THE PENNSYLVANIA STATE UNIVERSITY
SCHREYER HONORS COLLEGE

DEPARTMENT OF ELECTRICAL ENGINEERING

MACHINE LEARNING FOR ELECTROMAGNETIC METAMATERIAL DESIGN

PHILIP J. O'CONNOR
SPRING 2020

A thesis
submitted in partial fulfillment
of the requirements
for a baccalaureate degree
in Electrical Engineering
with honors in Electrical Engineering

Reviewed and approved* by the following:

Douglas H. Werner
Professor of Electrical Engineering
Thesis Supervisor

Julio Urbina
Professor of Electrical Engineering
Honors Adviser

* Electronic approvals are on file.

ABSTRACT

Computational electromagnetics has found success in the design and simulation of metamaterial devices for many applications. This work explores machine learning as a tool for computationally efficiently modeling metamaterial devices. Conventional methods have proven effective, though computationally expensive and slow, for analyzing metamaterials. By using a dataset of geometric metamaterials generated by traditional computational electromagnetic methods, a neural network can be trained to generate and analyze new metamaterial designs more quickly than previous methods.

This work develops and presents a method for parameterizing a dataset of geometric metamaterial patterns and electromagnetic properties for a neural network to generalize these electromagnetic properties to new patterns. The motivation for training a neural network on this computationally expensive process of analyzing the electromagnetic properties of geometric metamaterials is to ultimately utilize the quick computation of neural networks. Through parallelized GPU computing, thousands of metamaterials can be simulated by the trained neural network in the same time conventional methods can analyze a single design.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

I would like to thank my friends and family for offering endless support throughout my time at Penn State.

I would like to thank Dr. Werner for welcoming me to his incredible lab and supporting me throughout my research. I would also like to thank everyone in the CEARL for promoting a great lab experience and helping make everyday a pleasure to come into lab. I would especially like to thank Ronald Jenkins and Sawyer Campbell for their great attitudes toward research and their direct help every step of the way. I really could not have asked for a better research experience.

The author acknowledges partial support of this work by the Penn State MRSEC, Center for Nanoscale Science (NSF DMR-1420620), and by the DARPA/DSO Extreme Optics and Imaging (EXTREME) Program (HR00111720032). The findings and conclusions of this work do not necessarily represent the beliefs of either of these groups.

Chapter 1

Introduction

Computational electromagnetics is a field study that utilizes Maxwell's equations to model the interactions of electromagnetic waves and the surrounding environment. This field of study is constantly battling the limitations of computational power, algorithmic efficiencies, and electromagnetic modeling. Increasingly more powerful modeling tools enable the design of increasingly more complex and useful electromagnetic devices. All three frontiers--computation, algorithms, and electromagnetics--are areas of study that progress the field of computational electromagnetics.

This thesis presents an algorithmic advancement, specifically a method for leveraging deep learning with neural networks to simulate electromagnetic metamaterial devices. Additionally, there will be brief discussion of the GPU computation that enables the computational efficiency of neural networks.

1.1 Motivation

Machine learning is quickly finding its way into almost every research discipline. Machine learning is especially applicable to fields that involve significant computation, like computational electromagnetics. This thesis explores a method for combining the potential computation speed-ups of machine learning methods and the modeling of computational electromagnetics.

Neural networks are a quickly growing and popular method for modeling all sorts of data. Neural networks are universal function approximators, so they make a good candidate for accurately modeling the nonlinear problems within computational electromagnetics.

1.2 Background

1.2.1 Metamaterial Antennas

Metamaterial antennas come from a field of study known as *computational electromagnetics*, which basically refers to methods for designing antennas that rely on optimization methods and computation without necessarily modeling the broader type of antenna [1].

Metamaterial antennas specifically are a field of study that uses complex geometric shapes to essentially emulate and customize light-molecule interactions. Similar to the way 700nm visible red light behaves differently with glass and metal, these light interactions can be manipulated by changing the size scales of the "molecules" to be on the order of 100nm rather than ~1nm for molecules. At this 100nm size scale, various manufacturing techniques can create complex and well defined shapes. If the right "molecule" shapes are chosen, this meta-material can have unique and desirable properties. This is a highly nonlinear problem with frequency-geometry resonances often dominating the behavior of metamaterial antennas, and antennas in general.

In contrast to computational metamaterial antennas, a conventional antenna, like a patch antenna, dipole antenna, or Yagi Uda array, are designed based on equations derived from Maxwell's equations. These well established designs and shapes have popular simplifications and approximations to Maxwell's equations that make their electromagnetic properties as antennas simple to predict.

In contrast, metamaterial antennas are generated through time-intensive computation of Maxwell's equations or through full-wave finite element analysis. Figure 1.1 shows a metamaterial antenna array generated from the extrusion of a less model-able 2D shape.

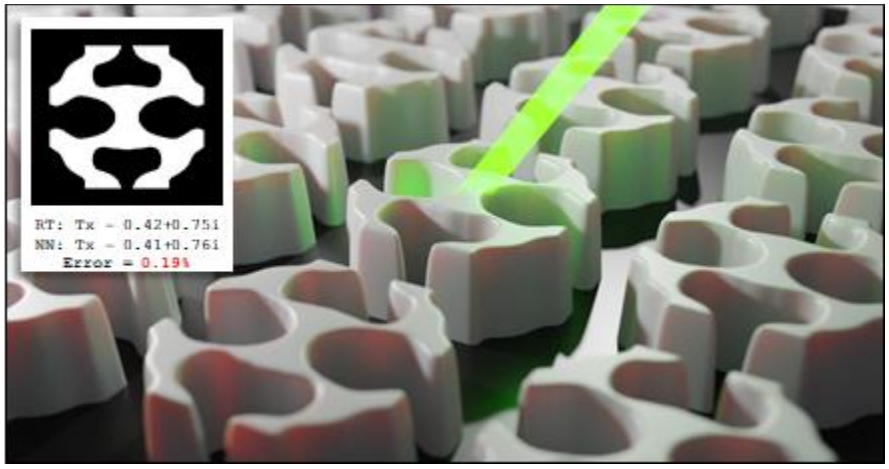

Figure 1.1: Rendering of a metamaterial array with the cross-section of the repeated single element depicted in the upper left.

In summary, the method to simulate a complex metamaterial shape like this one is relatively simple but time-intensive. And this makes it difficult to design a metamaterial antenna with the desired behavior.

This problem becomes even more difficult when designing complex metamaterial antenna arrays, with multiple complex shapes.

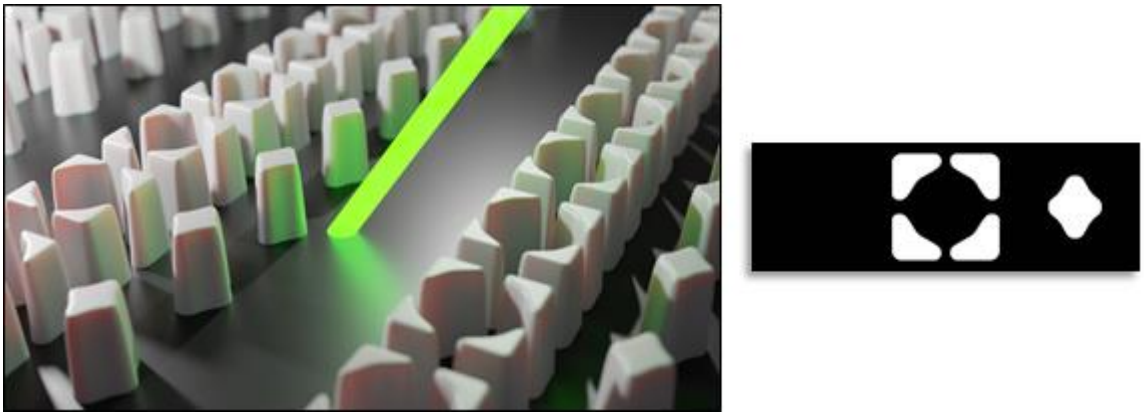

Figure 1.2: (left) Rendering of a metamaterial array with the cross-section of the three repeated elements depicted on the right.

1.2.2 Applying Machine Learning to Metamaterial Antennas

The computational complexity discussed above seemed like a great potential application of machine learning and neural networks. The hope would be to generate a training set of data, train a neural network, and then use a much quicker and computationally parallelized method for simulating these complex shapes.

The intuition supporting this hope lies in the ability of a neural network to learn a dimensionally compressed version of the problem based on the specifics of the defined constraints on the metamaterial antenna. For example, often a researcher in my lab will have a good understanding of constraints on the metamaterial antenna--perhaps there are material decisions, like silicon or metal plating, or height and size considerations for individual antenna elements that are known before modeling.

However, the current non-machine-learning methods for simulating these antennas cannot make use of these known constraints for computational speed-up.

The full-wave solvers are told the shape is silicon, for example, but this information is used in the equation for every calculation of a silicon metamaterial. In contrast, a neural network trained on just silicon metamaterials will have an easier problem to solve.

However, the dataset generation and neural network training must be faster than the conventional method for this new method to be useful. Or at least there must be a quicker transfer learning method onto subtly new problems, like the difference of training a silicon neural network model vs a metal-plated neural network model.

1.3 Original Contributions of this Work

- A method for using fully connected neural networks to model electromagnetic metasurfaces
- Discussion of the machine learning training methods for understanding and modeling a problem in electromagnetics
- Exploration of a neural network inversion method for metasurface pattern generation

Chapter 2

Related Work

Machine learning is becoming more popular for many different problems, and the field antennas and propagation is no exception. Many conventional problems are being explored for more efficient machine learning solutions while entirely new methods are being developed to better integrate machine learning solutions. Within the last few decades, machine learning has seen direct implementations of the some of the most similar sub-domains, i.e. image recognition, remote sensing, and signal processing [2].

This section will briefly discuss related work within the larger field of electromagnetics while the following section will discuss the theoretical work within the domain of machine learning.

2.1 Machine Learning Work in Antennas and Propagation

Many recent papers explore various machine learning methods in the broad field of antennas and propagation. While neural networks are becoming a very popular machine learning technique for a wide array of problems, other machine learning methods are also being explored. Support vector machines, or SVMs, can be used to model and design reflectarrays [3]. The design method is faster and maintains accuracy compared to method-of-moments solvers.

Neural networks have become an exceptionally popular technique. A convolutional neural network based on work from the machine learning and image recognition field can be applied to an inverse electromagnetic scattering problem [4]. The neural network can be applied to complex-valued, like electromagnetic scattering while maintaining accuracy.

Methods for improving neural networks are also being developed by the antennas and propagation community. A method was developed to reduce the required dataset size for using a neural network approach to design a patch antenna [5]. The method uses data mining techniques to essentially create new trainable data from the data that is already present.

New machine learning algorithms intertwined with electromagnetic intuitions can be developed for specific antenna problems. A new antenna optimization method was developed for the design of a cavity-backed slot antenna and then applied to 5G applications [6]. The designed antenna was then manufactured and shown to have accurate and realizable properties.

Generative adversarial networks, or GANs, are also becoming very popular among many fields, as well as the field of antennas and propagation. GANs make use of two competing networks in a reinforcement learning context to develop more accurate models. An energy-based model was used a proof-of-concept for employing GANs in general modeling methods [7].

Unsupervised methods can be applicable to antenna problems as well. By using the popular K-means algorithm to cluster similar groups of basis functions, multipole design approaches can be applied to antenna design [8].

2.2 Machine Learning Work in Computational Electromagnetics

Machine learning methods are also being explored for computational electromagnetics problems, like designing metamaterial devices. Supervised learning methods, like neural networks, have been particularly effective.

The metamaterial elements can be parameterized as various simple shapes, simulated using conventional methods, and then used to train a neural network. Once trained, the neural network can accurately and quickly simulate similar metamaterial elements [9]. There is however discussion and complications with directly applying this trained neural networks to new but related metamaterial problems.

Different parameterizations of metamaterial patterns allows for larger or smaller exploration spaces. A 1-dimensional barcode-like encoding was used with a generative adversarial network approach to develop a method that allows computationally more efficient design of metamaterial arrays [10]. The work can likely be applied to solving other inverse design problems as well.

Chapter 3

Fundamentals of Neural Networks

3.1 Neural Networks as Universal Function Approximators

Neural networks have been mathematically shown to be capable of approximating any arbitrarily complex function, even nonlinear functions. In theory, a large enough neural network can model anything [11]. Unfortunately, the neural network needs to be exponentially larger for a linearly more complex problem. However, the practical difficulty comes in learning this ideal neural network for a given problem.

3.1.1 General Neural Network Structure

Neural networks are a system of nodes and connections. These connections are directed from the input layer, generally portrayed on the left, to the output layer, portrayed on the right. There are different ways these connections are structured and different ways these connections react to a node's value.

In general, the neural network input layer is populated with the values from the input data, then various activation functions are used to perform combinations of this input layer to populate the successive hidden layers of the neural network until the output layer is populated. The structure of the neural network also enables backward computations, the backbone of neural network training.

