

THESIS FOR THE DEGREE OF DOCTOR OF PHILOSOPHY

Machine learning for the free-form inverse design in nanophotonics

TIMO GAHLMANN

Division of Condensed Matter and Materials Theory
Department of Physics
Chalmers University of Technology
Gothenburg, Sweden, 2023

Machine learning for the free-form inverse design in nanophotonics

TIMO GAHLMANN

Copyright ©2023 Timo Gahlmann

All rights reserved.

ISBN 978-91-7905-934-7

Doktorsavhandlingar vid Chalmers tekniska högskola, Ny serie nr 5400.

ISSN 0346-718X

Department of Physics
Division of Condensed Matter and Materials Theory
Chalmers University of Technology
Gothenburg, Sweden

Cover image courtesy of Erik Strandberg

Printed by Chalmers Digitaltryck,
Gothenburg, Sweden, 2023.

Abstract

In the fast developing field of nanotechnology, nanophotonics has emerged as a revolutionary technology, controlling the complex interplay between light and matter at the nanoscale. Crucial to the advancement of this field is the ability to manipulate the propagation of electromagnetic waves by shaping the transmitted or reflected wavefront through the design of structures that are only a fraction of the wavelength in size. The design of these subwavelength structures forming a quasicontinuous material is complex and usually exceeds human intuition.

In this thesis, we present our approach utilizing deep neural networks for both the prediction of optical properties of nanostructures and their inverse design. First, we demonstrate the feasibility of employing neural networks to accurately predict the optical properties of nanostructured materials, with a particular emphasis on metasurfaces. Given the increasing need for miniaturization together with smaller optical losses, the emergence of metasurfaces—single layers of phase-modifying nanostructures—has heralded a significant advancement, enabling unprecedented manipulation of light beyond the capabilities of traditional materials.

Subsequently, we present methodologies to achieve inverse design of metasurfaces, guided by specific desired optical attributes and fabrication constraints. This is achieved through neural network design and the generation of training data, labeled with corresponding optical characteristics and manufacturability constraints. We implemented a conditional generative adversarial network comprised of five synergistic neural networks. This approach effectively mitigates challenges related to design non-uniqueness, mode collapse, and experimental feasibility. Additionally, our research explores various techniques to optimize and stabilize neural network training and introduces novel network graph compositions, contributing to a versatile generator model capable of conceiving multiple metasurface unit cells with predefined, interconnected properties. This thesis thus provides insights and methodologies for inverse design, contributing to the continued evolution of nanophotonic design and applications.

Acknowledgment

I would like to express my deepest gratitude to my supervisor, Professor Philippe Tassin, for his invaluable guidance throughout my PhD journey. His willingness to spend a lot of time to help me with tedious details in the early stages of my research and his encouragement to pursue even the most ridiculous ideas have been instrumental to my growth as a researcher. His mentorship has been a cornerstone of my PhD experience, and for that, I am profoundly thankful.

Moving to Sweden, experiencing the culture and living through a very different covid response compared to the rest of the world was a very interesting experience. The darkness in winter will always be a deterrent, but the rest of my experiences here were very pleasant.

My sincere thanks go to Erik, whose collaboration was not only intellectually rewarding but also brought enjoyment to my work, beautifully reflected in the figures he created for this thesis. Our squash matches provided a much-needed respite from the rigors of research. Jonas also provided multiple figures, but more importantly I want to thank him for being my longest friend. I am grateful to Tobias, whose perspective and problem set inspired the network graph in paper C and facilitated a connection with JPL that has been invaluable.

I owe a debt of gratitude to Andreas and, in particular, Jari, for their patience and meticulous attention to detail in reviewing my writing. Their efforts have greatly enhanced the clarity and readability of this thesis.

A special thank you to Cindy, who shared countless lunch breaks with me, offering both companionship and support during the most demanding periods of my research. The constructive feedback from Albin and Hjalmar has been crucial in refining my research direction and methodology.

I am fortunate to have a circle of friends who have provided support throughout this journey. I want to thank Jonas, Paul, Amr, Christian, Fabian, David, Yoshi, Kamil and Pinky for listening and giving warmth and useful feedback. Eliott deserves a special mention for being there in times of need, and Chrischi, who spend uncounted hours with me over teamspeak flying rocket cars and scoring goals, which brought joy and beautiful imbalance to my life.

I must acknowledge the unconditional love and support of my parents, whose belief in my abilities has never weakened. Their encouragement has been a source of strength and motivation.

Last, but certainly not least, I extend my heartfelt thanks to Sofia, whose patience, kindness, and unwavering support have been my anchor in the last two years. Her presence has been a constant reminder of the world beyond

academia, and for that, I am eternally grateful.

We acknowledge support from Chalmers' Excellence Initiative Nano and from the Swedish Research Council under Grant No. 2020-05284. The training data generation and ANN training were performed on resources provided by the Swedish National Infrastructure for Computing (NAISS), at the Chalmers/C3Se and KTH/PDC sites, partially funded by the Swedish Research Council under Grant No. 2022-06725.

Timo Gahlmann

Göteborg, 1 December 2023

List of Publications

Appended publications

This thesis is partially based on the following publications:

- [**Paper A**] **T. Gahlmann** and P. Tassin, *Deep neural networks for the prediction of the optical properties and the free-form inverse design of metamaterials*
Physical Review B 106, 085408 (2022).
- [**Paper B**] **T. Gahlmann** and P. Tassin, *Machine learning techniques in the inverse design of physical devices*
Under review.
- [**Paper C**] **T. Gahlmann**, T. Wenger, and P. Tassin, *Machine-learning-based inverse design of wideband metasurfaces with interdependent unit cells*
Manuscript.

Contribution statement

My contributions to this thesis lie in the development of algorithms, artificial neural networks, and training loops that are presented in this thesis. Specifically, I wrote the algorithms and code that generate the grids symbolizing the unit cell designs. I automated the COMSOL simulations on CPU clusters for training data set creation, wrote the code that labels the unit cells based on their respective aspect ratios, and designed the training loops and architectures of all the networks used in this thesis.

Initially, my work was mostly focused on developing networks that analyze the unit cell designs with respect to their optical properties and experimental feasibility. The foundational methodology of the main part of the thesis is the standard Conditional Generative Adversarial Network (CGAN) framework. I conceived various modifications enhancing this established paradigm, e.g., integration of pre-trained models, alterations in the architecture of the networks, the construction of computational graphs (Paper A). Additionally, I experimented with numerous modifications to the training loop and evaluated the efficacy of these modifications by conducting comprehensive empirical studies (Paper B).

In a collaborative project focusing on Vertical Cavity Surface Emitting Lasers (VCSELs) with the MC2 team, I took on a supervisory role for two master's theses. Leveraging the foundational framework I developed, the master's thesis students were able to conceptualize and implement their own algorithms for the generation, simulation, and labeling of unit cell designs. I then used my framework to train multiple generators and wrote a straightforward script, enabling an experimental physicist to design unit cells with ease.

In another collaboration with Jet Propulsion Laboratory, aimed at developing unit cells with interconnected optical properties for a Zernike wavefront sensor (ZWFS), I conceived two new network graph compositions to facilitate the creation of such complex designs (Paper C).

Regarding the written documentation, my contributions evolved over time. For the first paper, my involvement was limited to crafting the initial draft and helping with the revisions. For the subsequent papers and this thesis, I wrote the initial draft and led the revision of the manuscripts until completion.

“We will become AI. AI will become us.”
- Lex Fridman

Contents

Abstract	iii
Acknowledgement	v
List of Publications	vii
Personal Contribution	ix
1 Introduction	1
1.1 Optics	1
1.2 Nanophotonics	2
1.3 Inverse design	5
1.3.1 Inverse design in nanophotonics	6
1.3.2 Methods for inverse design	7
1.4 Machine learning and neural networks	9
1.4.1 Biological neural networks	9
1.4.2 Artificial neural networks	11
1.5 Outline of the thesis	14
2 Artificial Neural Networks	15
2.1 Gradient descent and backpropagation	16
2.2 Convolutional layers	20
2.3 Augmentation	21
2.3.1 Noise	22
2.4 Additional features	22
2.5 A simple classification problem	23
3 Data Representation, Creation, and Simulations	27
3.1 Representation of the input data	27
3.2 COMSOL simulations	29
3.2.1 S parameters	30
3.3 Representation of the output data	31
4 Expert/Advisor Networks	33
4.1 Forward model	33
4.2 Classifier model	35
4.3 Training optimization techniques	38
4.4 Individual pixel error	40

5	Conditional Generative Adversarial Networks	43
5.1	Generative adversarial networks	44
5.2	Conditional GAN	46
5.2.1	Improved CGAN	47
5.2.2	Inputs	48
5.2.3	Encoder	54
5.3	Training optimization techniques	54
5.3.1	Discriminator	54
5.3.2	Generator	60
5.4	Generator with multiple interdependent outputs	63
5.5	Iterative improvement of training data	65
6	Complexity of topological optimization methods	67
6.1	Computational resources	68
6.2	Wall-clock time	69
6.3	Additional remarks	69
7	Application Examples	73
7.1	Meta-atom design	73
7.2	Cylindrical metalens	75
7.3	Metasurface for simultaneous polarization conversion and focusing	77
7.4	Metasurface for a Zernike wave front sensor	79
8	Conclusions	83
8.1	Outlook	84
	Bibliography	87
	Appendix	101
	A Paper A	101
	B Paper A: Supplemental Material	111
	C Paper B	123
	D Paper C	153

Chapter 1

Introduction

1.1 Optics

In modern technology, optical devices have become ubiquitous — from everyday essentials such as cameras, telescopes, and optical fibers to sophisticated instruments like interferometers and photon detectors, optical technologies permeate virtually every aspect of human life. Crucial to the advancement of traditional optical devices is the ability to manipulate light through reflection, refraction, and dispersion. Conventionally, this has been achieved by shaping materials with varying refractive indices into lenses, mirrors, and other optical components such as beam splitters, waveplates, and polarizers.

The manipulation of light has been extensively studied through multiple theoretical frameworks. Geometrical optics, also known as ray optics, offers a simplified yet robust description of light propagation by conceptualizing light as a collection of rays that traverse along straight trajectories. The behavior of these rays is dictated by the laws of reflection and refraction when encountering interfaces between media with different refractive indices. While geometrical optics is an invaluable tool for understanding a wide range of optical phenomena, it falls short in accounting for phenomena such as interference and diffraction. In contrast, the paradigm of physical optics treats light as a wave phenomenon, thereby providing a more comprehensive model for its behavior. This wave-based approach yields predictive power for interference and diffraction of light, which remain unexplained by geometrical optics.

For example, the way a prism bends an incoming light beam can be described by Snell's Law

$$n_1 \sin \theta_1 = n_2 \sin \theta_2, \tag{1.1}$$

where n_1 and n_2 are the refractive indices of the prism and the surrounding medium, θ_1 and θ_2 are the angles between the normal (to the interface) and the incident and refracted waves, respectively. This principle is visualized in figure 1.1.

In the context of physical optics, it is useful to look at the wave fronts, which are the planes of constant phase of a light beam. The wave fronts are orthogonal to the wave vector, which determines the direction of propagation. When light travels through a material, its speed is given by $\frac{c}{n}$, where c is the speed of light in vacuum. If this material has varying thickness, like a prism,

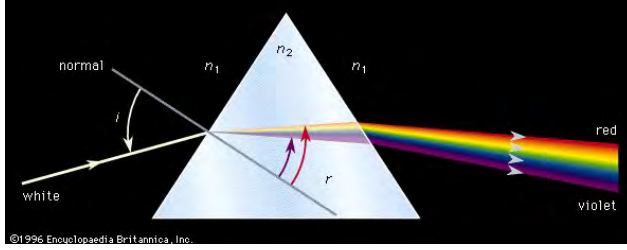


Figure 1.1: Light propagating through a prism. The incident white ray separates into constituent colors, because n_2 is frequency dependent. (Reprinted from [1]).

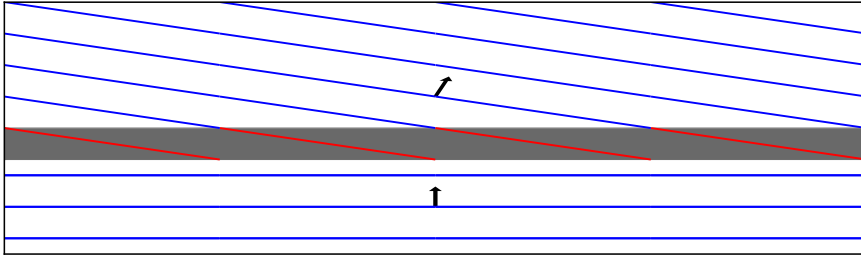


Figure 1.2: Schematic of light manipulation by propagating through a material with varying reflective index (gray). The blue lines are the wave fronts, red lines represent the induced phase manipulation, and the black arrows represent the wave vectors. This material alters the propagation direction by inducing a phase gradient.

the light is slowed down proportionally to the length of the material the light has to travel through. Consequently, its wave front and direction of propagation is manipulated. Instead of shaping the wave front through varying thicknesses of the same material it is also theoretically possible to manipulate the wave front by a material with constant thickness and varying refractive index. These approaches can produce the same wave front manipulation, although creating a material with changing refractive index is in practice much harder than shaping its thickness. This concept is visualized in figure 1.2.

With the same concepts, it is possible to create lenses with different optical characteristics. In figure 1.3 the two different ways of creating lenses are visualized schematically.

1.2 Nanophotonics

In nanotechnology, nanophotonics has emerged as a transformative discipline, exploring the interaction of light with matter at the nanoscale. This has unveiled novel phenomena and applications, e.g., solar cells [3], spectroscopy [4,5], microscopy [6], and band-gap engineering [7], to name just a few. As the demand for miniaturization and reduced losses in optical technologies increases, the advent of metasurfaces has been a highly anticipated breakthrough. These

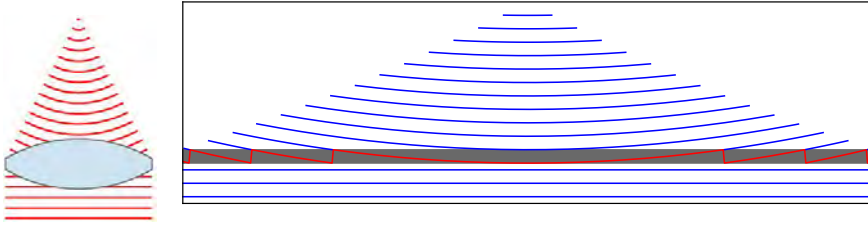


Figure 1.3: Left: Schematic of light focusing by propagating through a lens. The wave fronts are visualized by red lines (reprinted from [2]). Right: Schematic of light focusing by propagating through a material with varying reflective index (gray). The blue lines are the wave fronts, red lines represent the induced phase manipulation.

revolutionary structures, composed of a single layer of phase-altering nanostructures, open up new avenues for manipulating light in ways that surpass the limits of conventional materials. The term “meta,” derived from the Greek language, translates to “beyond” nature’s properties, alluding to the extraordinary characteristics of these metamaterials.

The beginning of research in metasurfaces can be traced back to the proposal of locally reducing the phase velocity using metallic patches [8], followed by the development of grating structures with a lattice constant smaller than the wavelength [9, 10]. Subsequently, nanotechnological advancements led to the creation of metal and dielectric-based structures capable of manipulating light across a broad frequency band [11–17]. These metasurfaces consist of elementary building blocks, known as meta-atoms [18–20], which represent the fundamental units of interaction with electromagnetic waves, in contrast to atoms in conventional materials.

Unlike traditional optical components that modulate light through the gradual accumulation of phase during propagation in refractive materials [21, 22], metasurfaces rely on individually designed meta-atoms to locally modify the phase and amplitude of scattered waves at the interface between two media (see figures 1.4, 1.5, and 1.6). This enables precise control over electromagnetic wave fronts, leading to functionalities unattainable with conventional materials. To maintain the effective continuity of the metasurface, the relative size of the meta-atoms must remain small in comparison to the radiation wavelength. This paves the way for obtaining nontrivial scattering response from metasurfaces, with the ability to tune refractive indices to positive, near-zero, or even negative values [23, 24].

Another important aspect of the individual meta-atom or groups of meta-atoms are their resonant characteristics in a finite bandwidth. Earlier research was mostly focused on metallic nanostructures due to localized plasmonic resonances, which is the resonant scattering of light by free electrons oscillating at the surface of the metallic unit cells. These unit cells can be described as radiating metallic antennas [26]. Such electric dipole resonances can be approximated by single harmonic oscillators and the scattering phase shift cannot exceed π [27] without metallic resonances. In order to shape the wave front in a meaningful way, a 2π phase coverage is necessary, but due to relatively high

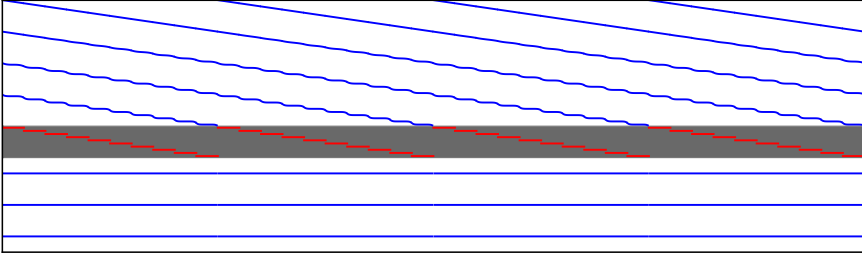


Figure 1.4: Schematic of light manipulation by propagating through a metasurface (gray). The blue lines are the wave fronts and red lines represent the induced phase manipulation of the unit cells. This metamaterial alters the propagation direction by effectively inducing a phase gradient.

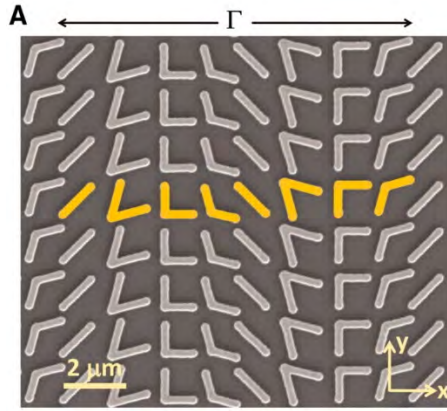


Figure 1.5: A metasurface using a resonance tuning to create a phase gradient depicted in figure 1.4. Unit cells are arranged in a repeating super cell (yellow). (Reprinted from [25] with permission from AAAS).

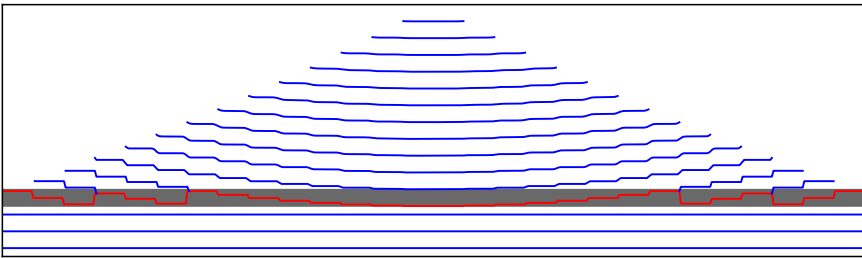


Figure 1.6: Schematic of light manipulation by propagating through a meta-surface (gray). The blue lines are the wave fronts and red lines represent the induced phase manipulation of the unit cells. This metamaterial alters the propagation direction by shaping the wave front like a lens.

losses, metallic metasurfaces are not suitable for wave front control at optical frequencies. By replacing metal with dielectrics, the limitations of plasmonic metasurfaces can be overcome. Dielectric unit cells can be resonant, have lower losses, and can obtain 2π phase coverage. Hence, dielectric metasurfaces fulfill all the necessary requirements and are currently an intensely researched material in the field.

The design of metasurfaces and their unit cells is limited by the lithographic techniques used to create them. Various technical aspects are relevant in lithography, but the most relevant technical limitation in the context of this thesis is the aspect ratio, which is the smallest feature size (of the unit cell 2D cross section) divided by the height of the unit cell structure. Therefore, the aspect ratio determines the size of the smallest features in the unit cells and, consequently, also limits what kind of optical response is accessible.

The research in this area has resulted in metasurfaces being designed for a wide range of different purposes, e.g., lenses and mirrors that achieve almost arbitrary refraction and reflection [25, 28, 29], focusing light while having optional polarization control [30–33], for holograms [34], axicons [35], retroreflectors [36], and optical tweezers (see figure 1.7), to name only a few.

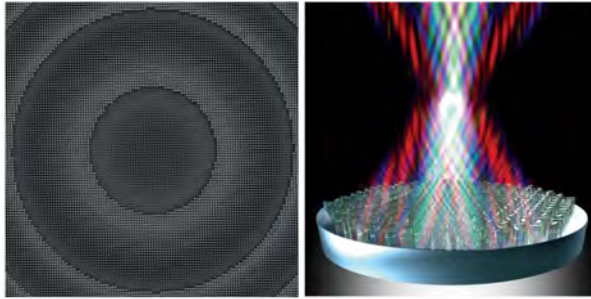


Figure 1.7: Left: Top-view scanning electron microscopy image of a metalens based on titanium dioxide, with dielectric pillars as unit cells. Right: Schematic of an achromatic metalens realized by engineering the dispersive response of its unit cells (Reprinted from [37] with permission from AAAS).

1.3 Inverse design

The challenging nature of designing metasurfaces sparked or intensified the development of various different approaches. However, all these methods are based on solving the underlying wave equations, which are partial differential equations. Given their coefficients, wave equations can be solved to high accuracy, but wavelength-sized objects often lead to computationally demanding simulations for anything beyond the simplest geometries. This is called the *forward problem*, which in this case involves solving the partial differential equations of an incoming light beam scattered off a structure to evaluate how much of the light gets transmitted and reflected, and at which phase for different polarizations measured at a predefined boundary. Solutions can be found with different methods, such as the finite-element method, the finite-difference

time-domain method, the transfer matrix method, etc.

The *inverse problem* [38–60], determining the coefficients of the partial differential equation (representing the geometry of the structure) from a field on a boundary, is even more demanding, since traditional optimization requires a large number of forward problems to be solved sequentially. Nevertheless, it’s a problem worth solving as the solutions may offer technological insights that go beyond human imagination. Free-form inverse design in physics, also known as topology optimization, is well established in continuum mechanics [61] and has in the last decades also been used in physics paradigms that rely on the wave equation [62].

For metasurfaces, inverse design takes the following form. The overarching objective is to manipulate the wave front of a light beam which either gets transmitted through or reflected from such a surface. The shape of the wave front can also be determined by an amplitude and phase profile in a plane parallel to the metasurface in question (see figure 1.4).

In order to physically realize this amplitude and phase profile, meta-atoms with the desired properties for the transmitted or reflected light are placed on the surface to match the target amplitude and phase profile. For example, the phase profile for a lens (see figure 1.3) is given by

$$\varphi(x, y) = -\frac{2\pi}{\lambda} \left(\sqrt{x^2 + y^2 + f^2} - f \right), \quad (1.2)$$

where x and y are the coordinates of the phase-shifting elements (unit cells) of the metasurface, λ is the wavelength and f is the focal length of the lens. An array of subwavelength unit cells thereby creates a quasicontinuous phase profile that shapes a wave front of a transmitted plane wave into a wave that, just like a traditional lens, focuses light at the focal point (see figure 1.6). Therefore, one can design a metasurface lens by arranging unit cells with the optical properties suggested by the phase profile of equation 1.2. Consequently, the metalens design can be split up into a lot of inverse design problems of unit cells with similar transmission amplitudes and phases between 0 and 2π .

1.3.1 Inverse design in nanophotonics

Before looking at the inverse design methodology proposed in this thesis, let us briefly look at different inverse design approaches. Inverse design refers to the automated design of structures with desired functionalities. In contrast to the conventional forward design process, where system parameters are manually tweaked to optimize performance, inverse design starts with a target performance and works backwards to identify the optimal system parameters. Within the context of nanophotonics, inverse design methodologies assist in identifying very small structures that produce specified optical responses, which could include absorption, scattering, emission, or waveguiding at specific wavelengths, angles, or polarizations.

Inverse design is riddled with a unique class of computational challenges, known as “*ill-posed problems*”. A problem is considered “*well-posed*” [63] if the three conditions formulated by French mathematician Jacques Hadamard are satisfied: *existence*, *uniqueness*, and *stability* of the solution. However, inverse design frequently encounters problems that deviate from these conditions. These

are ill-posed problems, where one or more of these criteria of well-posedness are violated. Indeed, different nanostructure configurations can result in similar optical properties, violating the condition of uniqueness (often called “one-to-many” problem), or minor variations in design parameters may cause drastic differences in the optical properties, infringing upon the condition of stability. Occasionally, no experimentally feasible set of nanostructure parameters can realize the desired photonic behavior, challenging the condition of existence.

1.3.2 Methods for inverse design

To solve the inverse problem, one defines an objective function that characterizes how close the solution of the wave equation is to the desired solution. An optimization algorithm is then used to find the coefficients of the wave equation that minimize the objective function. This can be applied to the design of, for example, optical and acoustic devices, since the coefficients of the wave equation define the materials and the geometry of the device. Depending on the number of degrees of freedom, inverse design can be achieved in different ways.

Three or fewer parameters

An exhaustive **parameter scan**, also known as grid search, parameter sweep or look-up tables, is a basic, yet often computationally intensive method of inverse design. This technique involves enumerating all possible combinations of design parameters within a predefined range and granularity. This method can guarantee to find the global optimum within the specified parameter space, given sufficient computational resources. However, as the number of design variables increases, the computational complexity grows exponentially—a phenomenon known as the “curse of dimensionality”—making this method less feasible for problems with high-dimensional parameter spaces.

Regression methods involve fitting a model to data to extract the underlying relationship between inputs and outputs. While these methods can be computationally efficient, they may not provide optimal solutions for more complex design problems.

Three to a few tens of parameters

Binary search [64, 65] is an algorithm typically employed to find a specific value within a sorted dataset. In the context of inverse design, it can be used to pinpoint the optimal parameter value within a predefined range [66].

Genetic algorithms [41, 62, 67–69], inspired by natural selection and evolution, use operators such as mutation, crossover, and selection to optimize complex problems. Within the field of nanophotonics, genetic algorithms have been applied for the inverse design of various optical structures such as photonic crystals and metamaterials.

Particle swarm optimization (PSO) [70–73] optimizes a problem by iteratively improving a candidate solution with regard to a given measure of quality. PSO, inspired by the social behavior of bird flocking or fish schooling,

has been effective in solving many optimization problems, and has been applied in the inverse design of nanophotonic structures.

More than a few tens of parameters

Physics-informed gradient-descent methods [39, 40, 74–76] use gradient information extracted from the underlying physical response to guide the search towards the optimal solution. The gradient of the objective function with respect to design variables is calculated to inform the direction and step size of the subsequent iteration. The adjoint method [39, 77–86] is usually used to obtain the gradient information, since it is a powerful tool for optimizations with a large number of parameters. Therefore, it is the only algorithm capable of handling the enormous number of design parameters in topological optimization. The general goal is to maximize a design problem specific Figure-of-Merit (FoM) or to minimize an objective function. By conducting a forward simulation of the current design iteration (to compute its FoM) and an adjoint simulation, the FoM derivatives with respect to all design parameters can be calculated. This permits the use of a gradient ascent/descent method to iteratively optimize the FoM. Key advantages of the adjoint method are its computational efficiency (only two simulations per iteration are required, where the computational cost of the adjoint simulation is in many cases smaller than the simulation of the current iteration, due to mathematical similarities) and its compatibility with commercial forward solvers. In chapter 6, we compare the physics-informed gradient-descent method with the neural network-based approach suggested in this thesis.

Optimization algorithms often require a large number of evaluations of the forward problem, to be solved sequentially. Additionally, once an algorithm has converged it is not clear whether a local or global minimum has been reached. Therefore, an algorithm has to be restarted multiple times in order to reach the global optimum with some degree of certainty. With a gradient-descent or stochastic optimization method, this means the optimization should be restarted a few times, potentially leading to wall-clock times of up to days or weeks in nanophotonic problems, depending on the average wall-clock time of the forward problem. Sometimes, inverse design reproduces known structures such as periodic photonic crystals or gratings [50, 75], but it can also produce designs with small features that are difficult to fabricate with lithographic methods if no method regulating the fabrication feasibility is integrated into the inverse design process.

Recently, machine learning techniques have been applied to inverse design in nanophotonics. Neural networks and other machine learning algorithms can learn complex relationships between design parameters and optical response. This dissertation will delve into the use of Conditional Generative Adversarial Networks (CGANs), a type of probabilistic deep inverse model, to address ill-posed problems in inverse design. By learning the distribution of solutions, CGANs can generate feasible and physically valid metasurface designs, even in situations of non-uniqueness.

1.4 Machine learning and neural networks

Over the last few decades, machine learning has become a pivotal tool in various fields, e.g., medical diagnosis [87, 88], image and face recognition [89], classification tasks [90], the development of self-driving vehicles [91], high-dimensional function approximation [92], robotics [93], control, data creation, and the construction of expansive language models such as ChatGPT [94]. Its applications extend to many more areas, reflecting a paradigm shift in computational approaches. Recent years have also seen the emergence of new developments in physics owing to the advancement in neural networks. For instance, neural networks have been successfully deployed to discover phase transitions in quantum-mechanical systems [95], predict ultrafast dynamics in nonlinear fibre optics [96], and identify fundamental physical principles such as energy conservation [97].

The focus of this thesis is to explore and address the computational and methodological challenges encountered in leveraging neural networks for the inverse design of metasurfaces [46, 47, 51, 60, 98]. Deep artificial neural networks (DNNs) have proven to have substantial merits compared to conventional calculation techniques in specific use cases [99–101]. In particular, once DNNs have been trained they provide the major advantage that they can quickly generate a design for a meta-atom with desired transmission amplitude and phase in the subset of realizable $\mathcal{S}$ parameters. Consequently, this negates the requirement to execute an individual optimization for each meta-atom, representing a significant advancement in efficiency after training.

1.4.1 Biological neural networks

Let us first take a step back to talk about what neural networks are and how they can learn to perform any of the tasks mentioned above. Artificial neural networks draw inspiration from biological counterparts, predominantly but not solely present within the brains of living organisms (see figure 1.8). These biological networks are composed of neurons, specialized cells that exhibit electrical excitability. Through synapses, neurons establish connections with one another and transmit signals, either electrical or chemical (see figure 1.9). This web of communication serves as a model for artificial neural networks, bridging the gap between biology and computation. Based on the input received and the state of a given neuron, two outcomes are possible: the neuron may generate an electrical signal, known as an action potential, and transmit it to an adjacent neuron, or it may remain inactive, not producing the signal. One effective method for observing the activity of an individual neuron involves examining its output as a temporal function. The resulting graphical representation is called spike train, as depicted in figure 1.10. A neuron is characterized as being active within a brief time window if it emits an action potential or a sequence thereof.

By organizing these relatively simple biological constituents into networks, organisms are equipped with the capability to process information from sensory input. They evaluate this information based on stored memories within the brain and genetically informed instincts, allowing them to make decisions and execute actions within their environment.

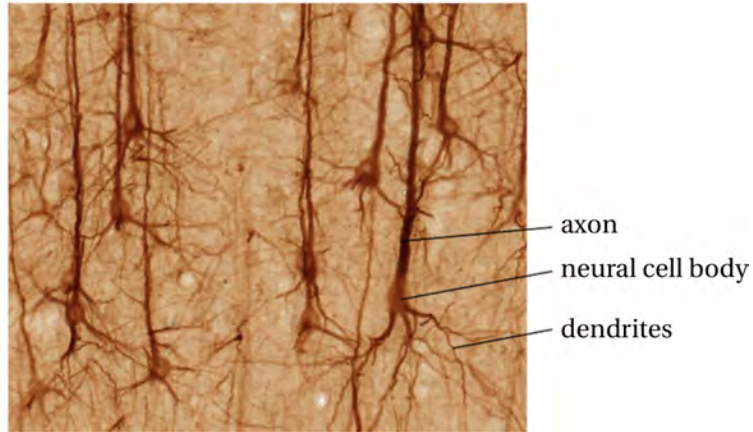


Figure 1.8: Neurons in cerebral cortex. (reprinted from [102])

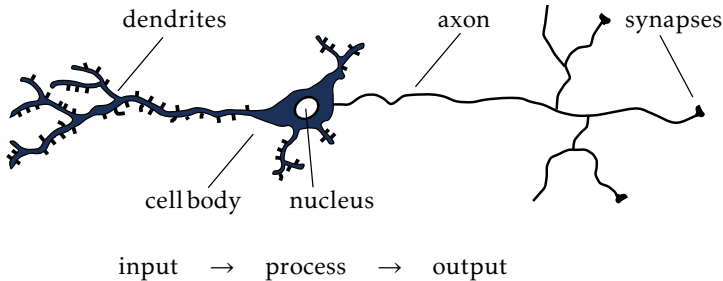


Figure 1.9: Schematic of a biological neuron (from [103] and reproduced with permission of Cambridge University Press).

