27].

2.4.7 Complex Valued Convolutional Neural Networks

The functions used in a neural network are defined for real valued input, and the functions themselves are real. There exist multiple variations to extend the CNN structure for complex values. There are variations for the different activation functions and variation in the convolutional and pooling layer. The different variations can be used to suit different types of problems [28].

2.4.7.1 Rectified Linear Unit

The standard ReLU function was defined in Equation (2.11). A common ReLU function for complex inputs known as ℂReLU is given by

$$g(x) = f(\mathrm{Re}(x)) + j \cdot f(\mathrm{Im}(x)),\ x \in \mathbb{C}, \tag{2.14}$$

where f is the real ReLU function defined in equation (2.11) and $\mathrm{Re}(x)$ and $\mathrm{Im}(x)$ are the real and imaginary parts of the input x [28], [29].

2.4.7.2 Hyperbolic Tangent

A common implementation of the complex hyperbolic tangent function is defined in the same way as the ReLU function by (2.14), with $f(x) = \tanh(x)$ [29], [30].

2.4.7.3 Convolutional Layer

The complex convolutional layer is defined as

$$g(x) = \Big[f(\mathrm{Re}(x)) - f(\mathrm{Im}(x))\Big] + j\Big[f(\mathrm{Im}(x)) + f(\mathrm{Re}(x))\Big], \tag{2.15}$$

where $\mathrm{Re}(x)$ and $\mathrm{Im}(x)$ are the real and imaginary parts of the input x and f is the real convolutional layer from section 2.4.3 [28], [29].

2.4.7.4 Maximum Pooling

For the complex maximum pooling the absolute value of the input is given as input to the real maximum pooling and it returns the complex value which corresponds to the largest absolute value [29], [31].

2.4.7.5 Optimizers

Both SGD and Adam use a variation of

$$x_{n+1} = x_n - \alpha \frac{\partial \mathcal{L}}{\partial x}, \tag{2.16}$$

for step size $\alpha \in \mathbb{R}$ and input $x \in \mathbb{R}$ but if the loss function, input or both is complex this can be written as

$$x_{n+1} = x_n - \frac{\alpha}{2}\frac{\partial \mathcal{L}}{\partial x}, \tag{2.17}$$

$$y_{n+1} = y_n - \frac{\alpha}{2}\frac{\partial \mathcal{L}}{\partial y}. \tag{2.18}$$

The complex update equation is then written as

$$z_{n+1} = z_n + \alpha\frac{\partial \mathcal{L}}{\partial z^*}, \tag{2.19}$$

which means that to compute the updates for the optimizers only the derivative of z^* need to be evaluated from equation 2.8 [19].

3
Methods

The model developed is a one dimensional complex valued convolutional neural network, CV-CNN. The model input consist of the antennas array factor kernel (AFK) described in section 3.1, together with a reference curve. The CV-CNN then outputs beamweights for this antenna and reference curve. The beamweights together with the AFK are used to calculate the radiated far field in a DPBF approach. The far field and the reference curve is then input to the loss function which calculates the loss for this input. The loss is then used with Adam optimizer to train the CV-CNN. The outline of the process is shown in Figure 3.1.

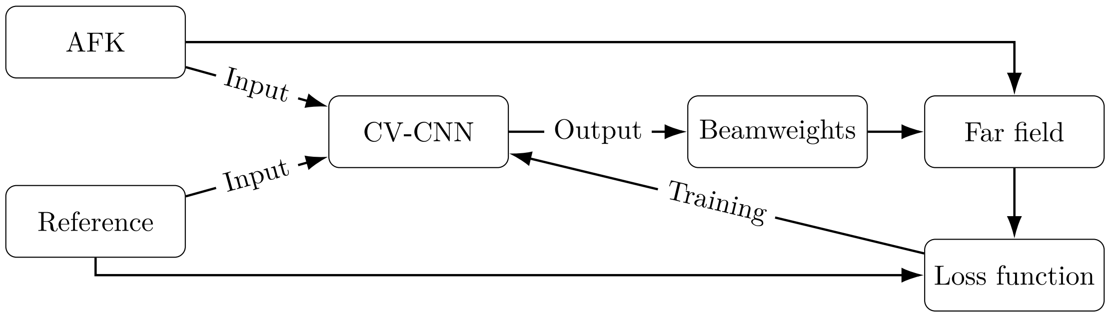


Figure 3.1: Outline of the process: The figure shows how the entire model is built. It illustrates how the input is connected to both the CNN and its connection to the loss function. The figure also shows how the loss function is connected to the training of the CNN.

The aim of this chapter is to explain each step in Figure 3.1 in detail as well as the tests performed in order to evaluate the model.

3.1 Input and Data Handling

The data used in this thesis was provided by Ericsson and it contained information of the antennas construction as well as measurements of the far field excited from each port of the antennas. The far field was measured at angels $\theta = 0°, 1°, ..., 180°$ and $\phi = -180°, -179°, ..., 180°$ at two polarizations defined by the unit vectors

$$\hat{\mathbf{co}} = \left(\hat{\theta} + \hat{\phi}\right) / \sqrt{2},$$
$$\hat{\mathbf{xp}} = \left(\hat{\theta} - \hat{\phi}\right) / \sqrt{2},$$

where $\hat{\theta}$ and $\hat{\phi}$ are the unit vectors defined in section 2.1.1. The measured co- and cross-polar components of the far field are denoted as

$$G_{co}^{(m,n,pol)}(\theta,\phi), \tag{3.1}$$

$$G_{xp}^{(m,n,pol)}(\theta,\phi), \tag{3.2}$$

where (m, n, pol) denotes the excited port mapped in accordance to Figure 3.2 where m denotes the subarray row, n the subarray column and $p = \pm 45°$ the polarization.

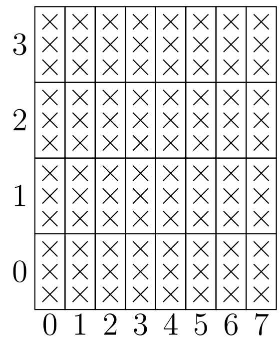


Figure 3.2: Port Mapping: The figure shows an example of how the rows and columns of subarrays are indexed when mapping the ports to the antenna arrays.

In total there were measurements from 62 antennas and all the measured antennas have 8 columns of subarrays but have a varying amount of subarray rows. A limitation of this project is to only consider a cut in azimuth, at $\theta = 90°$, of the resulting far field. Recalling from section 2.1.6.4, the shape of the peak normalized radiation pattern in azimuth only depends on the subarrays in azimuth, assuming that all rows of the array are equal. Another simplification was then to assume that azimuth and elevation excitation vectors can be treated independently, that means it is enough to optimize azimuth beamweights for a single row of subarrays. To increase the number of data points, each row of the antenna arrays can then be thought of as a separate antenna. If the $\pm 45°$ polarizations are assumed to be identical for a given antenna then the measurements of one polarization can be used for both when calculating the final far field. In this way the data can be doubled. This simplifies the problem and it allows the model to train on larger amounts of data but some information is lost in the process.

Each of the measured antennas are excited and measured at a varying number of different frequencies. The frequency of the excitation will impact the resulting far field of the antenna array through the wavenumber $k = 2\pi/\lambda$ in equation (2.6). Because of this, each row of each antenna excited at each frequency will be considered as a data point to increase the number of data points further. In total this resulted in 1764 data points.

The goal of the thesis is to construct a predictive model that can output beamweights for any given antenna such that the resulting far field mimics the shape of any given reference curve. To do this the input of the model must then contain the information of the antenna as well as the reference curve.

The information from the subarrays needed to calculate the resulting far field of the antenna array using equation (2.6) is contained in the factors

$$\mathbf{G}_n(\theta,\phi)e^{jk\mathbf{r}_n\cdot\hat{\mathbf{r}}} \quad \text{for } n = 1, ..., 8. \tag{3.3}$$

This will then be part of the input to the model. Using the notation from equation (3.1) and (3.2), then for a given row m and polarization pol we can rewrite equation (3.3) as

$$\mathbf{G}^{(m,n,pol)}(\theta,\phi)e^{jk\mathbf{r}_n\cdot\hat{\mathbf{r}}} \quad \text{for } n = 1, ..., 8. \tag{3.4}$$

The antenna arrays are constructed such that the phase difference between the far fields of the subarrays are minimized, assuming that they are excited with the same phase and have phase reference points in the center of the respective subarray. Using this we can approximate the far field functions in equation (3.4) using

$$\mathbf{G}^{(m,n,pol)}(\theta,\phi) \approx \left|\mathbf{G}^{(m,n,pol)}(\theta,\phi)\right| = \sqrt{\left|G_{co}^{(m,n,pol)}(\theta,\phi)\right|^2 + \left|G_{xp}^{(m,n,pol)}(\theta,\phi)\right|^2}.$$

The final input to the model containing the information of the antenna will be a matrix containing eight columns where column n contains

$$\left|\mathbf{G}^{(m,n,pol)}(\theta,\phi)\right| e^{jk\mathbf{r}_n\cdot\hat{\mathbf{r}}} \quad \text{for } \theta = 90°,\ \phi = -180°, ..., 180° \tag{3.5}$$

where m and pol are given. This matrix will be referred to as the array factor kernel or AFK.

The reference curves used are Gaussian curves capped at -25 dB with different HPBW. An example of such a reference with a 65° HPBW is shown in Figure 3.3. Each of the 1764 antenna data points AFK were combined with 46 reference curves with HPBW $= 20°, 25°, ..., 65°$ creating the data used as input for the model. In total it resulted in 81144 data points.

In the end the dataset was split into three subsets. The first set was called the test set and contained data from 5 randomly chosen antennas with a total of 248 data points. This data was not used under any of the modeling stages, it was only used to give an unbiased evaluation of the final model. The rest of the data was split into two subsets, a test and a validation set using an 80/20 split. The training dataset was used for training the model and the validation dataset was used to give an unbiased evaluation of the trained model while determining the hyperparameters of the model.

3.2 Model and Output

The artificial neural network was developed using the package PyTorch version 2.2.1 [32] in Python version 3.12.2 [33] due to the flexible framework for developing neural

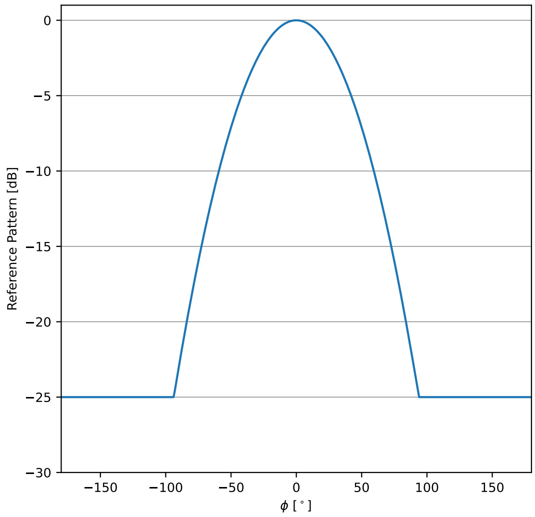


Figure 3.3: A peak normalized reference curve with 65° HPBW.

networks. To adapt PyTorch to the project, the functions was rewritten to work for complex valued input. The convolutional layer, max pooling layer, ReLU and hyperbolic tangent were implemented as described in section 2.4.7 [29]. The CV-CNN was developed to perform well while keeping the number of trainable parameters as low as possible. To find the optimal hyperparameters of the model Ray Tune was used, which will be described in section 3.5.