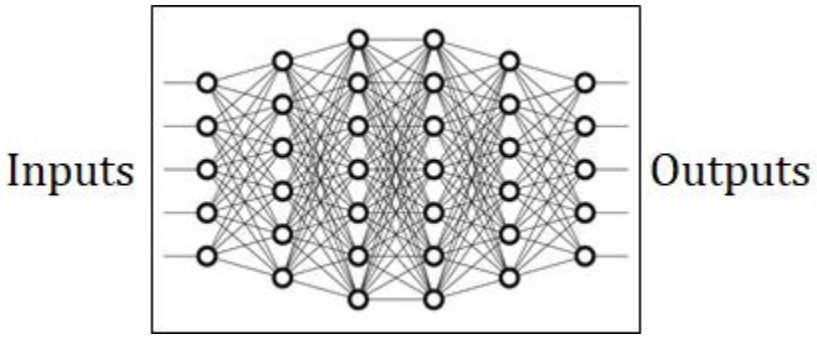

Figure 3.1: Graphical representation of the connections and nodes of a fully connected neural network.

Figure 3.1 depicts the structure of a fully connected neural network. The inputs are fed into nodes then layers of nodes are connected through activation function until reaching the outputs.

3.1.2 Nonlinear Activation Functions

The nonlinear modeling capability of neural networks comes from the choice of a nonlinear activation function. The first prominent nonlinear activation function was the sigmoid function, shown in figure 3.2. The combination of many of these simple nonlinear functions enable the neural network as a whole to model very complex, nonlinear functions.

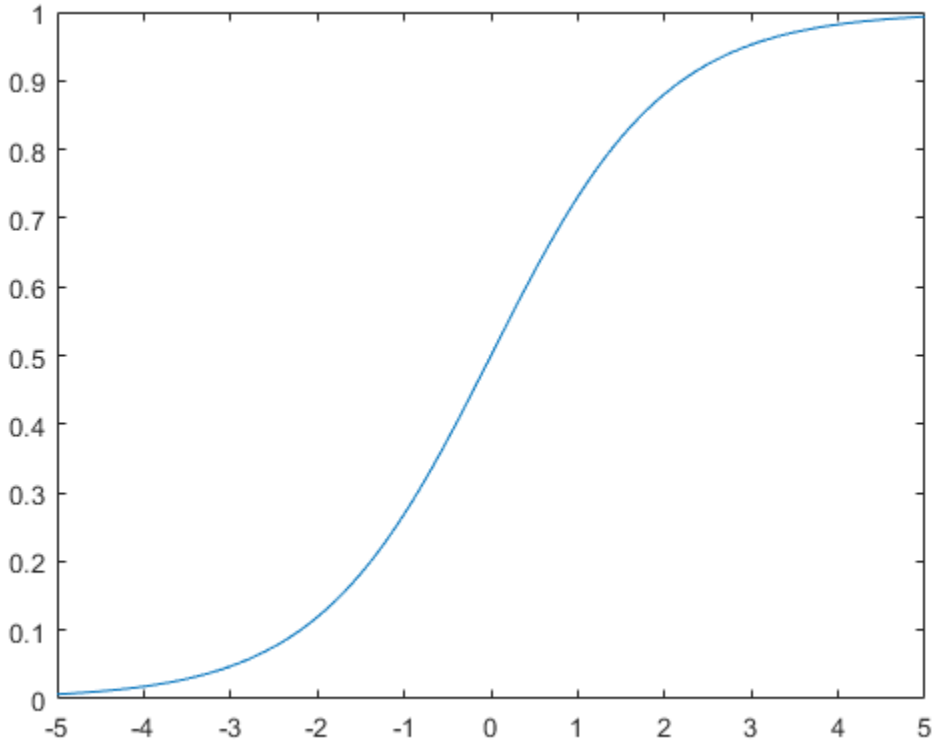

Figure 3.2: Plot of the sigmoid activation function (unitless).

3.2 Gradient Descent and Backpropagation

The backbone of neural network training is the backpropagation algorithm. This is the algorithm that enables a neural network to iteratively improve its prediction ability using pairs of input and output data. Backpropagation is a gradient descent method that calculates the error of every parameter in the neural network--every connection and bias term. This error then determines whether any given parameter should be increased or decreased to improve the network's ability to predict a given training example.

Without discussing the full details of the algorithms, the vector forms of the equations are shown below [12]. For each of the multiplicative weight terms and the

constant bias terms, the error is calculated backwards from the output layer (equations 3.1 and 3.2) where $\odot$ refers to the Hadamard product [13].

$$\delta^L = \nabla_a C \odot \sigma'(z^L) \tag{3.1}$$

$$\delta^l = ((w^{l+1})^T \delta^{l+1}) \odot \sigma'(z^l) \tag{3.2}$$

Then the gradient of this error with respect to each network parameter is calculated (equations 3.3 and 3.4).

$$\frac{\delta C}{\delta b^l{}_j} = \delta^l{}_j \tag{3.3}$$

$$\frac{\delta C}{\delta w^l{}_{jk}} = a^{l-1}{}_k \, \delta^l{}_j \tag{3.4}$$

3.3 Hyperparameters

The hyperparameters of a neural network are all the heuristically determined design decisions that affect the architecture and training of neural networks. The many different machine learning methods have many different possible hyperparameters to describe the method for training and structure of the neural network. Common hyperparameters in the training of neural networks are the learning rate, the depth of the network, and the width of layers of a network. There is often a trade-off between computational speed and the likelihood of improved training.

3.3.1 Learning Rate

The learning rate of neural network training refers to the multiplicative factor given to the error gradients during backpropagation to actually adjust the network parameters. Generally, just taking the error gradients as actual values would cause too steep an adjustment in model parameters, so generally the learning rate is much less than 1 to offer a more smooth convergence to better network parameters.

The learning rate can be a fickle hyperparameter for neural network training, especially due to the practicality of wanting to increase the speed of neural network training. Having a very large learning rate can cause the backpropagation algorithm to worsen the accuracy of a neural network and prevent convergence. Having too small a learning rate will cause the training to take an exceptionally long time. A common compromise is to have an adjustable learning rate. The intuition is that training can start with a large learning rate to quickly train the broader relations captured in a neural network, and as training progresses, a smaller learning rate allows gradient descent to converge to the extrema to model the finer details.

There are many important considerations for choosing the correct learning rate, but there are really only heuristic methods for finding good learning rates for a given problem. For example, the learning rate for training a convolutional neural network to identify images of cats has no obvious implication for a good learning rate for training a fully connected neural network to model metamaterial elements. However, choosing learning rates from similar published applications is generally the best starting point available.

3.3.2 Number of Layers and Size of Layers

Similar to learning rate, the number of layers and size of layers in a neural network are heuristically determined. There are no steadfast rules, and many celebrated advances in machine learning come from new combinations of the number and size of layers. In general, more layers and larger layers allow for a greater modeling capacity, but there definitely a critical point where neural network training fails to extend to larger networks. Larger networks guarantee slower training, but they also can prevent the network from learning. This field of study is what puts the “deep” in “deep learning.” Some architectures are more or less susceptible to problems with going deeper.

3.3.3 Dataset Size

While not always considered a hyperparameter, the size of the dataset is a critical consideration for ensuring a neural network will generalize beyond the training data. In general, there are rigorous methods for predicting a sufficient dataset size. There are ways to predict the ceiling, or the maximum that would be needed, but they are often orders of magnitude larger than practical, and many estimates are orders of magnitude larger than needed.

Depending on how data is generated, the simplest method for determining a sufficient dataset size is often to keep producing data until more data no longer seems to help improve the accuracy of the neural network. Another possible method is to simply review similar published work describing their sufficient dataset sizes.

3.4 Types of Neural Networks

There are many types of artificial neural networks (ANNs), and hybrids of these many types, so only the most popular types will be mentioned below. Also, these neural networks will be discussed in a supervised learning context, though all of them could be used for unsupervised and reinforcement learning methods as well.

3.4.1 Fully Connected Neural Networks

The fully connected, or feedforward, neural network is the original, and in many ways simplest, neural network. Every node in a given layer of a fully connected neural network connects to every node in the next layer. As described above in the general neural network architecture section, a fully connected network connects nodes with a simple nonlinear activation function, like a sigmoid or ReLU (rectified linear unit).

Fully connected neural networks are often a good starting point for a new machine learning problem because they can offer early insights into the complexity of modeling the problem. There are fewer hyperparameters to change for a fully connected neural network, so a hyperparameter sweep can be more quickly performed. Ideally, a relatively simple fully connected network will offer good performance and maybe that is sufficient for a specific application. But if the fully connected network is not sufficient, it might at least suggest what network architecture will be needed.

3.4.2 Convolutional Neural Networks

Convolutional neural networks (CNNs) are a class of ANNs that utilize convolution to learn hierarchical patterns within data. CNNs preserve spatial relations between data and generalize across different spatial scales, so they are well posed for image data, in any dimensionality.

CNNs use kernels to determine connections between nodes of a network. The kernels are convolved with the data to geometric representations of the data. CNNs make use of backpropagation to iteratively apply gradient descent to the weights of the kernels. The equations for backpropagating through a CNN are more complex than the simple fully connected network, but they can still be vectorized and parallelized to utilize the computation speed-ups of computation on a GPU.

Deep learning is an area of particular interest. Methods are being developed to enable very large and deep networks to be effectively trained [14]. Convolutional neural networks have been empirically shown to be particularly good at training very deep networks [15].

3.4.3 Recurrent Neural Networks

Recurrent neural networks (RNNs) are a class of ANNs that utilize backward (or feedback) connections to enable a memory of internal states between successive passes to the network. RNNs can process collections of successive input data and are well-posed for learning series data [16].

Relatedly, long short-term memory (LSTM) neural networks are a type of recurrent neural networks that use a memory cell with “forget” gates [17]. They are empirically shown to perform well on series data that involves multiple time scales, like the stock market or language interpretation.

3.4.4 Generative Adversarial Neural Networks

Generative adversarial neural networks are a class of machine learning systems wherein two ANNs compete to improve each other’s’ models [18]. The generative network learns to create inputs indistinguishable from the training data while the discriminative network learns to identify true data from data created by the generative network. GANs have recently become popular for problems across many research topics, and some machine learning researchers doubt their supremacy, but they have undeniably found success in many problems. Part of this success is likely in their intuitive method of operation.

3.5 Training Methods

3.5.1 Transfer Learning

Transfer learning is a method for taking a previously trained neural network and using some of its weights for connections as the initialization for a new network on a new problem [19]. Transfer learning works best when this new problem is closely related to the problem of the original network. Transfer learning can dramatically reduce the

computational time required to achieve good accuracy on a new problem. This is especially true for very deep networks, with hundreds of layers. Rather than retraining a whole new convolutional neural network to, for example, identify cats in images, a researcher can use one of many open-sourced pre-trained neural networks on similar image classification problems.

3.5.2 Batch Normalization

Batch Normalization is a method for improving the training time and accuracy of neural networks. Batch Normalization works by rescaling and normalizing the activations of connections during the forward pass and backpropagation of neural network training [20]. This helps spread the learning of the neural networks across all the nodes rather than allowing a small subset of the nodes to dominate the learning and reduce the generalizability and modeling capacity of the network. It has been empirically shown improve accuracy and computational time.

3.5.3 Dropout

Dropout is a method to randomly disable nodes in a neural network during training to reduce dependency on any given node [21]. A certain dropout rate will be defined during training to remove that percentage of nodes in a given layer. It has been empirically shown that this helps the network generalize and avoid overfitting.

3.6 Computational Speed-up of GPU Parallelization

The major benefit of using neural networks in the pursuit of computationally quicker algorithms is the ability for neural network computations to be vectorized and computed in parallel on a GPU (graphics processing unit). Every single training pair of inputs and outputs can be computed in parallel since they are all independent of each other. In practice, usually subsets of the entire training data are taken due to GPU memory limitations, but these training subsets, or batches, greatly increase the computation speed.

The calculations for the forward pass and backpropagation of a neural network can be vectorized, meaning there are vector notation forms of all the equations. GPUs are specifically designed to calculate thousands or millions of similar and simple computations at the same time [22]. While the actual application of graphics is best solved with these simultaneous computations, neural networks can also greatly benefit from simultaneous, parallel computation.

This computational speed-up for the neural network forward pass and backpropagation of gradient descent enabled by GPUs is reason for the explosion of neural network research in the last decade. Many of the algorithms for the foundations of neural networks were developed decades ago, but the vectorization and parallelization enabled by GPUs are what brought neural networks to the cutting edge of computation and modeling. The work of this thesis is largely dependent on the computational speed-ups of neural networks calculations being performed on GPUs.

Chapter 4

Encoding Metamaterials for Neural Networks

The conventional method for simulating a metamaterial is to create the geometrical structure in an electromagnetics software, like HFSS or COMSOL, and assign the corresponding material types to different parts of the structure, like silicon or various metals. Then the software uses full-wave solvers, finite-element analysis, and/or a lot of Maxwell's equations to slowly but surely simulate the structure. This method is computationally very slow.