The function that neurons serve in the behavior of organisms, such as humans, can be elucidated through the following example: A photon, after being reflected off an object, enters the human eye, making contact with the retina. Upon this interaction, a photoreceptor cell converts the photon into an electrical signal, which subsequently traverses the optic nerve into the brain. Alongside other signals initiated by different photons, a cascading effect ensues, characterized by neurons firing in intricate patterns. These patterns manifest a form of information processing, transforming raw sensory input into perceptual understanding. When this system operates as intended, the resultant activation pattern within the brain may, among other potential outcomes, trigger electrical signals to travel through the nervous system, inducing muscle contractions and facilitating interaction with the previously perceived object. It is important to recognize that this example is a simplification, yet it serves to illustrate the essential role that complex networks of neurons and synapses play in biology. Consequently, neural networks furnish organisms with a strategy to perform informed actions in response to their environment, in stark contrast to the rudimentary behavior exhibited by simple bacteria.

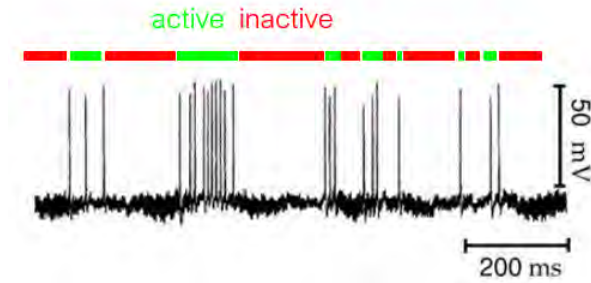


Figure 1.10: Spike train of a single neuron in the brain (from [103] and reproduced with permission of Cambridge University Press).

1.4.2 Artificial neural networks

The inception of artificial neural networks (ANNs) can be traced back to 1943 when Warren McCulloch and Walter Pitts pioneered the concept by publishing their paper entitled “A Logical Calculus of Ideas Immanent in Nervous Activity” [104]. In this work, they introduced the notion of artificial neurons possessing binary inputs, a threshold, and activation function, serving as a simplified precursor to the complex building blocks constituting contemporary artificial neural networks.

Subsequent advancements were made in 1958 when psychologist Frank Rosenblatt invented the perceptron, a foundational element that gave rise to the first artificial neural network [105]. The development of backpropagation methods at the outset of the 1960s further paved the way for multilayer networks during the mid-60s. However, ANN research encountered stagnation from 1969 onwards due to the findings of Minsky and Papert [106], who proved that perceptrons were incapable of processing the exclusive-or circuit, and due to the insufficient computing power of that era to train more elaborate networks.

The resurgence of interest in the field occurred in the mid-80s when hardware advancements reached a computational threshold, facilitating breakthroughs in predictive tasks such as next-word prediction in a sequence. Subsequent progress, characterized by a continuous increase in computing power, development of novel network architectures, and the refinement of learning algorithms, enabled achievements in diverse applications, such as predicting stock market trends [107] and the realization of a (predominantly) self-driving car [108] in 1995.

In the beginning of the 2010s, graphics processing units (GPUs) and tensor processing units (TPUs) were introduced, increasing hardware computing capabilities to unprecedented levels. This technological leap rendered ANNs an increasingly practical tool for scientific exploration and real-world implementation. Presently, ANNs have achieved supremacy in various domains where they outperform human cognition. Notable examples include image recognition and classification tasks, as well as strategic, well-defined board games such as chess, go, and shogi, but also video games like starcraft 2 and dota 2. The trajectory of ANN research continues to reflect a dynamic interplay between computational innovation and practical applicability, underscoring its importance in modern

scientific and technological landscapes.

In order to understand how all this was possible it is best to start at the elementary building block of any ANN, the artificial neuron:

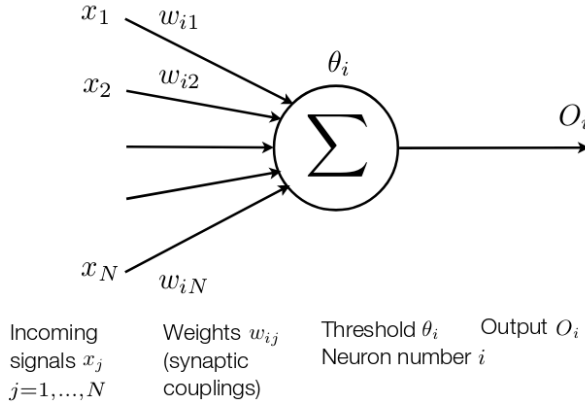


Figure 1.11: Schematic of an artificial neuron (from [103] and reproduced with permission of Cambridge University Press).

Artificial neurons, in contrast to their biological counterparts, are mathematical constructs and produce real-number outputs instead of electrical signals. The architecture of these artificial neurons involves connections to other neurons or inputs through weighted links. Specifically, the inputs x_j to a given neuron are individually multiplied by their corresponding weights w_j , and subsequently aggregated. This cumulative value is then combined with a specific threshold or bias parameter θ associated with that neuron.

$$b = w_1x_1 + \dots + w_Nx_N - \theta. \quad (1.3)$$

This is a mathematical abstraction of the synaptic connections and firing mechanism found in biological neurons, translating them into a computational framework. This local field b is then plugged into an activation function g and the result is the output O of that neuron:

$$O_i = g(b_i) = g \left(\sum_{j=1}^N w_{ij}x_j - \theta_i \right). \quad (1.4)$$

Activation functions can in principle have any kind of shape, but should be piecewise differentiable. Some of the most common ones are the step function, the signum function, the sigmoid function, and —the most commonly used one— the ReLU function. The purpose of the activation function is to introduce nonlinearities into the system to make it possible to approximate any kind of function.

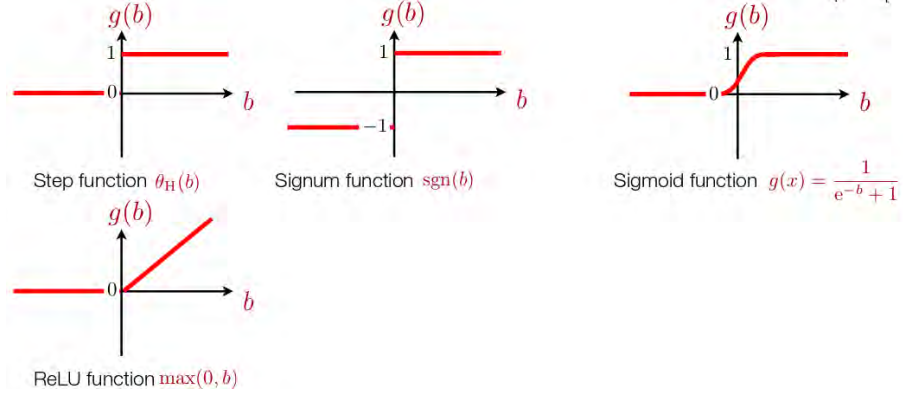


Figure 1.12: Examples of activation functions (reprinted from [109] with permission from the author).

To facilitate meaningful computational operations, these elementary building blocks are systematically arranged into networks. Within these networks, neurons are typically grouped into layers, forming a hierarchical structure where the outputs from one layer serve as the inputs to the subsequent layer.

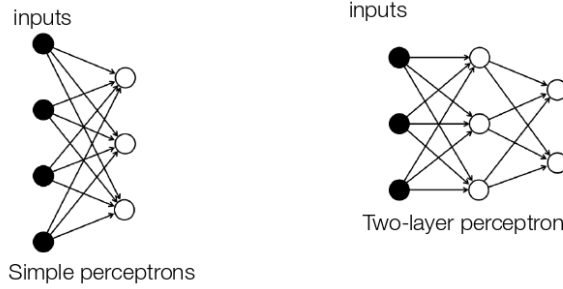


Figure 1.13: Multilayer perceptron (reprinted from [109]).

By combining these simple building blocks with the learning strategies described in the following chapter, it becomes generally feasible to approximate any function, whether linear or nonlinear. This feature of ANNs is described by the universal approximation theorem [110, 111]. In this thesis, we will use this property of ANNs to approximate the relation between an optical response quality, such as reflection or transmission, and the geometry of an optical meta-atom.

1.5 Outline of the thesis

In the upcoming chapters, I will present a machine learning framework for the inverse design of metasurfaces. Chapter 2 will delve into the theory of artificial neural networks by covering key types of architectures, learning algorithms, and useful tools for stabilizing and optimizing the training process. The chapter will be concluded by illustrating the theory with a simple 2D classification example.

Chapter 3 deals with the different aspects related to training data creation and its representation. First, the free-form representation of the metasurface unit cells is outlined and, subsequently, the representation of the finite element simulated optical properties is discussed.

After that, two specialized networks will be introduced in chapter 4. These models are trained to approximate the optical properties of a given unit cell design (forward model) and to evaluate the degree to which a design is experimentally feasible (classifier model). In this chapter, I will also evaluate how some of the tools mentioned in chapter 2 influence the training process of the two models.

Chapter 5 outlines our proposed conditional generative adversarial network (CGAN) frameworks to train inverse/generator networks. To this end, I will first introduce the concept of generative adversarial networks (GANs), before discussing strategies to get conditional outputs from a generator model. Thereafter, the main contributions of this thesis will be discussed, which center around improving the CGAN network graph, by integrating the forward and classifier models into the discriminator and, furthermore, introducing two functions that process the conditional inputs and the forward and classifier model outputs. Additionally, an encoder network is introduced, which prevents mode collapse and increases the generator output diversity. Then, multiple tools to optimize and stabilize the CGAN training process are evaluated, before a variation of the CGAN graph to produce multiple interdependent outputs is elaborated. The last section of this chapter presents the iterative improvement scheme used to optimize the training data set during its creation.

In chapter 6, I will present a back-of-the-envelope comparison of physics informed gradient descent methods compared with our CGAN methodology with regards to the required computational resources and wall-clock times.

Various application examples will be given in chapter 7 to demonstrate the viability of our approach. Here, I first describe the design and simulation of a cylindrical lens, then outline an experimental project focused on the design of a polarization converting metasurface lens, which is supposed to be etched on top of a Vertical Cavity Surface Emitting Laser (VCSEL). In the last section of this chapter a project with the Jet Propulsion Laboratory to design a metasurface for a Zernike wavefront sensor (ZWFS) is presented.

Finally, in chapter 8 a conclusion of the project is given, before outlining possible future improvements and potential paradigm shifts in the field of inverse design in nanophotonics.

Chapter 2

Artificial Neural Networks

Here, we will delve into the theory of artificial neural networks, covering various types of architectures, learning algorithms, and useful tools for stabilizing and optimizing the training process. In the last section of this chapter, we will illustrate the theory with a simple 2D classification example.

Artificial Neural Networks (ANNs) are typically constructed of individual neurons arranged in layers, sequentially stacked with a feed-forward relationship between successive layers. The first layer ($l = 1$) customarily serves as the input layer, wherein numerical data is introduced into the ANN. This input is systematically processed through successive layers until the final output can be obtained from the last layer ($l = L$). Within the architectures of ANNs, there exists a variety of layer types, among which the dense layer is one of the most prevalent.

In a dense or fully connected layer, every neuron is connected to every neuron in the subsequent layer. Another very common component is the convolutional layer, where neurons are spatially organized and connections are selectively enabled between neighboring neurons across consecutive layers (see section 2.2 for details). Figure 2.1 illustrates an example of an ANN that sequentially combines a convolutional layer, a max-pooling layer (which extracts the maximum value from a predefined region), and two fully connected layers. This modular structure enhances the network's capacity to learn and represent complex patterns and relationships within the data.

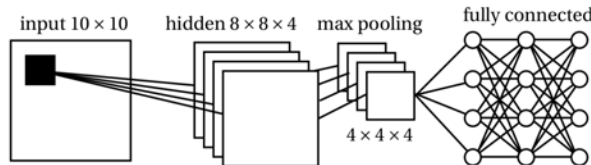


Figure 2.1: Convolutional neural network with a max pooling layer and two fully connected layers for simple image processing (from [103] and reproduced with permission of Cambridge University Press).

Connections between neurons can also go beyond the next layer and connections from neurons to preceding layers are possible (see chapters 4 and 5), but

the latter pose a risk to the convergence of the training process. For simplicity, we assume for now that connections are only possible from one layer to the next and the following notation will be employed:

$$\begin{aligned}
 x &: \text{input} \\
 L &: \text{number of layers} \\
 W^l = w_{ij}^l &: \text{weights connecting layer } l-1 \text{ and } l \\
 \theta^l &: \text{biases/thresholds at layer } l \\
 g^l &: \text{activation function at layer } l \\
 \mathcal{M}[x] &: \text{overall network/model function}
 \end{aligned} \tag{2.1}$$

The overall network function describing all operations in the ANN can then be written as:

$$\mathcal{M}[x] = g^L(W^L g^{L-1}(W^{L-1} \dots g^1(W^1 x - \theta^1) \dots - \theta^{L-1}) - \theta^L) \tag{2.2}$$

Here, all the inputs x , weights W , and biases θ can have different dimensions from scalars to vectors and tensors.

A useful insight into how ANNs process information arises from equation 1.4: All neurons (and thresholds) in one layer can be described by a vector with dimensions equal to the number of neurons. In the same way all the weights can be written in the form of a matrix. Therefore, the argument in the activation function going from one layer to the next is a matrix multiplication and vector addition. Consequently, the overall network is a combination of function composition, matrix multiplication, and vector addition.

2.1 Gradient descent and backpropagation

At the start of the training process, an ANN is typically initialized with weights drawn from a random distribution (commonly Gaussian), and the biases are set to 0. As a result, the network's outputs at this stage will be arbitrary. To extract coherent and valuable information from an ANN, it is imperative that all learnable parameters, including both weights and biases, be fine-tuned through a learning algorithm. Among the prevailing methodologies employed for this purpose are those rooted in gradient descent and backpropagation. The gradient descent method is a systematic procedure of modifying the weights W and biases θ in a direction that minimizes the error function $\mathcal{E}$ for a given training dataset, as illustrated in figure 2.2.

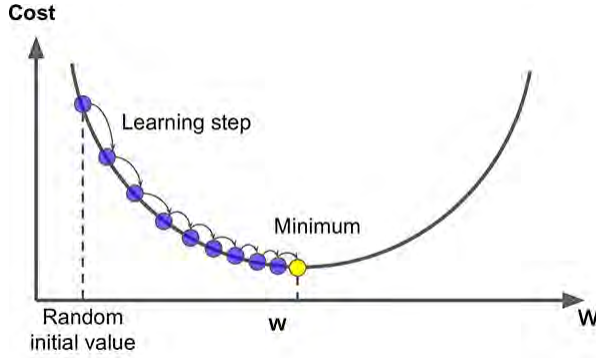


Figure 2.2: Illustration of the gradient descent method in the one-dimensional case, where the error/cost function gets reduced in every iteration until the minimum is reached (Reprinted from [112]).

To realize this optimization process algorithmically, an essential part involves evaluating how the learnable parameters should be adjusted. This specific procedure is termed backpropagation, a contraction of “backward propagation of errors.” The meaning of “backwards” in this context refers to the method’s underlying principle, where the computation of the gradient is performed in a reverse direction through the network. This procedure allows for the systematic computation of gradients with respect to each parameter, thereby guiding the incremental adjustments required to minimize the error function. This elegant algorithmic strategy forms the backbone of effective training in most ANNs.

The final requisite component in the training of an artificial neural network (ANN) is the availability of an extensive training data set, comprising a multitude of input-output pairs (x, y) . Examples of such pairs might include photographs of handwritten digits paired with their corresponding numerical values, or images of cats and dogs accompanied by labels distinguishing the two. The integrity of the training data set is paramount to the performance of the resulting model, and considerations of size, diversity, and congruence with post-training inputs are vital attributes for a high-quality training data set.

The loss function, denoted by $\mathcal{E}$, serves as a pivotal construct in this context. Although it may assume various forms, its fundamental role persists in evaluating the ANN’s performance by contrasting the network’s outputs with the corresponding ground-truth labels. The design of the loss function ensures that its magnitude increases with the disparity between the predicted outputs and actual labels, thereby providing a quantitative measure of the network’s accuracy (e.g., mean square error — mse). For the fine tuning of all learnable parameters, the following definitions are made:

$$\begin{aligned}
 y &: \text{ground truth label/target output} \\
 \mathcal{E}(y, \mathcal{M}[x, W, \theta]) &: \text{cost/error/loss function} \\
 b^l &: \text{local field in layer } l \\
 O^l = g^l(b^l) &: \text{output/activation in layer } l
 \end{aligned} \tag{2.3}$$

A useful evaluation is the derivative of the error function $\mathcal{E}$ with respect to the

inputs x :

$$\begin{aligned}\frac{d\mathcal{E}}{dx} &= \frac{d\mathcal{E}}{dO^L} \cdot \frac{dO^L}{dO^{L-1}} \cdot \frac{dO^{L-1}}{dO^{L-2}} \cdot \dots \cdot \frac{dO^1}{dx} \\ &= \frac{d\mathcal{E}}{dO^L} \circ (g^L)' \cdot W^L \circ (g^{L-1})' \cdot W^{L-1} \circ \dots \circ (g^1)' \cdot W^1.\end{aligned}\quad (2.4)$$

where $\circ$ is the Hadamard product¹, which is an element-wise product taken from left to right. The biases θ are ignored here for simplicity. The gradient ∇ is given by the transpose of the derivative:

$$\begin{aligned}\nabla_x \mathcal{E} &= \left(\frac{d\mathcal{E}}{dx} \right)^T \\ &= (W^1)^T \cdot (g^1)' \circ \dots \circ (W^{L-1})^T \cdot (g^{L-1})' \circ (W^L)^T \cdot (g^L)' \circ \nabla_{O^L} \mathcal{E}.\end{aligned}\quad (2.5)$$

Gradient descent algorithms serve to minimize the loss function by successively adjusting the learnable parameters within the ANN, as depicted in figure 2.2. This iterative process refines the network's outputs to more closely align with the target labels. Central to this optimization is the evaluation of the derivative of the loss function with respect to all the learnable parameters.

For the final layer of the network, the derivation of these gradients is straightforward. However, as the error propagation extends backward to the input layer, the complexity of the expressions increases. This increase in complexity is attributed to the multiplication of terms involved in the chain rule, which is utilized to calculate the derivatives at each layer:

$$\begin{aligned}\nabla_{W^L} \mathcal{E} &= (g^L)' \circ \nabla_{O^L} \mathcal{E} \cdot (O^{L-1})^T \\ \nabla_{W^{L-1}} \mathcal{E} &= (g^{L-1})' \circ (W^L)^T \cdot (g^L)' \circ \nabla_{O^L} \mathcal{E} \cdot (O^{L-2})^T \\ &\vdots \\ \nabla_{W^1} \mathcal{E} &= (g^1)' \circ (W^2)^T \cdot (g^2)' \circ \dots \circ (g^{L-1})' \circ (W^L)^T \\ &\quad \cdot (g^L)' \circ \nabla_{O^L} \mathcal{E} \cdot x.\end{aligned}\quad (2.6)$$

These equations can be simplified with the following notation:

$$\begin{aligned}\delta^l &= (g^l)' \circ (W^{l+1})^T \\ &\quad \cdot (g^{l+1})' \circ \dots \cdot (g^{L-1})' \circ (W^L)^T \cdot (g^L)' \circ \nabla_{O^L} \mathcal{E} \\ \delta^{l-1} &:= (g^{l-1})' \circ (W^l)^T \cdot \delta^l \\ \nabla_{W^l} \mathcal{E} &= \delta^l (O^{l-1})^T.\end{aligned}\quad (2.7)$$

After finding the derivatives in this highly multidimensional landscape, all the weights are changed slightly in the opposite direction of the gradient² in order to iteratively converge to a minimum of the error function

$$w_{ij}^l{}' = w_{ij}^l + \delta w_{ij}^l \quad \text{with} \quad \delta w_{ij}^l = -\eta \nabla_{w_{ij}^l} \mathcal{E}, \quad (2.8)$$

where the small parameter $\eta > 0$ is the learning rate.

¹ $(A \circ B)_{ij} = (A)_{ij} (B)_{ij}$

²The gradient points in the “upwards direction” of the error function

These calculations can be done analogously for the biases:

$$\begin{aligned} \nabla_{\theta^l} \mathcal{E} &= -\delta^l (O^{l-1})^T \\ \theta_i^{l'} &= \theta_i^l + \delta \theta_i^l \quad \text{with} \quad \delta \theta_i^l = -\eta \nabla_{\theta_i^l} \mathcal{E}. \end{aligned} \tag{2.9}$$

Algorithms that employ gradient descent often incorporate several optimization techniques to enhance the convergence and robustness of the learning process. These strategies may include the introduction of a momentum term to mitigate oscillations, normalizing the gradient update δW to a predetermined value to control the step size, or dynamically adapting the learning rate in response to the evolving landscape of the loss function, among other tactics. While these enhancements can offer significant benefits in practice, the underlying principle of the gradient descent methodology can be captured in its simplest form, as listed in algorithm 1. This description serves as a starting point for understanding more complex variations and adaptations used in modern machine learning frameworks.

Algorithm 1 Gradient descent

```

initialize weights  $w_{ij}^l$  to random values, thresholds to zero,  $\theta_i^l = 0$ 
for  $t = 1, \dots, T$  do
    randomize the order of the training set
    for  $x^\mu$  in Training set do
         $O_i^1 = g^1 \left( \sum_{j=1} w_{ij}^1 x_j^\mu - \theta_i^1 \right)$ 
        for  $l = 2, \dots, L$  do
            propagate forward:  $O_i^l = g^l \left( \sum_{j=1} w_{ij}^l O_j^{l-1} - \theta_i^l \right)$ 
        end for
         $\delta^L = (W^L)^T \cdot (g^L)' \circ \nabla_{O^L} \mathcal{E}$ 
        for  $l = L, \dots, 2$  do
            propagate backwards:  $\delta^{l-1} = (g^{l-1})' \circ (W^l)^T \cdot \delta^l$ 
        end for
        for  $l = 1, \dots, L$  do
            update:  $\delta W^l \leftarrow \delta W^l - \eta \delta^l (O^{l-1})^T$  and  $\theta^l \leftarrow \theta^l - \eta \delta^l$ 
        end for
    end for
end for

```

The gradient descent optimizer used in this thesis is Adam [113] (short for Adaptive Moment Estimation) and a variant called AdaMax, which are further evolutions of earlier optimizers combining adaptive gradients with the momentum method. The algorithm calculates both the first-order and the second-order moments of the gradient of the loss function. Specifically, it computes a moving average of past gradients (first-order moment) and a moving average of the past squared gradients (second-order moment). By doing so, Adam dynamically adapts the learning rates of each parameter in the model. Due to its inherently adaptive nature, robustness to stochastic noise, and capability to effectively handle sparse gradients, the Adam optimizer has become a preferred choice for training a wide range of machine learning models.

2.2 Convolutional layers

Convolutional layers are a cornerstone in the architecture of convolutional neural networks (CNNs), specifically designed to automatically and adaptively learn spatial hierarchies of features from input images. These layers perform a convolution operation, applying a set of learnable filters or kernels across the spatial dimensions of the input. Each filter generates a feature map that captures local and translation-invariant characteristics of the input, such as edges, corners, or textures. Through stacking multiple convolutional layers, the network is capable of learning increasingly complex and abstract features. Convolutional layers also significantly reduce the number of parameters in the model by sharing weights across spatial locations, making them highly efficient in terms of computational resources.

Generally, convolutions are employed in many areas of image processing to perform operations like blurring, sharpening, and embossing among others. The core operation behind this is mathematically expressed as:

$$O(x, y) = g \left(\sum_{i=-n}^n \sum_{j=-m}^m K(i, j) \cdot I(x - i, y - j) \right) \quad (2.10)$$

where $O(x, y)$ is the pixel value in the output feature map, $K(i, j)$ are the kernel's coefficients, and $I(x - i, y - j)$ are neighboring pixels in the input feature map or input image. Essentially, the kernel acts as a trainable parameterized function that transforms each pixel in the output feature map based on its local neighborhood in the input feature map.

Two of the simplest yet effective kernels used for edge detection in convolutional layers are the 3×3 matrices commonly known as the horizontal and vertical Sobel operators [114]

$$K_H = \begin{pmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{pmatrix} \quad \text{and} \quad K_V = \begin{pmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{pmatrix}. \quad (2.11)$$

In a convolutional layer, these kernels scan through every possible 3×3 sub-image from pixel to pixel over the whole input image to highlight horizontal (K_H) and vertical (K_V) edges. Specifically, when the kernel K_V passes over a vertical edge in the image, the resulting convolution output will produce high values, thereby indicating the presence of an edge. In other words, the Sobel operators are a discrete differentiation capturing the essential spatial intensity changes in the horizontal and vertical directions of the image, therefore, approximating the gradients of the image pixels values. Figure 2.3 shows the feature maps of K_H and K_V applied to a gray scale image.

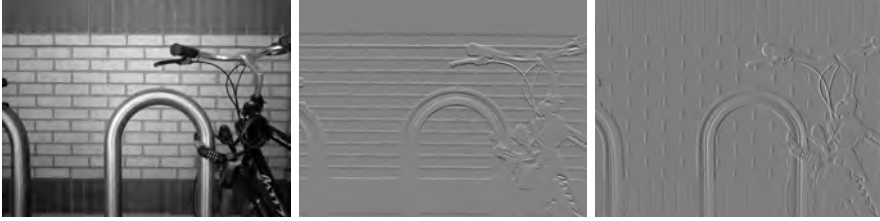


Figure 2.3: Left: Gray scale test image of brick wall and bike rack. Middle: Feature map of K_H applied to the gray scale test image. Right: Feature map of K_V applied to the gray scale test image. (Reprinted from [115])

The kernels K_H and K_V are usually not hard coded into the network, but in many training instances the network will approximate these or mathematically equivalent ones in the training process, due to their information processing capabilities.

In recent years, neural network architectures have been the subject of intense research and development. However, standard CNNs have a number of limitations, such as vanishing gradients, which can make it difficult for them to learn from deeper layers in the network. To address these issues, researchers have developed various architectures, such as residual networks (ResNets) [116], which use skip connections to enable information to flow more freely through the network. One of the newer developments in neural-network architectures is dense residual networks (DenseNets) [117], which take the idea of skip connections a step further by connecting a lot of layers directly with each other. This results in a very dense and connected network structure that allows for efficient feature propagation and reuse. The DenseNet architecture has shown great promise in various image-related tasks, including object detection and image segmentation [118–120]. We will use DenseNets in the forward (chapter 4.1) and classifier (chapter 4.2) models, as well as in the discriminator of the CGAN (chapter 5).

2.3 Augmentation

Another technique that has become increasingly popular in machine learning is data augmentation [88, 121–123], and it is particularly relevant in the context of training neural networks. The idea behind data augmentation is to artificially increase the amount of training data by generating new examples that are similar to the original ones but contain variations in some aspects. This can be done by applying various transformations to the original data, such as rotations, translations, scaling, flipping, or adding noise (see sections 4.3 and 5.3).

The primary reason for using data augmentation is to improve the performance and robustness of the neural network. By training on a larger and more diverse set of data, the network can learn to generalize better and be more capable of handling variations and noise in the real-world data. Additionally, data augmentation can help to reduce overfitting. Furthermore, data augmentation is particularly important in domains where the amount of available data is limited, such as in scientific or medical applications. In these cases, it is often not feasible to collect large amounts of new data, and therefore data

augmentation can be an effective way to leverage the existing data and improve the performance of the network. For inverse design, data augmentation can reduce the need for training data, which is important as the generation of training data through simulations constitute the major computational cost.

2.3.1 Noise

The injection of noise [99, 124–129] into the training process of neural networks serves as a powerful augmentation and regularization technique that enhances the model’s generalization capabilities and reduces overfitting. Therefore, we use blurring [130, 131] of the training data and Gaussian noise in the initial stages of training of the networks presented in this thesis (see sections 4.3 and 5.3 for empirical evidence). By introducing stochastic perturbations to either the input data, the weights, or even the activations, the model is encouraged to learn more robust and invariant features. This mechanism effectively prevents the network from fitting to spurious correlations or noise inherent in the training data, leading to improved performance on unseen data. Such techniques are particularly valuable in scenarios where the training data is limited or imbalanced, as they can mitigate the risk of overfitting. Overall, noise injection has gained recognition for its ability to improve model robustness without significantly complicating the architecture or training process.

2.4 Additional features

All the networks trained in the development of this thesis also use dropout [132] and L2 regularization [133]. Dropout is a regularization technique commonly employed in neural networks to mitigate the issue of overfitting. During the training phase, dropout stochastically deactivates a subset of neurons by setting their output to zero with a predetermined probability p . This forces the model to distribute the learned features more uniformly across the neural architecture, thereby enhancing generalization to unseen data. Implemented as a layer within the neural network, dropout essentially creates an ensemble of sub-networks with shared weights, contributing to a more robust final model. It has been observed to be effective across various types of neural networks and is considered a standard tool in the deep learning toolkit.

L2 regularization, also known as ridge regularization, is a commonly used technique to prevent overfitting in machine learning models, including neural networks. It imposes a penalty on the model by adding the sum of the squares of the model weights to the original loss function. Mathematically, this can be expressed as

$$\tilde{\mathcal{E}} = \mathcal{E} + \lambda \sum_i w_i^2, \quad (2.12)$$

where $\tilde{\mathcal{E}}$ is the new loss function, $\mathcal{E}$ is the original loss function, λ is the regularization coefficient, and w_i represents each weight in the network. The regularization term discourages the weights from reaching large values, thereby constraining the model’s capacity to fit noise in the training data. L2 regularization has the benefit of being differentiable, making it amenable to

gradient-based optimization algorithms.

2.5 A simple classification problem

To understand the inner workings of algorithm 1, consider a two-dimensional classification example³. This example comprises ten data points x^μ , each possessing two real-valued components, along with their corresponding target values y^μ . This illustration intends to visualize the fundamental principles underlying the gradient descent algorithm and provides a perspective on the process of optimization within the context of neural network training.

$$x^\mu = \begin{bmatrix} x_1^\mu \\ x_2^\mu \end{bmatrix} \quad (2.13)$$

$$y^\mu = 1 \quad \text{for } \blacksquare \quad \text{and} \quad y^\mu = -1 \quad \text{for } \square \quad (2.14)$$

Hence, a straightforward perceptron will be utilized, consisting of two inputs (the two components corresponding to a given data point) and one output. The activation function used here is the signum function, represented as $g(b) = \text{sgn}(b)$. Given that there is merely one output neuron connected to two inputs, the weights can be expressed as a vector:

$$w = \begin{bmatrix} w_1 \\ w_2 \end{bmatrix}. \quad (2.15)$$

Then the full network computation is given by:

$$O = \text{sgn}(w_1 x_1 + w_2 x_2 - \theta) = \text{sgn}(w \cdot x - \theta). \quad (2.16)$$

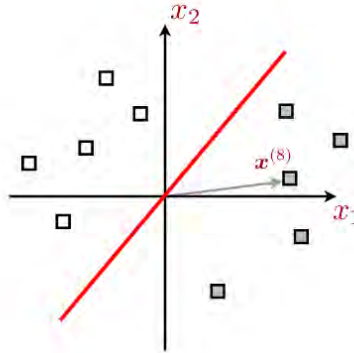


Figure 2.4: 2D classification of two different kinds of data points (white and grey) with a decision boundary (red). (From [103] and reproduced with permission of Cambridge University Press)

Due to the simplicity of this example, it is possible to visualize it. In figure 2.4, the two target groups are segregated into two distinct clusters, with

³The content of this section is derived from Chapters 6.1 and 6.2 of the book “Machine Learning with Neural Networks: An Introduction for Scientists and Engineers” [103].

a red line drawn between them representing a decision boundary. In this specific instance, and for simplicity, the threshold in equation 2.16 will be set to $\theta = 0$. Then the classification task takes the form

$$\text{sgn}(w \cdot x^\mu) = \text{sgn}(|w||x^\mu|\cos(\varphi)) = y^\mu \quad (2.17)$$

where φ is the angle between the two vectors w and x . Since only the sign of the argument of the signum function determines whether the output is -1 or 1 and the absolute values of the vectors are positive semidefinite, the angle φ determines the output of the network (when φ is between $-\pi/2$ and $\pi/2$, the scalar product is positive and it is negative otherwise). Consequently, the network classifies the patterns in figure 2.4 correctly if the weight vector is orthogonal to the decision boundary. In general, the threshold θ can be used to offset the intersection of the decision boundary with the x_2 -axis.