The CV-CNN takes the AFK and a reference curve as input, where each column in the AFK is considered as a one dimensional layer with 361 points. The last input layer is the reference curve which also is one dimensional with 361 points. An antenna with 8 subarrays in azimuth therefore has 9 input layers. The input layers is connected with a hidden convolutional layer. This is a one dimensional complex valued layer. A one dimensional layer is analogous to the normal two dimensional layer but the kernel and stride are scalars instead of vectors. The activation function is $\mathbb{C}$ReLU which is connected to a complex valued one dimensional max-pooling layer. There are a total of three convolutional layers in the CV-CNN, all with the same structure but with different hyperparameters.

The output from the last convolutional layer is connected to a fully connected feed forward neural network with one hidden layer. The hidden layer uses $\mathbb{C}$ReLU as activation function as the convolutional layers, the output layer however uses the complex valued hyperbolic tangent in order to correctly scale the final output.

The desired final output of the CV-CNN are the beamweights

$$A_n e^{j\Phi_n}, \quad \text{for } n = 1, ..., 2N$$

from equation (2.6) where N is the number of subarrays in the linear antenna array. The size of the final output is then two times the number of subarrays which corresponds to beamweights for each polarization of each subarray. This is desired since it can be used to calculate the far field of the antenna array using the DPBF technique. These beamweights will also be limited such that $A_n \in [-4, 4]$dB and $\Phi_n \in [-\pi, \pi]$. The reason for this is that the phase is periodic so it has full freedom even if limited to $[-\pi, \pi]$ and the amplitude is limited in the same manner in an implementation of a stochastic optimization algorithm currently used at Ericsson. Further, by using the same interval it is easier to compare the two methods.

The CNN uses complex values and the output will also be complex valued. In order to achieve the limits discussed above it is then easier to define the output of the network as

$$\ln(A_n) + j\Phi_n, \quad \text{for } n = 1, ..., 2N \tag{3.6}$$

and then use the activation function

$$g(z) = \tanh(\mathrm{Re}(z)) \cdot \ln(10^{4/20}) + j \cdot \tanh(\mathrm{Im}(z)) \cdot \pi \tag{3.7}$$

for the output layer. The final beamweights are then found using

$$e^{\ln(A_n)+j\Phi_n} = A_n e^{j\Phi_n}, \quad \text{for } n = 1, ..., 2N. \tag{3.8}$$

3.3 Loss Function

The purpose of the loss function is to calculate how far off the resulting far field of an antenna array is from a given reference curve. Thus far, the reference curve is given, the AFK is calculated as in section 3.1 and the beamweights are outputted from the model as in section 3.2. The first thing to do in the loss function is then to calculate the far field using equation (2.6). Note here however that the AFK only have N columns but the model outputs $2N$ beamweights. This is solved using the assumption that the polarizations are identical, the AFK can then be used for both polarizations.

The second step is to measure how well the resulting far field corresponds to the reference curve. This is done using two measures which in this thesis will be called far field difference and sidelobe loss. The final step is to measure how much of the maximum power output of the antenna is lost while forming the beams. This is done by calculating the so called taper loss. The output of the loss function is finally the sum of these three losses. All three losses have a minimum value of zero and the higher the loss the worse the model has performed. The measures will be defined in detail in the following sections.

3.3.1 Taper Loss

Taper loss is a measure of how the beamweight amplitudes varies and it indicates how much power is lost due to having non-uniform excitation amplitudes. The desire is to maximize the power output of the antenna. This is then done by exciting all elements with uniform amplitudes. In practice, the amplitude vector is normalized before the practical implementation so only the relative amplitude levels are of importance. Taper loss is defined as

$$\mathcal{L}_T = -10 \cdot \log_{10}\left(\sum_{n=1}^{2N}\left(\frac{|A_n|}{\max\limits_n |A_n|}\right)^2 /2N\right),$$

where A_n is the amplitude of the beamweights and $2N$ is the number of outputs from the model.

3.3.2 Far Field Difference

The far field difference is a measure on how well the main lobe of the resulting far field corresponds to the reference curve. This is done by calculating the mean squared error (MSE) between the reference curve and the resulting far field where the reference curve is above -10 dB. If the reference curve at ϕ is denoted as $R(\theta)$. Then the set of angles where the reference curve is above -10 dB can be defined as

$$\mathcal{F} := \left\{\phi : \left|\frac{R(\phi)}{R_{ref}}\right|_{\text{dB}} \geq -10,\ \phi = -180°, -179°, ..., 180°\right\},$$

where

$$R_{ref} = \max_{\phi} |R(\phi)|,$$

and if the resulting far field of the antenna array at $\theta = 90°$ and ϕ is denoted $\mathbf{G}_A(\phi)$. Then the far field difference is defined as

$$\mathcal{L}_F = \operatorname*{MSE}_{\phi\in\mathcal{F}}\left(\left|\frac{\mathbf{G}_A(\phi)}{G_{ref}}\right|_{\text{dB}}, \left|\frac{R(\phi)}{R_{ref}}\right|_{\text{dB}}\right) = \frac{1}{|\mathcal{F}|}\sum_{\phi\in\mathcal{F}}\left(\left|\frac{\mathbf{G}_A(\phi)}{G_{ref}}\right|_{\text{dB}} - \left|\frac{R(\phi)}{R_{ref}}\right|_{\text{dB}}\right)^2,$$

with

$$G_{ref} = \max_{\phi} |\mathbf{G}_A(\phi)|.$$

3.3.3 Sidelobe Loss

To minimize interference with other antennas the sidelobes should be suppressed, a threshold of -10 dB was set to ensure this. The threshold was enforced at 10° outside of the main lobe at -10 dB beam width and over the entire plane forward from the antenna, i.e. from $-90°$ to $90°$. The loss was measured as the maximum value of the sidelobes above the threshold in these areas, if no sidelobe were above the threshold the value was set to 0.

This can be defined in terms of the reference curve, $R(\phi)$, and the far field, $\mathbf{G}_A(\phi)$. First the angle $\phi_{-10\text{dB}} > 0$ is defined such that

$$\left|\frac{R(\phi_{-10\text{dB}})}{R_{ref}}\right|_{\text{dB}} = -10.$$

The set of angles where the threshold was enforced is then defined as

$$\mathcal{F}_{side} = \left\{-90°, -89°, ..., -\phi_{-10\text{dB}} - 10°, \phi_{-10\text{dB}} + 10°, ..., 89°, 90°\right\}.$$

Using this, the maximum sidelobe level can then be defined as

$$G_{side} := \max_{\phi \in \mathcal{F}_{side}} \left|\frac{\mathbf{G}_A(\phi)}{G_{ref}}\right|_{\text{dB}},$$

and the sidelobe loss is then defined as

$$\mathcal{L}_S = \max\left(G_{side} + 10, 0\right).$$

3.3.4 Total loss

The final loss function optimized by the network, called the total loss is defined as the sum of the taper loss, the far field difference and the sidelobe loss. Using the notation from section 2.4.6.1 and above, the loss function in this case is defined as

$$\mathcal{L} = \mathcal{L}_T + \mathcal{L}_F + \mathcal{L}_S.$$

3.4 Training

To train the model, Adam optimizer [27] was used together with Autograd which is a way of automatically differentiate functions in PyTorch. Autograd uses Wirtinger calculus as described in section 2.2.1 [19]. The model was trained on the training dataset for 100 epochs as the loss had decreased enough after that many iterations and more computational resources was deemed unnecessary.

3.5 Hyperparameter Tuning

The structure of the CV-CNN was decided by initial trials. However, the hyperparameters was optimized using the package Ray Tune [34] to find a set of hyperparameters that minimized the loss function evaluated on the validation set. Ray Tune samples instances of hyperparameters from a predetermined search space, trains the model using these parameters on a training set and then evaluates them over a validation set using a so called scheduler in order to find the best performing one [34]. In order to reduce computational waste, the Asynchronous Successive Halving Algorithm (ASHA) Scheduler [35] was used to stop samples early if they performed worse than others.

There were hyperparameters to optimize in both Adam and the CV-CNN. The hyperparameters to optimize in Adam was the learning rate and β-values. For the

CV-CNN the hyperparameters were the kernel sizes, stride and number of output channels for the three convolutional layers, the kernel sizes and stride for the pooling layers and the number of hidden neurons in the fully connected layer. The search spaces for each parameter can be found in Table 3.1. For the learning rate the search space was set between the two closest exponents of ten from the standard value 10^{-3}. Also the two β-values search spaces was set to values around the standard values at $\beta_1 = 0.9$ and $\beta_2 = 0.999$. The other search spaces were set to values which had shown promise during initial testing to perform well on and then expanded out to both higher and lower values to extend the search space.

The model was trained for a maximum of 50 epochs per sample in one Ray Tune run and 100 epochs in another. Each run contained 300 samples. All samples were given two epochs to learn before ASHA Scheduler was allowed to stop it, then the ASHA Scheduler had the opportunity to break at epoch $2, 4, 8, 16, 32, 64$.

Table 3.1: The hyperparameter search space for Ray Tune: The table shows the search space used when optimizing the hyperparameters using Ray Tune. Most of the parameters had sets of values while the learning rate and β-values was sampled uniformly over a continuous interval.

Hyperparameter	**Search space**
Learning Rate	$[0.01, 0.0001]$
β_1	$[0.7, 95]$
β_2	$[0.9, 999]$
Convolutional kernel size layer 1	$\{5, 10, 15, 20\}$
Convolutional kernel size layer 2	$\{5, 10, 15, 20\}$
Convolutional kernel size layer 3	$\{5, 10, 15, 20\}$
Convolutional stride layer 1	$\{1, 2, 3\}$
Convolutional stride layer 2	$\{1, 2, 3\}$
Convolutional stride layer 3	$\{1, 2, 3\}$
Output channels layer 1	$\{2, 4, 6, 8\}$
Output channels layer 2	$\{2, 4, 6, 8\}$
Output channels layer 3	$\{2, 4, 6, 8\}$
Pooling kernel size layer 1	$\{2, 3, 4\}$
Pooling kernel size layer 2	$\{2, 3, 4\}$
Pooling kernel size layer 3	$\{2, 3, 4\}$
Pooling stride layer 1	$\{1, 2, 3, 4\}$
Pooling stride layer 2	$\{1, 2, 3, 4\}$
Pooling stride layer 3	$\{1, 2, 3, 4\}$
Number of hidden neurons	$\{8, 16, 32, 64\}$

3.6 Tests

In order to test how the CV-CNN performs, various tests were performed. These included a comparison with an existing method and statistical tests for certainty and stability. Finally the model was tested how it scaled to other sizes of antenna array. All tests were done with the 248 data points in the test set which were never used in any training.

3.6.1 Performance on Test Set

To evaluate the performance of the CV-CNN it was tested on each degree HPBW reference curve from 20° to 65°. Then a 95% confidence interval was calculated for each loss and the total loss to evaluate the uncertainty of resulting losses for each reference width.

3.6.2 Comparison to Stochastic Optimization

In order to compare the result with an existing method the testing data points were optimized with one of the methods used by Ericsson, a stochastic optimizer called GODLIKE [36]. GODLIKE is a single- and multiobjective optimizer implemented in MATLAB. It uses a combination of the NSGA-II algorithm together with other algorithms to generate the population at each iteration. These algorithms include GA, DE and SA [36]. It is used as a multiobjective optimizer, optimizing far field difference, taper loss and sidelobe loss simultaneously, creating a Pareto front. This allows the user to pick out the solution best suited to the problem. In this report the solution in the Pareto front closest to the origin was used as a comparison to the solution found by the CV-CNN.