The work in this thesis is specifically focused on 2-dimensional metamaterials. While there is existing work on the more complex 3-dimensional metamaterials, they are much more difficult to manufacture and are less common in practical applications. Another benefit of 2-dimensional metamaterials is there are some equations to closely approximate their behavior at a fraction of the computational time.

4.1 Data Generation

Without digging into the details of the grating solver method, Reticolo allows for 2-dimensional metamaterials to be simulated relatively quickly compared to full-wave and finite-element solvers [23]. The simulations run by defining the geometric cross-section, the extrusion height, the operating frequency, and the material of the metamaterial shape. Given these inputs, Reticolo will then compute the electromagnetic

transmission and reflection of an incident wave. The geometric cross-section is defined by an arbitrary resolution binary image of the exact shape.

Using this Reticolo method, about 50,000 2-dimensional metamaterials patterns can be simulated in about 12 hours on a 10-core intel CPU. These simulations produce the electromagnetic transmission and reflection properties of interest.

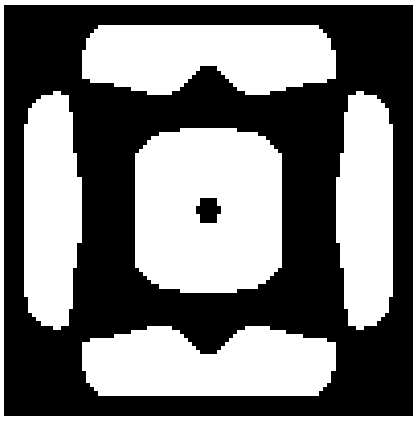

Figure 4.1: Depiction of a metamaterial binary image encoding for a pattern generated from 5x5 control points with enforced 4-fold symmetry.

An important additional design decision is what 50,000 patterns should actually get simulated. This question relies on some familiarity with metamaterials and some level of intuition about the specific application for the final metamaterial device. The question really is how complex of metamaterial shapes will be needed for the final application. Allowing for a very large solution space will require much more computation to search through that space, and many more patterns to accurately train a neural network. These design decisions will be discussed in more detail in the remainder of this section.

After the data is generated, the output is saved as magnitude and phase values for the 2 different polarizations of each the electric and magnetic components of the transmission and reflection, for a total of 32 values.

4.2 Input Encoding

There are several possible ways to encode an electromagnetic metamaterial for a neural network. The two methods explored in this paper are: an arbitrary resolution binary image of the pattern and a Bezier surface control point encoding.

4.2.1 Binary Image Encoding

The binary image encoding uses the same encoding as the original data generation--an arbitrary resolution binary image of the exact cross-sectional pattern. The benefits of this method are the neural network will be training on the exact metamaterial shape used to create the output data--there are no approximations made to represent the data.

The inherent drawback to this method is the data size and significant computation complexity of large images. Based on the data generation method and previous validation work, an image resolution of 100x100 pixels gives an accurate result from the Reticolo method [23].

Representing the geometric metamaterial data as a binary image also has machine learning applications. For those familiar with neural networks, an image-type input immediately suggests the use of convolutional neural networks. CNNs are often the best method for reducing the large dimensionality of images for a neural network.

4.2.2 Bezier Control Point Encoding

The Bezier surface control point encoding allows for a significantly reduced dimensionality by parameterizing and limiting the space of possible shapes.

A Bezier surface is a type of spline that is defined by coordinate positions and smoothly interpolated. The Bezier surfaces in this work are surfaces existing in 3-dimensions. The simplest analogy is to think of a bedsheet pulled tight by several people. Now imagine there are puppet strings attached to certain spots above and below the bedsheet. The strings can be manipulated up and down to change the shape of this bedsheet.

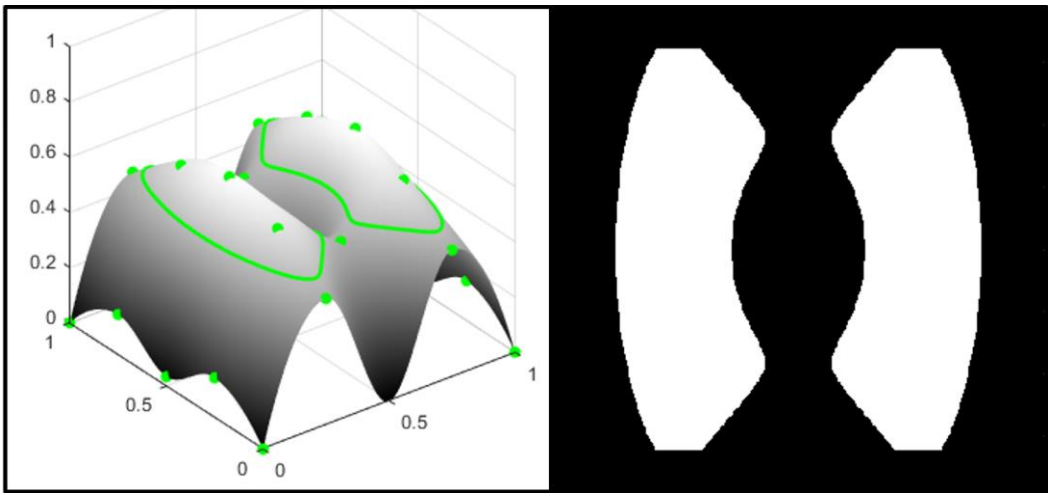

Figure 4.2: (left) Depiction of the Bezier surface defined by the green control points. (right) The cross section of the Bezier surface that is used as a metamaterial pattern.

Once this Bezier surface is defined, then a cross-section is taken to define a two dimensional shape, and that is the shape used as the 2-dimensional metamaterial pattern.

One key feature of this Bezier surface cross-section method is that it produces smooth shapes, which allows for a relatively continuous change in electromagnetic properties between very similar shapes.

Additionally, this method results in a significantly smaller input dimensionality compared the binary image method. Now, a small number of control points directly represent the metamaterial shape. This work explores Bezier representations by 3x3 and 5x5 control points, so even 5x5 = 25 control points is significantly less than 100x100 = 10,000 pixels for the binary image representation.

4.3 Output Encoding

After the data is generated, the output is initially encoded as magnitude and phase values for the 2 different polarizations of each the electric and magnetic components of the transmission and reflection, for a total of 32 values.

The two tested encodings for the output values were to leave the values as magnitude and phase and to convert the values to real and imaginary numbers.

4.3.1 Magnitude and Phase Encoding

After a few tests of training a neural network on magnitude and phase encodings of the transmission properties, it became clear there was significant error caused by

phase-wrap-around. This encoding caused very similar metamaterial patterns to be recorded as having drastically different phases. As shown in figure 4.3, these two patterns are very similar, and have very similar phases, but they are on the arbitrary cut-off of where phase happens to be referenced, so they are recorded as having the most different possible phases.

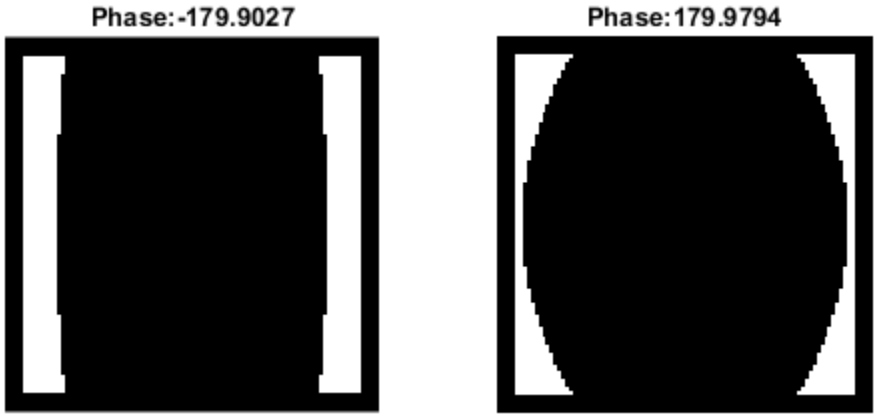

Figure 4.3: Two very similar metamaterial patterns with very similar phases. However, phase wrap-around leads to the two phase being parameterized as nearly 360 degrees apart.

To further explore this exact problem, the plot in figure 4.4 show the absolute difference between the predicted phase values and the true phase values. There is a clear trend on the boundaries of the -180 to 180 degree phase range. Some patterns that are truly -180 degree phase are predicted to be 180, and vice versa. There is simply no mechanism for a neural network to learn and accurately characterize this wrap-around

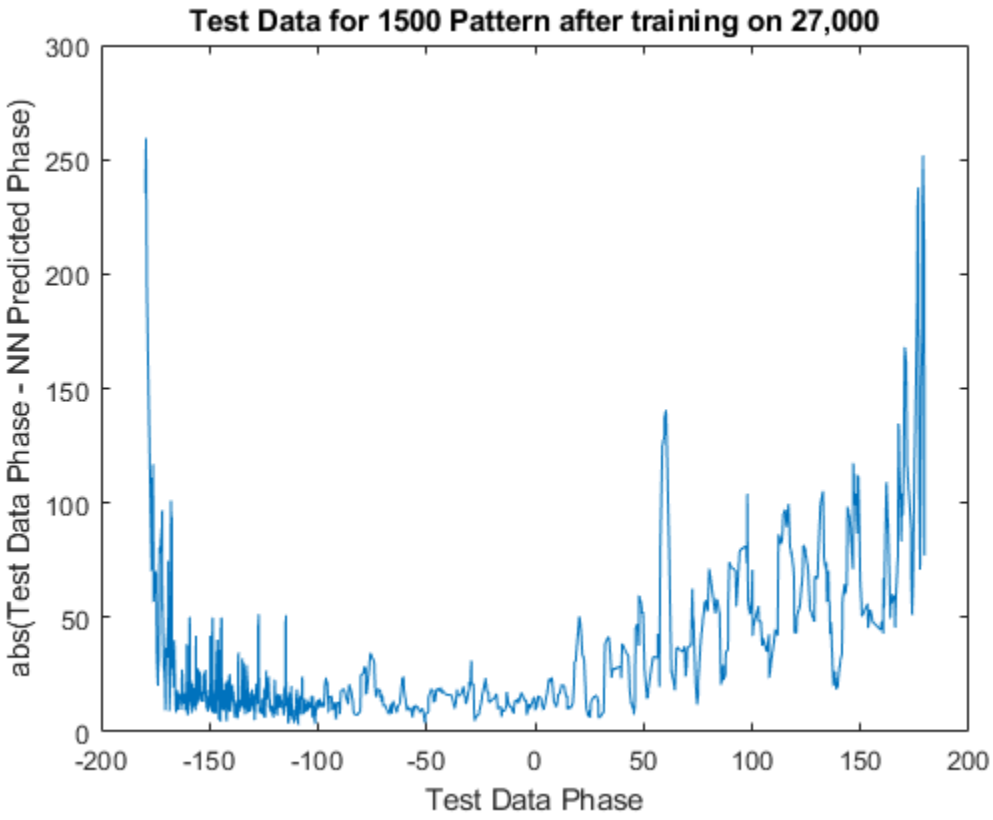

Figure 4.4: Plot of the absolute difference of predicted and actual phase on the test data set. There is noticeably higher error near the boundaries of -180 and 180 degrees phase.

Additionally, the figure above shows an interesting difference between patterns with phases closer to -180 degree phase and patterns closer to 180 degree phase. There is noticeably more error on the 180 degree phase end.

To explore this inconsistency in neural network accuracy depending on phase, the plot in figure 4.5 is two overlapping plots--the orange is a scatter plot showing the same metric as above, the absolute difference between prediction and true phase, and the blue is a histogram of the total number of patterns of a given phase. There is a clear data bias to having a dataset with patterns that coincidentally have a phase centered around -140 degrees.

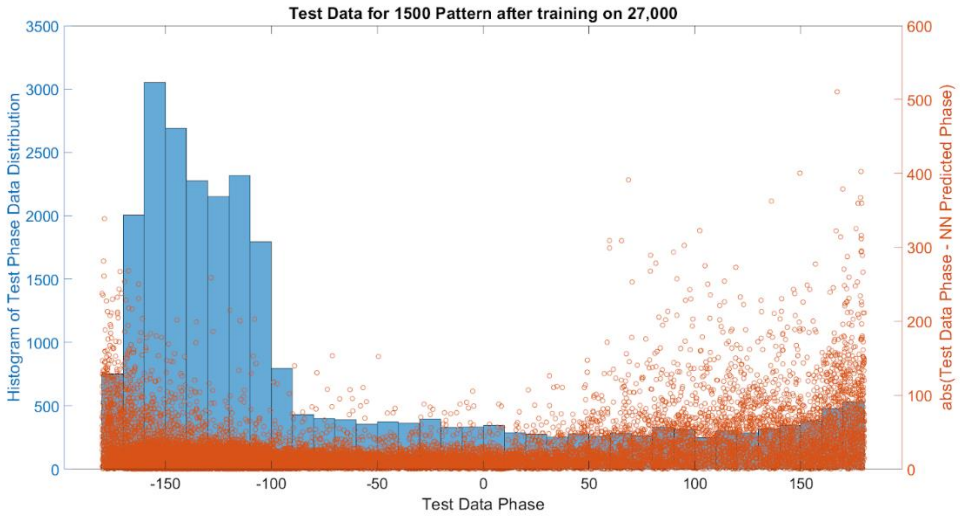

Figure 4.5: Two overlapping plots. In orange, there is a scatter plot of the absolute difference between the predicted and actual phase showing a tendency for higher error near the boundaries of -180 and 180 degree. This scatter plot also shows even higher error near the positive 180 degree boundary. In blue, there is a histogram of all the true phase of the test set. The higher number of phases near the -180 degree boundary explains the lower error on this side.