In order to solve the classification task, the decision boundary must cleanly separate the groups of positive and negative targets. If no such line can be identified, then the single neuron is insufficient to solve the classification, necessitating a more complex network architecture. Conversely, if such a line exists, a single neuron can achieve the task, and the data is described as linearly separable.

Finding the optimal weights and thresholds of the network is accomplished through the gradient descent algorithm 1, and the process can be visualized using a specific example. In figure 2.5, the input x^8 ($y^8 = 1$) is initially situated on the incorrect side of the decision boundary. By fine-tuning the weights, the decision boundary can be rotated to minimize the error, bringing the classification in line with the intended target outcomes,

$$w' = w + \delta w \quad \text{with} \quad \delta w_j^\mu = \eta(y^\mu - O^\mu)x_j^\mu. \quad (2.18)$$

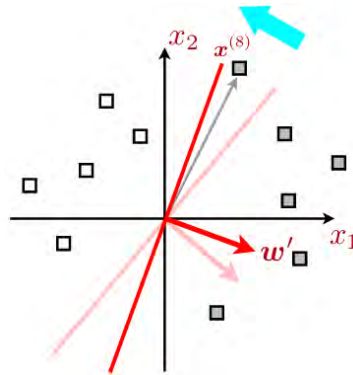


Figure 2.5: 2D classification, weight adjustment (from [103] and reproduced with permission of Cambridge University Press)

This example serves to visualize the gradient descent algorithm in its most basic form. As the network and its hyperparameters grow in complexity, the gradient descent procedure becomes more sophisticated. However, the underlying methodology remains conceptually the same.

The task of the network is to translate the provided training data points into an internal representation. Through this process, the network learns the decision boundaries, which in higher-dimensional spaces, manifest as hyperplanes. This encapsulation of information and representation learning is fundamental to the capabilities of artificial neural networks in modeling complex, nonlinear relationships within the data.

Chapter 3

Data Representation, Creation, and Simulations

In this chapter, we will discuss the data creation and the representations developed to train the ANN models discussed in the subsequent chapters. In this context, we will go into the details of how the unit cell designs in a metasurface (see figures 1.5 and 3.3) are connected to the scattering parameters ($\mathcal{S}$ parameters), which define the local part of the amplitude and phase profile section the unit cell is spatially occupying (see figure 1.6). The overarching goal of the trained ANNs is to approximate the scattering responses of metasurface unit cells and perform the inverse design of these cells. Consequently, it becomes essential to find a practical representation of the unit cells and the scattering response that can be processed by an ANN. Finding a suitable form of data representation is a crucial aspect of the inverse design, underlining the need for integrating physics and machine learning to achieve the desired precision and effectiveness in design and analysis.

3.1 Representation of the input data

We first discuss how we represent the unit cell geometry in a way that is not infringing on the free-form design aspect of our methodology. Due to experimental constraints in creating varying structural heights within a single unit cell or the entire metasurface, it is appropriate to characterize the meta-atom by its 2D cross section. This cross section would act as the lithographic mask in the fabrication process. One conceivable method for depicting the cross section is to utilize a few design parameters that describe the positions, sizes, and angles of geometric entities. An alternative approach, offering more design freedom, would employ coefficients of functions representing lines that separate the areas of the lithographic mask, thus differentiating regions with material from those devoid of material after the etching process. While the second approach provides greater flexibility than the former, this method's drawbacks lie in the confinement of possible geometries by predetermined shapes and functions. To achieve a truly free-form approach, where the designs employed in the training data and the output of the inverse design model are constrained

solely by experimental limitations, a pixelated representation has been selected. This choice is further justified by the resemblance of the 2D cross section to images, as it allows us to harness the extensive corpus of research on computer vision dealing with the processing of pixelated images. In figure 3.1 eight cross section examples are plotted.

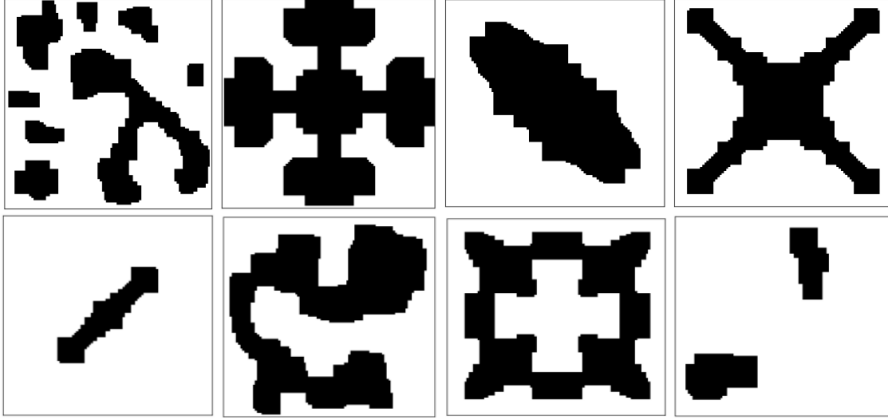


Figure 3.1: Example grids created by the grid creation algorithm.

In addressing the aforementioned challenge, a Python algorithm was developed to generate a data set comprising of semi-random pixelated images, representing the lithographic mask. The algorithm operates on a 100×100 binary grid, where the digit 0 denotes regions where the material is etched away, and 1 signifies areas where the lithographic mask prevents the etching process, preserving the material's integrity.

The number of possible grid configurations in this representation (ignoring experimental feasibility) is $2^{100^2} \approx 2 \cdot 10^{3010}$ and, therefore, vastly exceeds the number of atoms in the known universe, which are estimated to be around $10^{78} - 10^{82}$. Due to this vast number of configurations, the fact that only a small fraction of them will be experimentally feasible and that this “small fraction” is still an incredibly large number, any approach mentioned in chapter 1.3.2 other than physics informed gradient descent and ANNs will be very difficult or even impossible to implement with realistic computational constraints.

The binary grids are then evolved from random noise through the elimination of holes or the erasure of islands of increasing sizes, which is necessary because of the limitations state of the art lithographic techniques in producing such fine features have. This process is visualized in table 3.2. A commonly used term in this context is the aspect ratio, which is defined as the height of the structure on top of the substrate (out of plane) divided by the width of its smallest feature (in-plane). In the grid creation algorithm, random noise is iteratively refined into coarser structures until the fabrication constraints set by the aspect ratio are satisfied. In some instances, mirror and rotational symmetry operations were employed to further enhance the designs. The permissible geometries of our meta-atoms are confined only by the grid's resolution and the fabrication feasibility, not by any methodological constraints. The resolution of the grid was deliberately selected such that one pixel is smaller than the average

fabrication inaccuracy, thereby aligning with the technological constraints of the manufacturing process.

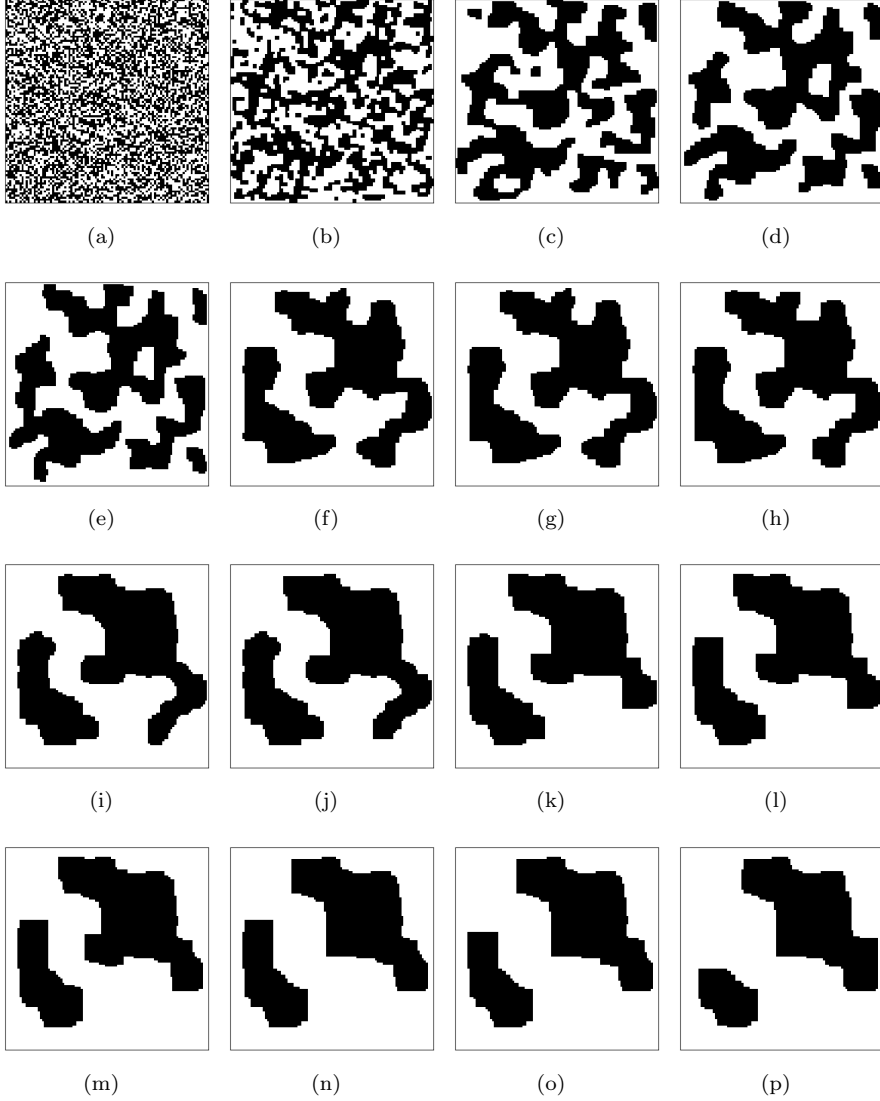


Figure 3.2: Grid creation algorithm. (a) Random noise. (b) Random noise corrected for pixel size 1. (c-p) Grids further corrected for pixel sizes 2-15.

3.2 COMSOL simulations

For each structure represented by a grid, an individual COMSOL [134] simulation was conducted to acquire the S parameters of that specific unit cell. In the project of paper A, our metasurface is composed of a silicon substrate,

with a 300 nm-thin silica spacer and a 500 nm-thick patterned layer of silicon on top of the substrate.

In collaboration with the NASA Jet Propulsion Laboratory (Paper C), we simulated a silicon substrate with amorphous silica structures on top. In this project, each unit cell was subjected to two distinct simulations. The first simulation determined the scattering parameters of the unit cell. The second simulation was specifically conducted to obtain the scattering response for the incoming plane wave with orthogonal polarization.

A large number of simulations for various grids were evaluated utilizing powerful computing clusters and a Python script that interfaced with the COMSOL application programming interface (API).

The finite element simulations calculate the $\mathcal{S}$ parameters by evaluating the inverse of the matrix describing the system of linear equations arising from Maxwell's equations expressed on the meshed geometry, periodic boundary conditions, absorbing boundary conditions (ports) for a plane wave incident from the top of the metasurface. Periodic boundary conditions are used to evaluate an infinite plane with the same unit cell instead of one isolated unit cell with poorly defined surroundings. A typical simulation in our projects has between 50,000 to 500,000 mesh domain elements and 500,000 to 3,500,000 number of degrees of freedom to solve for. Due to this high number of variables to solve for, we did not use a direct solver, but an iterative one, because of faster convergence and RAM limitations. The simulations take between 15 minutes and 24 hours in outlier cases with an average around one hour on a Intel(R) Xeon(R) Gold 6338 CPU with 32 cores and 96GB RAM. After successful evaluation, an output file is generated, which contains the real part, imaginary part, amplitude, and phase of all four $\mathcal{S}$ parameters for each frequency of interest.

3.2.1 $\mathcal{S}$ parameters

The scattering response of our meta-atoms is described by scattering parameters, commonly referred to as $\mathcal{S}$ parameters, which are a set of complex numbers that encapsulate both amplitude and phase information. They serve as an invaluable tool in describing how an electrical network responds to input signals. In microwave and radio-frequency engineering, $\mathcal{S}$ parameters are used to characterize multiport devices. The unit cells of a metasurface can be viewed as a four-port system in this context, with one port for each polarization on both ends of the unit cell. One possible configuration would be:

- Port 1 on top of the air box, x polarized light (see figure 3.3 (c));
- Port 2 on top of the air box and orthogonal polarization to port 1 (y polarized light);
- Port 3 underneath the structure (x polarized light);
- Port 4 underneath the structure and orthogonal polarization to port 3 (y polarized light).

Instead of the basis of linear polarizations, we could also choose a basis of circular polarizations, but $\mathcal{S}$ parameters in different bases are related by a linear transformation.

The $\mathcal{S}$ parameters are usually represented in matrix form, denoted by $\mathcal{S}$ and for a four-port network the $\mathcal{S}$ matrix can be expressed as

$$\begin{pmatrix} S_{11} & S_{12} & S_{13} & S_{14} \\ S_{21} & S_{22} & S_{23} & S_{24} \\ S_{31} & S_{32} & S_{33} & S_{34} \\ S_{41} & S_{42} & S_{43} & S_{44} \end{pmatrix}, \quad (3.1)$$

where S_{kj} characterizes how the electromagnetic waves entering the j -th port are scattered by the structure into the k -th port. We can decompose the $\mathcal{S}$ parameters in polar coordinates: $S_{kj} = |S_{kj}| \exp(i\varphi_{kj})$ with $|S_{kj}|$ being the absolute value¹, which signifies how much of the amplitude coming in through port j is scattered into port k and φ_{kj} is the phase added to the electromagnetic wave from port j to k (complex phase of S_{kj}).

Reflection and transmission coefficients serve as primary metrics for assessing the performance of a metasurface unit cell. For instance, in the case of a metasurface designed for beam steering, one may focus on optimizing the S_{31} parameter to maximize energy transfer of the same polarization. Phase manipulation is another domain where $\mathcal{S}$ parameters play a vital role. By analyzing the phase information φ_{ij} , designers can tune unit cells to achieve desired phase profiles. This capability is useful in applications like lensing and holography.

3.3 Representation of the output data

The COMSOL-simulated $\mathcal{S}$ parameters (see figure 3.3) of metasurface unit cells are used as the ground truth labels for the forward model (see chapter 4.1). This output is represented as a simple array, encompassing the imaginary part, real part, amplitude, and intensity of each $\mathcal{S}$ parameter, along with the sum of all four intensities. Such a representation effectively defines the optical properties of a given unit cell, yielding 17 outputs for each frequency. Although this data is partially redundant and could be symbolized solely by the imaginary and real part or the amplitude and phase of each $\mathcal{S}$ parameter, the other dependent parameters are incorporated to stabilize the learning process. This includes squaring the neural network output for the amplitude to align with the corresponding intensity and ensuring energy conservation by matching the sum of all intensities with the identical sum from the simulation data.

¹ $|S_{kj}| \in [0, 1]$ for passive devices.

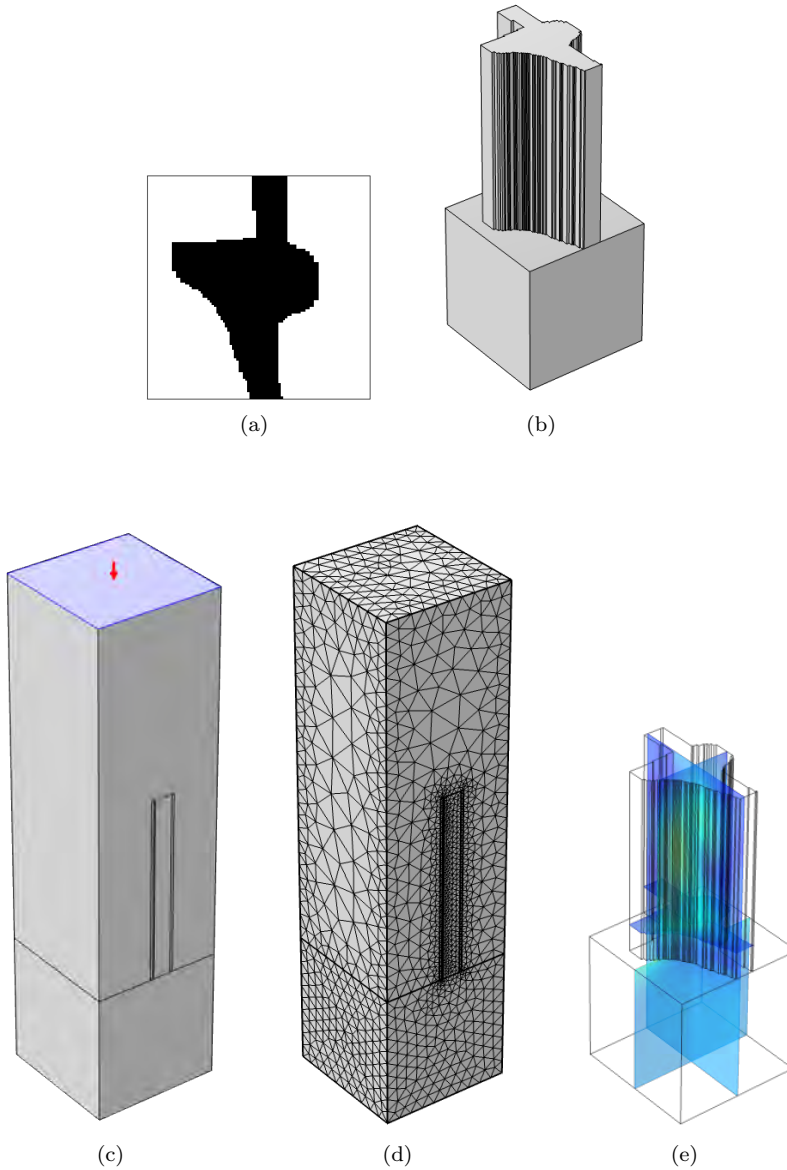


Figure 3.3: (a) Grid - 2D cross section of the unit cell structure. (b) Unit cell structure on top of substrate. (c) Full unit cell geometry with air box on top of the etched structure and the port for electromagnetic excitations. (d) Meshed geometry. (e) Simulated electric field norm (V/m).

Chapter 4

Expert/Advisor Networks

In this chapter, we introduce two ANN models. The first aims to approximate the $\mathcal{S}$ parameters of a given unit cell, and the second is designed to classify the experimental feasibility of the unit cell. Together, these expert/advisor networks assist the inverse-design ANN models in addressing challenges relating to $\mathcal{S}$ parameter accuracy and experimental feasibility.

4.1 Forward model

The task of performing free-form inverse design of nanophotonic structures, ensuring that they are experimentally feasible while also possessing the desired optical properties, is nontrivial, due to the high-dimensional and complex cost function. Addressing this complexity necessitates a method for approximating optical properties that is less time-consuming than full-wave simulations for a given design. Such an approach is essential for training an inverse or generator model effectively. To achieve this goal, a forward neural network model, denoted by $\mathcal{F}$, has been developed. This model is designed to approximate the optical properties [135–137] of our meta-atoms (unit cells), specifically the scattering parameters $\mathcal{S}$, as a function of the grid (g), which represents the lithographic mask.

$$\mathcal{F}[g] \approx \mathcal{S} \quad (4.1)$$

In subsequent sections of this thesis, the forward model is employed as a surrogate model [42, 49, 138, 139] in the training process of the conditional generative adversarial network (CGAN) model, as detailed in chapter 5. The necessity for a rapid and differentiable approximation of the optical properties becomes evident, as we will show that it is indispensable for our CGAN model’s convergence (see chapter 5.3). This, in turn, ensures the generation of high-quality results within an acceptable wall-clock time.

The architecture of the neural network employed in this study bears significant resemblance to those commonly encountered in the literature that focus on image classification and information extraction from images as shown in figure 4.1. A DenseNet [117] or an alternative convolutional layer configuration is the first part of the model. Subsequent to this, the network paths merge into a sequence of dense layers, prior to bifurcating into 12 discrete dense

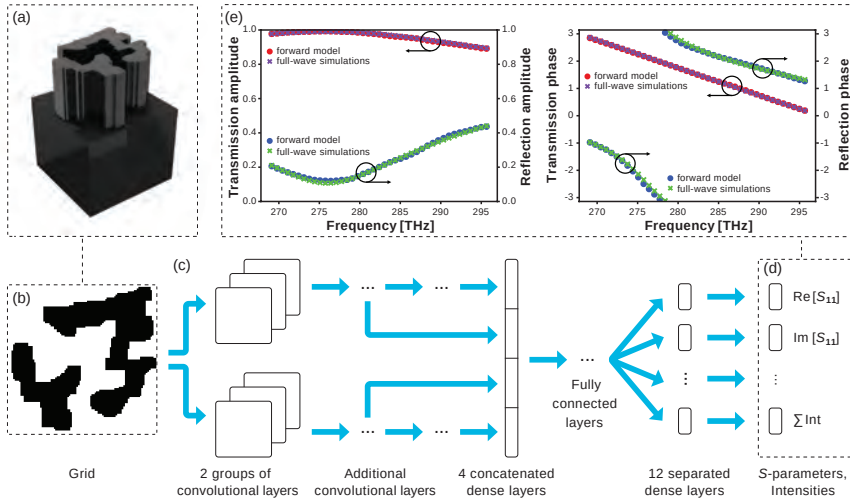


Figure 4.1: The forward model, which is schematically depicted in this figure, approximates the S parameters of a given metasurface meta-atom. (a) Rendered illustration of a meta-atom. (b) Grid representation (2D cross section) of the meta-atom. (c) Schematic architecture of the forward model (2 groups of convolutional layers were used instead of a DenseNet at the time of paper A). (d) Output of the forward model. (e) Comparison of S parameters predicted by the forward model (circles) and calculated by full-wave simulations (crosses). (Reprinted from [38])

layers. These are then complemented by a 1D convolution for the purpose of smoothing, followed by a cropping layer to eliminate any residual edge effects. The final layers render estimates for various variables, specifically: $\text{Re}[\mathcal{S}]$, $\text{Im}[\mathcal{S}]$, and $|\mathcal{S}|$ for each of the four S parameters, encompassing both transmission and reflection in two distinct polarization states ($S_{11}, S_{12}, S_{13}, S_{14}$ and $S_{21}, S_{22}, S_{23}, S_{24}$).

In addition to these, the network generates five supplementary outputs: $(\text{Re}[\mathcal{S}]^2 + \text{Im}[\mathcal{S}]^2)$ for each of the S parameters, and $\sum |\mathcal{S}|^2$, where the summation extends over all S parameters from one incoming polarization. These 17 outputs, although partly redundant, play a key role in the loss function of the forward network. By evaluating all of them, the possible output manifold of the network is confined to solutions that more accurately resemble the physical constraints on the outputs. These constraints are dictated by principles such as energy conservation and other physical relationships between the outputs.

It should be noted that the same confinement could be achieved by integrating specific terms into the loss function rather than incorporating additional outputs. However, the presented method offers an equivalent and robust approach ensuring that the network's outputs align closely with the underlying physical laws of the system.

In conjunction with the aforementioned outputs, the network architecture outputs three additional distinct numerical values, formulated to approximate

the maximal absolute difference between the $\mathcal{S}$ parameters across adjacent frequencies ($\mathcal{S}_r$). These values include the real and imaginary parts of the complex amplitudes, along with the absolute magnitudes of both the amplitudes and intensities. The maximum absolute differences between adjacent frequencies serve as metrics to gauge the narrowness of resonances within the unit cell. This criterion is not only useful for the detailed examination of the unit cell's resonance characteristics but also finds utility in Chapter 5.2.1, where these labels are used as inputs to the CGAN. By using these values as inputs, the methodology permits the establishment of a lower bound on the resonance's narrowness within the generated unit cells.

During the training phase of the forward model, the aim is to align these values $\mathcal{F}(g)_r$ with the corresponding maximal differences $\mathcal{S}_r$ obtained from the full-wave simulations. A schematic representation of the forward model architecture is illustrated in Figure 4.1, while supplementary results are shown in Figure 7.1.

A typical validation mean square error for the forward model is approximately 0.002. A limitation to be acknowledged is that the forward model tends to have larger errors in instances where the unit cell exhibit pronounced resonant behaviour. This phenomenon is likely attributable to the increased sensitivity of the $\mathcal{S}$ parameters to fluctuations in the design, leading to complexities in accurate representation and prediction. A way to better way to capture resonances could be achieved by incorporating a variation of the proposed physics-informed neural networks [52, 140, 141] like the Lorentz Neural Networks [56].

4.2 Classifier model

As described in the previous chapters, it is important to integrate fabrication constraints into the inverse design method. These constraints can be characterized by the aspect ratio (width of the smallest feature divided by the height). Filtering out too small geometric features with an algorithm as we did for the creation of the training samples is, however, too time-consuming in the context of ANN training. However, training without an experimental feasibility check leads to ANN model outputs that are to a large extent impossible to fabricate. To address this issue, we trained a classifier network $\mathcal{C}$ on flawless and flawed grids (see figure 4.2). To this end, a large number of flawed grids with a variety of different flaws were created by algorithms and ANNs. Among the possible flaws were issues related to the aspect ratio and different kinds of noise. All grids were labeled with the number of pixels that would need to be changed ($\mathcal{P}_j$) in order to satisfy the aspect ratio of $HM/(jD)$, where H is the height of the structure, M is the number of pixels in one dimension (here 100), j is a number of pixels (defining the length of the smallest feature size) and D is the length of one side of the grid (or in more intuitive terms: the height of the structure divided by the length of j pixels). Consequently, we are dealing with a form of quantized aspect ratio, due to the pixelated nature of the unit cell representations. Therefore, the $\mathcal{P}_j$ s enumerate the number of pixels that would need to be changed in order to fulfill the requirements for the aspect ration $HM/(jD)$. This labeling ($\mathcal{P}_j$) was done up to a given pixel

size N by a variation of the algorithm that created the semirandom grids for the training samples which well exceeds the respective aspect ratio restrictions given by experimental parameters (here $N = 20$). Additionally, the network approximates the size of the smallest isolated island $\mathcal{I}$. This approach ensures that the trained classifier network has the capacity to effectively approximate the experimental feasibility of a given grid.

$$\mathcal{C}[g] \approx \vec{\mathcal{P}} \quad \text{with} \quad \vec{\mathcal{P}} = [\mathcal{P}_0(=\mathcal{I}), \mathcal{P}_1, \mathcal{P}_2, \dots, \mathcal{P}_N]. \quad (4.2)$$

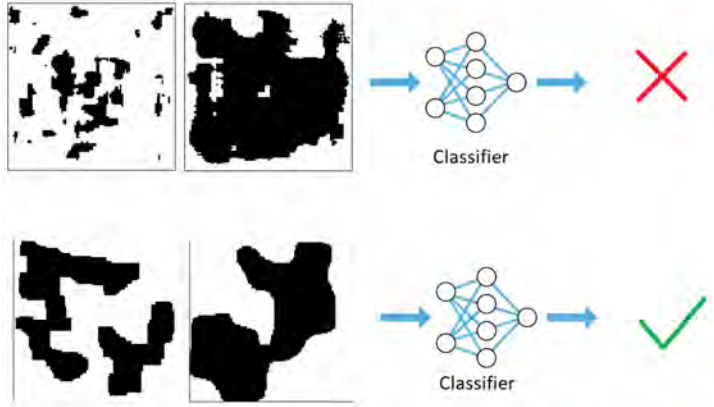


Figure 4.2: Experimental feasibility classification by an ANN model

This network's architecture is also inspired by CNNs for image classification. One notable aspect in the training of this network is the error function, which will be called ln-mse from here on:

$$\mathcal{E}(\mathcal{C}[g], \vec{\mathcal{P}}) = \sum_{j=0}^N [\ln(\mathcal{C}[g]_j + 1) - \ln(\mathcal{P}_j + 1)]^2 \quad (4.3)$$

This ln-mse was chosen to emphasize whether the target value $\mathcal{P}_j$ is 0 or not, while lowering the error for high target values. The idea behind this is the relative importance of the two cases which lay on a continuum: It is almost irrelevant whether the network correctly identifies that 500 or 510 pixels have to be changed ($\mathcal{E}[500, 510] \approx 0.0004$), while it is very relevant whether 0 or 10 pixels have to be changed ($\mathcal{E}[0, 10] \approx 5.75$). The validation ln-mse of the classifier model is between 0.18 and 0.5. Since the classifier yields the vector $\vec{\mathcal{P}}$, figure 4.2 serves merely as a binary simplification, representing a distilled version of the labels on which the classifier is actually trained. Four distinct unit cells, each exhibiting varying levels of experimental feasibility, are illustrated in Figure 4.3, along with their actual classifier labels, ranging from noisy to flawless. It is noteworthy that many of the $\vec{\mathcal{P}}_i$ entries are not precise predictions, particularly for higher indexed pixel numbers, due to the inherent randomness in the algorithm. Nevertheless, a discernible correlation exists between violations of the experimental feasibility constraints and the elements of $\vec{\mathcal{P}}$. This correlation enables the classifier to effectively evaluate the

compliance of a unit cell design with experimental constraints, despite some level of imprecision.

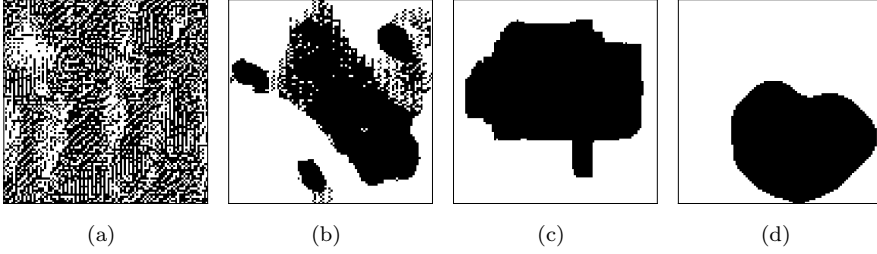


Figure 4.3: Grids with varying levels of experimental feasibility and classifier labels for the first 10 pixel sizes $[\mathcal{P}_0(=\mathcal{I}) - \mathcal{P}_{10}]$:

(a) $\vec{\mathcal{P}} = [1, 4663, 7492, 7620, 7980, 8163, 8278, 8335, 8476, 8481, 8546]$

(b) $\vec{\mathcal{P}} = [1, 655, 1061, 1192, 1321, 1361, 1467, 1641, 1982, 2001, 2283]$

(c) $\vec{\mathcal{P}} = [4738, 6, 8, 8, 8, 8, 24, 24, 24, 64, 225]$

(d) $\vec{\mathcal{P}} = [3018, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]$

Another notion introduced here are the “label triggers” ($\vec{\mathcal{T}}$). These are defined as a variant of the Heaviside function of the classifier labels:

$$\mathcal{T}_i = \begin{cases} 0 & \text{for } \mathcal{P}_i = 0 \quad \text{and} \quad i > 0 \\ 1 & \text{for } \mathcal{P}_i > 0 \quad \text{and} \quad i > 0 \end{cases} \quad (4.4)$$

Consequently, the $\vec{\mathcal{T}}$ in figure 4.3 are vectors of dimension N with $\mathcal{T}_i = 1 \quad \forall i \in [1, N]$ for grids (a-c) and $\mathcal{T}_i = 0 \quad \forall i \in [1, N]$ for (d). A $\mathcal{T}_i = 1$ means that the manufacturability for pixel size i is violated in some form. Two, in this case, slightly more interesting unit cells (the entries of $\vec{\mathcal{T}}$ are not all 0 or 1), and their respective labels and triggers are displayed in figure 4.4.

