Inverse Design of MetaSurface Stacks Using Convlutional Neural Networks

Bachelor Thesis

Friedrich-Schiller-Universität Jena
Physikalisch-Astronomische-Fakultät
Institute of Applied Physics

Tim Luca Turan
timturan@web.de

April 19, 2020

Evaluators

First Assessor:	Prof. Dr. rer. nat. Thomas Pertsch Institute of Applied Physics Friedrich-Schiller-Universität Jena
Second Assessor:	M.Sc. Jan Sperrhake Friedrich-Schiller-Universität Jena Institute of Applied Physics

1. Introduction

Metamaterials are materials with sub-wavelength structure whose physical properties are determined by their internal geometry. They can be engineered to exhibit properties not found in nature like negative refractive indices [1]. For optics the most common architecture are 2D metamaterials called metasurfaces. Normal optical devices, like lenses or phase plates, leverage chemical material properties and optical effects such as diffraction, refraction or phase differences accumulated over variable propagation lengths. All these effects rely on distances much larger than one wavelength to change the light's properties. In contrast, metasurfaces rely on artificial sub-wavelength structures and nano optical effects. Because of their sub-wavelength thickness they enable thin components not possible before [2]. These metasurfaces promise custom optical components which can be tailored to a wide range of applications just by changing geometry of the artificial structures. However, because of the sub-wavelength scale, optical metasurfaces with complex geometries are quite hard to manufacture. Also the optical behavior of metasurfaces is very challenging to model analytically and usually involves computationally intensive simulations which slow down the design process.

2016 Menzel and Sperrhake [3] presented a *Semi Analytic Stacking Algorithm*, short SASA, to analytically calculate the optical behavior of metasurface stacks if one knows how the surfaces behave individually. This allows for a different approach when designing novel optical properties. The idea here is to create a target behavior not by using complex geometry but by using simple metasurfaces and stacking them on top of each other. The design process becomes an optimization problem of choosing the right surfaces and then tuning stack parameters like the distance between layers and their rotation angle. This approach is faster because SASA is analytical in nature and many different stacks can be considered very quickly. However, it also poses some new challenges when trying to automate the process and optimize the properties of a design using a conventional approach. These usually rely on the continuity of the underlying process so that if a parameter is tuned slightly it is possible to calculate a gradient and tell wether that was a step in the right direction. This is a problem because some of the choices involved when building a stack are categorical. For example, choosing the material of the layers or the kind of geometry.

Another recent development has produced a very general and powerful way of dealing with categorical decision problems. Deep Neural Networks have found their way into virtually all areas of life from autonomous driving [4] to video recommendations [5]. Given enough data it, should be possible to train a Neural Network to choose the discrete stack parameters to a given optical target. Again, SASA being analytical is useful because it can be used to generate a lot of training stacks very quickly. This sets the goal for this work: Write an algorithm which can find a metasurface stack to a custom transmission spectrum and do so by training a Neural Network on data generated by the SASA algorithm.

The slow and involved process of designing metasurfaces based on numerical methods has led several teams to try and apply the recent advances in machine learning to this problem. Promising results have been published, but they usually yield designs that are too complex to be manufactured for optical wavelengths [6] or they have very specific optical targets, e.g. a chiral response [7]. The target algorithm would represent a tool that is simultaneously based on a simple metasurface geometry and capable of solving a common problem. Namely needing a specific transmission spectrum for an application.

Section 2 contains a short glossary on the notation used throughout this text. In section 3 we will first develop the physical theory to understand metasurface stacks and then look into the computer science background of Neural Networks to decide which network architecture fits the problem at hand. The Algorithm is described in detail and evaluated in section 4. We will and on a brief summary and outlook in section 5. All the code written for this project is open source and documented thoroughly at `https://github.com/TimLucaTuran` and the final tool can be installed as a standard Python package.

Contents

1 Introduction **3**

2 Notation **5**

3 Background **6**
- 3.1 Scattering Matrix Calculus . . . 6
- 3.2 SASA . . . 12
- 3.3 Plasmonic Metasurfaces . . . 14
- 3.4 Neural Networks . . . 17

4 Algorithm **24**
- 4.1 Implementation . . . 26
- 4.2 Network . . . 27
- 4.3 Database . . . 30
- 4.4 Optimizer . . . 31
- 4.5 Evaluation . . . 33

5 Conclusion **34**

A Tandem Model **36**

B Example Fits **39**

References **40**

2. Notation

Symbols	Explanation
$\mathbf{E}$, $\mathbf{B}$, $\mathbf{k}$	Vectors are written bold
E, B, k	Amplitudes of vectors are written non-bold so that $\lvert\mathbf{E}\rvert = E$
$\hat{S}$, $\hat{J}$, $\hat{w}$	Matrices have an hat
$\hat{w}^2_{1,2}$, E^{in}_x	Super and sub indices are used to specify matrix, vector or tuple elements
$(y_i - y'_i)^2$, $(n)^2$	All exponents are outside parentheses to differentiate between index and exponent
$\mathbb{1}$	Unity matrix sized accordingly e.g. in the context of 2×2 matrices $\mathbb{1} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$
∇	Nabla operator $\nabla = \begin{pmatrix} \frac{\partial}{\partial x} \\ \frac{\partial}{\partial y} \\ \frac{\partial}{\partial z} \end{pmatrix}$
Δ	Laplace operator $\Delta = \begin{pmatrix} \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} + \frac{\partial^2}{\partial z^2} \\ \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} + \frac{\partial^2}{\partial z^2} \\ \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} + \frac{\partial^2}{\partial z^2} \end{pmatrix}$
$\odot$	Hadamard product $\mathbf{a} \odot \mathbf{b} := \begin{pmatrix} a_1\, b_1 \\ a_2\, b_2 \\ \vdots \end{pmatrix}$