Beamweights were calculated using both GODLIKE and CV-CNN over the entire test set with reference curves with 20° and 65° HPBW. The loss was then calculated and compared for the resulting far fields using beamweights from both models. The hyperparameters used in this thesis for GODLIKE were equal for all runs and can be found in Table A.1 in Appendix A

3.6.3 Scalability

To test the scalability of the model, the same model structure optimized for an antenna array with 8 subarrays in azimuth was trained on antenna arrays with $4, 16, 32$ and 64 subarrays in azimuth as well. These were simulated from the real data of antenna arrays with 8 subarrays. For 4 subarrays the two outermost and the two center subarrays were considered as one antenna array with 4 subarrays, as the middle antenna array in Figure 3.4. For the antenna array with 16 subarrays in azimuth the first four subarrays from the measured 8 subarray was considered the first four in the larger array. The following four subarrays were copies of the fourth subarray and the ninth to thirteenth subarrays were copies of the fifth subarray. The last four subarrays were the last in the measured antenna array as well, as the bottom antenna array in Figure 3.4. For the larger antenna arrays the simulation was analogous, but with more copies of the two center subarrays. All models were then evaluated on the test set after it was adapted as above to include the respective number of subarrays.

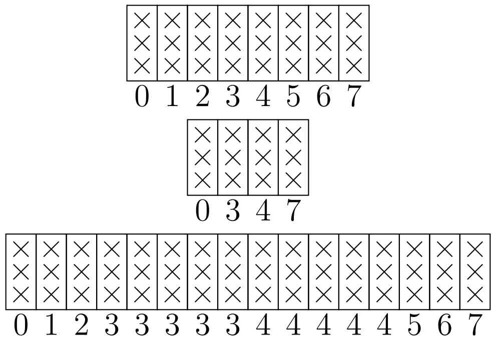


Figure 3.4: Simulating different antenna array sizes: The figure shows the original measured linear antenna array on top, followed by the simulated antenna arrays with 4 and 16 subarrays.

3.6.4 Input and Output Stability

A desirable system or model should be stable, meaning that small measurement errors in the input or small hardware errors affecting the output should not have detrimental effects in the performance. To test how stable the system is, noise was introduced to both the input and the output. The results from this also gives an indication to what requirements the measurements and hardware should have in order to be compatible with the model.

To test how the model reacted to measurement errors in the input, a noise was added to the AFK. This was done by multiplying each column, n, by a random noise factor

$$A_{n,noise}e^{j\Phi_{n,noise}},$$

where $A_{n,noise}$ was drawn from a uniform distribution over the interval $[1-\Delta A, 1+\Delta A]$ and $\Phi_{n,noise}$ was drawn from a uniform distribution over the interval $[-\Delta\Phi, \Delta\Phi]$. The intervals was chosen such that

$$\begin{aligned}\Delta A &\in \{0, 0.005, 0.01, ..., 0.1\}\,,\\ \Delta\Phi &\in \{0°, 0.5°, 1°, ..., 10°\}\,,\end{aligned}$$

and for each combination of ΔA and $\Delta\Phi$, 100 samples of the noise factors were drawn and multiplied with the AFK. The AFK with noise was then used in the input of the model and the beamweights obtained were compared to the beamweights obtained when inputting the AFK without noise. The beamweights from the noisy AFK were then used together with the noisy AFK to generate a far field and the loss of that far field was noted. This entire process was repeated for all data points in the test set with reference curves with a HPBW of either 20°, 40° or 65°.

To test how the system reacted to noise in the output a similar approach was utilized. This time the AFK was left untouched but the beamweights were multiplied by 100 random noise factors in the same intervals as above. These noisy beamweights were then used together with the AFK to generate a far field and the loss of that far field was noted. This process was also repeated for all data points in the test set with reference curves with a HPBW of either 20°, 40° or 65°.

4
Results

In this chapter the results from the thesis will be presented, beginning with the hyperparameter tuning. There will be a comparison to stochastic optimization in computational cost and performance. Following that the results of the CV-CNN on the test set and the results of scaling the model to 4, 16, 32 and 64 subarrays. Lastly, the stability results when applying noise to both the input and output of the model is presented.

4.1 Hyperparameter Tuning

The lines in Figure 4.1 illustrates how the validation loss decreases with the number of epochs for 50 and 100 epochs respectively. For both cases the first 50 epochs had similar loss and decreased with similar rate. In both cases the trend of the validation loss is decreasing which indicates that neither of the models overfit the data. The model for 50 epochs has larger fluctuations of the loss between epochs due to larger learning rate.

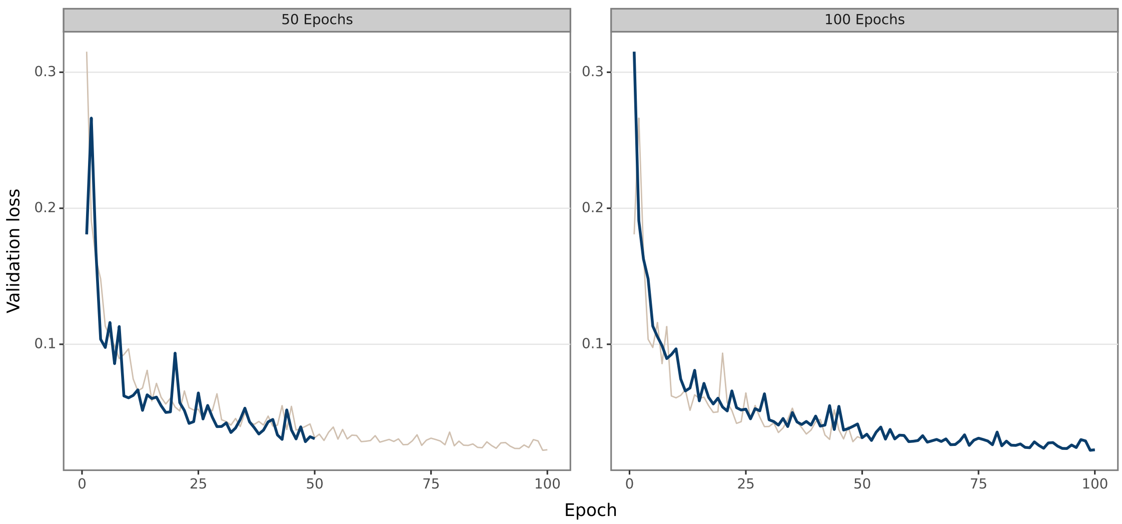


Figure 4.1: Validation loss per epoch for the best models: The figure shows the total validation loss development over epochs. The left figure highlights the loss for the best model trained over 50 epochs with a subtle line for the total validation loss for the model trained over 100 epochs. The right figure shows the same thing with the loss for the model trained over 100 epochs highlighted.

The hyperparameters which results in the lowest loss of the validation data when trained over 50 and 100 epochs respectively can be found in Table 4.1. Most hyperparameters differ between the number of epochs.

Table 4.1: Optimal hyperparameters: The table contains the optimal hyperparameters found using Ray Tune [34] when training a CV-CNN model for 50 and 100 epochs over 300 samples each.

Hyperparameter	**Best Parameters 50 Epochs**	**Best Parameters 100 Epochs**
Learning Rate	0.000715	0.000196
β_1	0.858205	0.797510
β_2	0.914448	0.999000
Convolutional kernel size layer 1	10	5
Convolutional kernel size layer 2	10	5
Convolutional kernel size layer 3	5	15
Convolutional stride layer 1	1	1
Convolutional stride layer 2	2	1
Convolutional stride layer 3	1	1
Output channels layer 1	8	6
Output channels layer 2	8	2
Output channels layer 3	6	8
Pooling kernel size layer 1	3	2
Pooling kernel size layer 2	2	2
Pooling kernel size layer 3	2	3
Pooling stride layer 1	1	1
Pooling stride layer 2	3	1
Pooling stride layer 3	2	3
Number of hidden neurons	64	64

Since the model trained for 100 epochs achieved the lowest loss on the validation set, this model will be chosen as the final model used in the remaining of the tests. Therefore, all other results in this chapter comes from this model.

4.2 Comparison to Stochastic Optimization

This section contains a comparison of results between the CV-CNN and the stochastic optimizer GODLIKE. How the antennas far field difference is distributed for 20° and 65° HPBW of reference curve and for each model is shown in Figure 4.2. A trend over all antennas is that the narrower reference curve has less far field difference than the wider one. For all antennas, except the last, the two models perform similarly for the narrower reference. On Antenna 5 however GODLIKE performs better for both reference curves. For the wider reference curve the variation of far field difference is larger compared to the narrower, for both models.

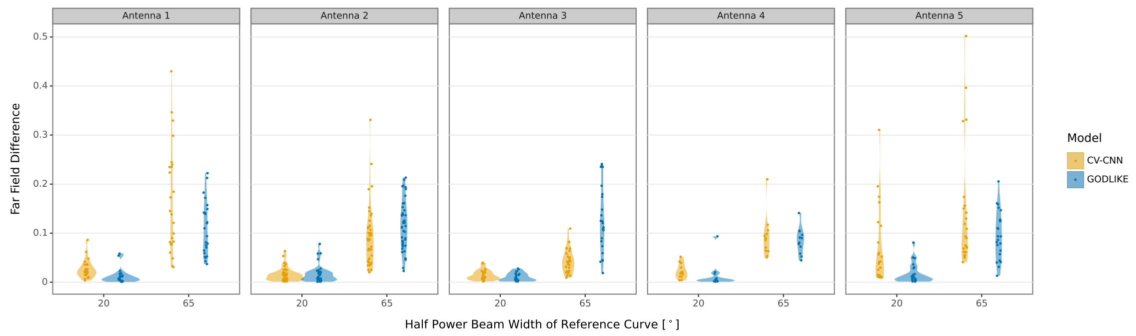


Figure 4.2: Comparison between CV-CNN and GODLIKE in far field difference: The figure shows violin plots containing the far field difference between a reference curve with either 20° or 65° HPBW and the far field from each measured antenna array in the test set. The beamweights used to generate the far field was obtained using a CV-CNN and GODLIKE. The result was split by antenna, HPBW of reference curve and model used to get the beamweights.

The structure of Figure 4.3 is similar to the structure in Figure 4.2 but with taper loss instead of far field difference. The CV-CNN shows better taper loss than GODLIKE for all test antennas. Note that this figure is on logarithmic scale.

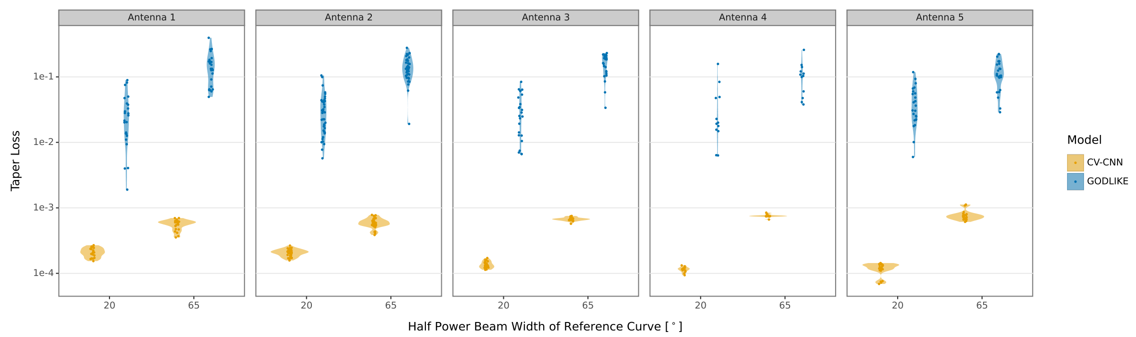


Figure 4.3: Comparison between CV-CNN and GODLIKE in taper loss: The figure shows violin plots containing the taper loss from beamweights optimized for the antennas in the test set against two different reference curves with 20° and 65° HPBW. The beamweights was obtained using a CV-CNN and GODLIKE. The result was split by antenna, HPBW of reference curve and model used to get the beamweights. Note that the taper loss is presented in logarithmic scale.