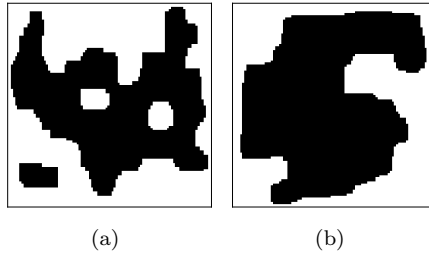


Figure 4.4: Grids with varying levels of experimental feasibility, classifier labels and label triggers for $[\mathcal{P}_1 - \mathcal{P}_{10}]$:

(a) $\vec{\mathcal{P}} = [0, 0, 0, 0, 0, 30, 140, 357, 541, 1054]$, $\vec{\mathcal{T}} = [0, 0, 0, 0, 0, 1, 1, 1, 1, 1]$

(b) $\vec{\mathcal{P}} = [0, 0, 0, 0, 0, 0, 0, 0, 105, 179]$, $\vec{\mathcal{T}} = [0, 0, 0, 0, 0, 0, 0, 0, 1, 1]$

In chapter 5.2.1, the label triggers will be reintroduced as inputs to the CGAN in order to control the the lower limit of the aspect ratio of the generator outputs.

4.3 Training optimization techniques

In order to optimize the training outcomes of our models, we used the various techniques mentioned in chapter 2. We investigate the effect of using DenseNets in the forward network, mapping the geometry of the metasurface on the optical properties, and in the classifier network. The full networks of the forward network with DenseNet and without DenseNet can be found in figures S1 and S2 in Paper B. The full networks for the classifier can be found in figures S3 and S4 in Paper B.

DenseNet In figure 4.5, we plot the mean square error for the validation sets for the forward model and for the classifier model with and without the DenseNet [117] architecture. Each line in the graph represents a ANN trained

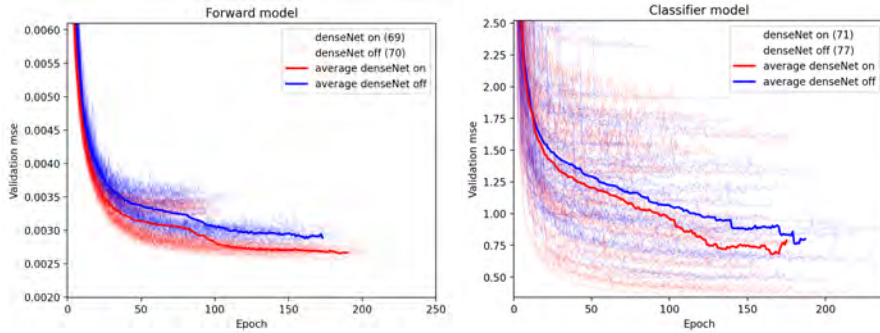


Figure 4.5: Comparison of the validation loss in the forward and classifier models between models with a DenseNet architecture and models with a standard 2D convolution. The numbers in the legend indicate how many models were trained in each category.

with 50,000 training samples randomly selected from the full data set (roughly 80,000 samples). Red lines signify models with DenseNets and blue lines models with a generic 2D convolutional network. The thick lines represent an average over the different models. We see that the adoption of the DenseNet architecture improves the classifier and forward networks. For the forward network, we see two clusters of lines, which is because augmentation (see below) is an even more important contributor to the training and divides the samples into said clusters. The DenseNets make up the lower part of each cluster. For the classifier network, the spread for different networks is larger, but we see that the best networks use the DenseNet architecture and also that on average it is better to use DenseNets.

Augmentation Another technique we used is augmentation, although, for the metasurfaces we are working with in this thesis, we cannot apply arbitrary rotation or scaling to the training data, since these operations on the unit cell alter the optical properties. However, translation can be used because the unit-cell simulations we use as training data have periodic boundary conditions, essentially creating an infinitely repeating surface. Therefore, a translation by $n\vec{a}/p_a$ or $n\vec{b}/p_b$, where $p_{a,b}$ denote the pixel count in the respective dimension and $n \in \mathbb{N}$, provides an equivalent representation of the unit cell with the same optical response. This concept is illustrated in figure 4.6. In addition

to translation, 90° , 180° , and 270° rotations of the unit cell could also be employed in the classifier training, since the experimental feasibility is not affected by this transformation, and could be employed as well in the forward model training by making an appropriate transformation of the $\mathcal{S}$ parameters.

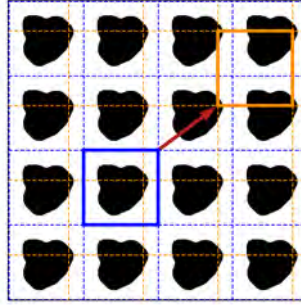


Figure 4.6: Visualization of translational augmentation.

Figure 4.7 is a plot of the same data set as presented in figure 4.5, but now the data is organized according to whether data augmentation has been utilized. Red lines are models that take advantage of data augmentation; blue lines represent models without data augmentation. From this figure it is evident that data augmentation exerts a more substantial influence on the validation errors of the two networks compared to the DenseNet architecture. The two clusters that we saw before in the data for the forward model are clearly related to the use of data augmentation.

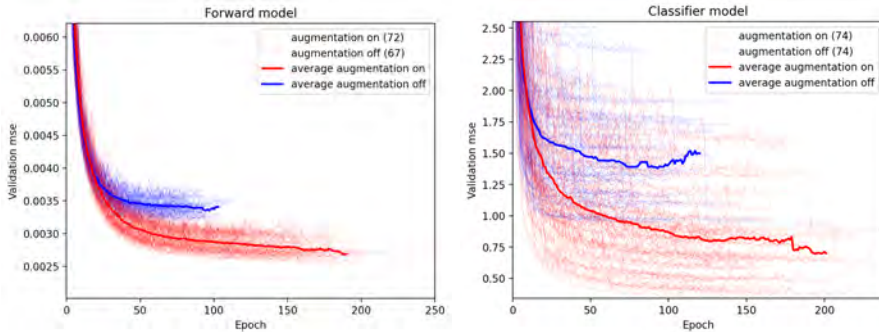


Figure 4.7: Comparison of the validation loss in the forward and classifier model with and without augmentation. The numbers in the legend indicate how many models were trained in each category.

Blurring In addition to augmentation, we use different kinds of blurring [130, 131] by applying semirandom convolutions to random subsets of the training data. These semirandom convolutions were generated by randomly picking an envelope function (Gaussian: $b_g(r) = \exp(-r^2/2a_g^2)$ with $a_g \in [0.5, 4]$; linear: $b_l(r) = 1 - r/a_l$ for $b_l(r) > 0$ with $a_l \in [2, 8]$ and $b_l(r) = 0$ otherwise; or inverse: $b_i(r) = \frac{1}{r+1}$), and a convolution size/range between 3 times 3 and maximally 15 times 15 (the maximum is decreasing with each

epoch). Then, for each pixel in the convolution, the envelope function (where r is the distance of the pixel from the center) is multiplied with a random number between 0 and 1 and the sum of the pixel values in the convolution is normalized to 1. The relevance of blurring will become more important for inverse design with CGANs (see chapter 5), because it will enhance their accuracy when presented with outputs of the generator, which often resemble blurred versions of the training data grids, especially in the initial phase of the training. Therefore, we compare the performance of networks trained on partially blurred data to networks trained on data that was not blurred. As shown in figure 4.8, the effect of blurring parts of the training data is significant, since it substantially improves the accuracy of the networks when they are presented with blurred unit cells. This does not affect the performance of the networks when presented with nonblurred data in any significant way. Consequently, in the initial and intermediate stages of training, the CGAN discriminator receives more accurate information from both the forward and classifier models about the generated unit cells.

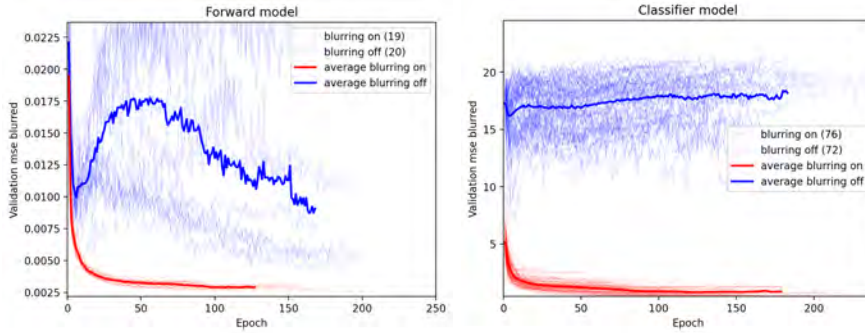


Figure 4.8: Comparison of the validation loss (blurred) in the forward and classifier model with and without blurring of the training data. The numbers in the legend indicate how many models were trained in each category.

4.4 Individual pixel error

Nonsensical designs, which are grids with a large number of pixel values between 0 and 1, can be analyzed by evaluating the sum of individual pixel errors (ipe):

$$\sum_i x_i^2 (x_i - 1)^2, \quad (4.5)$$

where x_i is the individual value of each pixel. One term in the sum is illustrated in figure 4.9. Creating a metasurface with precisely controlled varying etching depths is experimentally very difficult. Therefore, pixel values which are neither (very close to) one or zero should be avoided by the generator, since the etching mask either permits or prevents the top layer to be etched away in the lithography process.

The forward model, classifier model and the ipe will be used to evaluate the quality of the inverse design outputs in terms of $\mathcal{S}$ parameter errors (forward model), experimental feasibility (classifier model), and nonsensicality (ipe).

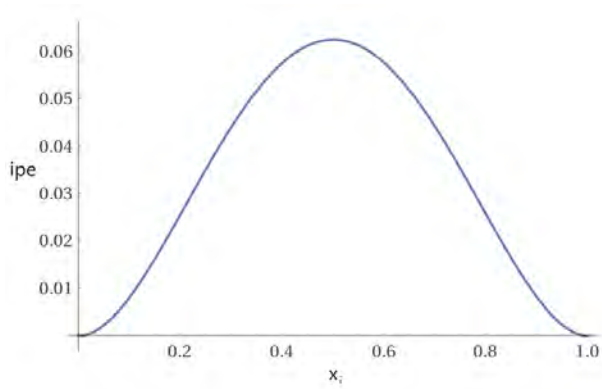


Figure 4.9: One term of individual pixel error (ipe).

Chapter 5

Conditional Generative Adversarial Networks

Now we want to “invert” the forward model, i.e., we want to obtain the etching mask (represented by a grid, see, e.g., figure 4.4) that results in a metasurface unit cell with a set of desired S parameters. To this end, we briefly discuss some complications related to the training of inverse design models. After that the framework of generative adversarial networks (GANs) will be introduced, providing an in-depth exploration of the underlying principles, potential advantages, and inherent challenges associated with their application. Subsequently, the conditional generative adversarial network (CGAN) we have developed will be detailed and we will examine various modifications and methodologies that can be applied to the standard graph and training loop.

The main obstacle for the convergence of an inverse design network lies in the “one-to-many” or “non-uniqueness” problem of the Hadamard conditions (refer to Chapter 1) for a “well-posed” problem. This issue describes the existence of multiple non-unique solutions for a given design target, as characterized by the inequality $\mathcal{F}[g_i] - \mathcal{F}[g_j] < \epsilon$ with $g_i \neq g_j$. The implication of this condition is that a simplistic inversion of the forward model does not converge in general. Indeed, such a network is being trained on data that appears to be conflicting, where a single set of input values corresponds to different possible output targets ($\mathcal{F}^{-1}[\mathcal{S}] = g_i$ and $\mathcal{F}^{-1}[\mathcal{S}] = g_j$). Hence, the network is unable to settle into one of the local minima of the loss function, since it is fed multiple different ground truth labels for essentially the same sets of inputs. Or in other words, a batch containing the two data points $[\mathcal{S}_i, g_i]$ and $[\mathcal{S}_j, g_j]$ with $\mathcal{S}_i - \mathcal{S}_j < \epsilon$ can never converge to the same loss function value as a model trained on a comparable data set with unviolated uniqueness. An iterative gradient based approach (see chapter 1.3.2) is not affected by “non-uniqueness” in the same way, since the gradients in the process of approaching one extremum of the FoM/loss function are not influenced by the gradients of any other extremum. This purely local knowledge of the gradient is beneficial to the convergence of these iterative algorithms, although, as a consequence multiple different starting points in the design space have to be chosen to potentially reach separate FoM/loss function extrema. In the area of ANNs

possible methodologies to overcome this issue are tandem networks [60, 142, 143], conditional variational autoencoders (CVAE) [144–146], and conditional generative adversarial networks (CGAN) [147, 148].

5.1 Generative adversarial networks

Generative Adversarial Networks (GANs) constitute a prominent class of machine learning models, introduced by Ian Goodfellow in 2014 [149]. This paradigm comprises two interdependent artificial neural network (ANN) models, combined within a zero-sum game, i.e., the interactions within this setup can be understood as competitive, with one network’s gain corresponding to the other’s loss.

In a GAN framework, one network is a generator, which is trained to synthesize new data from random noise, shaping it with characteristics similar to those found in the training set. This is realized through the concurrent training of a discriminator network, whose purpose is to discern between data originating from the training set (termed “real”) and data produced by the generator network (denoted as “fake”). Conversely, the generator’s learning objective is to deceive the discriminator, compelling it to incorrectly classify its output as genuine data from the training set. Therefore, GANs require a high-quality training data set, since the quality of the average data point is the theoretical optimum for the quality of the generator outputs. A representative illustration of the GAN framework, utilizing a training set of handwritten digits, is provided in figure 5.1.

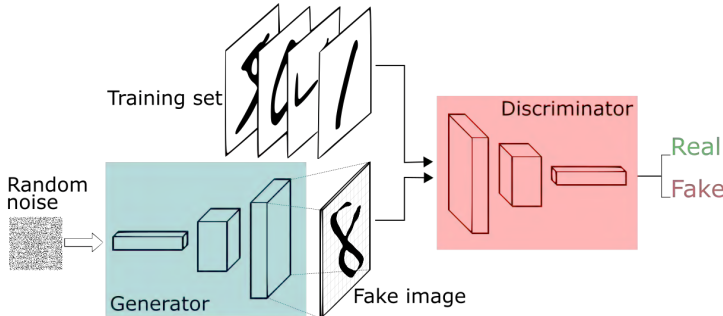


Figure 5.1: Schematic Generative Adversarial Networks. (Reprinted from [150])

The loss function of a GAN is given by

$$\begin{aligned} \mathcal{E}(g, \mathcal{N}) &= \ln(\mathcal{D}(g)) + \ln(1 - \mathcal{D}(\mathcal{G}(\mathcal{N}))) \\ g &: \text{Data point from training set} \\ \mathcal{N} &: \text{Noise} \\ \mathcal{D}(g) &: \text{Probability estimate of the discriminator} \\ &\quad \text{that the data point is real} \in [0, 1] \\ \mathcal{G}(\mathcal{N}) &: \text{Generator output} \\ \mathcal{D}(\mathcal{G}(\mathcal{N})) &: \text{Probability estimate of the discriminator} \\ &\quad \text{that the generator output is real} \in [0, 1] \end{aligned} \tag{5.1}$$

The generator tries to minimize $\mathcal{E}(g, \mathcal{N})$ and the discriminator tries to maximize it. If we take into account that the generator cannot influence the first part ($\ln(\mathcal{D}(g))$) and rewrite the discriminator loss function so that it is supposed to be minimized, the two loss functions are given by:

$$\begin{aligned}\mathcal{E}_{\mathcal{D}}(g, \mathcal{N}) &= -\ln(\mathcal{D}(g)) - \ln(1 - \mathcal{D}(\mathcal{G}(\mathcal{N}))), \\ \mathcal{E}_{\mathcal{G}}(\mathcal{N}) &= \ln(1 - \mathcal{D}(\mathcal{G}(\mathcal{N}))).\end{aligned}\tag{5.2}$$

The utilization of the logarithm in the loss function serves to explicitly penalize probability estimates that are both highly confident and incorrect simultaneously. It is noteworthy that this is not the only possible loss function for a GAN. Another widely recognized variant is the Wasserstein GAN, known as WGAN [151].

This training framework can be related to the real-world scenario of art forgery. In that analogy, the generator in the GAN framework is an artist or art forger, while the discriminator is an art critic or investigator. At the start of the training process, both networks are initialized with random weights and exhibit incompetence at their respective tasks. However, generally speaking, it is simpler to discern the difference between a masterpiece and a child's drawing than to learn the intricate skill of replicating a masterpiece. In a manner akin to this, the discriminator's task of minimizing its loss function and distinguishing between real and fake samples is often less complex than the generator's objective of maximizing a part of the discriminator's loss function by deceiving it. As a consequence, without suitable balancing measures, the discriminator's loss may diminish at a significantly more rapid rate than the generator's, leading to potential training instabilities.

In this work, the balance between the generator and the discriminator in the GANs was maintained through the modification of learning rates specific to each network, as well as the incorporation of noise and augmentation within the input channels (see chapter 5.3). Furthermore, an approach involving conditional skipping of the discriminator training was also explored.

In addition to the difficulty of balancing GANs, the generator often suffers from mode collapse [152–154]. This phenomenon is characterized by the generator's inability to produce a diverse set of outputs, meaning that the outputs either only differ very little or, even worse, not at all from one another. When a generator manages to produce a diverse output set, but fails to reach the diversity of the training set, then the generator is suffering from a partial mode collapse.

Despite the various instabilities that can occur during the training process of GANs, they offer a significant advantage over other methods, such as tandem networks [142, 143]: the presence of a dynamic error function. In a GAN, the discriminator essentially crafts the error function for the generator. This is a direct consequence of training the discriminator to discern the differences between the data generated by the generator and the data set. The generator then strives to correct these differences to more effectively deceive the discriminator. As long as this process remains stable, the proficiency of both networks improves.

Returning to the art forgery analogy helps to elucidate this process: When the art critic (discriminator) identifies a typical mistake in the forged art

(generator output/fake), the forger (generator) must modify their technique to better blend their work with the genuine masterpieces (data set/real) again.

This dynamic and adaptive error function greatly improves the training of a generator or inverse design network, leading to favorable outputs by all relevant metrics ($\mathcal{S}$ parameter mean square error, classifier output, and individual pixel error).

5.2 Conditional GAN

The GAN methodology is an elegant way to create data that shares characteristics with a given data set. However, it offers limited or no control over the output of the generator, as the discriminator merely requires the output to blend in with the data set. In the context of this thesis, we require a conditional generator that can create unit cells with desired $\mathcal{S}$ parameters and aspect ratios, rather than a generator that just outputs any type of grid (within the parameters of the data set). To continue with the art forgery analogy, this means one could instruct the forger (generator) to produce a specific kind of painting (e.g., a Salvador Dalí) or request a specific motif (e.g., Madonna with child) instead of a painting from just any artist or motif.

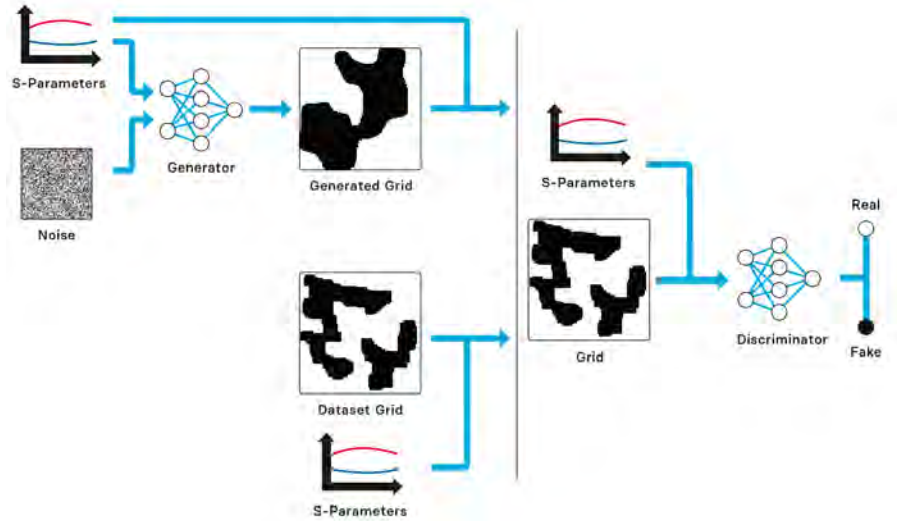


Figure 5.2: Conditional generative adversarial network (in this depiction the aspect ratios $\mathcal{A}$ are ignored). The generator creates meta-atom grids from noise and $\mathcal{S}$ parameters. The discriminator is trained to classify batches of grids and $\mathcal{S}$ parameter pairs from the data set as “real”, while batches containing pairs of desired $\mathcal{S}$ parameters and generated grids are trained to be classified as “fake”. Conversely, the generator is trained to “fool” the discriminator into falsely classifying the generated grids as “real”/coming from the data set. (Courtesy of [155])

The conditional generative adversarial network (CGAN) addresses this need. In our CGAN, the generator $\mathcal{G}$ creates grids based on a combination of noise

$\mathcal{N}$, $\mathcal{S}$ parameters, and aspect ratios $\mathcal{A}$. Conversely, the discriminator $\mathcal{D}$ is presented not only with the grids fabricated by the generator and genuine grids from the dataset but also with the corresponding labels (i.e., $\mathcal{S}$ parameters and aspect ratios $\mathcal{A}$):

$$\mathcal{G}[\mathcal{N}, \mathcal{S}, \mathcal{A}] = g_{\mathcal{G}} \quad , \quad \mathcal{D}[g, \mathcal{S}, \mathcal{A}] = o_{\mathcal{D}}. \quad (5.3)$$

The CGAN framework is depicted in figure 5.2 and the error functions are:

$$\begin{aligned} \mathcal{E}(g, \mathcal{N}, \mathcal{S}, \mathcal{A}) &= \ln(\mathcal{D}(g, \mathcal{S}, \mathcal{A})) + \ln(1 - \mathcal{D}(\mathcal{G}(\mathcal{N}, \mathcal{S}, \mathcal{A}), \mathcal{S}, \mathcal{A})) \\ \mathcal{E}_{\mathcal{D}}(g, \mathcal{N}, \mathcal{S}, \mathcal{A}) &= -\ln(\mathcal{D}(g, \mathcal{S}, \mathcal{A})) - \ln(1 - \mathcal{D}(\mathcal{G}(\mathcal{N}, \mathcal{S}, \mathcal{A}), \mathcal{S}, \mathcal{A})) \\ \mathcal{E}_{\mathcal{G}}(g, \mathcal{N}, \mathcal{S}, \mathcal{A}) &= \ln(1 - \mathcal{D}(\mathcal{G}(\mathcal{N}, \mathcal{S}, \mathcal{A}), \mathcal{S}, \mathcal{A})), \end{aligned} \quad (5.4)$$

where g is a grid from the data set.

The discriminator in this CGAN architecture accepts the grids as input and utilizes two distinct groups of convolutional layers or alternatively one DenseNet to analyze them. One of these groups is configured as a residual network (ResNet) [116], while the alternative could be a standard CNN. These are combined with subsequent fully connected layers, where the $\mathcal{S}$ parameters and aspect ratios $\mathcal{A}$ are also integrated in the evaluation.

5.2.1 Improved CGAN

The network graph depicted in figure 5.2 was the starting point for our research efforts involving CGANs in nanophotonics. However, in the course of our work, we discovered that differentiating between grids produced by the generator and those from the data set, while simultaneously evaluating whether the grids possess the desired optical properties and adhere to the required fabrication constraints, proved to be a challenging task for the discriminator (refer to chapter 5.3 for empirical evidence). To overcome these shortcomings we integrated two pre-trained expert/advisor networks, namely the forward network $\mathcal{F}$ and the classifier network $\mathcal{C}$ (refer to chapter 4), into the discriminator. During the training of the CGAN, the parameters (weights and biases) in these two networks were frozen.

Returning once more to the art forgery analogy, these two expert/advisor networks can be compared to two specialized art experts. One may be an expert in discerning various kinds of paints, to such a level of detail that they can identify the specific ingredients used in the paint's creation. The other might be trained to know everything about different painting techniques. The art critic (discriminator) can then present the art in question (grid g) to these two experts (forward and classifier models) and utilize their opinions (outputs of the advisor models) to arrive at a more informed conclusion as to whether the piece of art is a forgery (generator output/fake) or a true masterpiece (data set/real).

The two expert models are incorporated into the discriminator by feeding the grid g to the two networks and forwarding their outputs to a hidden layer of the discriminator (a visualization can be found in figure 5.3).

$$\mathcal{D}[g, \mathcal{F}(g), \mathcal{S}, \mathcal{C}(g), \mathcal{A}] = o_{\mathcal{D}}. \quad (5.5)$$

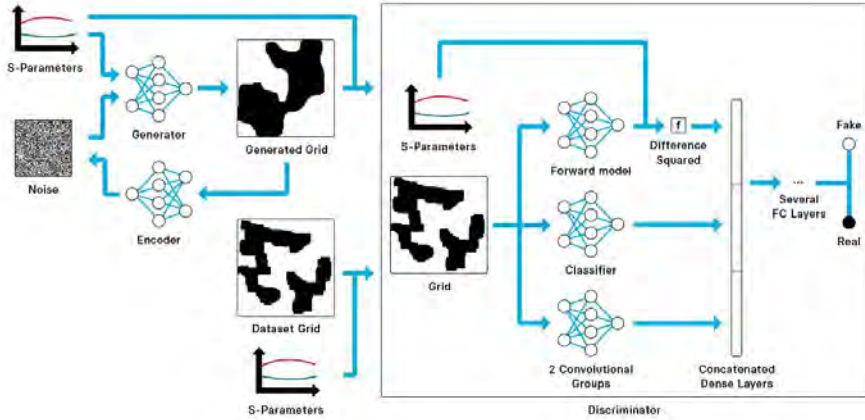


Figure 5.3: The conditional generative adversarial network model. The generator creates meta-atom grids from noise and $\mathcal{S}$ parameters, while the encoder prevents mode collapse. Two expert networks (forward and classifier model) are integrated inside the discriminator. In this depiction the aspect ratios $\mathcal{A}$ are ignored. (Courtesy of [155] and reprinted from [38])

The integrated expert networks introduce biases into the convergence process. To minimize this effect, we randomly select a network from a set of well-performing networks for each of the two expert networks at the start of each training process (optionally they can be replaced at any point during training). The integration of these expert networks significantly enhances the performance of the discriminator and, consequently, improves the efficiency and results produced by the generator.

5.2.2 Inputs

In this section, we will delve deeper into the sets of inputs that are fed to the generator and discriminator. These inputs enable the control over the generator’s output and are responsible for the prefix “conditional” in CGAN. The control over the generator’s outputs is a direct consequence of the discriminator’s ability to not only discern whether a grid is a reasonable unit cell design, compared to noise (compare grids in figure 4.3), but also whether the conditional inputs fed to the generator and discriminator match with the properties of the generated grid.

All the inputs discussed in the following section ($\mathcal{S}$ parameters, maximal absolute differences between neighboring frequencies $\mathcal{S}_r$, label triggers $\mathcal{T}$, and smallest island sizes $\mathcal{I}$) that are fed to the generator and discriminator are simply copied from the data set batch, which is fed to the discriminator in the same iteration of the training loop to ensure that the inputs given to the generator have a valid solution which exists in the data set.

$\mathcal{S}$ parameters

This computational comparison of the grid labels ($\mathcal{S}$ parameters and aspect ratio $\mathcal{A}$) could be done solely in a series of dense layers in the discriminator, but we found that a few simple mathematical operations can guide the discriminator's evaluation to great effect. Therefore, the ground truth labels or generator input $\mathcal{S}$ parameters and the output of the forward model $\mathcal{F}(g)$ are subjected to basic mathematical operations f , before feeding them to a hidden layer of the discriminator. The operation f includes passing on the input $\mathcal{S}$ parameters, evaluating the difference between the $\mathcal{S}$ parameter labels and the output of the forward network, squaring these differences, followed by multiplication with an optional input mask. This simple operation eases the discriminator's comparison of the labels and the (by the forward model) approximated $\mathcal{S}$ parameter by feeding also the error and the squared error to the hidden layer.

An input mask ($\mathcal{S}_m$) enables the option of specifying only a subset of desired $\mathcal{S}$ parameters, a feature implemented to accommodate the fact that in the design of a metasurface, one does not always care about defining all desired $\mathcal{S}$ parameters for every given frequency. For instance, one might only be interested in the amplitude and phase of the polarization-converted part of the transmitted incoming beam at a subset of all frequencies. Thus, the generator's ability to accept only a subset of all $\mathcal{S}$ parameters is a beneficial feature, since it increases the generator's design freedom. To achieve this, we introduce an input $\mathcal{S}$ parameter mask $\mathcal{S}_m$, which consists of a binary table of the same size as the $\mathcal{S}$ parameter table ($\#$ -frequencies $\times$ 17). This mask is generated semi-randomly¹ (ranging from all zeros to all ones) and is used as an additional input to both the generator and the discriminator. The combination of the input mask $\mathcal{S}_m$ and $\mathcal{S}$ parameters is denoted simply as $\mathcal{S}$.

Furthermore, the model allows for the utilization of the $\mathcal{S}$ parameters of not just one incoming polarization, but also the orthogonal polarization states ($S_{11}, S_{12}, S_{13}, S_{14}$ and $S_{21}, S_{22}, S_{23}, S_{24}$). This increases the inputs to the networks, generally boosting their performance and control. Simultaneously, the input mask preserves the flexibility to disregard one of the incoming polarizations if needed. This can be conveniently achieved by setting all the values in one of the input masks to zero, thereby providing an elegant balance between complexity and control within the design process.

Building on the foundational concepts laid out in section 4.1, we now discuss the practical application of the maximal absolute differences between neighboring frequencies ($\mathcal{S}_r$). The $\mathcal{S}_r$ s give us an indication of how narrow a resonance in a unit cell design is. In this context, we would like to have control over the strength and narrowness of resonances in the generated designs. To this end, we can feed the $\mathcal{S}_r$ labels to the generator and discriminator. In the course of the training, the generator will try to match the properties of the generated designs to the label inputs. Thus, the generator tries to match the forward model approximated $\mathcal{S}$ parameters of the unit cell ($\mathcal{F}(g)$) to the input $\mathcal{S}$ parameters. In the case of the resonance narrowness $\mathcal{S}_r$, we are not necessarily interested in getting exactly the input values reflected in the design properties, but would rather like to set a lower limit for the narrowness

¹In addition to completely random configurations some use case specific masks like setting the values in the mask corresponding to one amplitude (e.g. $\mathcal{S}_{31}$ for all frequencies) to 1.

(which is the same as setting a upper limit for the $\mathcal{F}(g)_r \approx \mathcal{S}_r$). To this end, alongside the $\mathcal{S}_r$ inputs, a random number, ranging from 0 up to the maximum value of that particular label within the data set, is added to about half of the individual $\mathcal{S}_r$ labels. By randomly adding this number, an upper limit (data set values plus random number) is effectively set for the forward model approximated $\mathcal{F}(g)_r$. This process sets an effective maximum limit since the discriminator is presented with $\mathcal{S}_r$ inputs that are equal or higher then the predicted values $\mathcal{F}(g)_r$ from the forward model. This incentivises the generator to adhere to a maximum level of the resonance strength, since the data set inputs approximately obey this limit and, therefore, the discriminator can punish violations of this limit. Thereby, an element of control to the generation process is introduced. Setting upper or lower limits in this way will be discussed and visualized again below for the smallest island size $\mathcal{I}$.