This exploration of phase-encoding the output data also led to a discovery in the dataset of a data bias stemming the particular solution space of metamaterial patterns being explored. While this is not inherently bad, it is an important consideration moving forward. If the final training is data-limited, as in more training empirically seems like it would produce an appreciably more accurate neural network, then it might be worthwhile to keep this dataset bias in. If the dataset empirically seems to be a sufficient size, or even larger than necessary, it might be more appealing to remove some of the data centered around this bias at -140 degrees phase and flatten out the histogram.

4.3.2 Real and Imaginary Encoding

As discussed above, the magnitude and phase encoding of the transmission and reflection properties of the metamaterials causes problems due to phase wrap-around, so the simplest solution is to convert the values to real and imaginary numbers.

Using this real and imaginary encoding removed the phase-dependent error of the neural network prediction

4.3.3 Geometric Sorting to Explore Output Continuity

During the exploration of the data, a study of the sensitivity of the output parameters was conducted. Metamaterials patterns were sorted by geometric similarity. An arbitrary starting pattern was chosen as the starting pattern. Then the highest correlating pattern was chosen as the following pattern. And this process was repeated until the patterns were sorted such that nearby patterns would be geometrically similar.

Figure 4.6 shows the results of plotting the electromagnetic transmission encoded both as real and imaginary numbers and as the magnitude and phase.

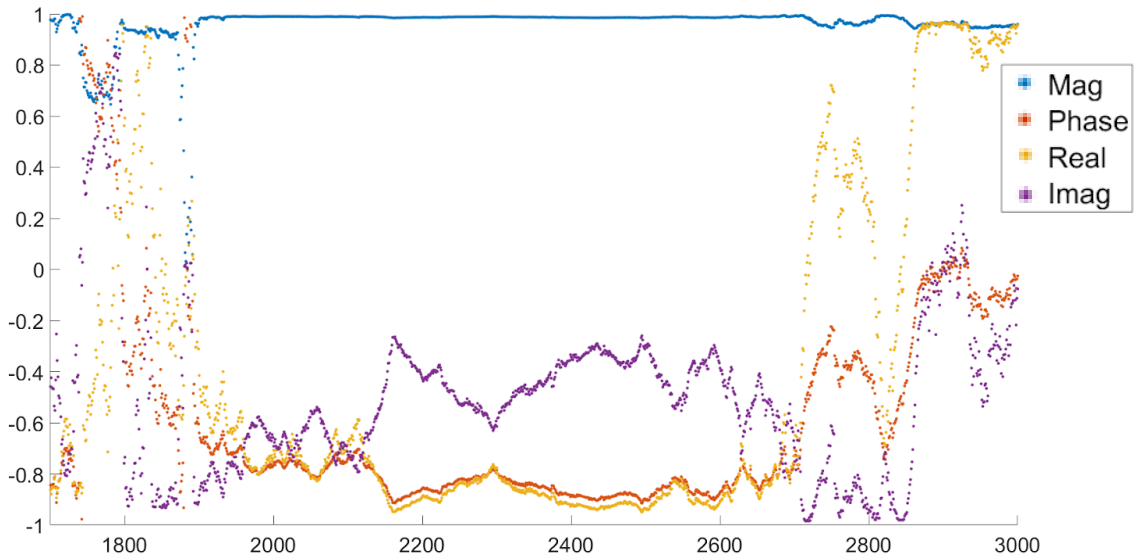

Figure 4.6: There are four plots showing the magnitude and phase encoding and the real and imaginary encoding, all linearly normalized between -1 and 1. They are plotted along x-axis by index of geometric similarity. The thing of note for this plot is which encodings are most sensitive to changes in value from small changes in geometry.

Some of the important understandings to be derived from figure 4.6 are which encodings are more sensitive to small changes in geometry and how many total changes in direction any of the patterns take. This two observations help inform the necessary modeling capacity of the neural network. Seemingly, there are 30 to 50 nonlinear changes in electromagnetic output parameters.

4.4 Exploration Space of Patterns

The exploration space of solutions being considered is an important consideration for designing a metamaterial element irrespective of machine learning methods. Ideally, the exploration space will be sufficiently complex to offer a rich variety of metamaterial

solutions while not being prohibitively complex to be computationally tractable. The problem unfortunately becomes entangled with machine learning once the neural network complexity and training become a limiting factor.

4.4.1 Solution Space for Data Generation

Relying on intuitions from previous work with metamaterials, the starting point for this work was on 3x3 control point encodings for data generation. These patterns could be fed through the data generation method described above and applied to either a binary image encoding or a Bezier control point encoding. Additionally, the 5x5 control

Compared to other methods for generating metamaterial geometries, the Bezier control point method offers an easily parameterized exploration space that changes smoothly and minimizes resonances.

One considered method for data generation was using optionally symmetric random-walk patterns. This would result in a significantly larger exploration space than the Bezier control point encodings for a comparable pixel resolution. The random-walk patterns could be smaller images scaled up, but there are other difficulties with random-walk patterns as well. They have many more discontinuities and nonlinear resonances

due to the geometric jaggedness.

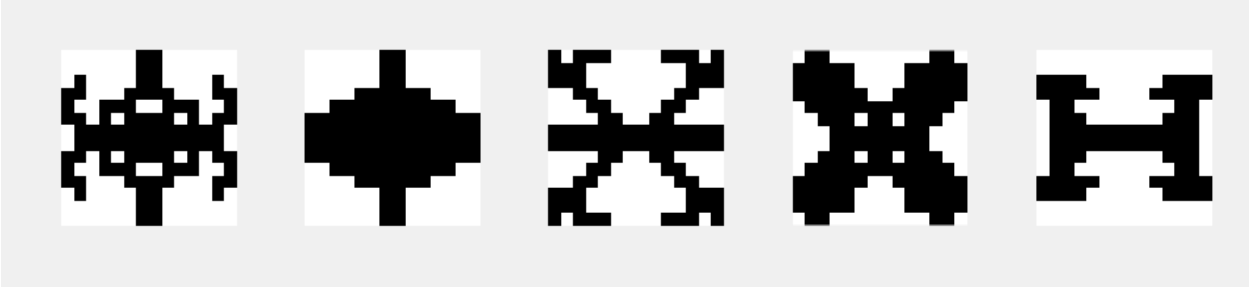

Figure 4.7: Five depictions of random-walk metamaterial patterns are shown.

Ultimately, the work of this thesis was performed on the Bezier control point encodings.

4.4.2 Pattern Symmetry

Another important consideration relating to limiting the exploration space is to optionally enforce a symmetry in the geometry of the metamaterial pattern. Enforcing either a 2-fold or 4-fold symmetry reduces the exploration space while allowing some larger and more complex patterns.

Figures 4.8, 4.9, and 4.10 depict metamaterial patterns generated using a less complex 3x3 control point Bezier surface method. These patterns are simpler than the 5x5 control point patterns. Figures 4.11, 4.12, and 4.13 show 5x5 control point patterns.

Figure 4.8 shows the 3x3 control point pattern with 4-fold symmetry enforced. These are the geometrically simplest patterns.

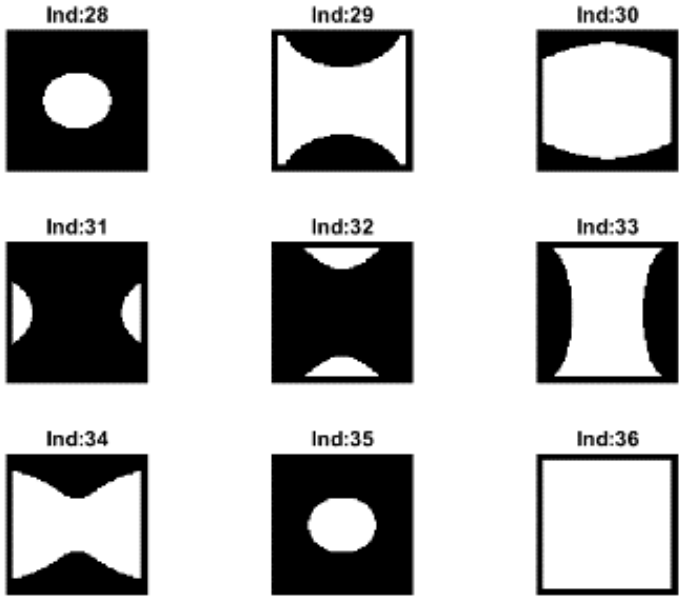

Figure 4.8: Nine depictions of metamaterial patterns generated with 3x3 Bezier control points and with 4-fold symmetry.

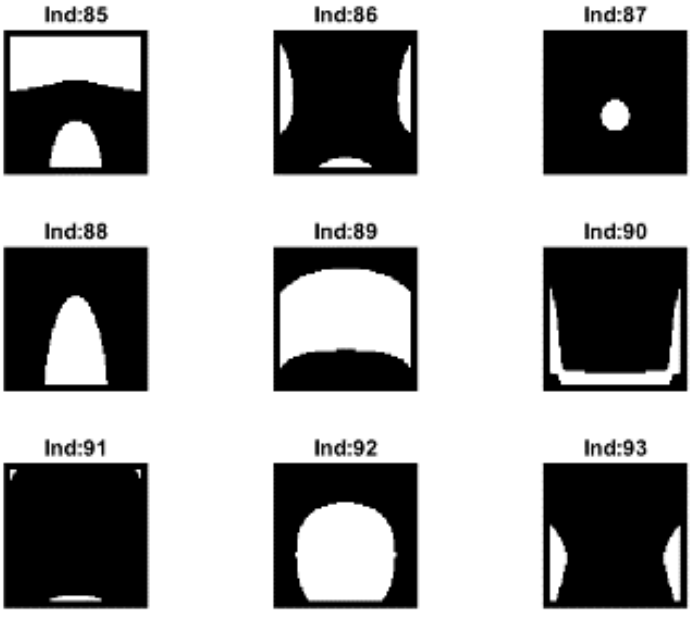

Figure 4.9: Nine depictions of metamaterial patterns generated with 3x3 Bezier control points and with 2-fold symmetry.

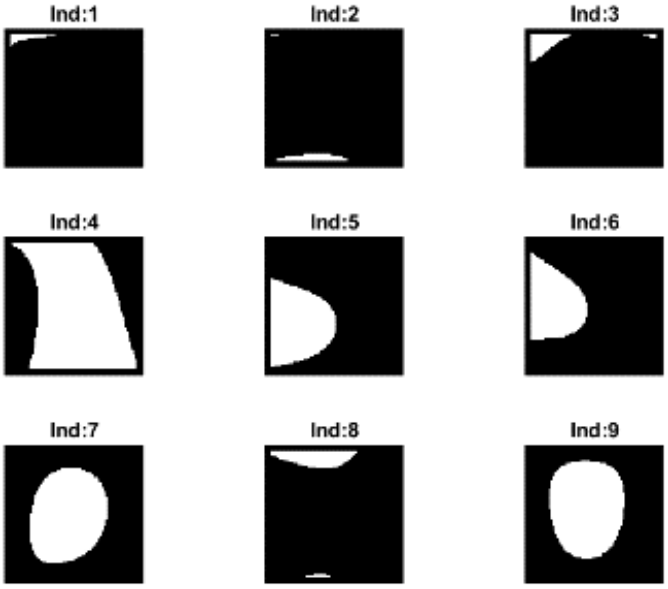

Figure 4.10: Nine depictions of metamaterial patterns generated with 3x3 Bezier control points and with no symmetry.

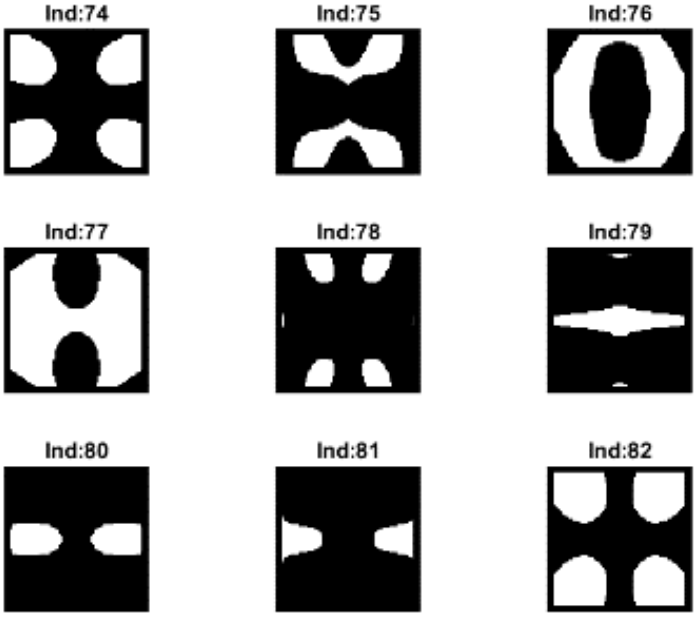

Figure 4.11: Nine depictions of metamaterial patterns generated with 5x5 Bezier control points and with 4-fold symmetry.