The violin plots in Figure 4.4 displays the total loss for each test antenna, model and reference curve. The total loss is the sum of the three loss measures, far field difference, taper loss and sidelobe loss. The sidelobe loss was zero for all beamweights from all data points obtained from the CV-CNN. The sidelobe loss was zero for all beamweights from all data points obtained from GODLIKE except for four data points, these sidelobe losses can be found in Table 4.2. From Figure 4.4 it is clear that for most antenna and reference combinations the CV-CNN is less varying than GODLIKE, with the exception of Antenna 5. The loss for the CV-CNN in Figure 4.4 is almost equal to the far field difference from Figure 4.2. The loss for GODLIKE in Figure 4.4 is more influenced by the taper loss in Figure 4.3. This is the reason for the smaller variations in loss for the CV-CNN.

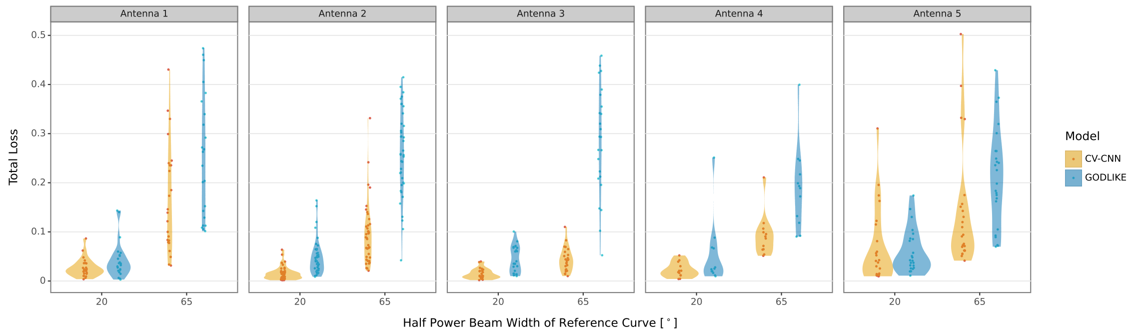


Figure 4.4: Comparison between CV-CNN and GODLIKE in total loss: The figure shows violin plots containing the total loss from beamweights optimized for the antennas in the test set against two different reference curves with 20° and 65° HPBW. The beamweights was obtained using a CV-CNN and GODLIKE. The result was split by antenna, HPBW of reference curve and model used to get the beamweights.

The subarray separation in azimuth, dY, is measured in wavelengths, λ, in order to combine frequency and antenna design to one measure. The antennas selected for the test set have subarray separation $dY \in [0.454, 0.587]\lambda$. To compare results between the models for different antenna characteristics the test antennas was split in quartiles of subarray separation. Figure 4.5 show how the two models far field difference varies for the different subarray separation. This is the same figure as Figure 4.2 but split by subarray separation instead of antenna. The figure shows how the CV-CNN fits the far field better for the second and third quartile than the first and fourth. GODLIKE show less variation for different subarray separations.

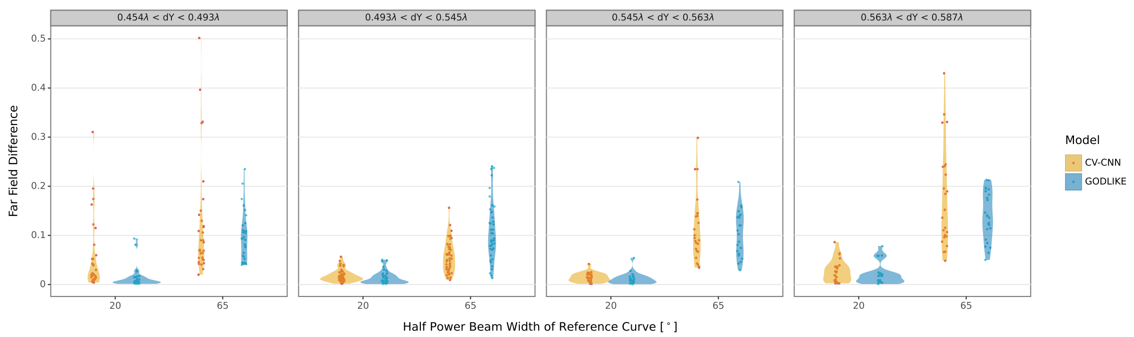


Figure 4.5: Comparison between CV-CNN and GODLIKE in far field difference for each quartile of subarray separation: The figure shows violin plots containing the far field difference between a reference curve with either 20° or 65° HPBW and the far field from each measured antenna array in the test set. The beamweights used to generate the far field was obtained using a CV-CNN and GODLIKE. The result was split by quartile of subarray separation, HPBW of reference curve and model used to get the beamweights.

The violin plots in Figure 4.6 show the taper loss for each quartile, reference width and model. As this is the same data as in Figure 4.3 the CV-CNN has lower taper loss for all antennas and the figure is in logarithmic scale. The taper loss does not follow the same tendency for CV-CNN with higher losses for the first and last quartile as the far field difference from the CV-CNN in Figure 4.5.

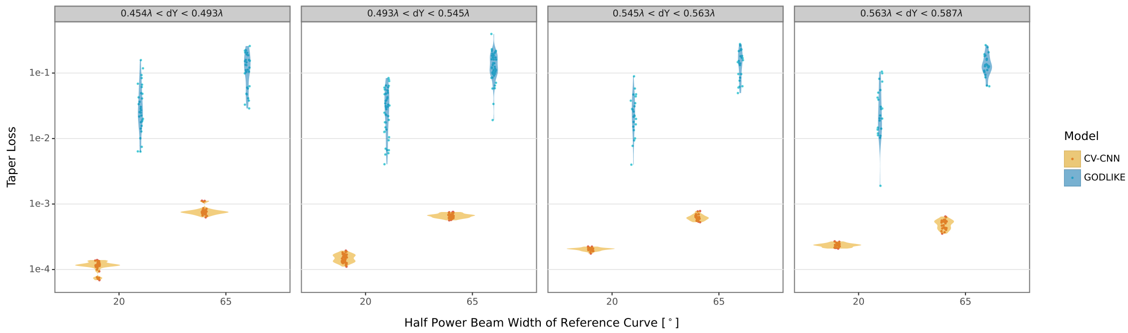


Figure 4.6: Comparison between CV-CNN and GODLIKE in taper loss for each quartile of subarray separation: The figure shows violin plots containing the taper loss from beamweights optimized for the antennas in the test set against two different reference curves with 20° and 65° HPBW. The beamweights was obtained using a CV-CNN and GODLIKE. The result was split by quartile of subarray separation, HPBW of reference curve and model used to get the beamweights. Note that the taper loss is presented in logarithmic scale.

The total loss for quartiles of subarray separation can be found in Figure 4.7. In the figure, the total loss is lower for the CV-CNN than GODLIKE for the second and third quartile, for both reference curves. it can also be seen in the figure that the highest density of distribution for all subarray separations and reference curves is lower for the CV-CNN than GODLIKE, but for the smallest subarray separations there are outliers which are larger than all losses from GODLIKE.

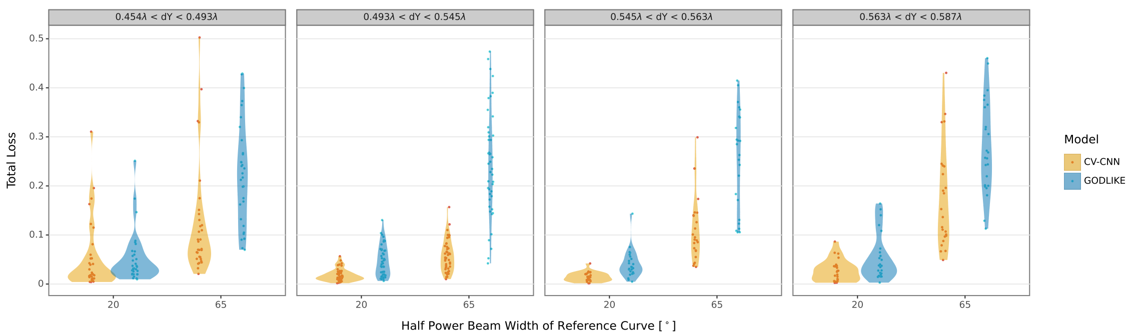


Figure 4.7: Comparison between CV-CNN and GODLIKE in total loss for each quartile of subarray separation: The figure shows violin plots containing the total loss from beamweights optimized for the antennas in the test set against two different reference curves with 20° and 65° HPBW. The beamweights was obtained using a CV-CNN and GODLIKE. The result was split by quartile of subarray separation, HPBW of reference curve and model used to get the beamweights.

Table 4.2: High sidelobes: The table shows all occurrences where the solution found by GODLIKE had a maximum sidelobe level above -10 dB.

Antenna	HPBW of Reference	dY [λ]	Sidelobe loss [dB]
2	20	0.56300	0.001293
2	20	0.58658	0.001629
4	20	0.49000	0.000489
5	20	0.46823	0.002728

4.2.1 Time Comparison

An important aspect of the models is their time consumption. To test this, first the number of function evaluations of the loss function was recorded when training the CV-CNN for 100 epochs on the training set. Then the number of function evaluations performed in an interval of typical runs of GODLIKE was recorded. The results can be found in Table 4.3.

Table 4.3: Number of function evaluations per model: The table shows how many function evaluations each model uses, for the CV-CNN the number is for 100 epochs of training and for GODLIKE it is an interval when optimizing one antenna.

Model	Period	Function evaluations
CV-CNN	100 epochs of training	5578900
GODLIKE	One optimization	$[5000000, 8000000]$

4.3 Performance of CV-CNN

In this section the results of the performance of the CV-CNN on a range of HPBW of the reference curve is presented. The sidelobe loss never exceeded zero and is therefore omitted. Figure 4.8 show how the mean far field difference loss varies over HPBW of reference curve for each antenna. Inn the figure the shaded area shows the 95% confidence interval (CI) of the result. From the figure it is clear that all antennas result are better for narrower reference curves and degrades for wider reference curves.

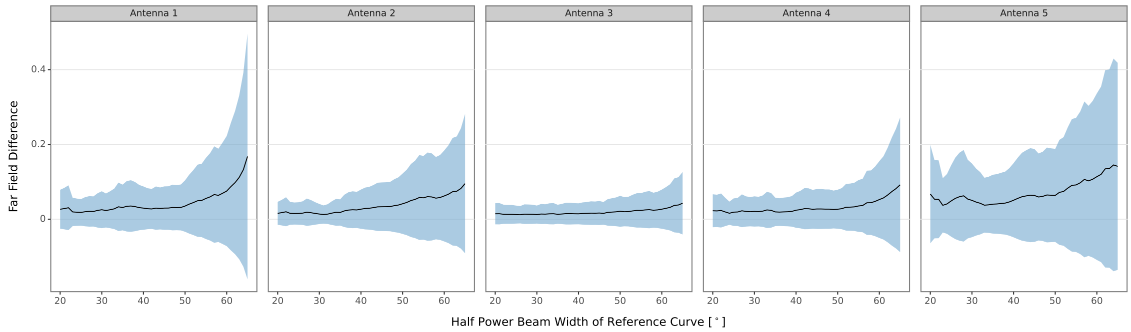


Figure 4.8: Far field difference per HPBW for each test antenna: The figure shows how well the far field calculated by the output of the CV-CNN model fits the reference curve in far field difference. The results are split by antenna. As each antenna has multiple measurements the mean is presented together with a 95% confidence interval.