The function f , processing the elements forwarded to the discriminator's hidden layer is, given by:

$$f(\mathcal{F}(g), \mathcal{S}) = [\mathcal{S}_m \mathcal{S}, \mathcal{S}_m(\mathcal{F}(g) - \mathcal{S}), \mathcal{S}_m(\mathcal{F}(g) - \mathcal{S})^2, \mathcal{S}_r, \mathcal{F}(g)_r] \quad (5.6)$$

Aspect ratio $\mathcal{A}$

Besides the generator and discriminator inputs associated with the $\mathcal{S}$ parameters detailed above, the smallest island size $\mathcal{I}$ and classifier label triggers $\mathcal{T}$ (see section 4.2) are used as inputs for both networks, modulating the smallest island size and aspect ratio of the produced unit cells. We use $\mathcal{A}$ as an umbrella term for $\mathcal{I}$ and $\mathcal{T}$. When feeding $\mathcal{I}$ to the generator and the hidden layer of the discriminator, in about half of the cases a random number (r) is subtracted from the data set label $\mathcal{I}$. This random number r is in a range², such that the input ($\mathcal{I}_{in}$) fed to both networks is between the minimal possible island size of the whole data set ($\mathcal{I}_{min}$) and the given data set label $\mathcal{I}$ ($\mathcal{I}_{in} = \mathcal{I} - r \in [\mathcal{I}_{min}, \mathcal{I}]$). Thereby, it is possible to control the lower limit of the smallest island size of the generator outputs, since the discriminator “observes” that the smallest island size (evaluated by $\mathcal{C}(g)_0$) is either bigger or approximately equal to $\mathcal{I}_{in}$ for the data set. Conversely, if the generator outputs break with that convention, i.e., the smallest island size observed by $\mathcal{C}(g)_0$ is smaller than the input $\mathcal{I}_{in}$, the discriminator will more likely identify it as fake (and therefore, the generator is trained to correct its output accordingly). This concept is illustrated in figures 5.4 and 5.5, where we plot individual data points of $\mathcal{C}(g)_0 - \mathcal{I}_{in}$, i.e., the classifier approximated island size minus the smallest island size input. This is plotted for the data set (g_d) in blue (a) and for individual training runs for the generator inputs and outputs (g_G) in red (b)-(c). In figure 5.4 a random number (r) was subtracted from the smallest island labels $\mathcal{I}$ for about half of the inputs and in figure 5.5 this subtraction was deactivated. The effect of subtracting r from $\mathcal{I}$ ($\mathcal{I}_{in} = \mathcal{I} - r$) can be seen in the spread of the blue data set distribution in figure 5.4(a), while the same values form almost a straight line around zero, due to the accuracy of $\mathcal{C}(g_G)_0 \approx \mathcal{I}$ in figure 5.5(a). Over the course of the training process most of the $\mathcal{C}(g_G)_0 - \mathcal{I}_{in}$ data points rise above zero in figure 5.4(b)-(c), while the same values are mostly located around zero over the course of the training in figure 5.5(b)-(c). The overarching idea

² $r \in [0, \mathcal{I} - \mathcal{I}_{min}]$

here is that the red data distribution approximates the blue data distribution during the course of training, in an attempt of the generator to produce outputs that have the same characteristics as the data set. Therefore we can conclude that we can indeed set a soft lower limit of the smallest island sizes for the generated grids. The red generator output distribution does not produce a perfect replica of the blue data set distribution, because the generator has to fulfill multiple design objectives at the same time. Setting only a lower limit generally leaves the generator with a greater design freedom, which should enable it achieve higher quality outputs regarding the other relevant metrics like, e.g., $\mathcal{S}$ -parameter mses.

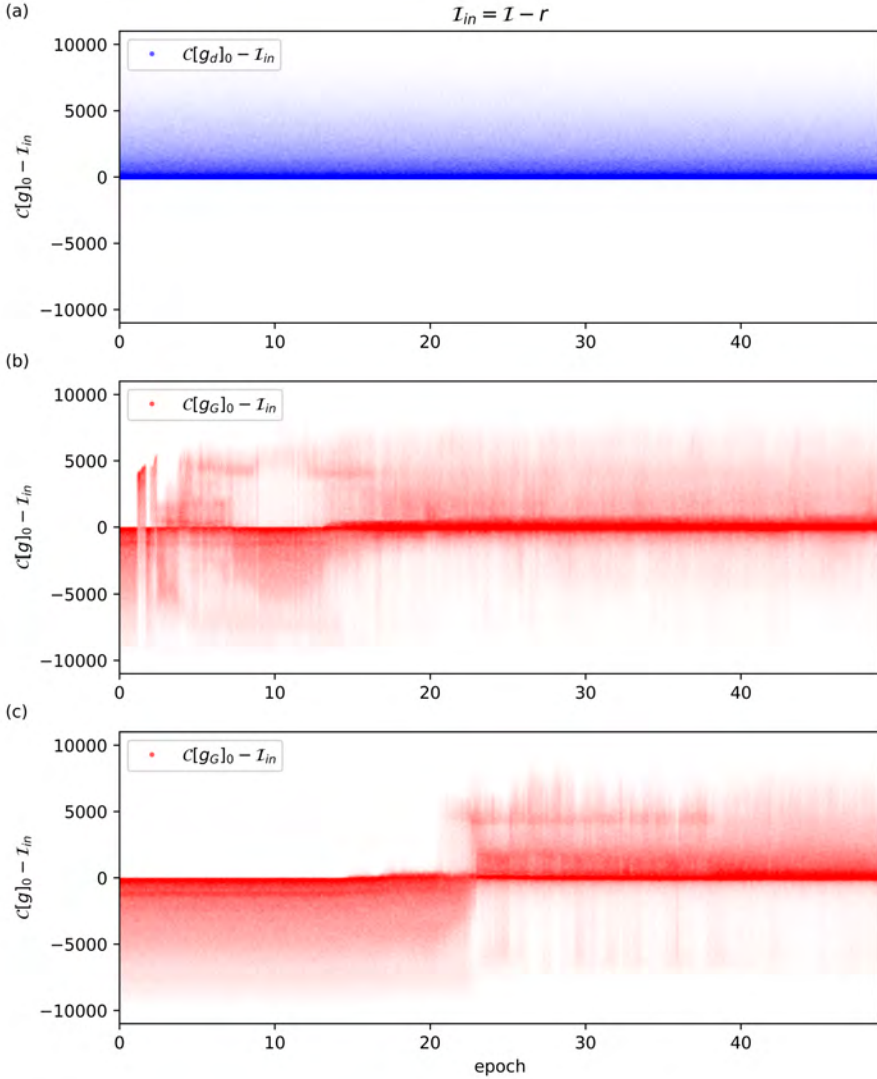


Figure 5.4: Data points of $\mathcal{C}(g_{d/G})_0 - \mathcal{I}_{in}$, where $\mathcal{I}_{in} = \mathcal{I} - r$, (a) g_d are grids/unit cells from the data set and (b)-(c) g_G are generator outputs from individual training runs.

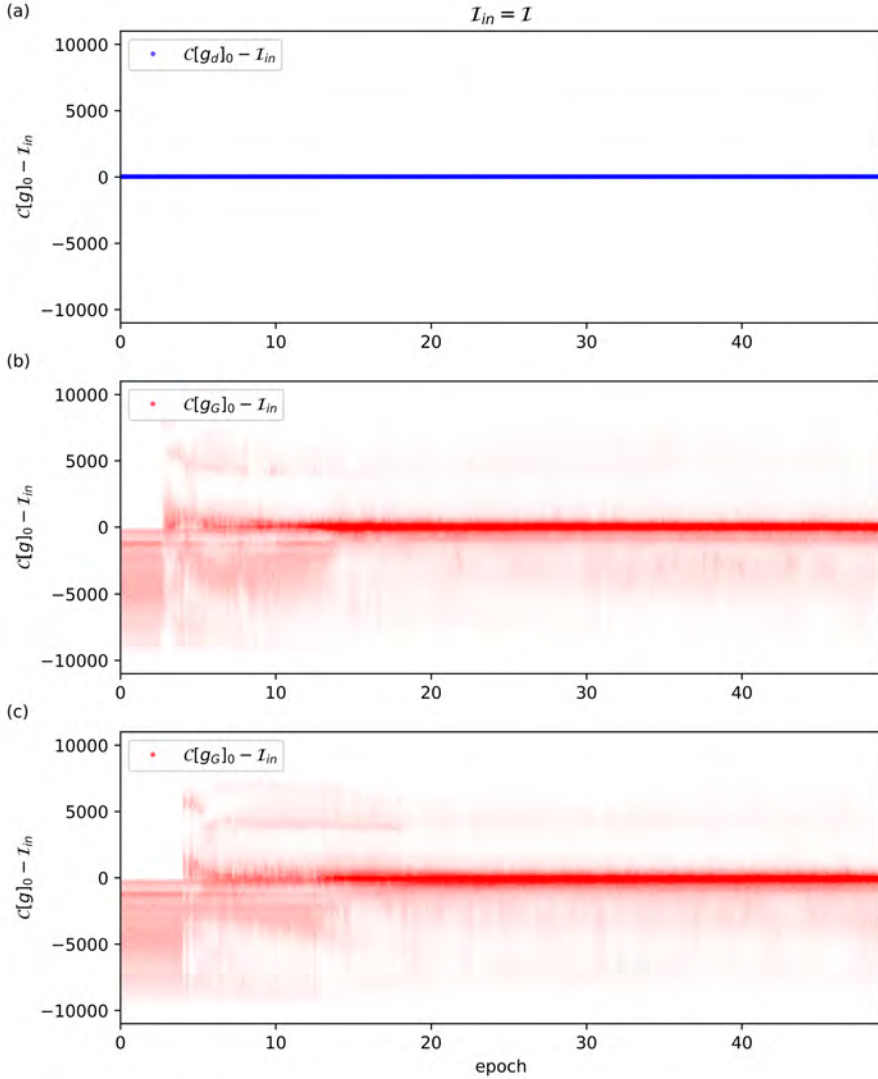


Figure 5.5: Data points of $\mathcal{C}(g_{d/G})_0 - \mathcal{I}_{in}$, where $\mathcal{I}_{in} = \mathcal{I}$, (a) g_d are grids/unit cells from the data set and (b)-(c) g_G are generator outputs from individual training runs.

In order to feed the classifier outputs in a form that is easier to process, similarly to the way $\mathcal{S}$ parameter errors are fed to the discriminator instead of feeding solely raw desired/ground truth labels and approximated $\mathcal{S}$ parameters, only those classifier outputs with $\mathcal{T}_i = 0$ are channeled into the discriminator's hidden layer (see equation 5.7). This selective approach is premised on the fact that classifier outputs $\mathcal{C}[g]_i > 0$ for $i > 0$ violate the desired aspect ratio demanded by the input $\mathcal{T}_i = 0$ to the generator³. Or in other words, when $\mathcal{T}_i = 0$ for $i < k$ and $\mathcal{T}_j = 1$ for $j \geq k$ then only $\mathcal{C}[g]_i$ for $i < k$ are forwarded to

³ $\mathcal{T}_i = 0$ for $i > 0$ follows from $\mathcal{P}_i = 0$, which means that no pixels have to be changed for the aspect ratio $HM/(iD)$. Refer to chapter 4.2 for details.

the discriminator (k is the lowest index with $\mathcal{P}_k > 0$ and, therefore, $\mathcal{T}_k = 1$).

Analogous to setting upper or lower boundaries for $\mathcal{I}$ and the resonance narrowness, we established a maximum boundary on the desired aspect ratio (since lower ratios simplify fabrication). Therefore, a threshold h is randomly selected between the index d (the minimum allowed feature size in the entire data set, i.e., the size of d pixels) and the lowest index k with $\mathcal{T}_k = 1$ for a given grid in the data set. Then, the modification of $\vec{\mathcal{T}}$ is given by: $\mathcal{T}_i = 1 \forall i \in [h, k]$ (and $\mathcal{T}_i = 1$ for $i \geq k$ already).

Let us elucidate this concept by looking at an example derived from the two unit cells shown in figure 4.4. In this example, the smallest feature size present in the data set is given by the length of five pixels, therefore, $\mathcal{P}_i = \mathcal{T}_i = 0 \forall i < 6$ for all unit cells in the data set, thus $d = 6$. For grid (a), $k = 6$, meaning that this grid has the smallest possible feature size of the data set ($k = d$) and since h is randomly picked between k and d , no change in the $\vec{\mathcal{T}}$ inputs can be made for this grid. For grid (b) of figure 4.4, the situation is different. Here, $k = 9$, since $\mathcal{P}_i = \mathcal{T}_i = 0 \forall i < 9$, and $d = 6$ is unaffected. Now, h is randomly chosen between $d = 6$ and $k = 9$ (including 6 and 9). Let us assume $h = 7$, then the resulting input vector is $\vec{\mathcal{T}} = [0, 0, 0, 0, 0, 0, 1, 1, 1, 1]$ instead of the ground truth label $\vec{\mathcal{T}} = [0, 0, 0, 0, 0, 0, 0, 0, 1, 1]$. This new input vector is then fed alongside other randomly selected and partly modified labels in a batch to the generator. In the same iteration of the training loop, the same labels are fed to the discriminator in combination with the respective data set unit cells.

The function t processing the elements fed to the discriminator's hidden layer is given by

$$t(\mathcal{C}(g), \mathcal{A}) = [\mathcal{I}, \mathcal{C}[g]_0, \mathcal{T}_i, \mathcal{C}[g]_i \cdot (1 - \mathcal{T}_i)] \quad \text{for } i > 0 \quad (5.7)$$

Rewriting equation 5.5 with f and t gives

$$\mathcal{D}[g, f(\mathcal{F}(g), \mathcal{S}), t(\mathcal{C}(g), \mathcal{A})] = o_{\mathcal{D}}. \quad (5.8)$$

Here, we can go back to the art forgery example one last time to help understand the utility of the functions f and t . In this analogy, the forward and classifier models were previously compared to experts specialized in kinds of painting material and painting techniques. One could imagine a situation where the knowledge of the experts is so specialized that the art critic does not understand all the details about the ingredients of paint ($\mathcal{S}$ parameters) and the minute differences in painting techniques (fabrication constraints) which are nevertheless relevant to judge the integrity of the origin of the painting (generator output or data set). Then it makes sense to present the reasoning and insight (outputs of the forward and classifier models) in a form (functions f and t) in which the art critic can process and understand the relevant information and conclusions of the experts ($\mathcal{S}$ parameter errors and aspect ratio transgressions) to come to a sound evaluation of the origin of a given painting (unit cell design).

In terms of performance, our CGAN attains a mean square error of approximately 0.004 to 0.008 between the aspired and actual output of the $\mathcal{S}$ parameters. This represents an error margin that is roughly two to four times that of the forward model error, underscoring the efficiency of the CGAN in the generation of metasurfaces that adhere closely to the specified parameters.

5.2.3 Encoder

Similarly to traditional GANs, our continuous-label CGAN is also prone to a phenomenon called mode collapse. As previously elaborated, this leads to a situation where the generator’s outputs show little or no dependence on the input noise. Consequently, the generator maps only to a restricted subset of the surrogate model, as well as the original data set. The susceptibility to mode collapse is further amplified by the continuous nature of the labels, which inherently requires a relatively broad diversity of the generated grids. This situation sometimes leads to the discriminator’s inability to adequately penalize this dynamic.

To mitigate this issue, an encoder network $\mathcal{O}$ has been introduced. This network is in some cases concurrently trained with the generator to map the generated grids and corresponding labels back to the noise input of the generator. The integration of this encoder network adds a layer of complexity to the model while striving to alleviate the challenges associated with mode collapse.

$$\mathcal{O}[\mathcal{G}(\mathcal{S}, \mathcal{N}), \mathcal{S}] = \mathcal{O}[g_{\mathcal{G}}, \mathcal{S}] \approx \mathcal{N}. \quad (5.9)$$

The resulting error of the encoder is added to the loss function of the generator. In this way, the generator is incentivized to encode the noise within the generated grids in such a way that the encoder can subsequently map it back. The outcome of this strategy is a more diverse array of generated grids (see figure 7.2), enriching the scope and applicability of the model.

5.3 Training optimization techniques

Now we will discuss the effects of the techniques listed in section 4.3 applied to the discriminator in addition to further optimization that we experimented with.

5.3.1 Discriminator

In this section we evaluate the performance of the generator based on the $\mathcal{S}$ -parameter accuracy, the classifier output of the generated meta atoms, and the pixel error (see chapter 4.4), which is a measure of how blurry the outputs are. We explored different techniques for improving the stability of the discriminator relative to the generator. The most important parameters in this context are the optimizer and the learning rates. For the generator we used the Adamax optimizer (see chapter 2.1) with learning rate $\in [5 \cdot 10^{-5}, 2 \cdot 10^{-4}]$, $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 10^{-7}$ and for the discriminator Adam with learning rate $\in [5 \cdot 10^{-8}, 8 \cdot 10^{-7}]$, $\beta_1 = 0.333$, $\beta_2 = 0.999$, $\epsilon = 10^{-7}$. These parameters are chosen such that the discriminator generally has a slight advantage over the generator in the long term.

Expert model integration: As explained in section 5.2.1, we integrated the forward model and the experimental-feasibility classifier in the discriminator of the CGAN in order to improve the discriminator’s and thereby the generator’s performance. The comparisons in performance can be assessed in figure 5.6 and 5.7, respectively. With the integration of the (pre-trained) forward model enabled, the $\mathcal{S}$ parameter error decreases significantly during the training,

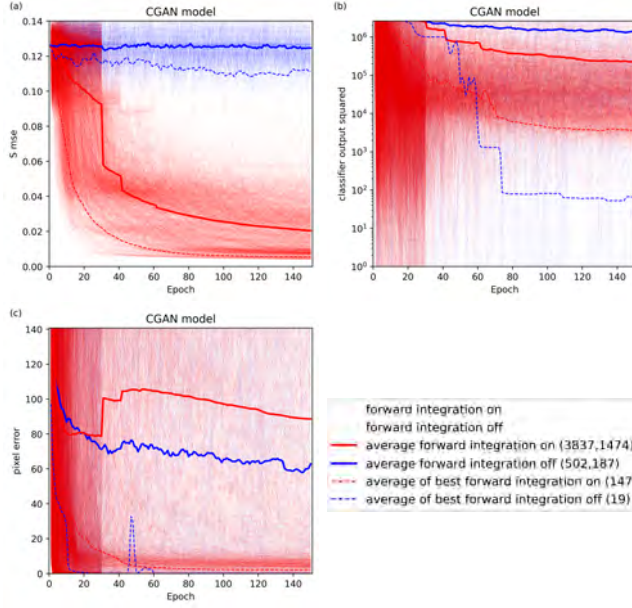


Figure 5.6: Comparison of the effects that the integration of the forward network in the Discriminator has on (a) S parameter mse, (b) classifier output, and (c) pixel error. In the legend, numbers after overall averages denote the total number of training instances and those reaching epoch 150. Similarly, the figure following the best training instance average indicates the number of instances used for that average.

while the training does not converge with the integration of the forward model disabled. This shows that the forward model in the discriminator is essential for CGAN-based inverse design. Disabling the classifier integration in the discriminator increases the S parameter accuracy of the generator outputs, since the generator is getting significantly less punished for unfeasible grid designs and, therefore, has greater freedom to find lithographic masks that possesses the desired optical properties. However, as can be seen from the classifier outputs, these designs tend to be harder or impossible to manufacture. Similarly to the forward network, the classifier is essential in achieving masks that satisfy the fabrication constraints.

The parts of the data set where either the integration of the forward or the classifier or both were disabled are excluded from the comparisons below, since some of the conditions for early training terminations were disabled and those modification alter the behaviour of the discriminator too drastically and would therefore pollute the other data.

DenseNets: To enhance the discriminator’s analytical capabilities and improve its ability to assess training data and generator outputs while identifying discrepancies, a DenseNet [117] was incorporated into the discriminator (network architectures can be found in figure S5 and S6 in paper B). The effects of the DenseNet compared to a standard convolutional network can be seen in figure 5.8. The improvements that we expected did not materialize and, so far,

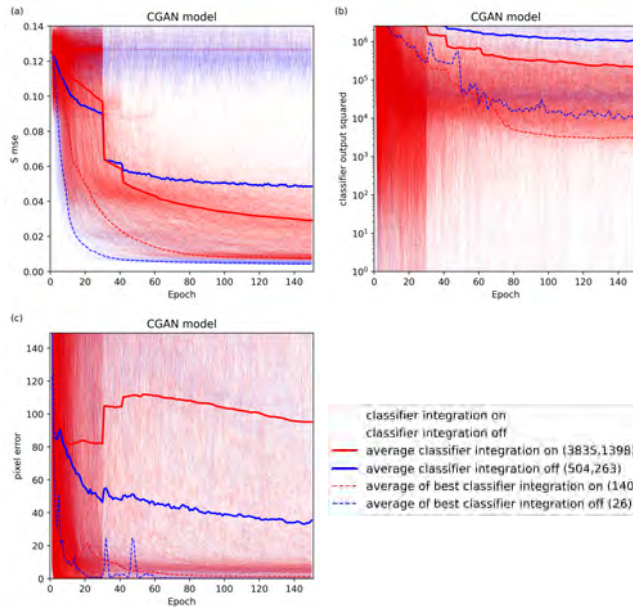


Figure 5.7: Comparison of the effects that the integration of the experimental feasibility classifier network in the Discriminator has on (a) $\mathcal{S}$ parameter mse, (b) classifier output, and (c) pixel error.

we were not able to explain this result.

Discriminator freezing: In an attempt to balance the advantage of the discriminator in the long term, we tried skipping the discriminator training in each batch with a probability proportional to the generator loss in the previous batch. However, we never allow the probability of skipping discriminator training to reach one, as this results in a static discriminator and consequently a static error landscape of the generator, which can ultimately lead to a stagnation of the CGAN learning process. Adaptive skipping of the discriminator training proved to be essential in stabilizing the loss function ratio of the generator and discriminator. However, the adaptive skipping of the discriminator training (freezing) did not improve the generator outputs as shown in figure 5.9, even though the loss functions of discriminator and generator stayed closer together. The $\mathcal{S}$ parameter error converges more slowly when discriminator freezing is enabled, although it asymptotically approaches the same values. The pixel error and the classifier outputs are also overall worse when freezing is enabled.

Blurring: In our experiments, we also employed blurring of the training set, albeit to a lesser extent. This approach aligns with the blurring used when training the forward and classifier models. By incorporating blurring during these early stages, we aim to facilitate the training of the generator network. This strategy was thought to help the generator network to gradually learn and produce more accurate and refined outputs as the training progresses. In reality, blurring did not make a significant difference in training performance as can be seen in figure 5.10 and we therefore cannot recommend this technique in the discriminator of CGAN-based inverse design.

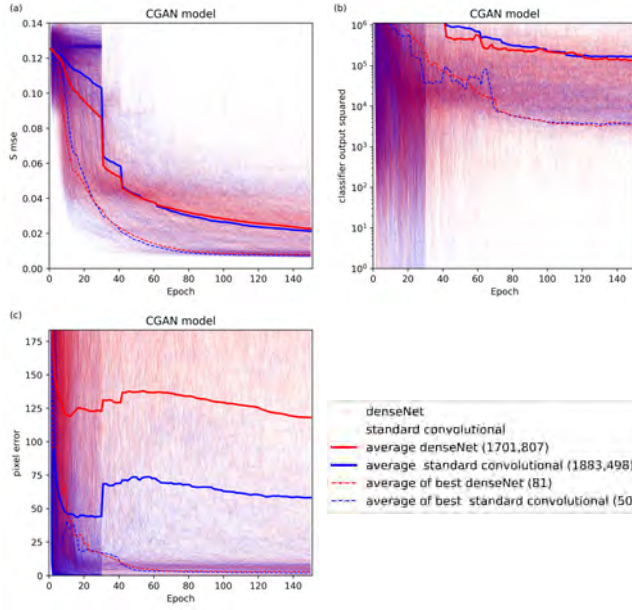


Figure 5.8: Comparison of the effects that a DenseNet or a standard convolutional network in the Discriminator has on (a) S parameter mse, (b) classifier output, and (c) pixel error.

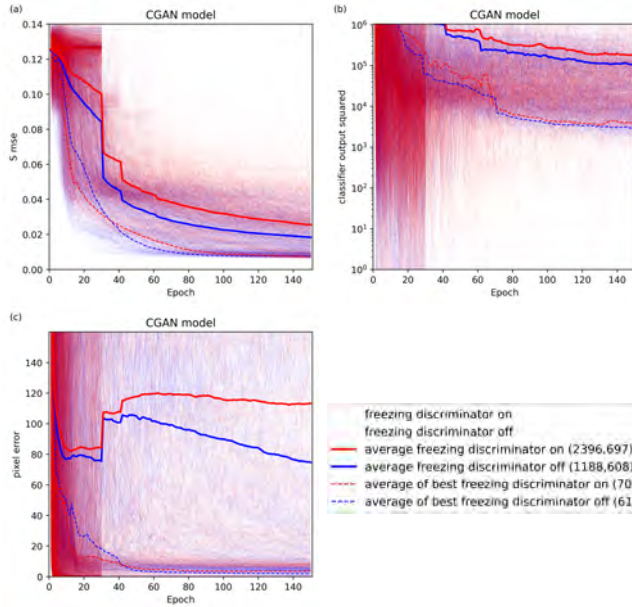


Figure 5.9: Comparison of the effects that conditional freezing of the discriminator has on (a) S parameter mse, (b) classifier output, and (c) pixel error.

Gaussian noise: Along with the data blurring, various types of Gaussian

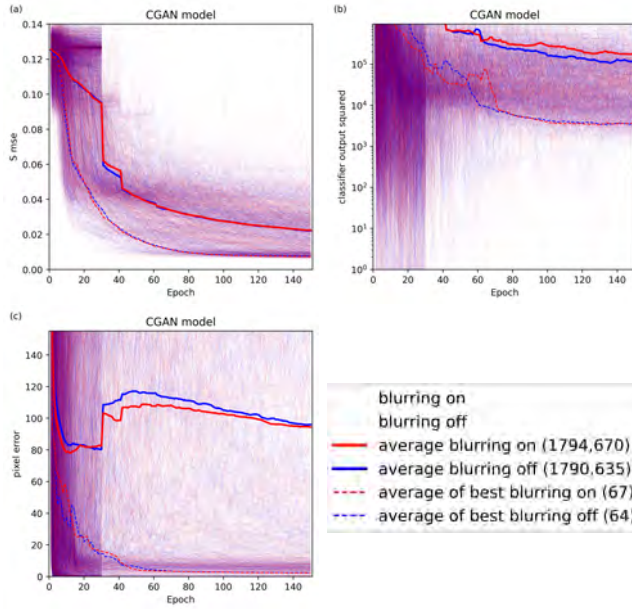


Figure 5.10: Comparison of the effect that blurring of the training data in the initial 20 epochs has on (a) S parameter mse, (b) classifier output, and (c) pixel error.

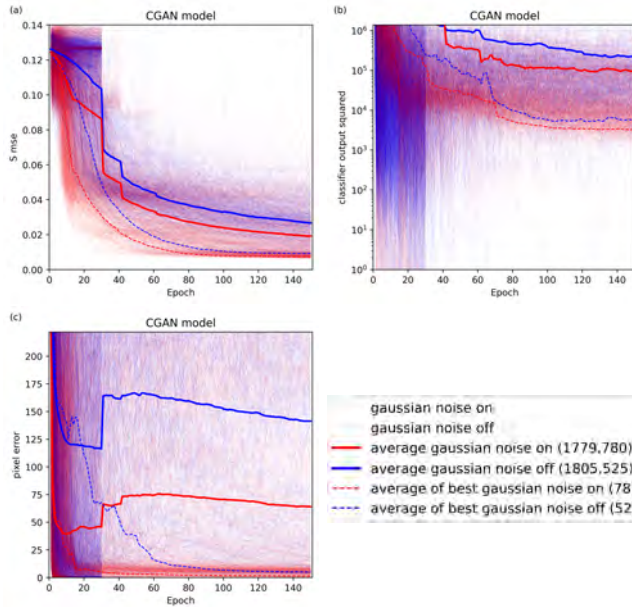


Figure 5.11: Comparison of the effects that Gaussian noise to the various input channels of the discriminator has on (a) S parameter mse, (b) classifier output, and (c) pixel error.

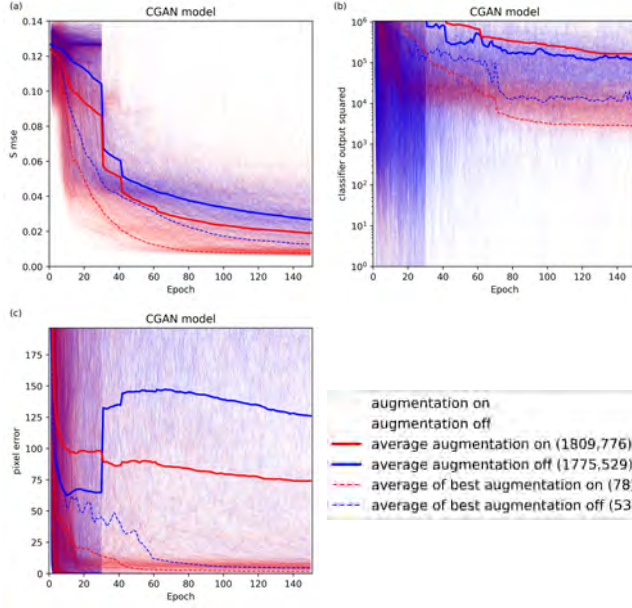


Figure 5.12: Comparison of the effects that augmentation of the training data and generator outputs fed to the discriminator has on (a) $\mathcal{S}$ parameter mse, (b) classifier output, and (c) pixel error.

noise [99, 124–126] were introduced to the three input channels of the discriminator during the first few epochs. We apply noise with a larger standard deviation in the forward and classifier models’ input channels compared to the discriminator’s own convolution input channel (starting at 1.1 and 0.3, respectively) and decreasing over the first 15 epochs. Thereby, compelling the discriminator to rely not only on the output of the two advisor networks but also train its own CNN. This added noise proved to be crucial in improving the generator output with regards to all metrics (see figure 5.11).

Data augmentation: We also investigated using data augmentation (see chapter 4.3) for all the inputs to the discriminator (both inputs coming from the generator and from the training data set). Here, we introduced the same random translation of the unit cell inputs as discussed in section 4.3). From figure 5.12, we see that data augmentation improves the performance of the CGAN substantially. The $\mathcal{S}$ parameter mse and the pixel error converge faster when data augmentation is enabled. For the classifier output, we see that data augmentation leads to a slightly larger average, but also that the best models perform better with data augmentation. This indicates a larger variability and demonstrates the importance of training sufficient different models in order to find one of the well-performing models.

Furthermore, we automated the collection of generator inputs and outputs from all independent CGAN training instances and saved them with the current epoch as label. Random samples are taken from epochs in a defined interval around the current epoch, which are then used to train the discriminator on a diverse set of data after each epoch. The effect of this was slightly positive,

though not significant (figure 5.13). Additionally, if the generator error of the current batch in any epoch becomes too high, the discriminator is trained in the subsequent batch on outputs from its own previous epochs instead of the current outputs if freezing of the discriminator is enabled or with a small random change otherwise. The effects of this were also slightly positive, albeit insignificant as shown in figure 5.14. Both alternative training schemes utilized another optimizer with a lower learning rate $\in [8 \cdot 10^{-10}, 1 \cdot 10^{-8}]$, $\beta_1 = 0.333$, $\beta_2 = 0.999$, $\epsilon = 10^{-7}$.

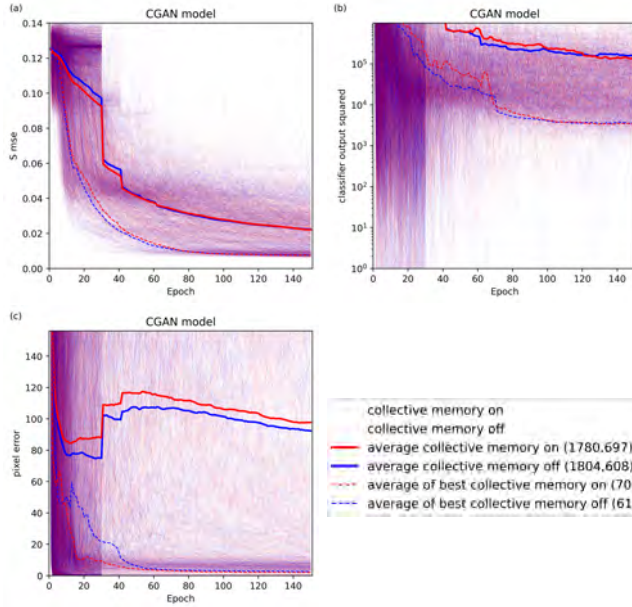


Figure 5.13: Comparison of the effects that the discriminator training on other generator outputs has on (a) $\mathcal{S}$ parameter mse, (b) classifier output, and (c) pixel error.