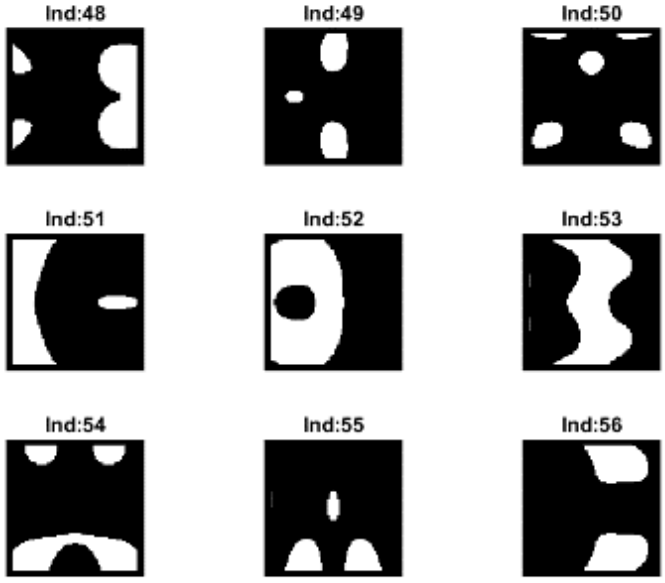

Figure 4.12: Nine depictions of metamaterial patterns generated with 5x5 Bezier control points and with 2-fold symmetry.

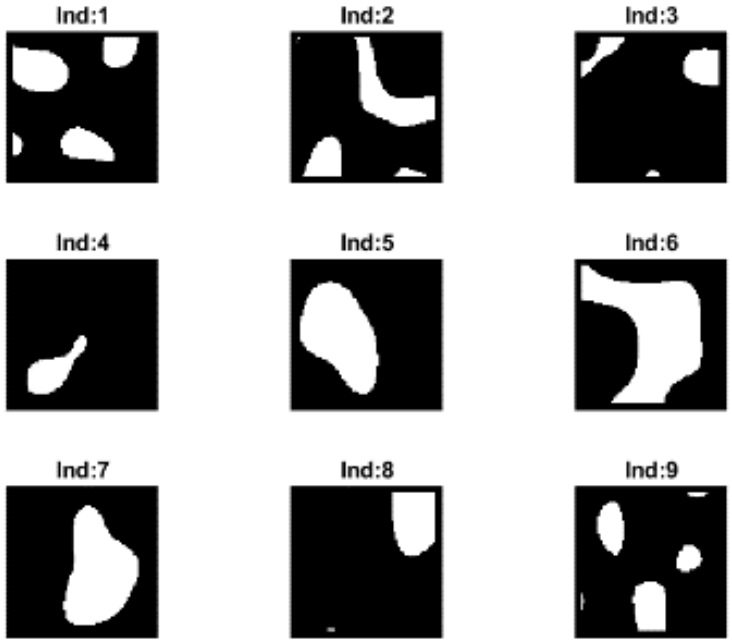

Figure 4.13: Nine depictions of metamaterial patterns generated with 5x5 Bezier control points and with no symmetry.

Chapter 5

Neural Network Implementation and Results

Based on the discussion in Ch. 3, various neural network considerations can be implemented for different data encodings.

5.1 Metric for Error

There are many ways to characterize the error of a neural network, and there are different metrics used for classification and regression problems. Root mean square error (RMSE) is often used for regression problems, and it is the square root of the sum of the squared errors. RMSE essentially encapsulates a square factor into its computation of error. The biggest problem with only using RMSE as a metric, at least for this work, is that RMSE changes when the number of output regression targets. A network trying to predict a single number and a network trying to predict 32 numbers cannot intuitively be compared through RMSE.

This work focuses on two different metrics: the mean of the mean errors and the maximum mean error. The mean of the mean errors, though confusingly named, is simply the mean for each different predicted output variable on each test data, and then the mean of those means. The maximum mean error is simply the maximum of the means of each predicted output variable. While these metrics also do not allow direct comparison between a network with single output variable and one with 32 outputs, it was found to be a more easily understood metric for comparing different output networks.

5.2 Fully Connected Neural Network Model

The original, and in some ways simplest, neural network architecture is the fully connected architecture. Every node from a given layer is fed into every node of the next layer. Every function can be approximated by a sufficiently large set of hidden layers. However, training for arbitrarily complex functions becomes difficult and unreliable. This is the major drawback of fully connected networks. In theory, a fully connected layer will be able to solve any problem, but in practice, this theoretical ideal network is not easily discovered.

The fully connected network is often tried first for a given machine learning problem. For this problem, fully connected networks were thoroughly explored and ultimately provided the best accuracy.

The modeling problem is to predict 32 outputs for the metasurface pattern's electric and magnetic reflection and transmission for each polarization.

Table 5.1 lists the exact training accuracies on the various metamaterial problems with Bezier control points encodings. The combinations listed are the 3x3 and 5x5 complexity patterns each with the constraints of four-fold symmetry, two-fold symmetry, and no symmetry. The errors given are shown as the discussed mean of the mean errors.

Table 5.1: Mean of the mean errors for 32 output predictions for the fully connected neural network architecture.

	3x3	5x5
4-Fold Symmetric	1.70%	3.72%
2-Fold Symmetric	2.01%	5.26%
Arbitrary/Non-symmetric	2.56%	9.19%

Table 5.2 shows the same networks with same combinations of pattern complexity and symmetry but with the metric of the maximum of the mean errors.

Table 5.2: Maximum of the mean errors for 32 output predictions for the fully connected neural network architecture.

	3x3	5x5
4-Fold Symmetric	5.43%	16.7%
2-Fold Symmetric	6.07%	24.9%
Arbitrary/Non-symmetric	7.87%	27.7%

5.3 Convolutional Neural Network Model

Given their surge in popularity over the last several years, CNNs seemed like good candidates for an application that can be easily encoded as an image, and where

spatial relations are crucial. Many architectures were attempted, but the convolutaional neural networks were unable to achieve a similar performance as the fully connected methods.

On the problem of the 3x3 Bezier control point encoding of the metamaterial patterns, a mean of the mean averages of 6.77% and a maximum of the means of 27.5% was achieved. The other problems were also explored but the CNNs reliably had significantly worse than the fully connected architectures for the same problem.

5.4 Comparison of Fully Connected and CNN

For every tested problem, the best performance came from neural networks with a fully connected architecture compared to a convolutional architecture. There are countless additional convolutional methods that could be explored and compared, but many types of networks were considered. Many depths and widths of CNNs were tested, as well as transfer learning from popular CNN architectures. However, there are some important and inherent limitations to convolutional neural networks for an application like this.

There is electromagnetic intuition that points to a few possible problems with CNNs for this application. The primary limitation comes from the kernel size. A kernel size that is, for example, 11x11 pixels applied to a binary metamaterial image that 100x100 pixels will inherently prohibit any learning or understanding of interactions that happen between antenna components outside of the kernel size.

From an electromagnetic understanding of antenna elements, a metamaterial antenna cannot be understood as the combination of local interactions. CNNs are greatly limited in their abilities to capture relationships that happen at scales much larger than their kernel. Perhaps, the kernel sizes could be greatly increased, but this dramatically reduces the computational speed that is a major benefit of convolutional neural networks. As the kernel approaches the size of the images being modeled, the CNNs effectively become very similar to fully connected layers.

Perhaps there are hybrid methods that could employ CNN layers in parallel with fully connected layers, or employ skip-connections, to improve the performance of the CNN methods for this problem, but the highly resonant and non-localizable aspects of this problem will inherently cause modeling difficulty for CNNs.

5.5 Hyperparameters

While no theoretically provable recommendations for hyperparameters can be easily determined, discussion of the empirical hyperparameters can be useful for applying to similar problems. This section will discuss some of the empirically determined better hyperparameters and considerations of the hyperparameters.

5.5.1 Number and Size of Layers

For both the fully connected and convolutional neural network architectures, many combinations of the number and size of layers were tested. In general, the method

for determining the best combination of the number and size of layers was to start small and slowly grow. Once growing the number and size of layers showed worsening error, then the number and size of layers is finalized. This is certainly not a perfect method, and there is significant debate in the wider field of machine learning, but there are only heuristic methods for determining the ideal network architecture [22].

Typically, the fully connected networks showed worsening once 10 or more layers were added. For the fully connected network, layer sizes were also heustrically determined. Some of the tested architectures used a cascading layer size, going from very large layers and progressively shrinking, while others maintained a consistent layer size. For cascading architectures, an initial hidden layer of 10,000 nodes seemed to be the approximate point of no returns. For architectures with constant layer size, 1000 to 2000 nodes was often the point with no returns.

5.5.2 Dataset Size

Since data is able to be generated relatively quickly, about 50,000 patterns in 12 hours, the training is not dataset-size limited. However, it is still worth exploring the necessary dataset size. The point of diminishing returns--more data does not seem to drastically help--happened around 50,000 to 100,000. This is the approximate elbow point in the Figure 5.1 for the 3x3 = 9 control-point problem. It is likely more complex patterns would require significantly more training data.

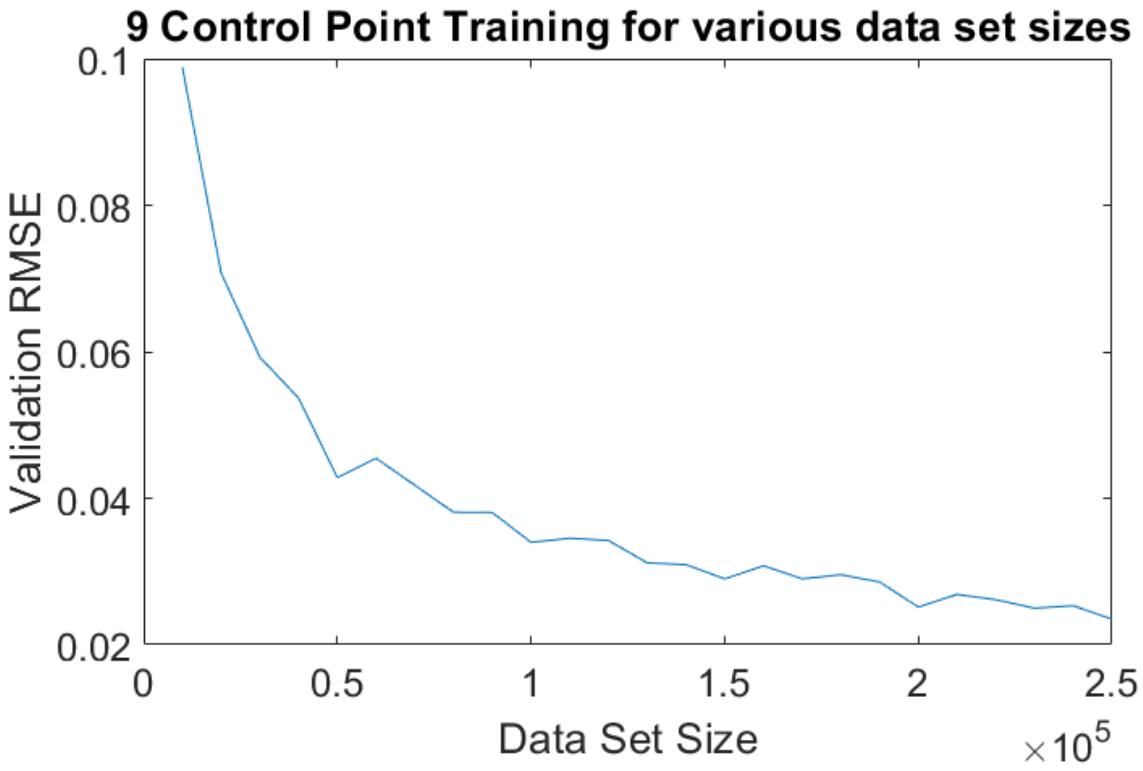

Figure 5.1: Plot of the data set size versus the validation RMSE.

Chapter 6

Extensions of Method

In addition to the neural network method developed, two extensions of the method were explored as ways of improving the method and extending its ability to be applied to meaningful problem. The extension for improving the method used CMA-ES as an optimization tool for determining better hyperparameters. The extension for applying the method to meaningful problems used neural network inversion for generating metamaterial patterns with desired electromagnetic properties.

6.1 CMA-ES Optimizing the optimization

CMA-ES, or covariance matrix adaptation for multi-objective optimization, is a method for optimizing several dependent variables [24]. The method utilizes an evolutionary strategy and the covariance of matrices representing the objective variables to explore the solution space of an objective function. Often, CMA-ES is an effective and efficient optimization for multi-objective problems.

In the context of this paper, the multi-objective problem needing to be solved was the hyperparameters of a fully connected network. Other work has performed a similar optimization of neural network parameters using CMA-ES [25].