The results in Figure 4.9 show how the mean taper loss varies over increasing HPBW of reference curve together with a 95% CI for each antenna. From the figure it can be seen that for all antennas the taper loss is largest around 40° HPBW reference curve and lower at the ends, 20° and 65° HPBW reference curve. The magnitude of the taper loss in Figure 4.9 is at the order of 10^{-3}, in comparison with the far field difference at 10^{-1} from Figure 4.8.

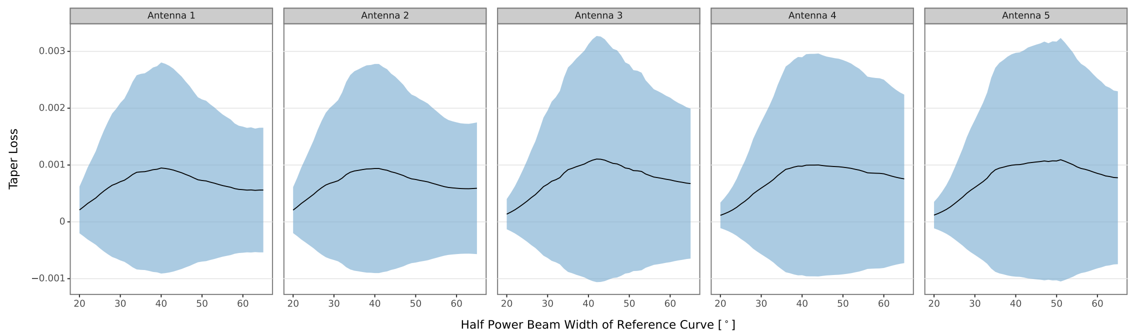


Figure 4.9: Taper loss per HPBW for each test antenna: The figure shows the taper loss from beamweights outputted from the CV-CNN for each HPBW of reference curves. The results are split by antenna. As each antenna has multiple measurements the mean is presented together with a 95% confidence interval.

The total loss per HPBW of reference curve for each antenna together with a 95% CI is shown in Figure 4.10. This figure is similar to Figure 4.8 as the taper loss is small and the sidelobe loss is zero.

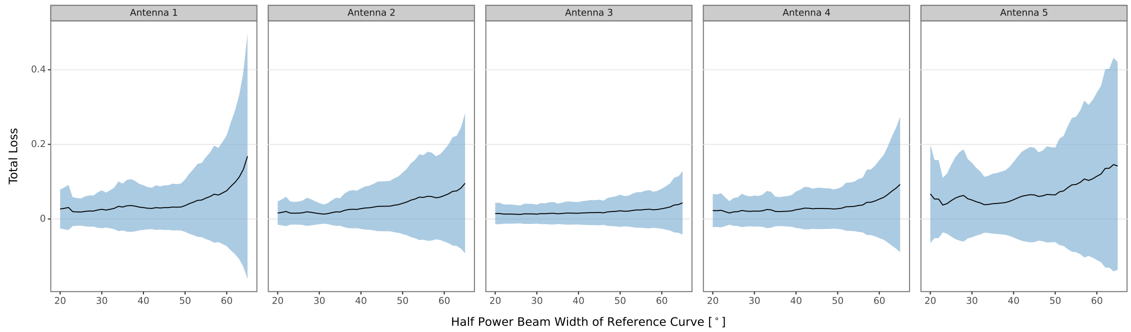


Figure 4.10: Total loss per HPBW for each test antenna: The figure shows the total loss from beamweights outputted from the CV-CNN for each HPBW of reference curves. The results are split by antenna. As each antenna has multiple measurements the mean is presented together with a 95% confidence interval.

How the far field difference varies over different HPBW of reference curve is illustrated in Figure 4.11. The figure is split in quartiles of subarray separation. As in Figure 4.8, all cases has increasing far field difference with increasing HPBW of reference curve. The second quartile in the figure show a lower loss and a narrower CI while the first and forth quartile show higher losses and wider CIs.

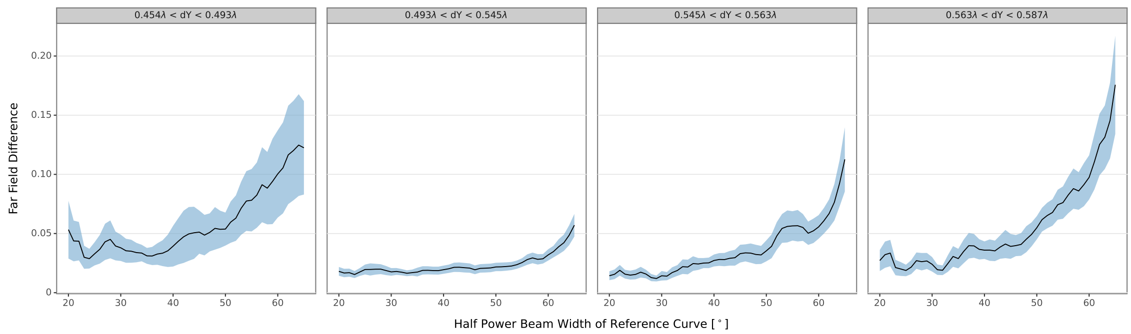


Figure 4.11: Far field difference per HPBW for each quartile of subarray separation from the test antennas: The figure shows how well the far field calculated by the output of the CV-CNN model fits the reference curve in far field difference. The results are split by quartile of subarray separation. As each quartile contains multiple measurements the mean is presented together with a 95% confidence interval.

How the taper loss is related to the width of the reference curve is shown in Figure 4.12. It is also separated by quartiles as Figure 4.11. It has the same tendency as Figure 4.9 with a peak of taper loss in the middle but with narrower CI.

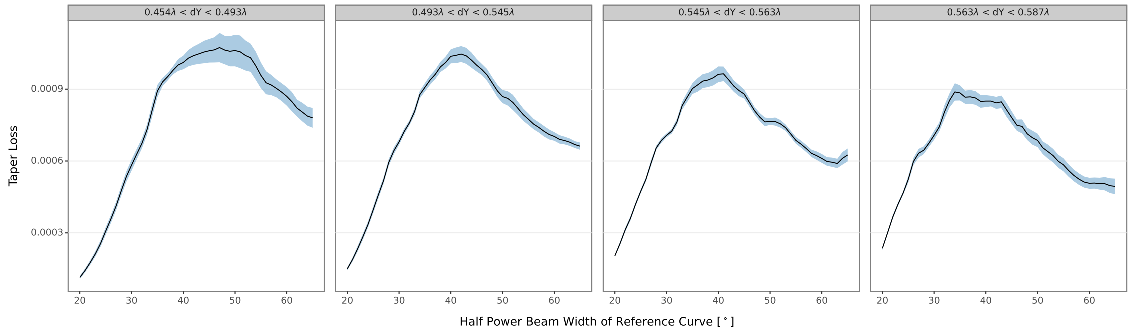


Figure 4.12: Taper loss per HPBW for each quartile of subarray separation from the test antennas: The figure shows the taper loss from beamweights outputted from the CV-CNN for each HPBW of reference curves. The results are split by quartile of subarray separation. As each quartile contains multiple measurements the mean is presented together with a 95% confidence interval.

The total loss when splitting the data into quartiles of subarray separation is shown in Figure 4.13. As in Figure 4.10 the total loss in Figure 4.13 is very similar to the far field difference as the taper loss and sidelobe loss is close to or equal to zero.

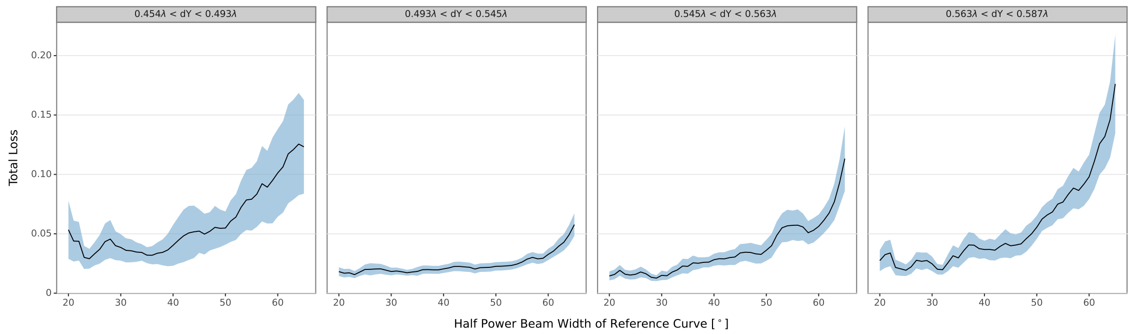


Figure 4.13: Total loss per HPBW for each quartile of subarray separation from the test antennas: The figure shows the total loss from beamweights outputted from the CV-CNN for each HPBW of reference curves. The results are split by quartile of subarray separation. As each quartile contains multiple measurements the mean is presented together with a 95% confidence interval.

4.3.1 Differently Sized Antenna Arrays

The losses for the differently sizes antennas is presented in Figure 4.14. In the figure, the mean total loss for each size and for all test antennas for each degree of HPBW of reference curve together with the 95% CI. From the figure it can be seen that for the smallest antenna the model gets issues for the narrower reference curves. Both the mean loss and the uncertainty is large for the largest antenna. In Figure 4.14, it is clear that the total loss is significantly higher for 4 and 64 subarrays compared to the others. Further inspection showed that the total loss is dominated by the sidelobe loss for 4 and 64 subarrays.

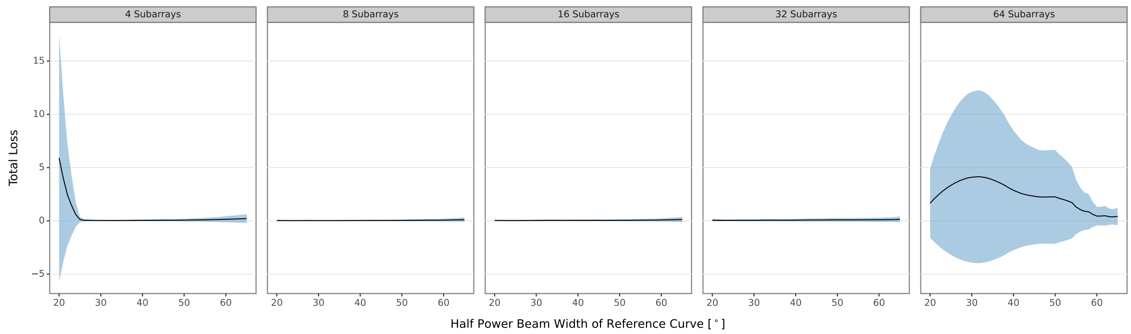


Figure 4.14: Total loss for antennas with 4 to 64 subarrays: The figure shows how the total mean loss changes with increasing HPBW of reference curve together with the 95% CI. The figure is separated by the number of subarrays in the antenna ranging from 4 to 64.

As 4 and 64 subarrays show much larger loss than 8, 16 and 32 subarrays in Figure 4.14, the plots for 8, 16 and 32 subarrays are shown in Figure 4.15. In this figure the mean loss is slightly increasing with increasing number of subarrays and the uncertainty is increasing more rapidly. From the figure it can be seen that a pattern is that all sizes has increasing loss with increasing reference HPBW.