5.3.2 Generator

Finally, we studied how the network architecture of the generator can enhance its stability and performance. We increased the number of neurons in the fully connected layers from $2^9 - 2^{10}$ to $2^{10} - 2^{11}$, finding that anything beyond this range caused significant instability (network architecture can be found in figure S12 in paper B). In terms of the transverse convolutional network architectures, we conducted experiments using different configurations. The simplest architecture consisted of standard networks with batch normalizations applied between the transverse convolutional layers (called DefaultNet in figure 5.15, architecture can be found in figure S14 in paper B). Additionally, we explored four groups of transverse convolutional layers, incorporating varying numbers of batch normalizations between the respective groups (referred to as FourNet in the figure 5.15, architecture can be found in figure S15 in paper B). In the third architecture, we introduced transverse convolutional

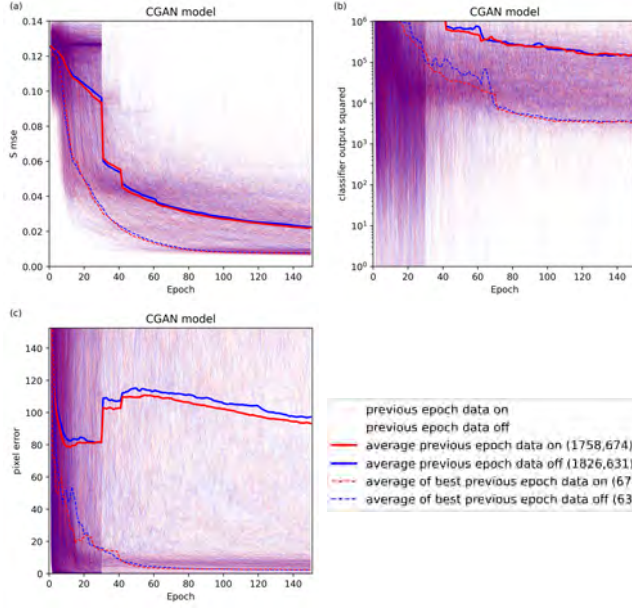


Figure 5.14: Comparison of the effects that the discriminator training on previous epochs of the same generator outputs has on (a) $\mathcal{S}$ parameter mse, (b) classifier output, and (c) pixel error.

connections between the groups (HelixNet in the figure 5.15, architecture can be found in figure S16 in paper B). These different architectures are compared in figure 5.15. We find that the HelixNet performs best, although the improvements are rather small. In order to further refine the outputs, we also introduced a DenseNet block at the end of the generator’s transposed convolution layers (the architecture of the DefaultNet with a DenseNet block can be found in figure S17 in paper B). This addition helped to enhance the quality and accuracy of the generated outputs by a small margin, contributing to the overall success of the network (figure 5.16).

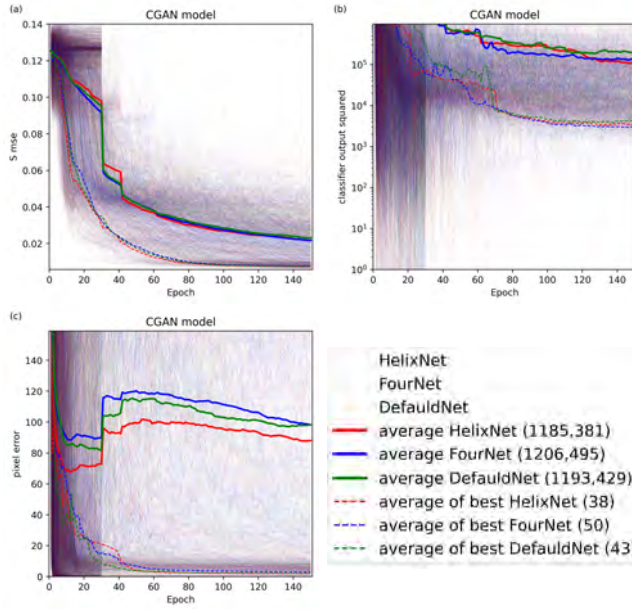


Figure 5.15: Comparison of the different transverse convolutional network architectures in the generator. (a) $\mathcal{S}$ parameter mse, (b) classifier output, and (c) pixel error.

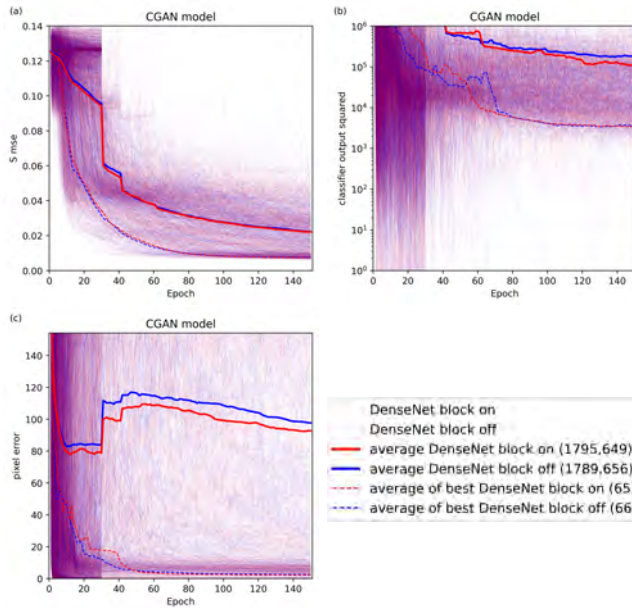


Figure 5.16: Comparison of the effects that a DenseNet block at the end of the latent vector decoder has on (a) $\mathcal{S}$ parameter mse, (b) classifier output, and (c) pixel error.

5.4 Generator with multiple interdependent outputs

We now discuss how the CGAN model for inverse design can be adapted to not only create one meta-atom g_0 with desired scattering parameters $\mathcal{S}_0$, but also $N \in \mathbb{N}$ additional meta-atoms $g_{1...N}$ with interdependent $\mathcal{S}$ parameters $\mathcal{S}_{1...N}$, where $\mathcal{S}_n = \phi_n(\mathcal{S}_0)$ with $n \in [1, N]$ (e.g., $\phi_n(\mathcal{S}_0)$ could be $\phi_1^n(\mathcal{S}_0)$, a phase rotation with constant amplitude, for beam steering). Generally, $\mathcal{S}_0$ has to be in the manifold $\mathbb{S}$ of $\mathcal{S}$ parameters spanned by the training data (used to train the discriminator and forward model) to facilitate that the output of the generator is not a hallucination and the generated unit cell g_0 actually approximates the desired $\mathcal{S}$ parameters ($\mathcal{S}_0$) sufficiently well. For the same reason, the function ϕ_n has to be a transformation which projects the set of desired $\mathcal{S}$ parameters $\mathcal{S}_0$ back onto $\mathbb{S}$ ($\phi_n(\mathcal{S}_0) \in \mathbb{S}$). This is not a strict condition in the sense that the generator will still provide output if $\phi_n(\mathcal{S}_0) \notin \mathbb{S}$, but, in the same way that the output g_0 is most likely a hallucination when $\mathcal{S}_0 \notin \mathbb{S}$, the generated unit cells g_n will not approximate the projected $\mathcal{S}$ parameters $\mathcal{S}_n$ if ϕ_n is not an endomorphism on $\mathbb{S}$. Nevertheless, this can be guaranteed by constructing a suitable training data set.

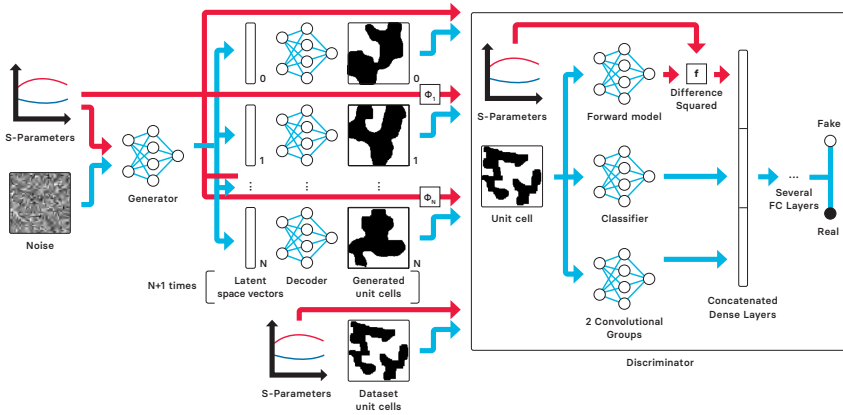


Figure 5.17: Neural network graph of the conditional generative network (CGAN) for the creation of unit cells with interdependent scattering parameters. The red arrows highlight the $\mathcal{S}$ parameter related connections of the graph and the blue arrows depict the rest of the graph. (Courtesy of [155])

In order to output multiple unit cells g_N with interdependent properties, the original generator architecture is split into two networks

$$\begin{aligned} \mathcal{G}_l[\mathcal{S}_0, \mathcal{A}, \mathcal{N}] &= [l_0, \dots, l_N] \\ \mathcal{L}[l_n] &= g_n \\ \mathcal{L}[\mathcal{G}_l[\mathcal{S}_0, \mathcal{A}, \mathcal{N}]] &= [g_0, \dots, g_N], \end{aligned} \tag{5.10}$$

where $\mathcal{G}_l$ is a multiple latent vector generator, which outputs $N + 1$ latent vectors and $\mathcal{L}$ is a latent vector decoder used to decode each latent vector l_n

into a unit cell g_n . All these unit cells get individually fed to the discriminator with ϕ_n transformed $\mathcal{S}$ parameters (ϕ_0 is the identity operation):

$$\mathcal{D}[g_n, \phi_n(\mathcal{S}_0), \mathcal{A}] = O_n. \quad (5.11)$$

The neural network graph of this adapted CGAN set up is visualized in figure 5.17. Subsequently, a weighted sum of the cross entropy losses of the O_n outputs is used in the loss function:

$$\begin{aligned} \mathcal{E}_{\mathcal{D}}(g_D, \mathcal{S}_0, \mathcal{A}, \mathcal{N}) &= -\ln(\mathcal{D}(g_D, \mathcal{S}_0, \mathcal{A})) \\ &\quad - \frac{\sum_{n=0}^N a_n \ln(1 - O_n)}{\sum_{n=0}^N a_n}, \\ \mathcal{E}_{\mathcal{G}}(\mathcal{S}_0, \mathcal{A}, \mathcal{N}) &= \frac{\sum_{n=0}^N a_n \ln(1 - O_n)}{\sum_{n=0}^N a_n}, \end{aligned} \quad (5.12)$$

where the factors a_n are the weights of the generator output g_n (and the respective discriminator output O_n) in the loss function and g_D are grids from the data set. While $a_0 = 1$ for all epochs, a_1 to a_N start out at 0.001 at epoch 0. Thereby, the generator outputs g_1 to g_N play almost no role in the beginning of training, making this effectively a standard inverse design CGAN setup (comparable to the graph in equation 5.4). This improves the generator output g_0 and, in the process, minimizes the $\mathcal{S}$ parameter error $|\mathcal{S}_0 - \mathcal{F}[g_0]|$. Once the $\mathcal{S}$ parameter error sinks under a certain threshold, a_1 to a_N increase as the $\mathcal{S}$ parameter error of g_0 decreases (and approach 0.5 in the limit of the error going to 0). In this way, the generator first learns to output l_0 . Additionally, the latent decoder is trained to output g_0 from l_0 , while the discriminator learns the wide range of characteristics that have to be fulfilled for a meta-atom mask to be a suitable design. Once this works so well that most of a set of training instances would converge to a useful generator, the weight factors a_1 to a_N increase in order to gradually shift the focus and learning of the generator to output l_1 to l_N (note that the same latent decoder network $\mathcal{L}$ is used to produce g_0 to g_N from l_0 to l_N). The unit cells g_1 to g_n will generally approximate the $\mathcal{S}$ parameters $\phi_1(\mathcal{S}_0)$ to $\phi_N(\mathcal{S}_0)$, because these are fed to the discriminator, which is punishing deviations. Thereby, the following sum is minimized

$$|\mathcal{S}_0 - \mathcal{F}[g_0]| + \sum_{n=1}^N |\phi_n(\mathcal{S}_0) - \mathcal{F}[g_n]|. \quad (5.13)$$

In the above neural network graph, we still fix the scattering parameters $\mathcal{S}_0$ of one of the meta-atoms. This is often undesirable as it unnecessarily restricts the design freedom, e.g., for a lens the transmission phase of the center meta-atom can be arbitrary as long as the relative phase compared to the center meta-atom has a quadratic phase as a function of the distance to the center meta-atom. If the forward model $\mathcal{F}$ has the necessary accuracy, we can adapt equation 5.11 by replacing $\phi_n(\mathcal{S})$ with $\phi_n(\mathcal{F}[g_0])$ for $n > 0$:

$$\begin{aligned} \mathcal{D}[g_0, \mathcal{S}_0, \mathcal{A}] &= O_0 \\ \mathcal{D}[g_n, \phi_n(\mathcal{F}[g_0]), \mathcal{A}] &= O_n \quad \forall n > 0. \end{aligned} \quad (5.14)$$

The network graph of this set up is shown in figure 5.18. In this way it is

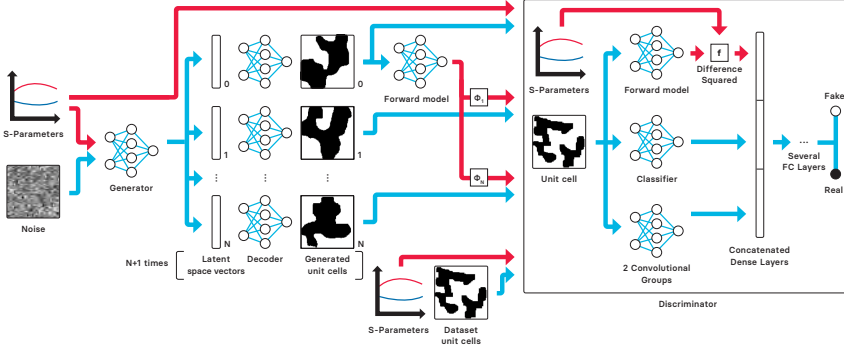


Figure 5.18: Graph of the second version conditional generative network (CGAN) for the creation of unit cells with interdependent scattering parameters using a more adaptive strategy. The red arrows highlight the $\mathcal{S}$ parameter related connections of the graph and the blue arrows depict the rest of the graph. (Courtesy of [155])

possible to give more design freedom to the generator to find a set of unit cells where

$$|\mathcal{S}_0 - \mathcal{F}[g_0]| + \sum_{n=1}^N |\phi_n(\mathcal{F}[g_0]) - \mathcal{F}[g_n]| \quad (5.15)$$

is minimized. In the training process we will generally feed $\mathcal{S}_0$ parameters to the generator and discriminator, although, by utilizing the input mask $\mathcal{S}_m$ for the $\mathcal{S}_0$ parameters it is possible to feed only a subset of all parameters to the network or none at all. Feeding a wide variety of $\mathcal{S}_0$ ($\in \mathcal{S}$) parameters to the networks helps generalizing their capabilities and feeding none ($\mathcal{S}_m = 0 \rightarrow |\mathcal{S}_m(\mathcal{S}_0 - \mathcal{F}[g_0])| = 0$) enables the maximal design freedom of the generator to minimize

$$\sum_{n=1}^N |\phi_n(\mathcal{F}[g_0]) - \mathcal{F}[g_n]|. \quad (5.16)$$

In section 7.4 we discuss some applications and results obtained with this network graph.

5.5 Iterative improvement of training data

Finally, we used an iterative process to enhance the quality of the training data set. This was done by training the classifier, forward, and CGAN models on unfinished fractions of the data set to identify and include training samples located in the more thinly covered regions of the $\mathcal{S}$ -parameter space. The resultant effect is multi-faceted: it increases the precision of all the models involved, minimizes the necessary corrections to the generated grids to meet the feasibility constraints, and refines the $\mathcal{S}$ parameters of the generated grids to more closely mirror the desired targets.

The procedure used for this improvement can be summarized in the following algorithm:

1. **Expert Models Training:** We initiate the process by training both the classifier and forward networks utilizing the existing set of grids, labels, and $\mathcal{S}$ parameters.
2. **CGAN Training:** Multiple CGAN generators are trained, leveraging the classifier and forward models that were trained during the previous step.
3. **Classifier Data Enhancement:** From all the generators involved in this iteration, we generate grids for a diverse spectrum of target $\mathcal{S}$ parameters, aspect ratios $\mathcal{A}$ and noise levels from all training epochs. The individual pixels of the generated unit cells are rounded to values of 0 or 1, appropriately labeled for the classifier, and subsequently incorporated into the classifier training data.
4. **CGAN and Forward Model Data Enhancement:** The quality and fidelity of grids produced by the generator are intrinsically tied to the accuracy of the forward and classifier model. The number of randomly created grids featuring resonances and, e.g., high S_{31} parameters can be relatively rare. Thus, there is a substantial rationale for augmenting the training data set with grids that have $\mathcal{S}$ parameters in the regimes of interest or increasing the data set density in regions of thin $\mathcal{S}$ parameter coverage. Therefore, we try to homogenise the data density as much as possible to improve the accuracy of the CGAN and forward models. For this purpose, a wide array of grids are crafted by the generators of this iteration, corrected if needed, simulated using COMSOL, and then added to the training data set if the designs vary sufficiently from all the designs in the data set.

The application of this iterative scheme yields a noticeable enhancement in the overall performance of the system. It manifests itself in the heightened accuracy of all the constituent models, a reduction in the corrections required to the grids generated by the generator in compliance with feasibility constraints, and a closer alignment of the $\mathcal{S}$ parameters and aspect ratios $\mathcal{A}$ of the generated grids with the intended target parameters.

Chapter 6

Complexity of topological optimization methods

We will now do a number of back-of-the-envelope calculations to compare the complexity of our DNN-based method with that of gradient-descent optimization utilizing the adjoint method. We will compare this for the typical nanophotonic designs considered in this thesis, where a forward simulation takes approximately one hour on average.

The advantages of a purely Artificial Neural Network (ANN) based inverse design model is that it has superior computational post-training efficiency. Therefore, once trained, one can obtain many designs with different desired optical responses at a virtually constant cost without computing additional forward simulations or retraining the networks. The disadvantage of our CGAN method is that it needs a relatively large amount of simulations in the training data set. Contrary to sequential execution of many forward simulations for each separate unit cell in conventional algorithms, the samples from the training data set can be simulated in parallel, since each simulation is independent.

As mentioned in chapter 1, gradient-descent is an iterative method that in each iteration solves a forward problem and an adjoint problem in order to obtain the gradients of the FoM with respect to all design parameters. In general, the adjoint problem is of the same size as the original system and, since it can be factorized using the same lower and upper triangular matrices as the forward problem, it has the same condition number and the same spectrum of eigenvalues. Therefore, an iterative solver can use similar preconditioners, will have similar performance, and consequently, the adjoint problem should be no harder than solving the original problem [156]. We define $A \in (0, 1]$ as the computational cost of the adjoint problem relative to the original one. In nanophotonics, where we often solve systems that are too large for direct solvers and are forced to use iterative solvers due to RAM limitations and the relative speed of iterative solvers compared to direct ones, A is close to 1. The typical gradient-descent execution requires an average of I iteration steps depending on the specific design problem (200 for 2D and nonresonant simulations, 500 for a conservative estimate). Moreover, if the design needs to be a global optimum rather than a local optimum, the algorithm should be restarted R times (R can range from 1 in lower-dimensional scenarios to about

5 for cases with higher complexity) with varying initial design configurations to enhance the likelihood that the run with the superior FoM is the global optimum.

In contrast, a conditional Generative Adversarial Network (CGAN) requires S_N simulations to construct the training data set (for our projects between 40,000 and 80,000 data points¹ with an average simulation time $t(S)$ of one hour) for its training data set and T computational resources to train the networks (we express computational resources in units of the average computational resources needed for a single simulation, i.e., one hour; for simplicity we will also set one CPU node equal to one GPU node). The average training times of the models are the following: 45 minutes for the forward model, 3 hours for the classifier model, 48 hours for the CGAN (on two GPUs), furthermore, it is appropriate to train at least 5 models each and pick the best of each.

6.1 Computational resources

For a single inverse design task (for instance, one unit cell), the adjoint method demands approximately $(1 + A) \times I \times R$ simulations, compared to $S + T$ computational resources in the neural network approach. For more designs, the computational resources required increase linearly for gradient descent, but are constant for a DNN-based method.

Number of designs	Computational resources	
	adjoint method	CGAN
N	$(1 + A) \times I \times R \times N$ $= 2 \times 500 \times 5 \times N$ $= 5000 \times N$	$S + T$ $= 50.000 + 5 \times (1 + 3 + 2 \times 48)$ $= 50500$
1	5000	50500
10	50000	50500
100	500000	50500

Table 6.1: Comparison of computational resources (normalized to the average resource required for one simulation) used in the adjoint method and the CGAN approach.

The neural network has a distinct advantage if multiple designs are needed, e.g., for a metasurface: once trained, it can generate any number N of designs within the design parameters space spanned by the training data set, possessing varying optical properties, at virtually negligible computational cost compared to $S + T$. To conclude, if $(1 + A) \times I \times R \times N > S + T$, it is computationally more efficient to deploy a neural network approach. For a typical nanophotonic example, the break even point from this calculation is around 10 designs. This number varies dependent on the specifics of the underlying design task.

¹The 40,000 simulations were done for one polarization, whereas 150,000 simulations were evaluated to create a mixture of orthogonal polarizations for close to 80,000 grids.

6.2 Wall-clock time

Wall-clock time refers to the actual time taken by a process, as measured by a clock on the wall. In other words, it's the total time elapsed from start to finish. This time depends on the available resources and the degree to which the computation can be parallelized. Here, we first consider these to be infinite before considering some more realistic constraints. Without constraining the number of available CPUs, GPUs and simulation program licences, designing any number of unit cells takes $500 \times t(S) = 500$ hours with the adjoint method and $t_{max}(S) + t(T_{fwd}) + t(T_{CGAN}) = 24h + 1h + 48h = 73h$,² where $t_{max}(S)$ is the maximum time of any simulation in S , $t(T_{cl})$ is the average time for training a classifier model, and $t(T_{CGAN})$ is the average time for training a CGAN model. In practice, we cannot compute all training data samples simultaneously. Let us consider a bottleneck maximum of S_M simulations that can be run in parallel, either due to the number of available CPUs or licences and T_M available GPUs. The wall clock time of the adjoint method is estimated by

$$t_{AM} = I \times \max \left(\frac{(1 + A) \times R \times N \times t(S)}{S_M}, t(S) \right), \quad (6.1)$$

where $t(S)$ is the average wall clock time of one simulation. In comparison, the wall clock time of our CGAN approach can be estimated by

$$\begin{aligned} t_{NN} = & \max \left(\frac{S \times t(S)}{S_M}, t_{max}(S) \right) \\ & + \max \left(\frac{t(T_{fwd}) \times N_{fwd} + t(T_{cl}) \times N_{cl}}{T_M}, t(T_{fwd}), t(T_{cl}) \right) \\ & + \max \left(\frac{t(T_{cgan})}{T_M/2}, t(T_{cgan}) \right), \end{aligned} \quad (6.2)$$

where $t(T_{fwd})$ is the average time of the forward model training time, N_{fwd} is the number of forward models to train, N_{cl} the number of classifier models to train, $t(T_{cgan})$ is the time it takes to train a CGAN model, and this time is divided by $T_M/2$ because two GPUs are necessary in the training process of the CGAN. Table 6.2 provides a comparison of t_{AM} and t_{NN} , where all the wall clock times and other parameters such as the number of simulations are set to the values given above while plugging in different values for the number of designs, available CPUs and GPUs ($S_M = T_M$) in to the formulas.

Therefore, the current state of CGAN approaches and traditional algorithms like the adjoint method fill different niches.

6.3 Additional remarks

In addition to the comparison of the two methodologies with respect to computational resources and wall clock times, the quality of the produced designs is essential. This comparison is unfortunately out of the scope of this thesis, due to the singular focus on the CGAN methodology. The quality of the CGAN

²all simulations can be run in parallel, and forward and classifier can be trained at the same time before training the CGAN

Number of designs	Number of computing nodes	Wall clock time in hours	
N	$S_M (= T_M)$	t_{AM} adjoint method	t_{NN} CGAN
1	1	5,000	50,000 + 20 + 240
10	1	50,000	50,000 + 20 + 240
100	1	500,000	50,000 + 20 + 240
1	10	500	5,000 + 3 + 48
10	10	5,000	5,000 + 3 + 48
100	10	50,000	5,000 + 3 + 48
1	100	500	500 + 3 + 48
10	100	500	500 + 3 + 48
100	100	5,000	500 + 3 + 48
1	1000	500	50 + 3 + 48
10	1000	500	50 + 3 + 48
100	1000	500	50 + 3 + 48

Table 6.2: Comparison of wall clock time elapsed in the adjoint method and the CGAN approach. The constants used in this evaluation depend on the specific design task. $500h \approx 20d$, $5,000h \approx 208d \approx 7m$, $50,000h \approx 2083d \approx 5y + 8m$

outputs, among other factors, depends heavily on whether the requested optical properties are in the a region which is sufficiently covered by data in the training set. This is a disadvantage compared to gradient descent with the adjoint method, where an optimum of the FOM is directly targeted. We are trying to address this shortcoming by directly influencing the training data set during its creation, by employing the iterative improvement described in chapter 5.5. Consequently, this adds additional computational resources to the methodology, depending on how many networks are trained during the data creation, but it does not add any wall clock time, due to parallelizability of the data improvements. Moreover, if any parameters of the simulation like geometry, material properties or other physical settings are changed, the training data set has to be adjusted, simulated again, and therefore, the break-even point is reset. This is clearly a huge disadvantage of the CGAN methodology. Nevertheless, there might be opportunities to create less application-specific design networks in the future, which will be further discussed in chapter 8.1.

An additional question that arises is, whether it is even necessary to train the CGAN models after simulating a data set of 40,000 to 80,000 unit cells. One could treat this data set as a look-up table and simply pick the design with the $\mathcal{S}$ parameters closest to the desired ones. In order to analyze this question, we created histograms, where we plot the minimum mean square error (mse) of each $\mathcal{S}$ parameter data point with every other dater point in the training data set. Thereby, we are visualising the density of the $\mathcal{S}$ parameter distribution, which should correlate with the error distribution of treating the data set as a look-up table. In figure 6.1, we do this for the training data set of the project of section 7.4. We see that the mses are on average smaller than the $\mathcal{S}$ parameter mses of the CGAN, which are around 0.008. Therefore, treating the data set as a look-up table seems reasonable, but also note that for some

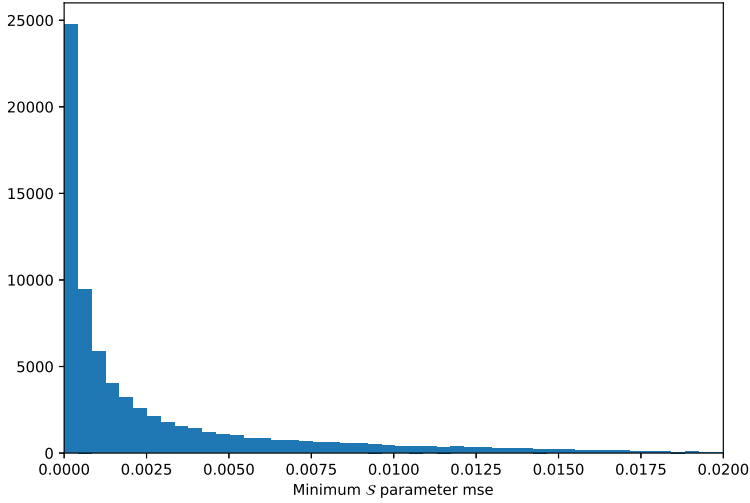


Figure 6.1: Minimum mse between one set of simulated $\mathcal{S}$ parameters and all the other $\mathcal{S}$ parameters in the data set.

desired $\mathcal{S}$ parameters, the CGAN can find a better meta-atom and has control over the aspect ratio.

Here, it is important to note that we are adjusting the density of the $\mathcal{S}$ parameter data set by employing the iterative data set improvement discussed in section 5.5. This iterative process uses generator models that are trained on unfinished data sets and aims to homogenise the density of the distribution as much as possible. Consequently, the shape of the histogram in figure 6.1 is directly influenced by our networks and would have a longer tail at the right side of the distribution without the ANN-guided iterative improvement scheme.

In figure 6.2, another histogram is plotted, now showing the minimum ZWFS loss (i.e., representing the figure of merit for the interconnected meta-atoms from chapter 7.4 of all data points in the data set combined with every other $\mathcal{S}$ parameter data point in blue. The orange bars represent the ZWFS loss of the CGAN outputs. In this case, the best CGAN outputs significantly outperform the look-up table approach, legitimizing our ANN approach.

At the current state of the development of both approaches, it is probably fair to say that physics-informed gradient-descent is a much more mature method than DNN-based approaches. While looking at the vast progress achieved by machine learning methods in many areas over the last one or two decades [94, 157–160], it is clear to us that the approach presented in this thesis has a lot of room for improvement. This thesis merely tries to provide one of the many stepping stones that will be necessary to arrive at an optimal ANN-based inverse design approach. It might also be possible to combine physics-informed gradient-descent and machine-learning-based approaches to perform inverse design [161] (see chapter 8.1).

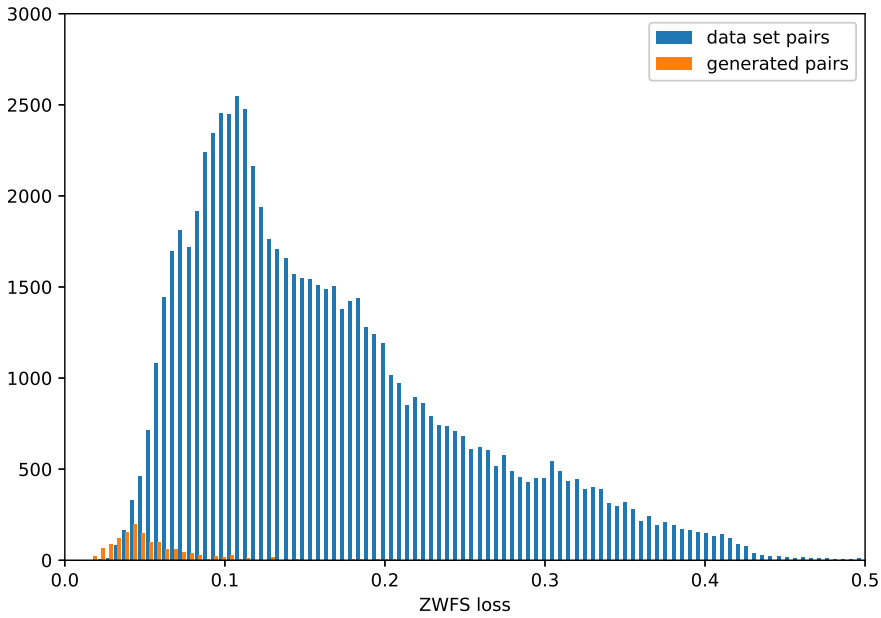


Figure 6.2: Minimum ZWFS loss from chapter 5.4 for all data points in the data set compared with every other $\mathcal{S}$ parameter data point in blue (lowest ZWFS loss: 0.0216). ZWFS loss of the CGAN generated pairs in orange (lowest ZWFS loss: 0.0152).

Chapter 7

Application Examples

In this chapter, we want to demonstrate the capabilities of our CGAN to design meta-atoms with desired $\mathcal{S}$ parameters and aspect ratios for different kind of metasurfaces. To this end, we will evaluate the generators' general capabilities in section 7.1. The subsequent sections will detail various application examples. In section 7.2, we outline the use of our methodology to design a cylindrical metasurface lens and simulate the focusing of an incoming plane wave by the metasurface as a proof of concept. Sections 7.3 describes the design of a metasurface that focuses light emitted by a Vertical Cavity Surface Emitting Lasers (VCSELs) while transforming its polarization from linear to circular. Finally, section 7.4 describes how we used the neural network graph proposed in section 5.4 to design a metasurface phase mask for a Zernike wavefront sensor (ZWFS).