The three hyperparameters being optimized were the learning rate, the size of each layer, and the number of layers. Each hidden layer of the neural network was constrained to be the same size.

CMA-ES was used in place of a random or linearly-spaced sweep of hyperparameters, which is often done but is computationally intensive [22].

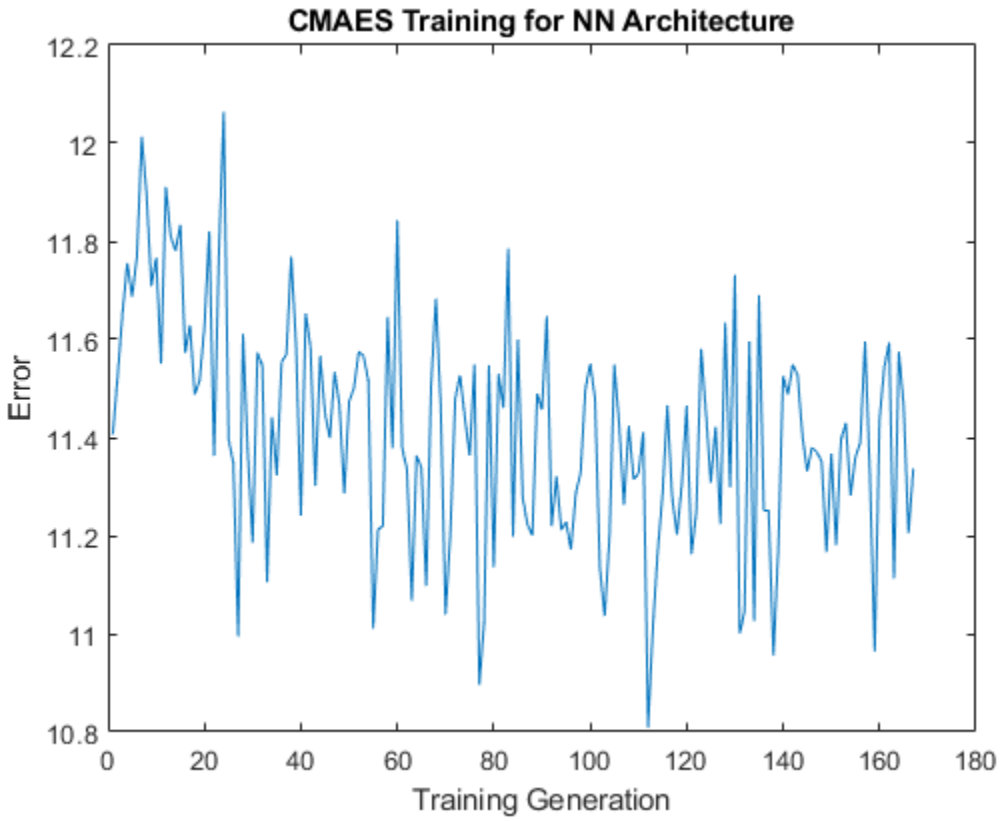

Figure 5.2: Plot of the average mean error throughout 170 generations of CMA-ES optimizing the neural network's hyperparameters of learning rate, layer size, and number of layers.

Figure 5.2 shows the non-smooth convergence over the CMA-ES generations. The first several generations show a quick decrease in error. However, the last many generations do not exhibit improved learning. The unrepeatable nature of neural network training caused by random initializations make it a very difficult multi-objective optimization problem.

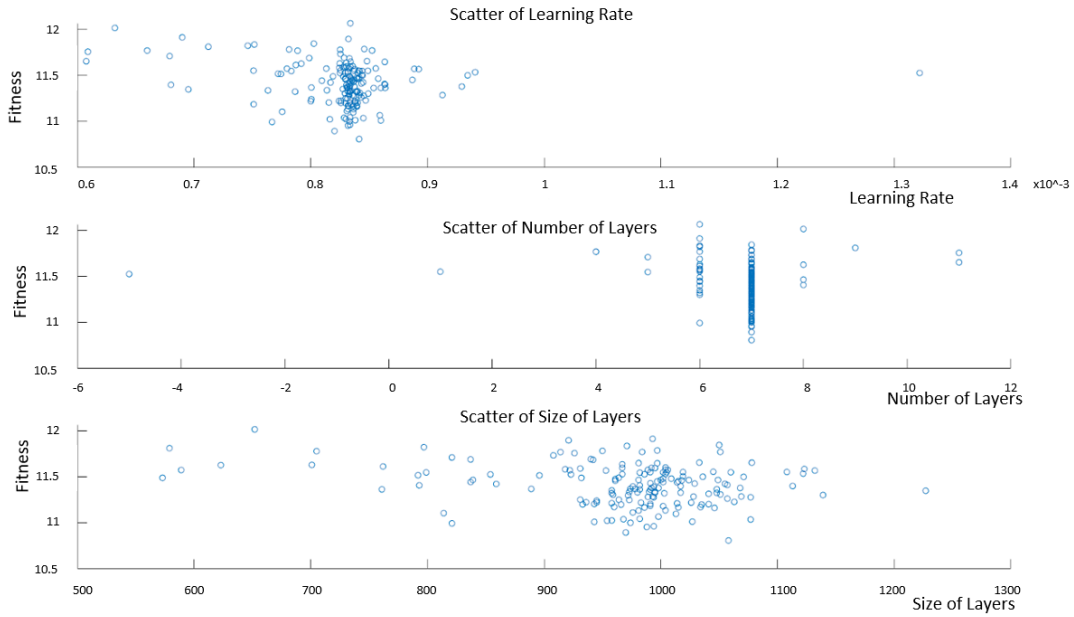

Figure 5.3: Three scatterplots. One scatterplot for each of the parameters being optimized simultaneously during 170 iterations of CMA-ES on the neural network hyperparameters. The top plot shows the learning rate. The middle plot shows the number of layers. The bottom plot shows the size of the layers.

Despite the unreliable convergence, this CMA-ES optimization does show valuable information when viewed by the individual optimization variables. As shown in figure 5.3, the number of layers of the network showed clear convergence to 7 layers. The learning rate showed convergence near 0.8 x 10^-3, but its convergence was less clear. And the size of the layers showed the least definitive convergence. All of these observations help inform the significance of these hyperparameters in influencing the neural network accuracy.

6.2 Inverse Neural Network Method for Direct Pattern Generation

This inverse neural network method aims to use a trained neural network and gradient descent to generate metamaterial patterns with desired electromagnetic properties. This paper employs a very similar method of backpropagating error gradients into the input layer as described in this work [26].

6.2.1 Motivation

Once a neural network is trained, there are various methods for inverting the neural network to solve the inverse problem. For the work in this thesis, the main problem--or the forward problem--is to input a metamaterial pattern and accurately generate the electromagnetic transmission and reflection. However, the inverse problem is, in many ways, even more useful. Ideally, there would be a method for inputting the desired electromagnetic properties and outputting an appropriate metamaterial pattern.

This method of neural network inversion attempts to take the neural network trained on the forward problem and use it to generate metamaterial patterns with the desired properties.

6.2.2 Pseudocode of Method

The method uses backpropagation and gradient descent to iteratively change a random shape to one with the desired properties.

To invert the trained neural network:

1. Guess a random metamaterial shape and set it as the input layer of the trained network
2. Forward propagate this random guess through the network
3. Find the output error based on compared to the desired transmission properties
4. Backpropagate the error gradients through the network and all the way into the input layer
5. Update the random guess in the input layer based on this error gradient multiplied by a learning rate
6. Repeat steps 2-5, iteratively improving the original guess until a defined stopping point is reached

6.2.3 Results of Inverse Neural Network Method

For evaluating this method, the neural network was limited to only a single polarization of the transmission of the electric field, represented as a real number and an

imaginary number. This allows 2-dimensional plots to represent the output space of the neural network.

There are several types of figures generated to represent this method. All types are similar but were generated with slightly different goals. All figures are plotted on the real and imaginary axes of the neural network output space. They also All have several thousand red points depicting random input combinations projected in the neural network output space. This helps depict the possible and likely outputs of the neural network given the constraints of the input.

Figure 6.1 shows the inverse method attempting to start at one random point and find solutions to 50 different targets. This was the most ambitious, and most useful, application of the method, but it ultimately fails to converge to the majority of the target outputs. The targets that are closest to the starting in output space are not necessarily the points closest input space--which input space is the actual space being changed--but targets close in output space tend to have the best success in convergence.

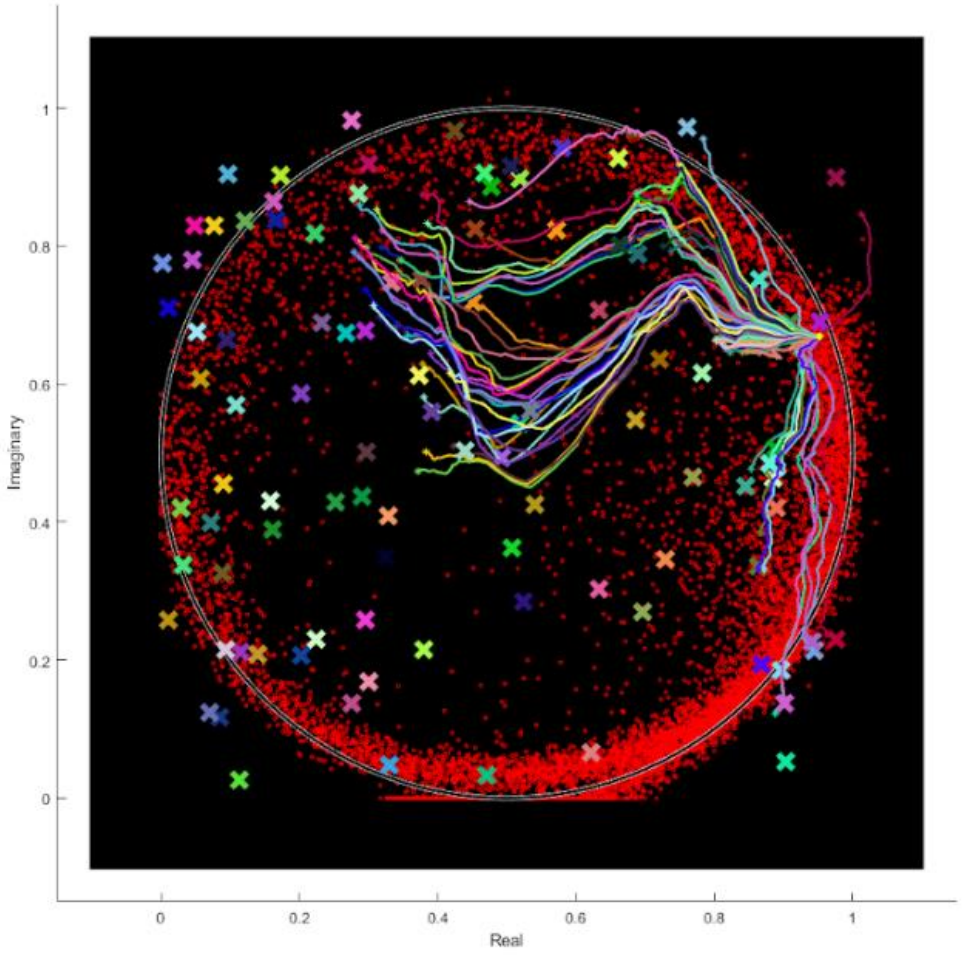

Figure 6.1: Plot of the output space encoded as real and imaginary numbers. The X marks are targets and the corresponding trail of the same color shows the inversion method's attempt to reach each X. The red dots show several thousand random input points mapping out the neural network in output space.

After performing several more similar runs, some performed appreciably better than others. The run in figure 6.2 seems to have found a circular path that allowed gradient to reach most of the target Xs. This suggests the random starting point has a significant impact on convergence.

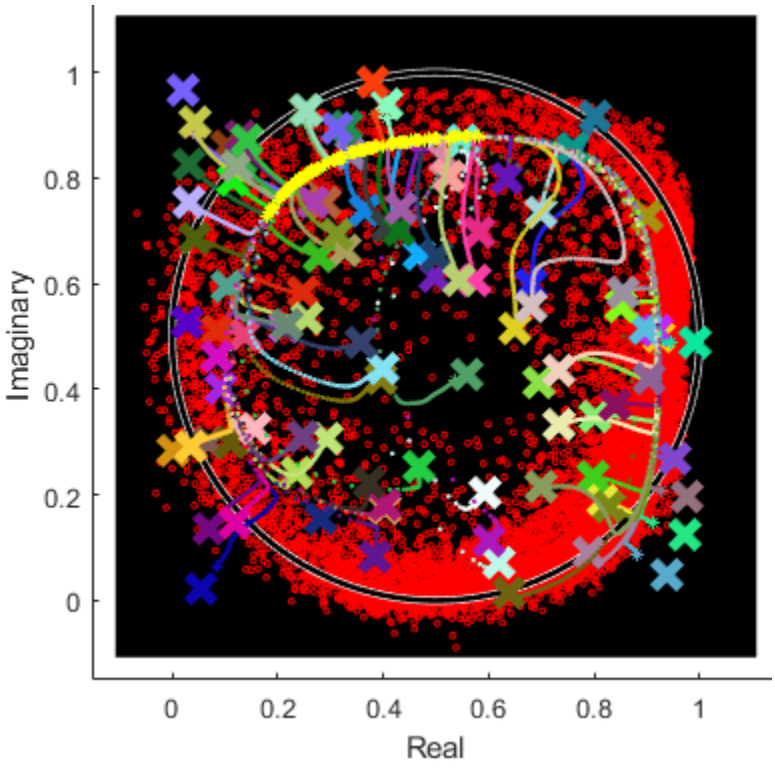

Figure 6.2: Plot of the output space encoded as real and imaginary numbers. The X marks are targets and the corresponding trail of the same color shows the inversion method's attempt to reach each X. The red dots show several thousand random input points mapping out the neural network in output space.