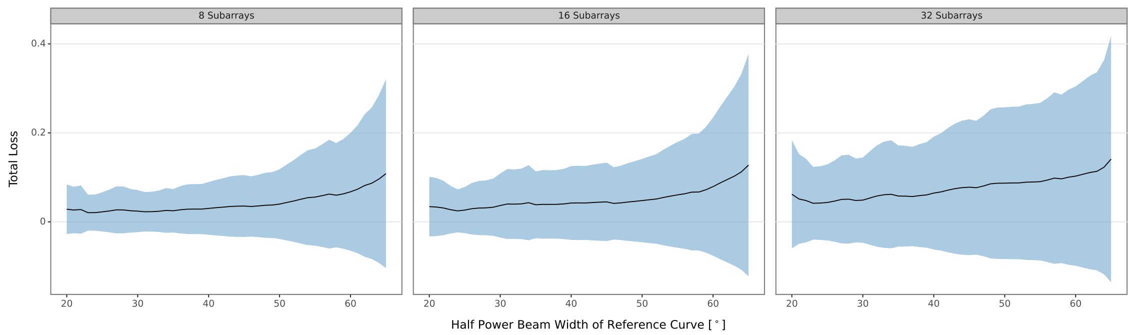


Figure 4.15: Total loss for antennas with 8, 16 and 32 subarrays: The figure shows the average of the total loss with the 95% CI for each degree of HPBW of reference curve. This is the same as Figure 4.14 without 4 and 64 subarrays, note the magnitude of the loss.

4.4 Stability Analysis

The results of the stability investigation is presented in the following figures. First the stability when applying noise to the beamweights outputted of the CV-CNN is presented in Figure 4.16. For the three different reference curves in the figure the loss increases with increased noise, both with noise in phase and amplitude. In the figure, the loss increases faster the wider the reference curve.

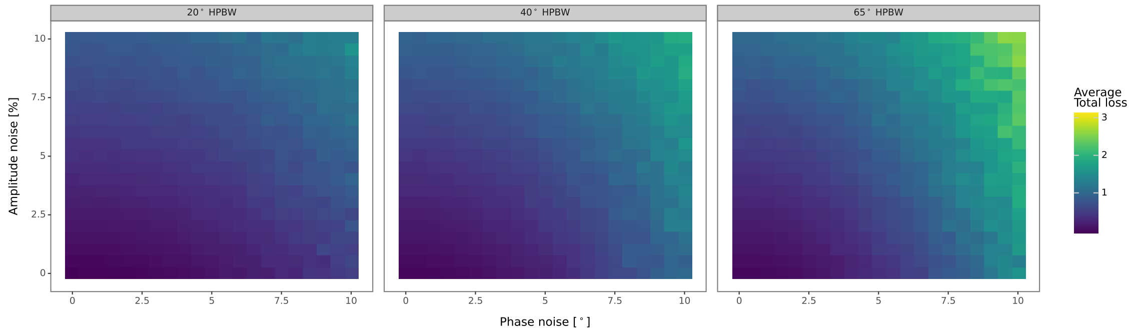


Figure 4.16: Stability of loss with noised beamweights: The figure shows how the loss changes when noise is applied to the beamweights outputted from the model. The loss was calculated as an average over 100 samples of noised beamweights for each data point in the test set for each combination of the phase noise and amplitude noise for 20°, 40° and 65° HPBW reference curves.

In Figure 4.17, 4.18 and 4.19 the results of applying noise to the input to the model is illustrated. Figure 4.17 shows how the total loss is affected by noise in the input. The figure shows that the loss is correlated to the phase noise but not the amplitude noise. The figure also shows that a wider reference curve corresponds to a larger loss.

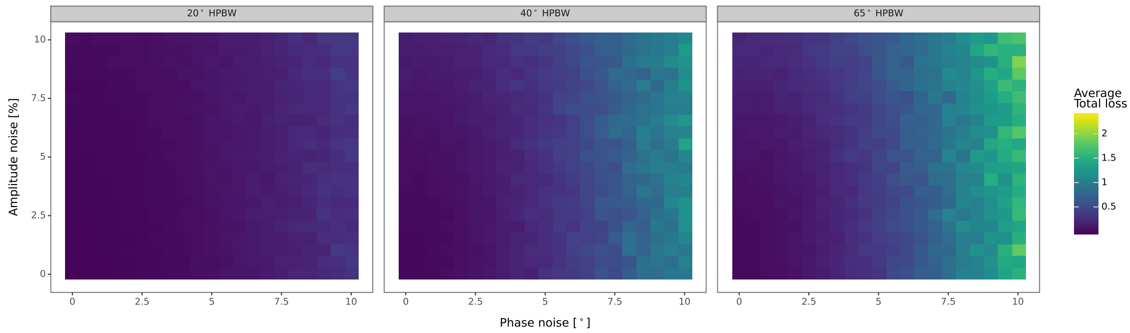


Figure 4.17: Stability of loss with noised AFK: The figure show how the total loss changes when noise is applied to each subarrays radiation when inputted to the model. The loss was calculated as an average over 100 samples of noised AFK for each data point in the test set for each combination of the phase noise and amplitude noise for 20°, 40° and 65° HPBW reference curves.

In Figure 4.18 the difference in amplitude of the beamweights is shown with increasing amount of input noise. The figure shows that the impact is very low for a 20° HPBW reference curve, a bit higher for a 65° HPBW reference curve and significantly higher for a 40° HPBW reference curve.

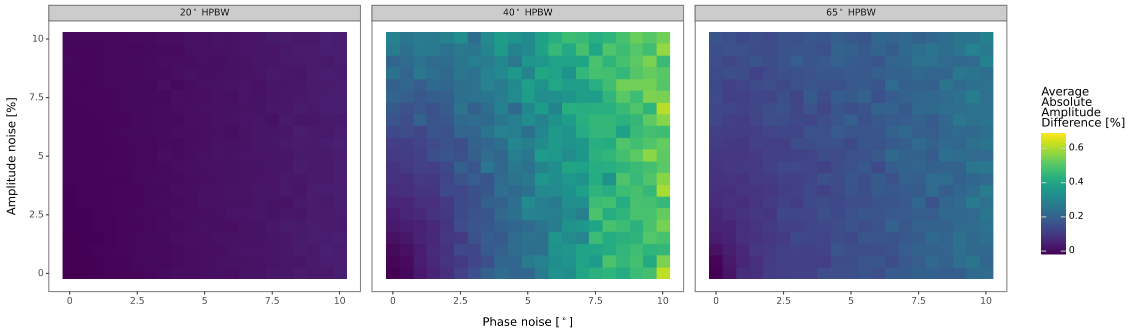


Figure 4.18: Stability of amplitude with noised AFK: The figure show how the amplitude without noise varies when noise is applied to each subarrays radiation when inputted to the model. The loss was calculated as an average over 100 samples of noised AFK for each data point in the test set for each combination of the phase noise and amplitude noise for 20°, 40° and 65° HPBW reference curves.

In Figure 4.19 the results of how the beamweights phase changes with increasing amount of noise is illustrated. The average phase difference for the beamweights is increasing with noise in both amplitude and phase, however quicker with increasing noise in phase. The figure show that the phase difference is highly correlated to the phase noise but not as correlated to the amplitude noise.

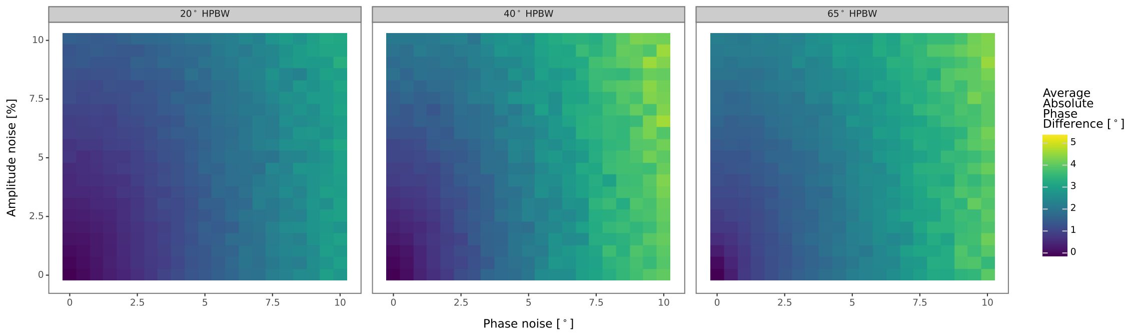


Figure 4.19: Stability of phase with noised AFK: The figure show how the phase varies from the input without noise when noise is applied to each subarrays radiation when inputted to the model. The loss was calculated as an average over 100 samples of noised AFK for each data point in the test set for each combination of the phase noise and amplitude noise for 20°, 40° and 65° HPBW reference curves.

5

Discussion

This chapter aims to discuss the results presented in the previous section. It is divided into four sections which discusses the results from the corresponding sections in the previous chapter.

5.1 Hyperparameter Tuning

The resulting validation loss for the models trained over 50 and 100 epochs shown in Figure 4.1 is very similar for the first 50 epochs. Both models are nearly equally as good but the learning continues with more epochs which is the reason that the 100 epoch model is better in the end. The fact that the loss is still decreasing with the number of epochs also show that the model is not overfitted to the training data.

Most of the hyperparameters between the two models are not equal as can be seen in Table 4.1. The learning rate is one parameter that differ where the model trained over 50 epochs have a larger learning rate than the one for the model trained over 100 epochs. This can be seen in Figure 4.1 as the variation in validation loss between epochs is much larger for the 50 epoch model. Almost all other parameters differ as well. However, the validation loss is very similar, this could mean that there are many different sets of hyperparameters that gives good results, or that there is a set of hyperparameters not found yet that is even better for the problem. The learning rate for the best model is close to the lower end of the search space which shows that the search space could be expanded to include lower values of the learning rate. Furthermore, both models use the maximum amount of hidden neurons allowed in the search space. This indicates that the search space for the number of hidden neurons could be increased to potentially find better solutions. In general, the search space for all hyperparameters could be expanded for a more thorough search which could lead to finding better solutions that are now outside the search space. Another way to extend the search space could be to include the number of hidden layers and the number of convolutional layers. This could potentially increase the likelihood of finding a better model than the one found.

5.2 Comparison to Stochastic Optimization

To compare the results of the far field difference between the CV-CNN and GOD-LIKE the violin plots in Figure 4.2 and Figure 4.5 were studied. In Figure 4.2 the results are split by antenna for two reference curves. It can be seen that for Antenna

1 and 5 the CV-CNN struggles a bit. For the 20° reference curve for Antenna 1 and 5 the loss is higher and the spread is bigger for the CV-CNN than GODLIKE. For 65° the average results are quite alike but the CV-CNN has larger spread in the results. For Antenna 2, 3 and 4 the results are quite equal between the models for both references. The CV-CNN has some bad outliers and some results that are very collected. When looking at the far field difference split by quartile of subarray separation in Figure 4.5 it is clear that the CV-CNN mostly struggles when the subarrays are closer to each other. The first quartile of subarray separation the CV-CNN is varying a lot for both reference curves, the same can be seen for the wider reference curve for the fourth quartile. For the second and third quartile the results between the models are similar and in some cases the CV-CNN performs a bit better. This might be due to having more training data for the CV-CNN with subarray separation in quartile two and three or that the model might be biased towards the mean value in the data set.

The taper loss seem to be completely independent of antenna and subarray separation, for both models. This can clearly be seen in Figure 4.3 and Figure 4.6. This is reasonable as the taper loss is less dependent of antenna design than the far field. The values of the taper loss can be found in Figure 4.3 and Figure 4.6 and it is clear that the CV-CNN outperforms GODLIKE. The taper loss in the figures is consequently 100 times lower for the CV-CNN than GODLIKE, regardless of subarray separation, antenna or reference curve HPBW. The sidelobe levels can be neglected as the sidelobe loss was zero for all antennas optimized using the CV-CNN and only four antennas optimized by GODLIKE had a sidelobe loss over zero. The sidelobe losses above zero are presented in Table 4.2 and it is clear that the sidelobe loss is very low for all four antennas.