7.1 Meta-atom design

First, we want to examine and visualize the generator and forward model outputs without considering a specific application. The transmission amplitudes and phases for three meta-atoms are plotted in figure 7.1: one meta-atom from the validation set and two meta-atoms created by the generator. In order to compare the errors of our training with other neural network errors, the normalized mean square error (nmse) is evaluated. This nmse is the mse divided by the variance of the original data distribution. For each meta-atom, we show the requested $\mathcal{S}$ parameters as well as the spectrum of the $\mathcal{S}$ parameters obtained from a direct finite-element simulation and the output of the forward model. We see that the agreement between the $\mathcal{S}$ parameters predicted by the CGAN and the exact calculation by full-wave simulations is good, not only for meta-atoms from the validation set [figure 7.1(a)], but also for meta-atoms created by the generator [figure 7.1(b)-(c)]. Rather narrow resonances are not captured perfectly by the forward model and therefore also not by the CGAN. The network is quite capable of predicting that there is a resonance (see chapter 4.1 for maximum absolute differences), but approximating narrow ones with high precision is not possible unfortunately. More training data in general and more specifically a greater number of grids with narrow resonances would increase the performance of the forward model. Here, it would be useful to

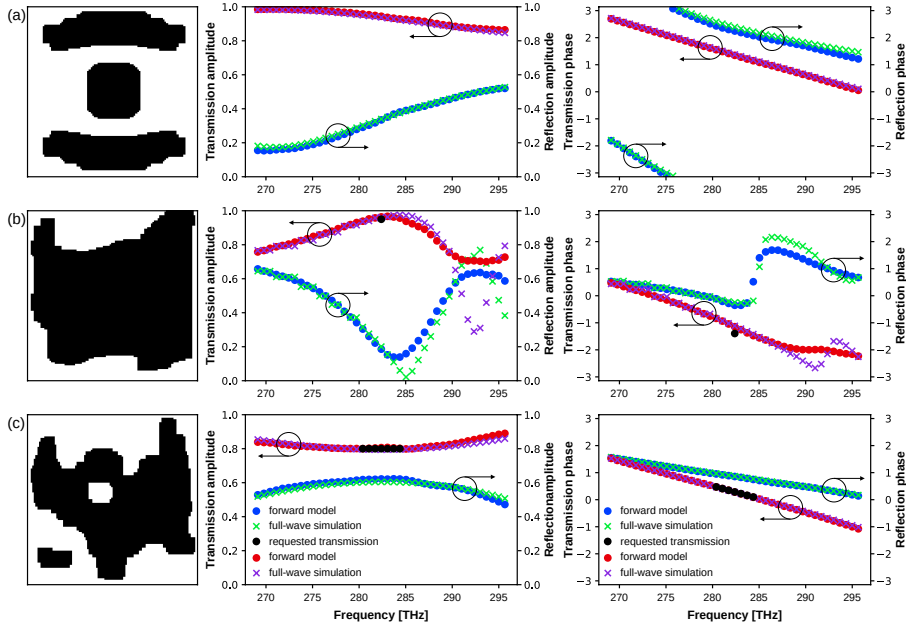


Figure 7.1: Comparison of the $\mathcal{S}$ parameters as requested from the CGAN (black dots), predicted by the forward model (blue and red dots), and calculated by full-wave simulations (green and purple crosses). (a) A meta-atom from the validation set (MSE=0.0004; NMSE=0.001). (b) A meta-atom with a resonance obtained by requesting a desired transmission amplitude and phase at a single frequency from the CGAN (MSE=0.007; NMSE=0.02). (c) A meta-atom obtained from the CGAN by requesting a constant transmission amplitude and linear transmission phase (MSE=0.0003; NMSE=0.0008). (Reprinted from [38])

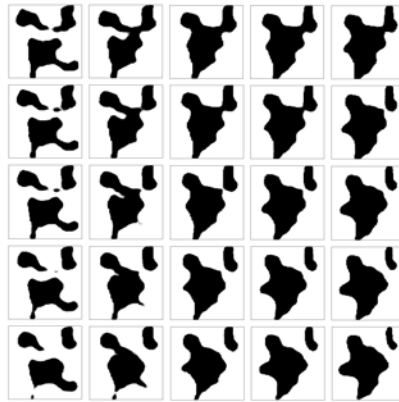


Figure 7.2: Different CGAN output grids for the same $\mathcal{S}$ parameters, dependent on 2D noise $\mathcal{N}$. From the different designs we can select the best suitable grid in terms of how close it approximates the desired $\mathcal{S}$ parameters and the ease of fabrication. (Reprinted from [38])

implement a neural network with a physics layer [162] to improve the resonant feature approximation. However, for many applications in metasurfaces, strong and narrow resonances are not desired, since this makes the designs sensitive to fabrication tolerances, and it is therefore not worth to spend these additional computational resources. This does not mean that resonances in general and broadband resonances specifically are not important; in order to access a large range of $\mathcal{S}$ parameters (e.g., in gradient-phase metasurfaces a 2π coverage of phase is required), metasurfaces must necessarily have resonances. Nevertheless, our networks are still able to capture the effect of resonances at frequencies not in the immediate neighbourhood of the resonance frequency. Furthermore, the network does not struggle with broader resonances, only narrow ones are approximated insufficiently.

We can specify the desired $\mathcal{S}$ parameter as a subset of all parameters used in the training data simulations [black dots in figure 7.1(b) and (c)] by adjusting the $\mathcal{S}$ -parameter input mask $\mathcal{S}_m$ (see chapter 5). It is equally possible to request the whole desired spectrum from the CGAN generator. In figure 7.1(c), for example, we request a spectrum with constant amplitude and linear phase, and again we get very good agreement between the requested $\mathcal{S}$ parameters and full-wave simulations. The CGAN can generate multiple designs for the same desired $\mathcal{S}$ parameters by integrating the encoder (see section 5.2.3) in the training process—an example of this is shown in figure 7.2. This provides us with multiple alternatives that we can choose from in order to ease the nanofabrication or to get a better approximation to the desired $\mathcal{S}$ parameters.

7.2 Cylindrical metalens

In this section, we discuss the first application of our methodology, the design of a cylindrical metasurface lens. A cylindrical lens is essentially a phase mask (see figure 1.6) that focuses an incoming plane wave to a focal line. This can be achieved in a metasurface by stacking meta-atoms in a two-dimensional array where each meta-atom is designed to have a predetermined phase (and a somewhat constant amplitude). Cylindrical lenses focus a plane wave on a line, opposed to a point. Therefore, the phase profile is continuous and obeys mirror symmetry in one dimension (in this case x) and constant in the other dimension (y):

$$\varphi(x) = -\frac{2\pi}{\lambda} \left(\sqrt{x^2 + f^2} - f \right), \quad (7.1)$$

where x are the coordinates of the phase shifting elements of the metasurface and f is the focal point distance of the lens (see chapter 1).

This requires essentially a map between meta-atom designs and their transmission phase and amplitude. With our CGAN, we can very quickly generate a set of meta-atoms with any transmission phase between 0 and 2π and with an almost constant amplitude. For each pixel in a square-lattice grid, we determine the phase required to focus light to a focal point 50 wavelengths further down and use our CGAN to generate the corresponding meta-atom. Figure 7.3(a) and (b) show the central section of such a metasurface lens designed with our CGAN. Subsequently, we perform a full-wave simulation of how a normally-incident optical beam is focused by our metasurface. The

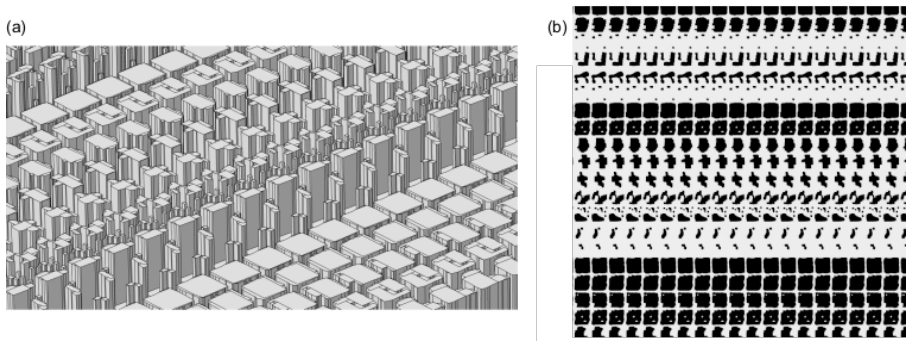


Figure 7.3: Part of the cylindrical lens (above the symmetry axis) constructed from meta-atoms with predetermined phase generated by the CGAN. (a) Top view at an angle of the lens. (b) 2D top view of an section of the lens. (Reprinted and adapted from [38])

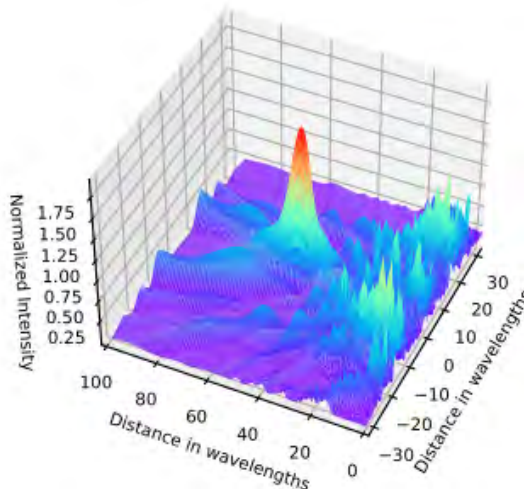


Figure 7.4: Optical intensity of one point on the beam line focused by the ANN-designed lens. The designed focal distance was 50 wavelengths away from the metasurface. (Reprinted from [38])

simulation of the entire metasurface is also performed using COMSOL. The electric field distribution behind the metasurface is exported and propagated to the far field using LightPipes [163]. Plotting the optical intensity around the designed focal length of 50 wavelengths away from the metasurface in figure 7.4, we observe a sharp peak in intensity confirming that the metasurface lens is focusing light exactly onto the desired focal point. This confirms the successful inverse design of the wave equation by our CGAN.

7.3 Metasurface for simultaneous polarization conversion and focusing

In another project, we worked on a metasurface that is intended to overlay a Vertical Cavity Surface Emitting Lasers (VCSELs) with the dual purpose of focusing the emitted laser beam while converting its linear polarization to circular. Leveraging the foundational framework we developed, they were able to implement their own algorithms for the generation, simulation, and labeling of unit cell designs. The simulations, conducted via COMSOL, were configured such that a plane wave of linearly polarized light served as the electromagnetic wave excitation (ports 1 and 2, see section 3.2.1), while the transmitted light was evaluated for two circular polarizations (ports 3 and 4). Using this data along with the CGAN framework elaborated in chapter 5, we trained multiple generators specifically aimed at designing unit cells for a metasurface.

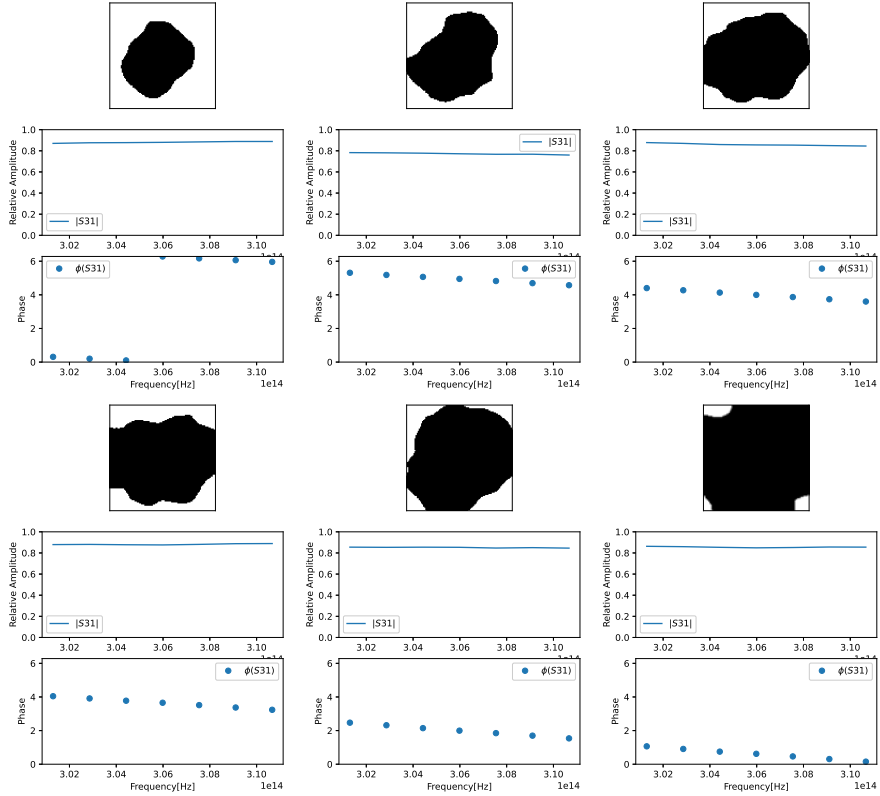


Figure 7.5: Linear to circular polarization converting unit cells. The air pillar around the unit cell is not included in the unit cell cross section. Ports 1 and 2 are in a basis for linear polarized light, while ports 3 and 4 are in a circular basis.

To streamline the design process, we crafted a user-friendly script that empowered an experimental physicist to easily generate unit cell designs. We were able to create a set of polarization converting unit cells with phases ranging

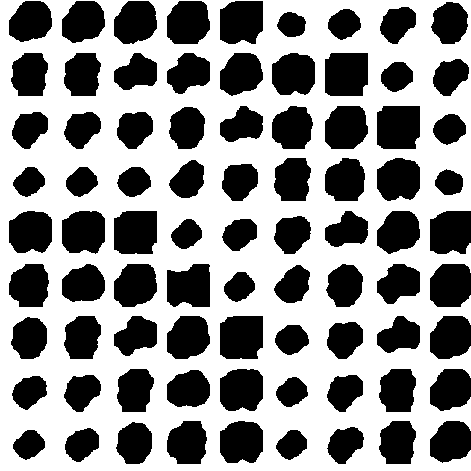


Figure 7.6: A section of the etching mask used to fabricate a polarization converting lens. (Courtesy of Erik Strandberg)

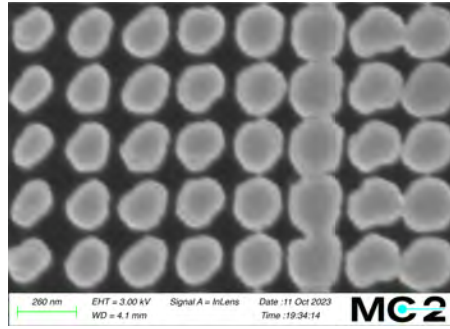


Figure 7.7: Scanning electron microscope (SEM) of an attempt to create a linear phase gradient (see figure 1.4) with polarization conversion from linear to circular. (Courtesy of Erik Strandberg)

from 0 to 2π . A selection of these unit cells and their $\mathcal{S}$ parameters are shown in figure 7.5. A cylindrical lens variant was simulated in COMSOL by the Master's students, validating the concept. While the etching process for the surface encountered some obstacles not related to our work, it is anticipated that these will be resolved in the near term.

In figure 7.7 the scanning electron microscope image of an attempt to create a linear phase gradient (see figure 1.4) with polarization conversion from linear to circular is shown. Some of the regions with more densely packed unit cells were not etched as intended.

7.4 Metasurface for a Zernike wave front sensor

Finally, we used our CGAN model for the creation of interdependent unit cells (see section 5.4) in the context of designing a metasurface used in a Zernike wavefront sensor (ZWFS) [164–167]. A ZWFS can be used to detect misalignment/deviations of a detected wavefront from a plane wavefront, which can then be used to correct the incoming light signal. Therefore, it is a very powerful technique for ground-based telescopes, where the incoming light gets distorted by the earth’s atmosphere, and eliminating these distortions can increase a telescope’s resolution significantly.

ZWFS sensors implement the phase contrast technique, where light is passed through a focal plane mask consisting of two regions imparting different phases on impinging light. This light interferes at an exit pupil to create intensity variations which can be readily imaged with a camera. A phase mask can for instance be created by etching a shallow dimple into a piece of glass to create an optical path length difference between the etched and non-etched regions. However, such an implementation leads to a limited dynamic range in terms of measurable phase errors. This can be improved by using polarization as a degree of freedom and simultaneously impart two different phases on two orthogonal polarizations. These can later be separated to create two images containing different phase information which improves the dynamic range. Here, we utilize our CGAN to design metasurfaces which not only impart different phases on two polarizations, but also do so in a highly achromatic manner. This achromatic behavior is important for broadband functionality of metasurface devices in general.

We design a Zernike phase mask that consists of two regions, as illustrated in figure 7.8, with the following interdependent $\mathcal{S}$ parameters over the entire frequency band of interest:

- a relative phase difference of $\pi/2$ between the $\mathcal{S}_{31}$ parameters in both regions;
- a relative phase difference of $-\pi/2$ between the $\mathcal{S}_{42}$ parameters in both regions;
- high and approximately constant amplitudes of $\mathcal{S}_{31}$ and $\mathcal{S}_{42}$ in both regions.

Designing its meta-atoms is a notoriously difficult task because there are requirements on phase and amplitude simultaneously [168]. It can be achieved for a single incident circular polarization with Berry-Pancharatnam meta-atoms [11, 34, 169], but suffers from leakage. In figure 7.9, we plot the efficiency of a ZWFS sensor with Berry-Pancharatnam meta-atoms; we see that the efficiency is high for a center frequency, but then gradually drops towards the edges of the operational bandwidth, which is a fundamental limitation of this type of metasurfaces [168]. We ask our CGAN for interdependent meta-atoms to design the metasurface for a ZWFS sensor and, in figures 7.10(a)-(b), we plot the meta-atom pairs for two different designs returned by the CGAN. In figures 7.10(c)-(d), we plot the transmission efficiency of the meta-atoms in both regions. The CGAN-derived designs have efficiencies that are constant over a very large bandwidth; they are slightly fluctuating, but do not systematically

drop off towards the edges of the operational bandwidth. In figure 7.10(e)-(f), we observe that the transmission phases satisfy the requirement to a very high degree, also constant over the very large bandwidth. The desired phase is imprinted for both polarizations ($\mathcal{S}_{31}$ and $\mathcal{S}_{42}$). This confirms that CGAN-based inverse design can find designs beyond the current state of the art for meta-atoms with interdependent optical properties.

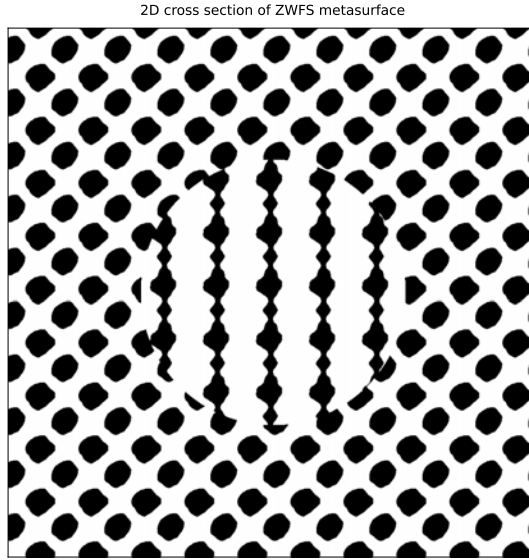


Figure 7.8: A sketch of the ZWFS metasurface (meta-atoms not to scale, unit cells from figure 7.10(b)).

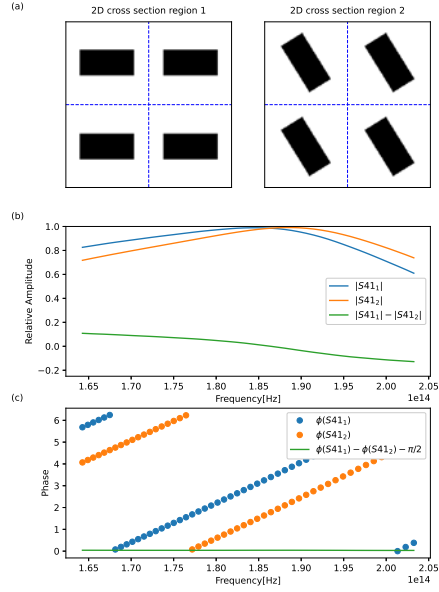


Figure 7.9: Example design of Berry-Pancharatnam meta-atoms for the two circular polarization converting and phase-altering regions for ZWFS designs. (a) 2×2 repetitions of the unit cells are shown for both regions. (b) The conversion amplitudes for the two orthogonal circular polarization states, as well as the differences between the polarization states. (c) The conversion phases for the two orthogonal polarization in both regions, as well as the deviations from design target.

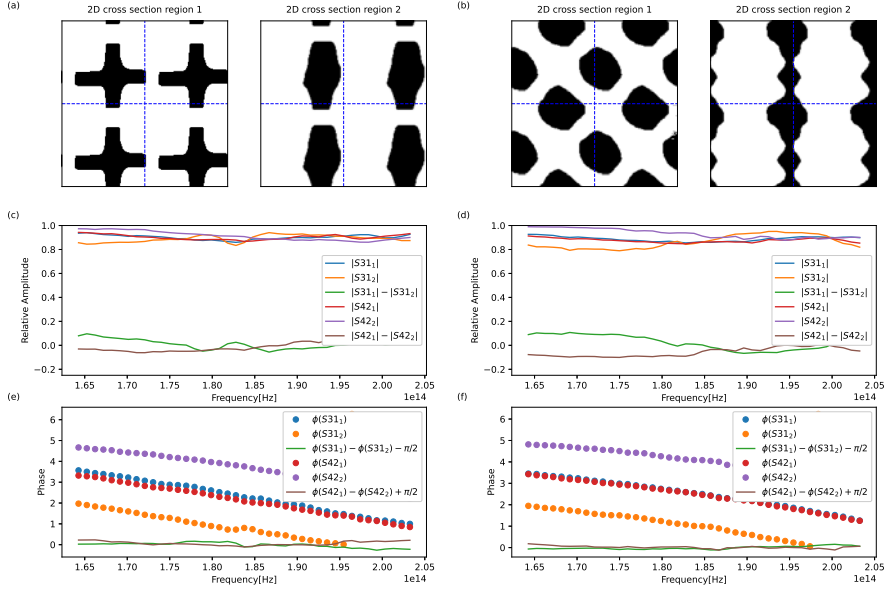


Figure 7.10: Two example designs of meta-atoms for the two phase-altering regions for ZWFS designs, as created by our CGAN variation with interdependent $\mathcal{S}$ parameter relations across the complete frequency band. (a)-(b) 2×2 repetitions of the unit cells are shown for both regions. (c)-(d) The transmission amplitudes for the two orthogonal polarization states, as well as the differences between the polarization states. (e)-(f) The transmission phases for the two orthogonal polarization in both regions, as well as the deviations from design target.

Chapter 8

Conclusions

In this thesis, we have explored the application of machine learning to the free-form inverse design of nanophotonic devices. We demonstrated that a combination of a conditional generative adversarial network (CGAN), a classifier dedicated to fabrication feasibility constraints, and a surrogate neural network model for the prediction of optical properties, can achieve free-form inverse design in the field of nanophotonics. Our neural network has demonstrated the ability to generate hundreds of meta-atom designs within mere seconds. This enables the crafting of large-area metasurfaces from amplitude and phase masks (within the parameter range delineated by the training set)—without the necessity of executing an individual optimization algorithm for each individual meta-atom. A necessary feature of the generator network is its inherent consideration of fabrication feasibility, thereby avoiding unfeasible designs sometimes yielded by alternative methodologies.

Moreover, we have evaluated various techniques aimed at optimizing and stabilizing neural network training and have identified a valuable subset of these approaches. We developed a network graph composition, enabling the creation of a generator model that conceives multiple metasurface unit cells with interdependent properties. This approach enabled us to design several pairs of unit cells for a phase mask of a Zernike wave front sensor (ZWFS), enhancing its operational frequency band, or potentially the development of metasurfaces devoid of chromatic aberration.

A notable requirement for the training of these neural networks is the execution of a large number of full-wave simulations, performed using a classical solver for partial differential equations. However, a prominent advantage of deep neural networks over traditional iterative optimization methods lies in the independent nature of these simulations, enabling simultaneous execution across various computers and leveraging high-performance cluster infrastructure.

We believe that our CGAN approach, in tandem with other strategies such as physics informed gradient descent, dimensionality reduction and knowledge discovery [55, 58, 170] as well as physics informed networks [56] may become a standard technique for achieving inverse design for various applications in nanophotonic structures. These include, but are not limited to, metasurfaces, metamaterials, plasmonic waveguides and resonators, and structures specifically engineered for nonlinear optics.

While the focus of this work has been primarily on metasurfaces with mostly predetermined scattering behaviors, it is relevant to note that our networks may be adapted for the design of structures characterized by strong near fields or significant nonlinear optical interactions. Finally, since our training samples are obtained from full-wave simulations of wave equations, our approach can potentially be generalized to other technological areas based on wave equations, such as microphotonics, acoustics, water waves, and seismic waves.

8.1 Outlook

There are a few things I would do differently, in retrospect, in the context of this thesis. Firstly, I would change the integration of physical constraints in the forward model (see chapter 4.1) from integrating it into the network architecture to additional terms in the loss function. Alternatively, a network approximating underlying physical resonances to model the $\mathcal{S}$ parameters, like the Lorentz net [162], has potentially even higher accuracy. Furthermore, the grid creation algorithm and classifier labeling (see chapter 3.1 and 4.2) work, but could certainly be done cleaner with a different style of algorithm, which would also have downstream consequences for the classifier model.

Due to the current state-of-the-art limitations for computational hardware, we were limited to simulating unit cells instead of the entire metasurface. After arranging unit cells in a metasurface in accordance to the desired phase profile (see figure 7.3), some of the neighbouring unit cells differ at least slightly from the neighbouring unit cells arising from the periodic boundary conditions in the simulations conducted for the training data sets. This difference introduces perturbations, which depend on various factors like the difference in cross section, resonance strength, and the size of the air gap between the unit cells among other factors. Fortunately, the sum of all perturbations is in many cases only causing some noise in the propagating fields, while the desired collective wave interference is close to the desired behaviour (see figure 7.4). Following the trend of hardware improvements and assuming further incremental or paradigm shift induced increase of computational capabilities, it will be possible to simulate full metasurfaces at a cost comparable to our unit cell simulations in the future. Using a data set of these bigger simulations in combination with bigger ANNs (or gradient descent algorithms) should enable the design of metasurfaces that are not constrained by arranging unit cells, but can be completely continuous in their cross section.

Even some currently scientifically explored techniques may revolutionize the field of neural network based inverse design. These include, but are not limited to the integration of transformer architectures [157], which have shown great success especially in large language models [94] (LLMs), transfer learning [171], and stable diffusion [160]. One example of the success of these potential improvements can be found in voice imitation, where only a few years ago one had to feed many hours of recorded voice of one person to a speech generating AI to get a decent imitation of their voice, but nowadays, it is possible to train an AI on a large corpus of speech from thousands of people and it only takes 3 seconds of audio recording from one person to adjust the output of the speech generating AI to resemble the person's actual voice with astonishing

accuracy [172]. With a comparable approach using transformers and a large corpus of various kinds of photonic simulations it should be possible to train a very general neural network capable of creating designs for various applications at potentially any scale. This may be a bold extrapolation, but it is certainly not impossible. The biggest disadvantage of the various fields where inverse design is applied compared to image, sound and video creation is the amount of readily available data and it will stay that way until the computational resources necessary to create such a diverse and vast data set become comparable to the available resources.

Finally, it is possible that physics-informed gradient-descent methods and neural network based approaches could work in synergy by benefiting from their respective strengths. It is hard to say what the final iteration of such a methodology would look like, but here are two possible starting points. On the one hand, in the iterative training data improvement scheme (see section 5.5), it could be useful to employ the adjoint method to increase the density of sparsely covered $\mathcal{S}$ parameter regions. To this end, an algorithm could select the data point with the largest minimal $\mathcal{S}$ parameter mse from the whole distribution (data point occupying the least densest region in $\mathcal{S}$ parameter space, right side of the histogram in figure 6.1). Then, a few small perturbations of the $\mathcal{S}$ parameters of that data point could be used as the target in the FoM. One to a few iterations of the physics-informed gradient-descent method should then be enough to create additional designs which increase the local density of the data set distribution.

On the other hand, ANNs could be used to decrease the number of necessary iteration steps of physics-informed gradient-descent methods, by providing an educated guess for the initial design configuration. This could be useful in a situation where multiple designs with the same overarching parameters are required (e.g., unit cells in a metasurface). After finishing a few of the required designs, all of the simulations from the previous iteration steps could be used to form an incomplete training data set. An ANN model, trained on this data set, could then be used to produce the initial design configuration for the next target designs. After additional data points are simulated, the previously trained ANN model could be trained further by using a combination of the old and the new data points, thereby decreasing the number of iteration steps of the subsequent designs, as a consequence of iteratively increasing the accuracy of the initial design configurations.

Bibliography

- [1] “Encyclopædia britannica.” <https://www.britannica.com/technology/prism-optics>. Accessed: 2023-09-30.
- [2] “Wikipedia: Lens.” <https://en.wikipedia.org/wiki/Lens>. Accessed: 2023-09-28.
- [3] V. E. Ferry, J. N. Munday, and H. A. Atwater, “Design considerations for plasmonic photovoltaics,” *Advanced Materials* **22**, 4794–4808, 2010.
- [4] R. Zhang, Y. Zhang, Z. Dong, S. Jiang, C. Zhang, L. Chen, L. Zhang, Y. Liao, J. Aizpurua, Y. e. Luo, *et al.*, “Chemical mapping of a single molecule by plasmon-enhanced raman scattering,” *Nature* **498**, 82–86, 2013.
- [5] G. Acuna, D. Grohmann, and P. Tinnefeld, “Enhancing single-molecule fluorescence with nanophotonics,” *Federation of European Biochemical Societies Letters* **588**, 3547–3552, 2014.
- [6] J. M. Guerra, “Super-resolution through illumination by diffraction-born evanescent waves,” *Applied Physics Letters* **66**, 3555–3557, 1995.
- [7] L. Thulin and J. Guerra, “Calculations of strain-modified anatase tio 2 band structures,” *Physical Review B* **77**, 195112, 2008.
- [8] W. E. Kock, “Metallic delay lenses,” *Bell System Technical Journal* **27**, 58–82, 1948.
- [9] W. Stork, N. Streibl, H. Haidner, and P. Kipfer, “Artificial distributed-index media fabricated by zero-order gratings,” *Optics Letters* **16**, 1921–1923, 1991.
- [10] M. W. Farn, “Binary gratings with increased efficiency,” *Applied Optics* **31**, 4453–4458, 1992.
- [11] P. Lalanne, S. Astilean, P. Chavel, E. Cambril, and H. Launois, “Blazed binary subwavelength gratings with efficiencies larger than those of conventional échelette gratings,” *Optics Letters* **23**, 1081–1083, 1998.
- [12] P. Lalanne, “Waveguiding in blazed-binary diffractive elements,” *Journal of the Optical Society of America A* **16**, 2517–2520, 1999.