Related to the last point, figure 6.3 shows a starting position that caused the inversion method to hardly reach any of the target points. The method seems to perform

better when it starts in a position densely populated in output space. Figure 6.3 starts in a very sparse spot.

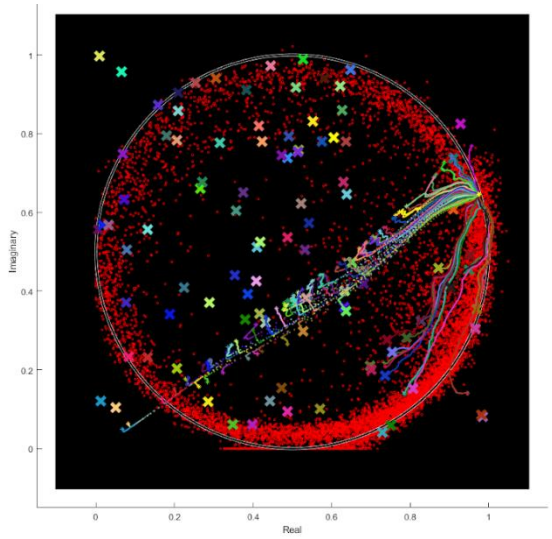

Figure 6.3: Plot of the output space encoded as real and imaginary numbers. The X marks are targets and the corresponding trail of the same color shows the inversion method's attempt to reach each X. The red dots show several thousand random input points mapping out the neural network in output space.

Figures 6.4 and 6.5 depict a trial that starts at 50 very close points in input space that all try to converge to the same output target. While very close in input space, these points also appear very close in output space. The yellow starting points try to converge to the grey X and convergence trails end with a green marker. This method ultimately had much more success than the previous method of trying to reach any point. While the majority of points were still unable to converge to the target, several points were able to reach the target output. Since only one actual point needs to reach the target to have a desirable metamaterial, then this is sufficient.

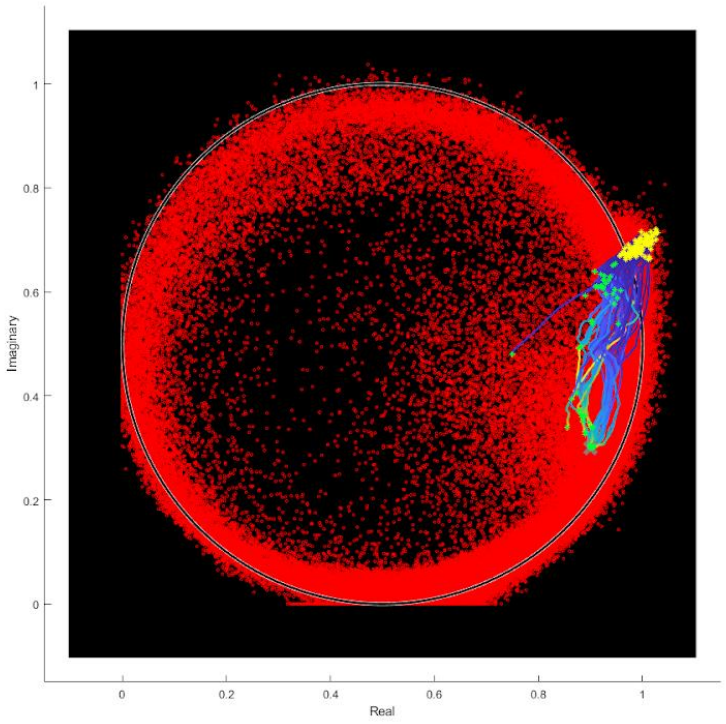

Figure 6.4: Plot of the output space encoded as real and imaginary numbers. 50 inversion iterations are started from very similar positions. Several of the paths successfully reach the target X. The red dots show several thousand random input points mapping out the neural network in output space.

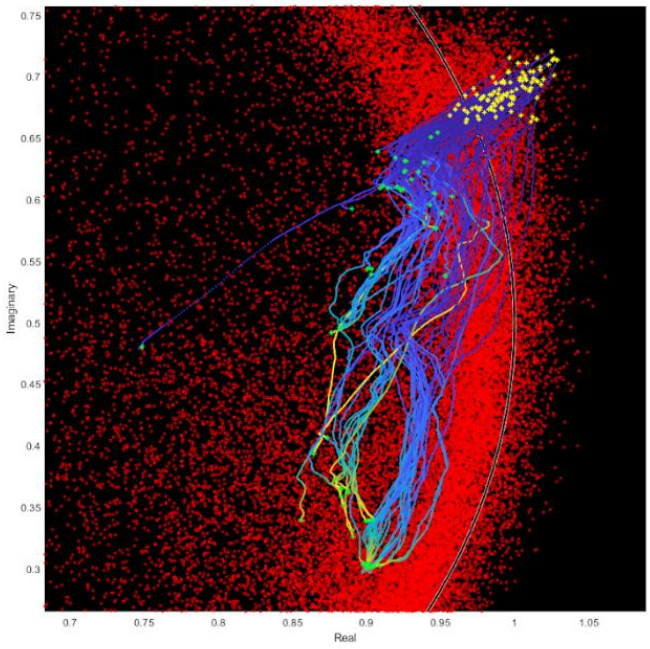

Figure 6.5: Zoomed in depiction of the previous figure. Plot of the output space encoded as real and imaginary numbers. 50 inversion iterations are started from very similar positions. Several of the paths

successfully reach the target X. The red dots show several thousand random input points mapping out the neural network in output space.

Performing several more of these trials with start points close in input space showed convergence was inconsistent. The example in figure 6.6 was intentionally chosen as appearing close in output space as the previous example but having a target slightly further away in output space. This example of trying to traverse a farther distance in output space was successful in having at least one iteration reach the target, but far fewer than the closer previous example.

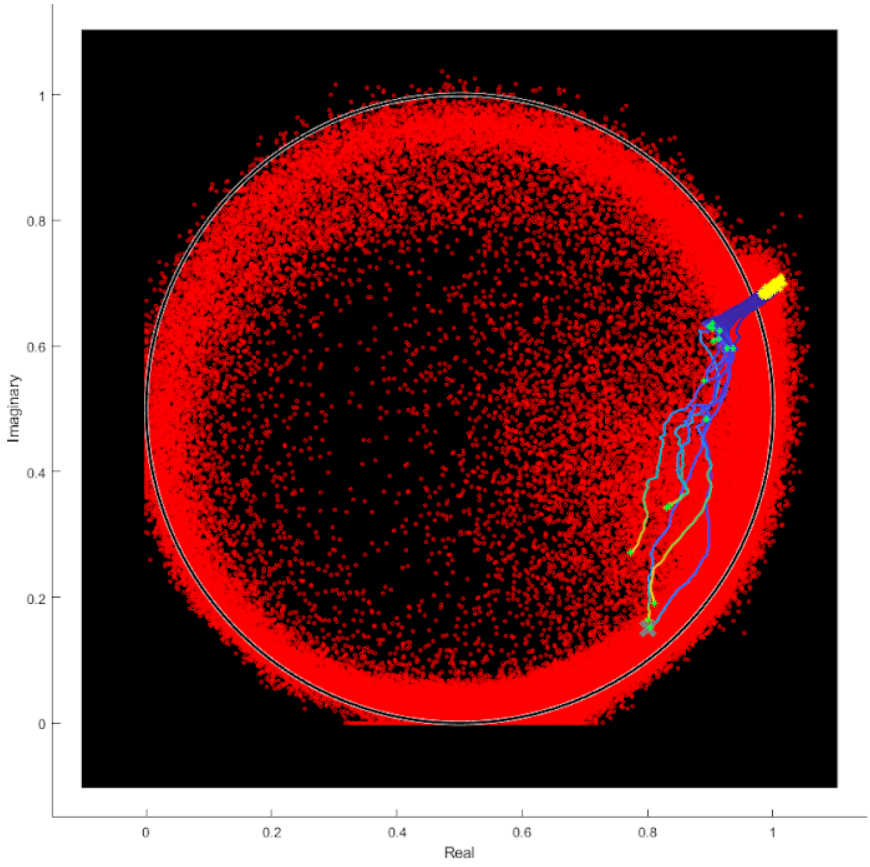

Figure 6.6: Plot of the output space encoded as real and imaginary numbers. 50 inversion iterations are started from very similar positions. Now the target path is farther from the starting point and fewer of the paths reach the target. The red dots show several thousand random input points mapping out the neural network in output space.

6.2.4 Limitations and Drawbacks

This method of inverting the neural network for direct pattern generation is susceptible to several problems. Firstly, it is inherently dependent on the accuracy of the original trained neural network. If the original neural network has an average of 1% error in its predictions, this inversion accepts that 1% error prediction to be ground truth and will never be able to perform better than the error inherent to the neural network.

Assuming a perfect underlying neural network, this inversion method is still susceptible to other drawbacks. Most of these problems are related to gradient descent and being trapped in local extrema. It is very possible a randomly chosen input point will be unable to reach the target through gradient descent. And since the target is defined in output space, there will be several points in input space matching to the target output, so gradient descent will not have a clear, single direction to follow. This is the many-to-one problem, where several possible choices map to the same target output.

This method also has no mechanism for determining if a target has a possible solution. In the forward problem, the inputs to the neural network are guaranteed to be physically realizable devices. In the inverse problem, depending on the constraints of the metamaterials and size of the exploration space in the training data, there will likely be points in the output space that are impossible to realize, and the neural network has no method for considering this possibility.

Fundamentally, this inversion method would only be useful if it is computationally quicker than optimization methods using the forward neural network. Unfortunately, a single iteration of the inversion method is unable to vectorize and

parallelize its computation to benefit from the same computationally speed-up other methods have. In the example where many trials start in similar input spaces and have the same target, the method could be parallelized, but there is still only one target being computed. The alternative forward neural network methods can simply simulate many thousand patterns and compute them in parallel, hoping one will be close to the target.

Ultimately, this neural network inversion method has unreliable convergence for the problem of randomly guessing a point in input space and converging to an arbitrary target in output space. However, there might be merit to combining this method in a hybrid technique with other methods. Perhaps, rather than using entirely random points, the forward pass of the neural network could be used to generate several thousand random points. Then the closest several random points could be fed into this inversion method to fine-tune the metamaterial patterns and their output characteristics.

Chapter 7

Conclusion and Future Work

7.1 Summary

This work explored several neural network methods for modeling electromagnetic metamaterials. This motivation for using neural network methods in computational electromagnetics is the computational speed-up enabled by the neural network algorithms and parallelization of GPU computations. Electromagnetics intuition and machine learning methods for neural networks were combined to develop a successful model of a class of metamaterials. Important factors like the exploration space and parameterization of the metamaterials, the architecture of the neural network, and required accuracy of the final model were considered for assessing the method.

Computational electromagnetics is dependent on quick algorithms, and this method used neural networks to develop a metamaterial method that was computationally quicker than previous methods. A critical caveat of this claim is that the method described requires using conventional computational electromagnetics methods to build a dataset to train the network. While this new machine learning is still dependent on the slower conventional methods, the computational speed-up for certain problems exists--there is just an economy of scale comparison that must be made. This machine learning method can be a few orders of magnitude faster if many metamaterial elements within the same class are needed to populate an array. However, if only a single metamaterial element is needed, the conventional methods can still be faster.

Some of the underlying and essential electromagnetics intuition of metamaterial devices helped inform and debug some the neural network design and training decisions. Most importantly, the shortcomings of the CNN models are most likely related to the inability for CNN kernel to model relationships that happen at spatial scales much larger than the kernel size. For the problem explored in this work, a kernel size of even 11x11pixels will struggle to capture the relationships between opposite corners of a 100x100 pixel binary image representing the metamaterial. Having kernel sizes much larger is computationally impractical. This is a major reason the convolutional neural network model likely performed worse than the fully connected model on the same problem.

The fully connected neural network was the preferred model and discussions of hyperparameters and training methods were discussed to offer insight into how machine learning can be applied to other problems within computational electromagnetics and antennas in general. While there are few theory-driven approaches to determining hyperparameters on new machine learning problems, the hyperparameters of similar work can offer a good, heuristic starting point for further research.