The total loss for the two models is compared by antenna in Figure 4.4 and by subarray separation quartiles in Figure 4.7. As previously the CV-CNN has more trouble with Antenna 1 and 5, which is noticeable as a much larger spread between the results. However, apart from some outliers the CV-CNN has lower total loss than GODLIKE for all antennas, subarray separations and HPBWs. This is mostly due to the low taper loss for the CV-CNN. For the second and third quartile of subarray separation in Figure 4.7 the CV-CNN seem to perform much better than GODLIKE for all data. For the fourth quartile the CV-CNN performs marginally better while for the first quartile the average performance is better but there are some antennas performing worse than GODLIKE. However, whether or not the CV-CNN performs better than GODLIKE or not depends on what the goal is. If minimizing the far field difference is of most importance then GODLIKE might be better, and if minimizing the taper loss is of most importance then the CV-CNN might be better. The CV-CNN is also deterministic and will give one set of beamweights while GODLIKE presents a Pareto front of solutions which the user can chose from depending on what is mostly valued in that case.

5.2.1 Time Comparison

The number of function evaluations needed to train the CV-CNN and evaluate GODLIKE are of the same magnitude as can be seen in Table 4.3. This is however not all computations done in either of the models. GODLIKE performs a stochastic amount of computations in order to improve the population and these are directly connected to the number of function evaluations, hence the range in the table. One important aspect is that once the CV-CNN is trained, only one feed through the network is needed in order to get a set of beamweights. This is simply a number of matrix multiplications. No function evaluations is needed unless the loss is of interest.

5.3 Performance of CV-CNN

It is easier for the CV-CNN to fit a narrow reference beam rather than a wide one. This can be seen for all antennas and different subarray separation in Figure 4.10 and Figure 4.13. The uncertainty is also larger for the wider reference beams when they are separated by antenna, as seen in Figure 4.8. From the figure, it can be seen that the far field difference is larger for Antenna 1 and 5 than the others. This is most likely due to a large spread of subarray separation for these antennas as the split over quartile of subarray separation in Figure 4.11 shows that the uncertainty is more constant over all widths of reference beam. This also shows that separating the data by characteristics is more informative about the model than separating by antenna. In Figure 4.11 the mean far field difference and uncertainty is better for the second and third quartile, especially for the second. This shows that the CV-CNN is much better at predicting beamweights for antennas with subarray separation in this interval. This could be because most points in the training data having a subarray separation in this interval or that there is a compromise leading the more extreme values of subarray separation to get worse results.

As can be seen in Figure 4.9, the taper loss is almost negligible as it is below 10^{-2} while the far field difference is around 10^{-1}. The CV-CNN learns taper loss well, however the peak in loss around 40° seen for all antennas and subarray separations lack clear explanation. When comparing the taper loss when split by antenna and quartile of subarray separations in Figure 4.9 and Figure 4.12 respectively it is clear that the uncertainty is much narrower when separated by quartile. This shows that the taper loss is correlated to dY. Again, this is a result showing that splitting data by antenna characteristics is more informative as results of the model than splitting by antenna, as for far field difference. The sidelobe loss is constantly zero over all test antennas and can therefore be seen as an assurance of not getting any unwanted sidelobes over the set limit.

The total loss is almost the same as the far field difference, this is the case for all antennas, subarray separations and HPBW as can be seen in Figure 4.10 and Figure 4.13. The far field difference is therefore by far the most difficult loss-measure to learn for CV-CNN structure.

5.3.1 Differently Sized Antenna Arrays

The results for antennas with different number of subarrays were quite varying as seen in Figure 4.14. For the antenna with 64 subarrays the loss were in general very high. This is probably due to the model being hyperparameter optimized for 8 subarrays and not that a model of this type is unable to handle the larger antenna. Figure 4.14 shows that for four subarrays the model had a hard time with the narrow reference curves, which is expected as the physical limitations for using fewer subarrays restricts the opportunity to generate narrower beams. As the sidelobes are the dominating loss for both 4 and 64 subarrays the CV-CNN has issues to suppress the sidelobes. In the case with 4 subarrays it might be a physical limitation that gives this result. For reference curves over 30° HPBW the results appear to be quite close to the larger antenna arrays and therefore 4 subarrays might be worth to train for wider references only. As the complexity for the larger antenna is higher the model might be too simple to learn enough to optimize a 64 subarray antenna. Tuning a CV-CNN for the larger antennas should return better results.

When observing the three antenna array sizes with better results in Figure 4.15 it seems like the difference is quite small. The mean total loss increases slightly with more subarrays and the uncertainty increases a bit more, however the increase in both is small. It is interesting to note that for 16 and 32 subarray antennas the model struggles with the wider reference curves in the same way as the results for 8 subarrays in section 5.3. The model seem to be able to scale up to a bit larger antennas but hyperparameter tuning is to recommend to improve its performance.

5.4 Stability Analysis

Figure 4.16 shows how the loss behaves when a noise is introduced to the beamweights outputted from the CV-CNN. The figure clearly shows that noise applied on both the amplitude and the phase affects the total loss, which is expected. From Figure 4.16 the loss seems to be higher in general for wider reference curves but also that the total loss increases faster with the increase of noise for a wider reference curve. The on average worse results for wider reference beams can be explained by the fact that the model generally performs worse on wider reference curves as can be seen in Figure 4.10 and Figure 4.13. This fact might in part explain the steeper increase in loss for wider reference curves but otherwise the reason is unknown. From Figure 4.16 it can also be seen that the loss increases a bit faster with phase noise than amplitude noise even though one part of the total loss, taper loss, only depends on the amplitude. This might be due to the amplitude and the phase being in different scales.

The effect on the loss when introducing noise in the AFK, used for both the input to the model as well as the total loss function, can be seen in Figure 4.17. The figure shows once again that the model performs worse on wider reference curves. However, the figure also shows that the loss seems to be highly dependent on the phase noise but not at all to the amplitude noise. This can be a result from the approximation $\mathbf{G}_n \approx |\mathbf{G}_n|$ in the AFK in equation (3.5). This approximation removes all irregularities in the phase from the measurements which results in that the phase difference between the subarrays in the AFK only depends in their physical separation that is assumed to be equal. This means that the model is only trained on uniform phase differences which might explain why it is sensitive to irregular phase differences. The model is however trained to handle irregularities in the amplitude which might explain why it is robust to the noise in amplitude. To get a more robust model one might then remove the approximation used in the AFK and train the model to handle irregular phase differences.

How the amplitude of the outputted beamweights is affected by noise being introduced in the input AFK is shown in Figure 4.18. The first thing to note in the figure is that the differences in amplitudes are very low. The maximum difference lies around 0.6%, which corresponds to a maximum of 0.098 in taper loss. That the amplitude differences are low are quite expected since the model is optimized against taper loss which is directly correlated to this. From Figure 4.18 it is clear that the effects in amplitude difference depends on how wide the reference curve is. The results from the 20° HPBW reference curve shows that the amplitude is almost completely unchanged regardless of the noise level. For the 40° HPBW reference curve there are a lot of difference and for the 65° HPBW reference curve the effects are somewhere in between. This can be explained through Figure 4.9 and Figure 4.12. In these figures it is clear that the taper loss and thus the amplitude variance is the highest for reference curves with HPBW around 40°. The uncertainty of the taper loss is also highest around reference curves with 40° HPBW which further implies that the amplitude varies much for reference curves with around 40° HPBW. This might then explain why the amplitude differences in Figure 4.18 are the highest for the 40° HPBW reference curve. In the Figure 4.9 and 4.12 it is clear that the taper loss is the lowest for narrower reference curves which might explain the low values of amplitude difference in Figure 4.18 for the 20° HPBW reference curve.

How the phase of the outputted beamweights is affected by noise being introduced in the input AFK is shown in Figure 4.19. From the figure it seems like the differences in phase are not as correlated to the width of the reference curve as the differences in amplitude were. The difference in phase does seem to be strongly correlated with the phase noise and weakly correlated to the amplitude noise. This might again partly be explained by the approximation used in the AFK making the model less robust against phase noise. It can also be partly explained by the fact that a phase noise should be compensated with a change in phase, if the phase of one input is shifted by one degree then the corresponding output phase should be shifted by one degree to compensate. A last remark is that the effects in phase difference is quite significant as the average can reach as high as 5°.

To summarize, the results in Figure 4.19 and Figure 4.18 indicates that noise in the input affects the outputted beamweights from the model, especially in phase. If these differences are thought of as noise to the beamweights then Figure 4.16 shows that a maximum average difference in amplitude around 0.6% does not have a large effect on the loss but a maximum average difference in phase of around 5° can have quite large implications on the loss. This together with the observation that the phase difference in Figure 4.19 correlates with the phase noise but not as much with the amplitude noise indicates that a measurement error in the input phase can have large implications to the performance of the antenna in practice. It also indicates that the model is not as robust against noise in the phase of the input as it is to noise in the amplitude of the input. This might partly be explained by the approximation made in the AFK.

6

Conclusion

A general conclusion of the project is that it is possible to predict beamweights for an unseen antenna using a ML model. The beamweights also seems to generate a satisfactory far field in relation to any given Gaussian reference curve.

A conclusion of the hyperparameter tuning is that it appears that there are multiple combinations of hyperparameters resulting in equally good models. With an extended search space, including the number of convolutional layers and number of hidden neurons an even better model could possibly be found. An increase in samples and epochs could also find better solutions.

The main conclusion of the comparison between the models is that which model is best depends on what the user is interested in. The CV-CNN performs better on total loss by lower taper loss while GODLIKE is better on far field difference. The difference in far field difference is larger for small and large values of subarray separation. If the GODLIKE parameters were more adapted to the specific antenna they were optimized against, it could potentially perform even better. GODLIKE returns a set of solutions which makes it more flexible than the CV-CNN. The CV-CNN is very much quicker than GODLIKE, once it is trained.

The results of the performance of the CV-CNN shows that the far field loss is dominant, especially for wider reference curves. The far field loss is very dependent on the subarray separation. The lowest subarray separation gives the worst result. The model returns low taper loss and no sidelobe loss for all antennas. The differences in taper loss is only dependent on the width of the reference curve and subarray separation. The conclusion is that the total loss is only dependent on the far field difference and that the farfield difference is worse for wider reference curves and narrow subarray separation.

The CV-CNN model was evaluated on different sizes of antenna arrays, but the hyperparameters in the model for all tests were optimized for antenna arrays with 8 subarrays in azimuth. The results showed that the model performed similarly on 8, 16, and 32 subarrays in azimuth, regardless of the element separation or the HPBW of the reference curve. For 4 subarrays in azimuth the model performed significantly worse for reference curves with lower HPBW, this is most likely explained by the physical limitations of an antenna with 4 subarrays, it can not easily generate such narrow beams. For 64 subarrays, the model generally performed a lot worse than for the other sizes. The reason for this might come from the fact that the model was optimized for 8 subarrays. The final conclusion is then that the model is scalable to different sizes of antenna arrays but the hyperparameters should be optimized for each size.

The results from the stability analysis shows that the CV-CNN is generally stable to amplitude errors in the input but that it is sensitive to phase errors in the input. A phase error in the input results in a low difference in the outputted beamweights amplitude but a significant difference in the beamweights phase. The results also shows that the system is sensitive to both amplitude and phase noise in the output beamweights. This indicates that phase noise in the input results in a loss in performance of the resulting far field. These effects can in part be explained by an approximation made in the calculation of the array factor kernel. Removing this approximation might then be a good step to improve the stability of the model.