- [13] P. Lalanne, S. Astilean, P. Chavel, E. Cambril, and H. Launois, "Design and fabrication of blazed binary diffractive elements with sampling periods smaller than the structural cutoff," *Journal of the Optical Society of America A* **16**, 1143–1156, 1999.
- [14] D. Berry, R. Malech, and W. Kennedy, "The reflectarray antenna," *IEEE Transactions on Antennas and Propagation* **11**, 645–651, 1963.
- [15] D. Pozar and T. Metzler, "Analysis of a reflectarray antenna using microstrip patches of variable size," *Electronics Letters* **29**, 657–658, 1993.
- [16] J. Huang and J. A. Encinar, *Reflectarray antennas*, John Wiley & Sons, 2007.
- [17] J. A. Encinar, "Design of two-layer printed reflectarrays using patches of variable size," *IEEE Transactions on Antennas and Propagation* **49**, 1403–1410, 2001.
- [18] P. Genevet, F. Capasso, F. Aieta, M. Khorasaninejad, and R. Devlin, "Recent advances in planar optics: From plasmonic to dielectric metasurfaces," *Optica* **4**, 139–152, 2017.
- [19] F. Ding, A. Pors, and S. I. Bozhevolnyi, "Gradient metasurfaces: a review of fundamentals and applications," *Reports on Progress in Physics* **81**, 026401, 2017.
- [20] S. M. Kamali, E. Arbabi, A. Arbabi, and A. Faraon, "A review of dielectric optical metasurfaces for wavefront control," *Nanophotonics* **7**, 1041–1068, 2018.
- [21] M. Born and E. Wolf, *Principles of optics*, Cambridge University Press, 2013.
- [22] J. D. Jackson, *Classical electrodynamics 3rd ed*, Wiley, New York, 1999.
- [23] Z. Jakšić, S. Vuković, J. Matovic, and D. Tanasković, "Negative refractive index metasurfaces for enhanced biosensing," *Materials* **4**, 1–36, 2010.
- [24] T. Suzuki and S. Kondoh, "Negative refractive index metasurface in the 2.0-thz band," *Optical Materials Express* **8**, 1916–1925, 2018.
- [25] N. Yu, P. Genevet, M. A. Kats, F. Aieta, J.-P. Tetienne, F. Capasso, and Z. Gaburro, "Light propagation with phase discontinuities: Generalized laws of reflection and refraction," *Science* **334**, 333–337, 2011.
- [26] R. King, *The theory of linear antennas*, Harvard University Press, Cambridge, Massachusetts, 1956.
- [27] P. Genevet, F. Capasso, F. Aieta, M. Khorasaninejad, and R. Devlin, "Recent advances in planar optics: From plasmonic to dielectric metasurfaces," *Optica* **4**, 139–152, 2017.

- [28] X. Ni, N. K. Emani, A. V. Kildishev, A. Boltasseva, and V. M. Shalaev, “Broadband light bending with plasmonic nanoantennas,” *Science* **335**, 427–427, 2012.
- [29] S. Sun, K.-Y. Yang, C.-M. Wang, T.-K. Juan, W. T. Chen, C. Y. Liao, Q. He, S. Xiao, W.-T. Kung, G.-Y. Guo, *et al.*, “High-efficiency broadband anomalous reflection by gradient meta-surfaces,” *Nano Letters* **12**, 6223–6229, 2012.
- [30] A. Niv, G. Biener, V. Kleiner, and E. Hasman, “Propagation-invariant vectorial Bessel beams obtained by use of quantized Pancharatnam–Berry phase optical elements,” *Optics Letters* **29**, 238–240, 2004.
- [31] F. Aieta, P. Genevet, M. A. Kats, N. Yu, R. Blanchard, Z. Gaburro, and F. Capasso, “Aberration-free ultrathin flat lenses and axicons at telecom wavelengths based on plasmonic metasurfaces,” *Nano Letters* **12**, 4932–4936, 2012.
- [32] B. Desiatov, N. Mazurski, Y. Fainman, and U. Levy, “Polarization selective beam shaping using nanoscale dielectric metasurfaces,” *Optics Express* **23**, 22611–22618, 2015.
- [33] X. Chen, L. Huang, H. Mühlenbernd, G. Li, B. Bai, Q. Tan, G. Jin, C.-W. Qiu, S. Zhang, and T. Zentgraf, “Dual-polarity plasmonic metalens for visible light,” *Nature Communications* **3**, 1–6, 2012.
- [34] G. Zheng, H. Mühlenbernd, M. Kenney, G. Li, T. Zentgraf, and S. Zhang, “Metasurface holograms reaching 80% efficiency,” *Nature Nanotechnology* **10**, 308–312, 2015.
- [35] D. Lin, P. Fan, E. Hasman, and M. L. Brongersma, “Dielectric gradient metasurface optical elements,” *Science* **345**, 298–302, 2014.
- [36] A. Arbabi, E. Arbabi, Y. Horie, S. M. Kamali, and A. Faraon, “Planar metasurface retroreflector,” *Nature Photonics* **11**, 415–420, 2017.
- [37] M. Khorasaninejad and F. Capasso, “Metalenses: Versatile multifunctional photonic components,” *Science* **358**, eaam8100, 2017.
- [38] T. Gahlmann and P. Tassin, “Deep neural networks for the prediction of the optical properties and the free-form inverse design of metamaterials,” *Physical Review B* **106**, 2022.
- [39] A. Y. Piggott, J. Lu, K. G. Lagoudakis, J. Petykiewicz, T. M. Babinec, and J. Vučković, “Inverse design and demonstration of a compact and broadband on-chip wavelength demultiplexer,” *Nature Photonics* **9**, 374–377, 2015.
- [40] A. Michaels and E. Yablonovitch, “Inverse design of near unity efficiency perfectly vertical grating couplers,” *Optics Express* **26**, 4766–4779, 2018.
- [41] S. Molesky, Z. Lin, A. Y. Piggott, W. Jin, J. Vucković, and A. W. Rodriguez, “Inverse design in nanophotonics,” *Nature Photonics* **12**, 659–670, 2018.

- [42] S. D. Campbell, D. Z. Zhu, E. B. Whiting, J. Nagar, D. H. Werner, and P. L. Werner, “Advanced multi-objective and surrogate-assisted optimization of topologically diverse metasurface architectures,” in *Metamaterials, Metadevices, and Metasystems 2018*, **10719**, 43–48, SPIE, 2018.
- [43] Z. Liu, D. Zhu, S. P. Rodrigues, K.-T. Lee, and W. Cai, “Generative model for the inverse design of metasurfaces,” *Nano Letters* **18**, 6570–6576, 2018.
- [44] V. Kalt, A. K. González-Alcalde, S. Es-Saidi, R. Salas-Montiel, S. Blaize, and D. Macías, “Metamodeling of high-contrast-index gratings for color reproduction,” *Journal of the Optical Society of America A* **36**, 79–88, 2019.
- [45] R. S. Hegde, “Photonics inverse design: pairing deep neural networks with evolutionary algorithms,” *IEEE Journal of Selected Topics in Quantum Electronics* **26**, 1–8, 2019.
- [46] S. So, J. Mun, and J. Rho, “Simultaneous inverse design of materials and structures via deep learning: Demonstration of dipole resonance engineering using core-shell nanoparticles,” *ACS Applied Materials & Interfaces* **11**, 24264–24268, 2019.
- [47] W. Ma, F. Cheng, Y. Xu, Q. Wen, and Y. Liu, “Probabilistic representation and inverse design of metamaterials based on a deep generative model with semi-supervised learning strategy,” *Advanced Materials* **31**, 1901111, 2019.
- [48] A. K. González-Alcalde, R. Salas-Montiel, V. Kalt, S. Blaize, and D. Macías, “Engineering colors in all-dielectric metasurfaces: metamodeling approach,” *Optics Letters* **45**, 89–92, 2020.
- [49] R. Pestourie, Y. Mroueh, T. V. Nguyen, P. Das, and S. G. Johnson, “Active learning of deep surrogates for PDEs: application to metasurface design,” *npj Computational Materials* **6**, 1–7, 2020.
- [50] A. Y. Piggott, E. Y. Ma, L. Su, G. H. Ahn, N. V. Sapra, D. Vercruysse, A. M. Netherton, A. S. Khope, J. E. Bowers, and J. Vuckovic, “Inverse-designed photonics for semiconductor foundries,” *ACS Photonics* **7**, 569–575, 2020.
- [51] X. Shi, T. Qiu, J. Wang, X. Zhao, and S. Qu, “Metasurface inverse design using machine learning approaches,” *Journal of Physics D: Applied Physics* **53**, 275105, 2020.
- [52] L. Lu, R. Pestourie, W. Yao, Z. Wang, F. Verdugo, and S. G. Johnson, “Physics-informed neural networks with hard constraints for inverse design,” *SIAM Journal on Scientific Computing* **43**, B1105–B1132, 2021.
- [53] Z. A. Kudyshev, A. V. Kildishev, V. M. Shalaev, and A. Boltasseva, “Machine learning-assisted global optimization of photonic devices,” *Nanophotonics* **10**, 371–383, 2021.

- [54] M. V. Zhelyeznyakov, S. L. Brunton, and A. Majumdar, “Deep learning to accelerate Maxwell’s equations for inverse design of dielectric metasurfaces,” in *2021 Conference on Lasers and Electro-Optics (CLEO)*, 1–2, IEEE, 2021.
- [55] M. Zandehshahvar, Y. Kiarashinejad, M. Zhu, H. Maleki, T. Brown, and A. Adibi, “Manifold learning for knowledge discovery and intelligent inverse design of photonic nanostructures: Breaking the geometric complexity,” *ACS Photonics* **9**, 714–721, 2022.
- [56] O. Khatib, S. Ren, J. Malof, and W. J. Padilla, “Learning the physics of all-dielectric metamaterials with deep lorentz neural networks,” *Advanced Optical Materials* **10**, 2200097, 2022.
- [57] J. Lim and D. Psaltis, “MaxwellNet: Physics-driven deep neural network training based on Maxwell’s equations,” *APL Photonics* **7**, 011301, 2022.
- [58] M. Zandehshahvar, Y. Kiarashi, M. Chen, R. Barton, and A. Adibi, “Inverse design of photonic nanostructures using dimensionality reduction: Reducing the computational complexity,” *Optics Letters* **46**, 2634–2637, 2021.
- [59] D. Shin, A. Cupertino, M. H. J. de Jong, P. G. Steeneken, M. A. Bessa, and R. A. Norte, “Spiderweb nanomechanical resonators via bayesian optimization: Inspired by nature and guided by machine learning,” *Advanced Materials* **34**, 2106248, 2022.
- [60] D. Liu, Y. Tan, E. Khoram, and Z. Yu, “Training deep neural networks for the inverse design of nanophotonic structures,” *ACS Photonics* **5**, 1365–1369, 2018.
- [61] M. P. Bendsoe and O. Sigmund, *Topology optimization: theory, methods, and applications*, Springer, 2003.
- [62] J. S. Jensen and O. Sigmund, “Topology optimization for nano-photonics,” *Laser & Photonics Reviews* **5**, 308–321, 2011.
- [63] J. Hadamard, “Sur les problèmes aux dérivées partielles et leur signification physique,” *Princeton University Bulletin*, 49–52, 1902.
- [64] H. Chen, H. Jia, T. Wang, and J. Yang, “A gradient-oriented binary search method for photonic device design,” *Journal of Lightwave Technology* **39**, 2407–2412, 2021.
- [65] T. Fujisawa and K. Saitoh, “Bayesian direct-binary-search algorithm for the efficient design of mosaic-based power splitters,” *OSA Continuum* **4**, 1258–1270, 2021.
- [66] B. Shen, P. Wang, R. Polson, and R. Menon, “An integrated-nanophotonics polarization beamsplitter with $2.4 \times 2.4 \mu\text{m}^2$ footprint,” *Nature Photonics* **9**, 378–382, 2015.
- [67] M. M. Elsaywy, S. Lanteri, R. Duvigneau, J. A. Fan, and P. Genevet, “Numerical optimization methods for metasurfaces,” *Laser & Photonics Reviews* **14**, 1900445, 2020.

- [68] 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,” *Optical Materials Express* **9**, 1842–1863, 2019.
- [69] S. Osher and J. A. Sethian, “Fronts propagating with curvature-dependent speed: Algorithms based on Hamilton-Jacobi formulations,” *Journal of Computational Physics* **79**, 12–49, 1988.
- [70] S. Preble, M. Lipson, and H. Lipson, “Two-dimensional photonic crystals designed by evolutionary algorithms,” *Applied Physics Letters* **86**, 061111, 2005.
- [71] M. Shokooh-Saremi and R. Magnusson, “Particle swarm optimization and its application to the design of diffraction grating filters,” *Optics Letters* **32**, 894–896, 2007.
- [72] P. R. Wiecha, A. Arbouet, C. Girard, A. Lecestre, G. Larrieu, and V. Paillard, “Evolutionary multi-objective optimization of colour pixels based on dielectric nanoantennas,” *Nature Nanotechnology* **12**, 163–169, 2017.
- [73] C. Liu, S. A. Maier, and G. Li, “Genetic-algorithm-aided meta-atom multiplication for improved absorption and coloration in nanophotonics,” *ACS Photonics* **7**, 1716–1722, 2020.
- [74] C. M. Lalau-Keraly, S. Bhargava, O. D. Miller, and E. Yablonovitch, “Adjoint shape optimization applied to electromagnetic design,” *Optics Express* **21**, 21693–21701, 2013.
- [75] L. Beilina, L. Mpinganzima, and P. Tassin, “Adaptive optimization algorithm for the computational design of nanophotonic structures,” in *2016 International Conference on Electromagnetics in Advanced Applications (ICEAA)*, 420–423, IEEE, 2016.
- [76] L.-J. Black, Y. Wang, C. De Groot, A. Arbouet, and O. L. Muskens, “Optimal polarization conversion in coupled dimer plasmonic nanoantennas for metasurfaces,” *ACS Nano* **8**, 6390–6399, 2014.
- [77] M. B. Giles and N. A. Pierce, “An introduction to the adjoint approach to design,” *Flow, Turbulence and Combustion* **65**, 393–415, 2000.
- [78] T. W. Hughes, M. Minkov, I. A. Williamson, and S. Fan, “Adjoint method and inverse design for nonlinear nanophotonic devices,” *ACS Photonics* **5**, 4781–4787, 2018.
- [79] K. Wang, X. Ren, W. Chang, L. Lu, D. Liu, and M. Zhang, “Inverse design of digital nanophotonic devices using the adjoint method,” *Photonics Research* **8**, 528–533, 2020.
- [80] O. Sigmund and J. Søndergaard Jensen, “Systematic design of phononic band-gap materials and structures by topology optimization,” *Philosophical Transactions of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences* **361**, 1001–1019, 2003.

- [81] Y. Elesin, B. S. Lazarov, J. S. Jensen, and O. Sigmund, "Design of robust and efficient photonic switches using topology optimization," *Photonics and Nanostructures—Fundamentals and Applications* **10**, 153–165, 2012.
- [82] T. Hughes, G. Veronis, K. P. Wootton, R. J. England, and S. Fan, "Method for computationally efficient design of dielectric laser accelerator structures," *Optics Express* **25**, 15414–15427, 2017.
- [83] C. Y. Kao, S. Osher, and E. Yablonovitch, "Maximizing band gaps in two-dimensional photonic crystals by using level set methods," *Applied Physics B* **81**, 235–244, 2005.
- [84] R. Matzen, J. S. Jensen, and O. Sigmund, "Systematic design of slow-light photonic waveguides," *Journal of the Optical Society of America B* **28**, 2374–2382, 2011.
- [85] J. S. Jensen and O. Sigmund, "Topology optimization of photonic crystal structures: A high-bandwidth low-loss t-junction waveguide," *Journal of the Optical Society of America B* **22**, 1191–1198, 2005.
- [86] L. F. Frellsen, Y. Ding, O. Sigmund, and L. H. Frandsen, "Topology optimized mode multiplexing in silicon-on-insulator photonic wire waveguides," *Optics Express* **24**, 16866–16873, 2016.
- [87] I. Kononenko, "Machine learning for medical diagnosis: history, state of the art and perspective," *Artificial Intelligence in Medicine* **23**, 89–109, 2001.
- [88] P. Chlap, H. Min, N. Vandenberg, J. Dowling, L. Holloway, and A. Haworth, "A review of medical image data augmentation techniques for deep learning applications," *Journal of Medical Imaging and Radiation Oncology* **65**, 545–563, 2021.
- [89] S. Srivastava, A. V. Divekar, C. Anilkumar, I. Naik, V. Kulkarni, and V. Pattabiraman, "Comparative analysis of deep learning image detection algorithms," *Journal of Big Data* **8**, 1–27, 2021.
- [90] P. Wang, E. Fan, and P. Wang, "Comparative analysis of image classification algorithms based on traditional machine learning and deep learning," *Pattern Recognition Letters* **141**, 61–67, 2021.
- [91] A. Gupta, A. Anpalagan, L. Guan, and A. S. Khwaja, "Deep learning for object detection and scene perception in self-driving cars: Survey, challenges, and open issues," *Array* **10**, 100057, 2021.
- [92] T. Nakamura-Zimmerer, Q. Gong, and W. Kang, "Adaptive deep learning for high-dimensional Hamilton-Jacobi-Bellman equations," *SIAM Journal on Scientific Computing* **43**, A1221–A1247, 2021.
- [93] W. Wang and K. Siau, "Artificial intelligence, machine learning, automation, robotics, future of work and future of humanity: A review and research agenda," *Journal of Database Management* **30**, 61–79, 2019.

- [94] Y. Liu, T. Han, S. Ma, J. Zhang, Y. Yang, J. Tian, H. He, A. Li, M. He, Z. Liu, Z. Wu, L. Zhao, D. Zhu, X. Li, N. Qiang, D. Shen, T. Liu, and B. Ge, “Summary of ChatGPT-related research and perspective towards the future of large language models,” *Meta-Radiology* **1**, 100017, 2023.
- [95] E. P. Van Nieuwenburg, Y.-H. Liu, and S. D. Huber, “Learning phase transitions by confusion,” *Nature Physics* **13**, 435–439, 2017.
- [96] L. Salmela, N. Tsipinakis, A. Foi, C. Billet, J. M. Dudley, and G. Genty, “Predicting ultrafast nonlinear dynamics in fibre optics with a recurrent neural network,” *Nature Machine Intelligence* **3**, 344–354, 2021.
- [97] R. Iten, T. Metger, H. Wilming, L. Del Rio, and R. Renner, “Discovering physical concepts with neural networks,” *Physical Review Letters* **124**, 010508, 2020.
- [98] L. Gao, X. Li, D. Liu, L. Wang, and Z. Yu, “A bidirectional deep neural network for accurate silicon color design,” *Advanced Materials* **31**, 1905467, 2019.
- [99] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*, MIT Press, 2016.
- [100] Y. LeCun, Y. Bengio, and G. Hinton, “Deep learning,” *Nature* **521**, 436–444, 2015.
- [101] P. R. Wiecha, A. Arbouet, C. Girard, and O. L. Muskens, “Deep learning in nano-photonics: inverse design and beyond,” *Photonics Research* **9**, B182–B200, 2021.
- [102] “Uc regents davis campus.” <http://brainmaps.org>. Accessed: 2023-09-29.
- [103] B. Mehlig, *Machine Learning with Neural Networks: An Introduction for Scientists and Engineers*, Cambridge University Press, 2021.
- [104] W. S. McCulloch and W. Pitts, “A logical calculus of the ideas immanent in nervous activity,” *The Bulletin of Mathematical Biophysics* **5**, 115–133, 1943.
- [105] F. Rosenblatt, “The perceptron: a probabilistic model for information storage and organization in the brain,” *Psychological Review* **65**, 386, 1958.
- [106] M. L. Minsky and S. A. Papert, “Perceptrons: expanded edition,” 1988.
- [107] W. Hasperué, “The master algorithm: how the quest for the ultimate learning machine will remake our world,” *Journal of Computer Science & Technology* **15**, 2015.
- [108] “No hands across america.” https://www.cs.cmu.edu/~tjochem/nhaa/nhaa_home_page.html. Accessed: 2023-09-30.
- [109] B. Mehlig, *Lecture notes for the course Artificial Neural Networks*, Chalmers University of Technology, 2019.

- [110] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural Networks* **2**, 359–366, 1989.
- [111] B. C. Csáji *et al.*, "Approximation with artificial neural networks," *Faculty of Sciences, Eötvös Loránd University, Hungary* **24**, 7, 2001.
- [112] "Saugat bhattarai." <https://saugatbhattarai.com.np/what-is-gradient-descent-in-machine-learning/>. Accessed: 2023-09-29.
- [113] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.
- [114] N. Kanopoulos, N. Vasanthavada, and R. L. Baker, "Design of an image edge detection filter using the sobel operator," *IEEE Journal of Solid-State Circuits* **23**, 358–367, 1988.
- [115] "Wikipedia: Sobel operator." https://en.wikipedia.org/wiki/Sobel_operator. Accessed: 2023-09-29.
- [116] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. A. Alemi, "Inception-v4, inception-resnet and the impact of residual connections on learning," in *Thirty-first AAAI Conference on Artificial Intelligence*, 2017.
- [117] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 4700–4708, 2017.
- [118] G. Huang, S. Liu, L. Van der Maaten, and K. Q. Weinberger, "DenseNet: An efficient DenseNet using learned group convolutions," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2752–2761, 2018.
- [119] F. Iandola, M. Moskewicz, S. Karayev, R. Girshick, T. Darrell, and K. Keutzer, "DenseNet: Implementing efficient convnet descriptor pyramids," *arXiv preprint arXiv:1404.1869*, 2014.
- [120] K. Zhang, Y. Guo, X. Wang, J. Yuan, and Q. Ding, "Multiple feature reweight DenseNet for image classification," *IEEE Access* **7**, 9872–9880, 2019.
- [121] A. Mikołajczyk and M. Grochowski, "Data augmentation for improving deep learning in image classification problem," in *International Interdisciplinary PhD Workshop (IIPhDW)*, 117–122, IEEE, 2018.
- [122] N. E. Khalifa, M. Loey, and S. Mirjalili, "A comprehensive survey of recent trends in deep learning for digital images augmentation," *Artificial Intelligence Review*, 1–27, 2022.
- [123] I. Kostrikov, D. Yarats, and R. Fergus, "Image augmentation is all you need: Regularizing deep reinforcement learning from pixels," *arXiv preprint arXiv:2004.13649*, 2020.
- [124] C. M. Bishop, "Training with noise is equivalent to Tikhonov regularization," *Neural Computation* **7**, 108–116, 01 1995.

- [125] G. An, “The effects of adding noise during backpropagation training on a generalization performance,” *Neural Computation* **8**, 643–674, 1996.
- [126] S. Kothari and H. Oh, “Neural networks for pattern recognition,” *Advances in Computers* **37**, 119–166, 1993.
- [127] A. P. Piotrowski and J. J. Napiorkowski, “A comparison of methods to avoid overfitting in neural networks training in the case of catchment runoff modelling,” *Journal of Hydrology* **476**, 97–111, 2013.
- [128] R. M. Zur, Y. Jiang, L. L. Pesce, and K. Drukker, “Noise injection for training artificial neural networks: A comparison with weight decay and early stopping,” *Medical Physics* **36**, 4810–4818, 2009.
- [129] H. Noh, T. You, J. Mun, and B. Han, “Regularizing deep neural networks by noise: Its interpretation and optimization,” *Advances in Neural Information Processing Systems* **30**, 4–9, 2017.
- [130] R. Wang, W. Li, and L. Zhang, “Blur image identification with ensemble convolution neural networks,” *Signal Processing* **155**, 73–82, 2019.
- [131] I. Vasiljevic, A. Chakrabarti, and G. Shakhnarovich, “Examining the impact of blur on recognition by convolutional networks,” *arXiv preprint arXiv:1611.05760*, 2016.
- [132] L. Wan, M. Zeiler, S. Zhang, Y. Le Cun, and R. Fergus, “Regularization of neural networks using dropconnect,” in *International Conference on Machine Learning*, 1058–1066, PMLR, 2013.
- [133] C. Cortes, M. Mohri, and A. Rostamizadeh, “L2 regularization for learning kernels,” *arXiv preprint arXiv:1205.2653*, 2012.
- [134] C. Inc., “Introduction to comsol multiphysics,” 2019.
- [135] I. Malkiel, M. Mrejen, A. Nagler, U. Arieli, L. Wolf, and H. Suchowski, “Plasmonic nanostructure design and characterization via deep learning,” *Light: Science & Applications* **7**, 1–8, 2018.
- [136] S. An, C. Fowler, B. Zheng, M. Y. Shalaginov, H. Tang, H. Li, L. Zhou, J. Ding, A. M. Agarwal, C. Rivero-Baleine, *et al.*, “A deep learning approach for objective-driven all-dielectric metasurface design,” *ACS Photonics* **6**, 3196–3207, 2019.
- [137] Y. Li, Y. Wang, S. Qi, Q. Ren, L. Kang, S. D. Campbell, P. L. Werner, and D. H. Werner, “Predicting scattering from complex nano-structures via deep learning,” *IEEE Access* **8**, 139983–139993, 2020.
- [138] Y. Zhu, N. Zabaras, P.-S. Koutsourelakis, and P. Perdikaris, “Physics-constrained deep learning for high-dimensional surrogate modeling and uncertainty quantification without labeled data,” *Journal of Computational Physics* **394**, 56–81, 2019.
- [139] T. Chugh, C. Sun, H. Wang, and Y. Jin, “Surrogate-assisted evolutionary optimization of large problems,” in *High-Performance Simulation-Based Optimization*, 165–187, Springer, 2020.

- [140] S. Cuomo, V. S. Di Cola, F. Giampaolo, G. Rozza, M. Raissi, and F. Piccialli, “Scientific machine learning through physics-informed neural networks: Where we are and what’s next,” *Journal of Scientific Computing* **92**, 88, 2022.
- [141] Y. Chen, L. Lu, G. E. Karniadakis, and L. Dal Negro, “Physics-informed neural networks for inverse problems in nano-optics and metamaterials,” *Optics Express* **28**, 11618–11633, 2020.
- [142] X. Xu, C. Sun, Y. Li, J. Zhao, J. Han, and W. Huang, “An improved tandem neural network for the inverse design of nanophotonics devices,” *Optics Communications* **481**, 126513, 2021.
- [143] R. Unni, K. Yao, X. Han, M. Zhou, and Y. Zheng, “A mixture-density-based tandem optimization network for on-demand inverse design of thin-film high reflectors,” *Nanophotonics* **10**, 4057–4065, 2021.
- [144] W. Jing, L. Runze, H. Cheng, C. Haixin, R. Cheng, Z. Chen, and M. Zhang, “An inverse design method for supercritical airfoil based on conditional generative models,” *Chinese Journal of Aeronautics* **35**, 62–74, 2022.
- [145] K. Yonekura and K. Suzuki, “Data-driven design exploration method using conditional variational autoencoder for airfoil design,” *Structural and Multidisciplinary Optimization* **64**, 613–624, 2021.
- [146] Y. Tang, K. Kojima, T. Koike-Akino, Y. Wang, P. Wu, Y. Xie, M. H. Tahersima, D. K. Jha, K. Parsons, and M. Qi, “Generative deep learning model for inverse design of integrated nanophotonic devices,” *Laser & Photonics Reviews* **14**, 2000287, 2020.
- [147] M. Mirza and S. Osindero, “Conditional generative adversarial nets,” *arXiv preprint arXiv:1411.1784*, 2014.
- [148] W. Hong, Z. Wang, M. Yang, and J. Yuan, “Conditional generative adversarial network for structured domain adaptation,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 1335–1344, 2018.
- [149] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, “Generative adversarial networks,” *Communications of the ACM* **63**, 139–144, 2020.
- [150] “Eclipse deeplearning4j.” <https://mgubaidullin.github.io/deeplearning4j-docs/generative-adversarial-network.html>. Accessed: 2023-10-29.
- [151] M. Arjovsky, S. Chintala, and L. Bottou, “Wasserstein generative adversarial networks,” in *International Conference on Machine Learning*, 214–223, PMLR, 2017.
- [152] K. Liu, W. Tang, F. Zhou, and G. Qiu, “Spectral regularization for combating mode collapse in gans,” in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 6382–6390, 2019.

- [153] A. Jabbar, X. Li, and B. Omar, “A survey on generative adversarial networks: Variants, applications, and training,” *ACM Computing Surveys* **54**, 1–49, 2021.
- [154] W. Li, L. Fan, Z. Wang, C. Ma, and X. Cui, “Tackling mode collapse in multi-generator gans with orthogonal vectors,” *Pattern Recognition* **110**, 107646, 2021.
- [155] J. Rose, “Bürorose.” www.buerorose.de. Accessed: 2023-10-29.
- [156] S. G. Johnson, “Notes on adjoint methods for 18.335,” *Introduction to Numerical Methods*, 2012.
- [157] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, “Attention is all you need,” *Advances in Neural Information Processing Systems* **30**, 5998–6008, 2017.
- [158] J. Schrittwieser, I. Antonoglou, T. Hubert, K. Simonyan, L. Sifre, S. Schmitt, A. Guez, E. Lockhart, D. Hassabis, T. Graepel, *et al.*, “Mastering atari, go, chess and shogi by planning with a learned model,” *Nature* **588**, 604–609, 2020.
- [159] J. Jumper, R. Evans, A. Pritzel, T. Green, M. Figurnov, O. Ronneberger, K. Tunyasuvunakool, R. Bates, A. Žídek, A. Potapenko, *et al.*, “Highly accurate protein structure prediction with alphafold,” *Nature* **596**, 583–589, 2021.
- [160] C. Saharia, W. Chan, S. Saxena, L. Li, J. Whang, E. L. Denton, K. Ghasemipour, R. Gontijo Lopes, B. Karagol Ayan, T. Salimans, *et al.*, “Photorealistic text-to-image diffusion models with deep language understanding,” *Advances in Neural Information Processing Systems* **35**, 36479–36494, 2022.
- [161] R. V. Woldseth, N. Aage, J. A. Bærentzen, and O. Sigmund, “On the use of artificial neural networks in topology optimisation,” *Structural and Multidisciplinary Optimization* **65**, 294, 2022.
- [162] O. Khatib, S. Ren, J. Malof, and W. J. Padilla, “Learning the physics of all-dielectric metamaterials with deep lorentz neural networks,” *Advanced Optical Materials* **10**, 2200097, 2022.
- [163] G. Vdovin, F. van Goor, *et al.*, “Lightpipes: Beam propagation toolbox,” *OKO Technologies, The Netherlands*, 1999.
- [164] D. S. Doelman, F. F. Auer, M. J. Escuti, and F. Snik, “Simultaneous phase and amplitude aberration sensing with a liquid-crystal vector-Zernike phase mask,” *Optics Letters* **44**, 17–20, 2019.
- [165] J. Steeves, J. K. Wallace, C. Kettenbeil, and J. Jewell, “Picometer wavefront sensing using the phase-contrast technique,” *Optica* **7**, 1267–1274, 2020.
- [166] J. K. Wallace, S. Rao, R. M. Jensen-Clem, and G. Serabyn, “Phase-shifting zernike interferometer wavefront sensor,” in *Optical Manufacturing and Testing IX*, **8126**, 110–120, SPIE, 2011.

- [167] R. Jensen-Clem, J. K. Wallace, and E. Serabyn, “Characterization of the phase-shifting zernike wavefront sensor for telescope applications,” in *2012 IEEE Aerospace Conference*, 1–7, 2012.
- [168] A. C. Overvig, S. Shrestha, S. C. Malek, M. Lu, A. Stein, C. Zheng, and N. Yu, “Dielectric metasurfaces for complete and independent control of the optical amplitude and phase,” *Light: Science & Applications* **8**, 92, 2023.
- [169] L. Huang, X. Chen, H. Mühlenbernd, G. Li, B. Bai, Q. Tan, G. Jin, T. Zentgraf, and S. Zhang, “Dispersionless phase discontinuities for controlling light propagation,” *Light: Science & Applications* **12**, 5750–5755, 2012.
- [170] Y. Kiarashinejad, M. Zandehshahvar, S. Abdollahramezani, O. Hemmatyar, R. Pourabolghasem, and A. Adibi, “Knowledge discovery in nanophotonics using geometric deep learning,” *Advanced Intelligent Systems* **2**, 1900132, 2020.
- [171] F. Zhuang, Z. Qi, K. Duan, D. Xi, Y. Zhu, H. Zhu, H. Xiong, and Q. He, “A comprehensive survey on transfer learning,” *Proceedings of the IEEE* **109**, 43–76, 2020.
- [172] C. Wang, S. Chen, Y. Wu, Z. Zhang, L. Zhou, S. Liu, Z. Chen, Y. Liu, H. Wang, J. Li, *et al.*, “Neural codec language models are zero-shot text to speech synthesizers,” *arXiv preprint arXiv:2301.02111*, 2023.