Additionally, two extensions of this neural network method were briefly discussed. The optimization algorithm CMA-ES was applied to a hyperparameter sweep of neural network parameters in an attempt to more quickly improve the neural network model. The method was better than random guessing, but it did not converge smoothly to ideal hyperparameters. The other additional method explored was neural network inversion for direct metamaterial pattern generation. Once a neural network is trained, there are methods that attempt to invert this neural network--to iteratively backpropagate

error weights into the input layer. Ultimately, this method showed some promise for reaching a pattern with the desired electromagnetic properties, but its convergence was unreliable. Neural network inversion is not a robust method for generating metamaterial patterns, but there are ways it can be incorporated to improve other methods.

7.2 Future Work

As mentioned above, the primary purpose of the work of this thesis was to provide a computational speed-up for analyzing metamaterials such that optimizing a device metamaterial becomes more computationally tractable. The faster the algorithms, the more complex a problem becomes solvable. The work of this thesis is intended to be applied to metamaterial optimization. This will involve thousands or millions of function calls to the neural network scheme developed by this work. Building upon previous metamaterial optimization techniques, incorporating a trained neural network with function calls up to 1000 times faster for parallelizable problems will enable a richer complexity of problems to be solved.

There are many cutting-edge neural network methods that could very immediately be applied to this work. Some of the most obvious methods are hybrid architectures, skip-connections, autoencoders, and generative adversarial methods. There are almost certainly ways to improve the performance of convolutional methods, but convolutional methods alone will still find problems in capturing spatial relationships that happen outside of their kernel size.

There are additional future work concepts to more directly improve the method developed in this thesis. One of the most important and likely successful candidates involves transfer learning. If the neural network method developed in this thesis can then use transfer learning on smaller training sets of similar but distinct classes of metamaterials, then the major computational limit of this method will be dramatically improved. There is a good chance a neural network trained on a certain type of silicon metamaterial could be trained much more quickly on a new problem on a certain metal metamaterial compared to training a new model from scratch.

This method will hopefully be applicable to problems in robustness and coupling of metamaterial elements. In these problems, this neural network method can be greatly helpful because the total number of function evaluations to simulate a metamaterial element are incredibly high. This suggests running 50,000 or 100,000 simulations using conventional computational electromagnetics methods to develop a training dataset will not be comparatively slow. Rather, once the training dataset is developed and a neural network trained, problems involving metamaterial arrays, like robustness and element coupling, can be computed several orders of magnitude more quickly.

References

[1] S. D. Campbell, D. Sell, R. P. Jenkins, E. B. Whiting, J. A. Fan, and D. H. Werner, "Review of numerical optimization techniques for meta-device design [Invited]," *Opt. Mater. Express*, vol. 9, no. 4, p. 1842, Apr. 2019, doi: 10.1364/OME.9.001842.

[2] S. D. Campbell, R. P. Jenkins, P. J. O'Connor, and D. H. Werner, "The Explosion of Artificial Intelligence in Antennas and Propagation: How Deep Learning is Advancing Our State of the Art," *to appear in Antennas and Wireless Propagation Magazine special issue on Artificial Intelligence in Electromagnetics*.

[3] D. R. Prado, J. A. López-Fernández, M. Arrebola, and G. Goussetis, "Efficient Shaped-Beam Reflectarray Design Using Machine Learning Techniques," in *2018 48th European Microwave Conference (EuMC)*, Sep. 2018, pp. 1545–1548, doi: 10.23919/EuMC.2018.8541763.

[4] H. M. Yao, W. E. I. Sha, and L. Jiang, "Two-Step Enhanced Deep Learning Approach for Electromagnetic Inverse Scattering Problems," *IEEE Antennas Wirel. Propag. Lett.*, vol. 18, no. 11, pp. 2254–2258, Nov. 2019, doi: 10.1109/LAWP.2019.2925578.

[5] L.-Y. Xiao, W. Shao, T.-L. Liang, and B.-Z. Wang, "Artificial neural network with data mining techniques for antenna design," in *2017 IEEE International Symposium on Antennas and Propagation USNC/URSI National Radio Science Meeting*, Jul. 2017, pp. 159–160, doi: 10.1109/APUSNCURSINRSM.2017.8072122.

[6] Q. Wu, H. Wang, and W. Hong, "Broadband Millimeter-Wave SIW Cavity-Backed Slot Antenna for 5G Applications Using Machine-Learning-Assisted Optimization Method," in *2019 International Workshop on Antenna Technology (iWAT)*, Mar. 2019, pp. 9–12, doi: 10.1109/IWAT.2019.8730801.

[7] C. Finn, P. Christiano, P. Abbeel, and S. Levine, "A Connection between Generative Adversarial Networks, Inverse Reinforcement Learning, and Energy-Based Models," *ArXiv161103852 Cs*, Nov. 2016, Accessed: Mar. 29, 2020. [Online]. Available: http://arxiv.org/abs/1611.03852.

[8] D.-J. Yun, I. I. Jung, H. Jung, H. Kang, W.-Y. Yang, and I. Y. Park, "Improvement in Computation Time of the Finite Multipole Method by Using K-Means Clustering," *IEEE Antennas Wirel. Propag. Lett.*, vol. 18, no. 9, pp. 1814–1817, Sep. 2019, doi: 10.1109/LAWP.2019.2930674.

[9] S. An *et al.*, "A Deep Learning Approach for Objective-Driven All-Dielectric Metasurface Design," *ACS Photonics*, vol. 6, no. 12, pp. 3196–3207, Dec. 2019, doi: 10.1021/acsphotonics.9b00966.

[10] J. Jiang and J. A. Fan, "Simulator-based training of generative models for the inverse design of metasurfaces," *Nanophotonics*, vol. 0, no. 0, Nov. 2019, doi: 10.1515/nanoph-2019-0330.

[11] G. Cybenko, "Approximation by Superpositions of a Sigmoidal Function," *Control Signal Syst.*, no. vol. 2, 4, pp. 303–314, Dec. 1989.

[12] M. Nielsen, "Neural Networks and Deep Learning," *Determ. Press*, 2015.

[13] Y. LeCun, "Efficient Backpropagation," *Springer*, 1998, Accessed: Mar. 29, 2020. [Online]. Available: http://yann.lecun.com/exdb/publis/pdf/lecun-98b.pdf.

[14] C. Szegedy *et al.*, “Going Deeper with Convolutions,” *ArXiv14094842 Cs*, Sep. 2014, Accessed: Mar. 29, 2020. [Online]. Available: http://arxiv.org/abs/1409.4842.

[15] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “ImageNet Classification with Deep Convolutional Neural Networks,” in *Advances in Neural Information Processing Systems 25*, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2012, pp. 1097–1105.

[16] A. Cleeremans, D. Servan-Schreiber, and M. J, “Finite State Automata and Simple Recurrent Networks,” *Neural Comput*, vol. 1, no. 3, pp. 372–381, Sep. 1989.

[17] S. Hochreiter and J. Schmidhuber, “Long Short-Term Memory,” *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, 1997.

[18] I. J. Goodfellow *et al.*, “Generative Adversarial Networks,” *ArXiv14062661 Cs Stat*, Jun. 2014, Accessed: Mar. 29, 2020. [Online]. Available: http://arxiv.org/abs/1406.2661.

[19] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson, “How transferable are features in deep neural networks?,” in *Advances in Neural Information Processing Systems 27*, Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2014, pp. 3320–3328.

[20] S. Ioffe and C. Szegedy, “Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift,” p. 9.

[21] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, “Dropout: A Simple Way to Prevent Neural Networks from Overfitting,” p. 30.

[22] A. Ng, “Deep Learning,” *Coursera*. https://www.coursera.org/specializations/deep-learning (accessed Jan. 26, 2020).

[23] J. P. Hugonin and P. Lalanne, *Reticolo Software for Grating Analysis*. Institut d’Optique, 2005.

[24] C. Igel, N. Hansen, and S. Roth, “Covariance Matrix Adaptation for Multi-objective Optimization,” *Evol. Comput.*, vol. 15, no. 1, pp. 1–28, Mar. 2007, doi: 10.1162/evco.2007.15.1.1.

[25] I. Loshchilov and F. Hutter, “CMA-ES for Hyperparameter Optimization of Deep Neural Networks,” *ArXiv160407269 Cs*, Apr. 2016, Accessed: Mar. 29, 2020. [Online]. Available: http://arxiv.org/abs/1604.07269.

[26] C. A. Jensen *et al.*, “Inversion of Feedforward Neural Networks: Algorithms And Applications,” in *Proc. IEEE*, 1999, pp. 1536–1549.

ACADEMIC VITA

Philip O'Connor

pjo5097@psu.edu

EDUCATION:

The Pennsylvania State University, Schreyer Honors College
Electrical Engineering – BS/MS – Expected May 2020

- Five-year Dual Degree for Bachelors and Masters – Integrated Undergraduate-Graduate (IUG) Program
- Relevant Coursework in: C++, MATLAB, Computer Vision, Machine Learning, Signal Processing

Computational Science Minor – Focusing on Machine Learning and Statistics
English Minor – Focusing on the interplay of Technical Writing and Creative Writing

PROFESSIONAL EXPERIENCE:

The Boeing Company – **Internship –** Seattle, WA — Summer 2018 – 11 weeks
Electromagnetics Effects Intern for the Presidential Aircraft Recapitalization and KC-46 Tanker Programs

- Performed analysis on antennas to determine distances for FAA Hazardous Effects of Radiation to Personnel
- Led the troubleshooting and rework for several flightline aircraft manufacturing defects in wiring
- Supported aircraft-level tests for electromagnetic hardening against precipitation-static interference

The NASA Jet Propulsion Laboratory – **Internship –** Pasadena, CA — Spring 2018 – 16 weeks
Researcher for project on **Magneto-Quasistatic Nuclear Magnetic Resonance (long-range MRI)**

- Explored near-field magnetic phenomena for applications in imaging, positioning, and communications
- Led the hardware implementation for a prototype desktop nuclear magnetic resonance device
- Transferred related research in through-the-wall magnetic field techniques to nuclear magnetic resonance

IBM – **Internship** – Poughkeepsie, NY — Summer 2017 – 12 weeks
Engineering Intern for Designing and Implementing Thermal Qualification Equipment in LabVIEW

- Utilized LabVIEW to measure the heat dissipation and the pump reliability for the Z-System Mainframe
- Performed long-term tests with pressure transducers, flowmeters, and thermocouples with Agilent DAQs
- Contacted suppliers to determine equipment specifications and budget

ACADEMIC EXPERIENCE:

Computational Electromagnetics and Antennas Research Laboratory – Dr. Douglas Werner — 2016 – Present
Ongoing Thesis Research: — Full-time: 2019 – Present

- Developed and trained deep neural network architecture for simulating meta-surface antenna elements
- Encoded antenna elements into 2D images for over 1000x speed increase over state-of-the-art full-wave EM solvers
- Created iterative tolerance measurements and optimization made possible through neural network speed increase

Past Projects: — Part-time: 2016 – 2018

- Simulated gold nano-loop antennas (~100nm radius) to analyze unique plasmon-induced directive properties
- Explored applications of metamaterial antennas for wearable medical devices

Teaching Intern for Electrical Engineering Design Process — Summer 2019 – Present

- Gained experience teaching lectures, creating assignments, and holding office hours

Class Projects — 2015 – 2019

- Computer Vision – MATLAB – Video object tracking, machine learning image classification, stereo depth estimation
- Remote Sensing – MATLAB – Single-rotational-axis model satellite using momentum wheel and photo star-tracker
- Antenna Engineering – FEKO/MATLAB – Phased Array Antenna design using randomization to prevent grating lobes
- EE Design – LabVIEW/breadboard – robot mouse navigation, optical theremin, karaoke circuit
- Object-Oriented Programming – C++ – Chess program, scheduling system

RELATED EXPERIENCE:

Personal Projects

- Checkers AI based on genetic algorithm neural network in MATLAB — Ongoing
- Stock trading algorithm based on modular LSTM neural networks in MATLAB — Ongoing
- Software-defined radio reception of NOAA weather satellite broadcast — 2018
- Independently developed the smoothing filter to win 8th grade science fair by classifying images of sand — 2011

Mentoring

- IEEE Mentor – One-on-one mentoring for 3 underclassmen in electrical engineering 2018
- Electrical Engineering Envoy – Provided tours of the EE department for prospective students 2018 – 2019
- Engineering Orientation Network – Head Mentor – Led 10 students in mentoring 200 incoming freshmen 2015 – 2017

Granted Patent entitled **"Reverberation-Induced Magnetic Field Alteration to Enhance Sound"** 2014 – 2017

- Developed an innovative method for coating the resonant wood of a musical instrument with magnetic material
- Enabled an electromagnetic pick-up to interact directly with the magnetized wood of a guitar

Lunar Lion – Penn State student-led **Rocket Club –** Guidance, Navigation, and Control 2015 – 2016

- Designed camera vision software to detect craters during a lunar landing in Python through GitHub
- Created a 3D simulation of a lunar landing using Blender - CAD Animation

---

PERSONAL ACTIVITIES: Backpacking, Video Game Programming, Book Club, Ukulele Club, Intermural Sports