6.1 Further Work

An improvement that could be done is to expand the search space in the hyperparameters tuning, one way is by including lower values of learning rate and including more hidden units in the hidden layer. Another is to include the number of convolutional layers and the number of hidden layers to the search space.

To improve the performance for differently sized antennas hyperparameter tuning should be done for each antenna array size.

As the taper loss is consequently low, the amplitude could be completely omitted in the optimization. This halves the number of degrees of freedom in the output which might improve the results from the optimization.

The input data to the model was approximated to not include phase differences from the measured subarrays. This could be included which might lead to a better, more stable model. Another improvement to the input could be to include the measurements of each polarization instead of assuming that they are equal. These changes might lead to a more reasonable modeling of the problem. Testing the different models of the problem can be done by generating beamweights from both models for the same antenna and then measure the resulting far field.

An interesting test would be to evaluate how a CV-CNN model performs on elevation as the characteristics of an antenna array in elevation is different from the characteristics in azimuth, as the subarrays used contains only one element in azimuth but several in elevation.

Another interesting extension is to see if the model can be extended to find beamweights for a 2D planar antenna array. The output can then be evaluated against a 3D reference curve. Then the far field is evaluated in both azimuth and elevation simultaneously.

In the thesis, only one loss function was used. An interesting extension is then to see how the beams and the system is affected when using different loss functions. A simple change is to weigh the different losses used to see if the model then minimizes some losses more than others. Another change is to lower the threshold used in the sidelobe loss to see if the model can minimize the sidelobes further. Apart from that the possibilities are endless and the loss function can be adapted to fit the need of the intended implementation.

An interesting extension of the project is to try out different reference curves than the one used. In the project, only Gaussian reference curves pointing straight forward from the antenna is used. Other forms of curves could be used and they could also be pointed in other directions than straight forward.

Another interesting analysis is to see how the model behaves when trained on different amount of data, maybe the model performs significantly better with more training data.

Bibliography

[1] P. von Butovitsch et al., *Massive MIMO handbook, Extended version*, Second Edition, D. A. et al., Ed. Ericsson AB, 2023.

[2] "Ericsson mobility report november 2023," `https://www.ericsson.com/en/reports-and-papers/mobility-report/reports` (accessed: 18 May 2024).

[3] S. Jayaprakasam, S. K. Abdul Rahim, C. Y. Leow, T. O. Ting, and A. A. Eteng, "Multiobjective beampattern optimization in collaborative beamforming via nsga-ii with selective distance," *IEEE Transactions on Antennas and Propagation*, vol. 65, no. 5, pp. 2348–2357, 2017. DOI: `10.1109/TAP.2017.2684187`.

[4] G. Sun, X. Zhao, G. Shen, *et al.*, "Improving performance of distributed collaborative beamforming in mobile wireless sensor networks: A multiobjective optimization method," *IEEE Internet of Things Journal*, vol. 7, no. 8, pp. 6787–6801, 2020. DOI: `10.1109/JIOT.2020.2983519`.

[5] T. Sallam and A. M. Attiya, "Convolutional neural network for 2d adaptive beamforming of phased array antennas with robustness to array imperfections," *International Journal of Microwave and Wireless Technologies*, vol. 13, no. 10, pp. 1096–1102, 2021. DOI: `10.1017/S1759078721001070`.

[6] S. Mohammadzadeh, V. H. Nascimento, R. C. de Lamare, and N. Hajarolasvadi, "Robust beamforming based on complex-valued convolutional neural networks for sensor arrays," *IEEE Signal Processing Letters*, vol. 29, pp. 2108–2112, 2022. DOI: `10.1109/LSP.2022.3212637`.

[7] P.-S. Kildal, *Foundations of Antenna Engineering, A Unified Approach for Line-Of-Sight And Multipath*, J. Yang, Ed. Kildal Antenn AB, Mar. 2021.

[8] A. Tsagkaropoulos, *Main coordinate systems*, TikZ. `https://tikz.net/main-coordinate-systems/` (accessed: 27 March 2024).

[9] P. J. Bevelacqua, *Frequency*, Antenna-Theory. `https://www.antenna-theory.com/basics/frequency.php` (accessed: 25 March 2024).

[10] *Leveraging the potential of 5g millimeter wave*, Ericsson. `https://www.ericsson.com/en/reports-and-papers/further-insights/leveraging-the-potential-of-5g-millimeter-wave` (accessed: 25 March 2024).

[11] P. J. Bevelacqua, *Field regions*, Antenna-Theory. `https://www.antenna-theory.com/basics/fieldRegions.php` (accessed: 22 March 2024).

[12] D. K. Cheng, *Field and Wave Electromagnetics*, Second Edition. Pearson Education Limited, 2014.

[13] S. O. Petersson, "Power-efficient beam pattern synthesis via dual polarization beamforming," in *2020 14th European Conference on Antennas and Propagation (EuCAP)*, 2020, pp. 1–5. DOI: `10.23919/EuCAP48036.2020.9135929`.

[14] M. Girnyk, H. Jidhage, and S. Faxér, "Broad beamforming technology in 5g massive mimo," *Ericsson Technology Review*, vol. 10, Oct. 2023. [Online]. Available: `https://www.ericsson.com/490a10/assets/local/reports-papers/ericsson-technology-review/docs/2023/broad-beamforming-technology-in-5g-massive-mimo.pdf`.

[15] P. J. Bevelacqua, *Directivity*, Antenna-Theory. `https://www.antenna-theory.com/basics/directivity.php` (accessed: 22 March 2024).

[16] P. J. Bevelacqua, *Antenna gain*, Antenna-Theory. `https://www.antenna-theory.com/basics/gain.php` (accessed: 22 March 2024).

[17] A. Pastukh, V. Tikhvinskiy, E. Devyatkin, and A. Kostin, "Interference analysis of 5g nr base stations to fixed satellite service bent-pipe transponders in the 6425–7125 mhz frequency band," *Sensors*, vol. 23, 172, 2023. DOI: `10.3390/s23010172`.

[18] M. B. et al., *A First Course in Complex Analysis*. Orthogonal Publishing L3c, 2016, ISBN: 9781944325015. [Online]. Available: `https://books.google.se/books?id=HrAEvgAACAAJ`.

[19] *Autograd mechanics*, PyTorch. `https://pytorch.org/docs/stable/notes/autograd.html` (accessed: 4 April 2024).

[20] M. Wahde, *Biologically Inspired Optimization Methods: An Introduction*. WIT Press, 2008, ISBN: 9781853128530. [Online]. Available: `https://books.google.se/books?id=LkPvAAAAMAAJ`.

[21] M. Emmerich and A. Deutz, "A tutorial on multiobjective optimization: Fundamentals and evolutionary methods," *Natural Computing*, vol. 17, pp. 585–609, Sep. 2018. DOI: `10.1007/s11047-018-9685-y`.

[22] K. V. Price, "Differential evolution," in *Handbook of Optimization: From Classical to Modern Approach*, I. Zelinka, V. Snášel, and A. Abraham, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, ISBN: 978-3-642-30504-7. DOI: `10.1007/978-3-642-30504-7_8`. [Online]. Available: `https://doi.org/10.1007/978-3-642-30504-7_8`.

[23] A. G. Nikolaev and S. H. Jacobson, "Simulated annealing," in *Handbook of Metaheuristics*, M. Gendreau and J.-Y. Potvin, Eds. Boston, MA: Springer US, 2010, ISBN: 978-1-4419-1665-5. DOI: `10.1007/978-1-4419-1665-5_1`. [Online]. Available: `https://doi.org/10.1007/978-1-4419-1665-5_1`.

[24] B. Mehlig, *Machine Learning with Neural Networks: An Introduction for Scientists and Engineers*. Cambridge University Press, 2021.

[25] I. Neutelings, *Neural networks*, TikZ. `https://tikz.net/neural_networks/` (accessed: 27 March 2024).

[26] J. Riebesell and S. Bringuier, *Collection of standalone tikz images*, version 0.1.0, 10.5281/zenodo.7486911 - https://github.com/janosh/tikz, Dec. 27, 2022. DOI: `10.5281/zenodo.7486911`. [Online]. Available: `https://github.com/janosh/tikz` (visited on 03/27/2024).

[27] D. P. Kingma and J. Ba, *Adam: A method for stochastic optimization*, 2017. arXiv: `1412.6980 [cs.LG]`.

[28] C. Trabelsi, O. Bilaniuk, D. Serdyuk, *et al.*, "Deep complex networks," *CoRR*, vol. abs/1705.09792, 2017. arXiv: `1705.09792`. [Online]. Available: `http://arxiv.org/abs/1705.09792`.

[29] wavefrontshaping, *Complexpytorch*, version 0.2.1, 10.1103/PhysRevX.11.021060 - https://github.com/wavefrontshaping/complexPyTorch/tree/master, Feb. 15, 2021. DOI: 10.1103/PhysRevX.11.021060. [Online]. Available: https://github.com/wavefrontshaping/complexPyTorch/tree/master (visited on 04/05/2024).

[30] L. Pfeifenberger, M. Zöhrer, and F. Pernkopf, "Deep complex-valued neural beamformers," May 2019, pp. 2902–2906. DOI: 10.1109/ICASSP.2019.8683517.

[31] C. Lee, H. Hasegawa, and S. Gao, "Complex-valued neural networks: A comprehensive survey," *IEEE/CAA Journal of Automatica Sinica*, vol. 9, no. 8, pp. 1406–1426, 2022. DOI: 10.1109/JAS.2022.105743.

[32] A. Paszke, S. Gross, F. Massa, *et al.*, "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.

[33] G. Van Rossum and F. L. Drake, *Python 3 Reference Manual*. Scotts Valley, CA: CreateSpace, 2009, ISBN: 1441412697.

[34] R. Liaw, E. Liang, R. Nishihara, P. Moritz, J. E. Gonzalez, and I. Stoica, "Tune: A research platform for distributed model selection and training," *arXiv preprint arXiv:1807.05118*, 2018.

[35] L. Li, K. Jamieson, A. Rostamizadeh, *et al.*, *A system for massively parallel hyperparameter tuning*, 2020. arXiv: 1810.05934 [cs.LG].

[36] R. Oldenhuis, *Godlike - a robust single-& multi-objective optimizer version: 1.5*, 2024. [Online]. Available: https://se.mathworks.com/matlabcentral/fileexchange/24838-godlike-a-robust-single-multi-objective-optimizer#version_history_tab (visited on 05/01/2024).

A

Appendix 1

A.1 GODLIKE Parameters

The parameters used in GODLIKE are all given standard values, most of the ones used are standard. However MaxFunEvals, popsize, MinIters and which algorithms to use is custom values. PSO is not recommended for multi-objective optimization and is therefore omitted and therefore also the parameters used for PSO. In table A.1 it is specified that ASA is used, however the ASA algorithm implemented in GODLIKE is SA [36]. The parameters for ASA are estimated by GODLIKE and there is no reason to change these, therefore also they are omitted. Also the parameters for single-objective optimization is omitted as there are multiple objective functions.

Table A.1: Parameters in GODLIKE optimizer: The table contains all parameters used in the GODLIKE optimizer when generating results.

Paremeter	**Value**
Algorithms used	DE, GA, ASA
MaxFunEvals	$5 \cdot 10^8$
MaxIters	20
MinIters	20
ItersLb	10
ItersUb	100
popsize	5000
Flb	-1.5
Fub	1.5
CrossConst	0.95
CrossProb	0.25
MutationProb	0.1
Coding	Binary
NumBits	52

DEPARTMENT OF MATHEMATICAL SCIENCES
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden
www.chalmers.se

CHALMERS
UNIVERSITY OF TECHNOLOGY