3. Background

3.1. Scattering Matrix Calculus

To implement the desired algorithm we need to be able to calculate the optical behavior of stacked metasurfaces. The mathematical framework we will use is called scattering matrix calculus and this section will give some insight into its physical origin and how to use it. We will start with Maxwell's equations in matter.

Maxwell Equations

$$\nabla \times \mathbf{E}(\mathbf{r},t) = -\frac{\partial}{\partial t}\mathbf{B}(\mathbf{r},t) \tag{3.1}$$

$$\nabla \cdot \mathbf{D}(\mathbf{r},t) = \rho_{\text{ext}}(\mathbf{r},t) \tag{3.2}$$

$$\nabla \times \mathbf{H}(\mathbf{r},t) = \mathbf{j}(\mathbf{r},t) + \frac{\partial}{\partial t}\mathbf{D}(\mathbf{r},t) \tag{3.3}$$

$$\nabla \cdot \mathbf{B}(\mathbf{r},t) = 0 \tag{3.4}$$

The four involved fields are: $\mathbf{E}$ electric field, $\mathbf{B}$ magnetic flux density, $\mathbf{D}$ electric flux density and $\mathbf{H}$ magnetic field and the sources are the external charges ρ_{ext} and the macroscopic currents $\mathbf{j}$. All the material properties are captured by the $\mathbf{D}$ and $\mathbf{H}$ fields which are defined as

$$\begin{aligned} \mathbf{D}(\mathbf{r},t) &= \varepsilon_0 \mathbf{E}(\mathbf{r},t) + \mathbf{P}(\mathbf{r},t) \quad \text{and} \\ \mathbf{H}(\mathbf{r},t) &= \frac{1}{\mu_0}[\mathbf{B}(\mathbf{r},t) - \mathbf{M}(\mathbf{r},t)], \end{aligned} \tag{3.5}$$

where $\mathbf{P}$ is the dielectric polarization and $\mathbf{M}$ is the magnetic polarization. One can read equation (3.45) in the following way: when the electric field $\mathbf{E}$ interacts with matter, it exerts a force on all its charges and displaces them by a small amount. This separation of charges results in a counter field $\mathbf{P}$ and the total field $\mathbf{D}$ is now a superposition of $\mathbf{E}$ and $\mathbf{P}$. This set of equations describes the whole electromagnetic spectrum. However, in this work we are only interested in visible (VIS) and near infrared (NIR) light where we can make some simplifications. Generally in optics, materials are non-magnetizable so $\mathbf{M} = 0$ and there are no free charges $\rho_{\text{ext}} = 0$. Inserting these assumptions into maxwell's equation gives

$$\nabla \times \mathbf{E}(\mathbf{r},t) = -\mu_0 \frac{\partial}{\partial t}\mathbf{H}(\mathbf{r},t) \tag{3.6}$$

$$\varepsilon_0 \nabla \cdot \mathbf{E}(\mathbf{r},t) = -\nabla \cdot \mathbf{P}(\mathbf{r},t) \tag{3.7}$$

$$\nabla \times \mathbf{H}(\mathbf{r},t) = \mathbf{j}(\mathbf{r},t) + \frac{\partial}{\partial t}\mathbf{P}(\mathbf{r},t) + \varepsilon_0 \frac{\partial}{\partial t}\mathbf{E}(\mathbf{r},t) \tag{3.8}$$

$$\nabla \cdot \mathbf{H}(\mathbf{r},t) = 0 \tag{3.9}$$

Light in Vacuum

Now we can derive the wave equation by applying $\nabla\times$ to (3.6) which yields

$$\begin{aligned} & \nabla \times \left[\nabla \times \mathbf{E}\right] = \nabla \times \left[-\mu_0 \frac{\partial}{\partial t}\mathbf{H}\right] \\ \iff \quad & \nabla(\nabla \cdot \mathbf{E}) - \Delta \mathbf{E} = -\mu_0 \frac{\partial}{\partial t} \nabla \times \mathbf{H} \quad \Big| \quad \text{subs. (3.8) and (3.7)} \\ \iff \quad & \frac{1}{c^2}\frac{\partial^2}{\partial t^2}\mathbf{E} - \Delta \mathbf{E} = -\mu_0 \frac{\partial}{\partial t}\mathbf{j} - \mu_0 \frac{\partial^2}{\partial t^2}\mathbf{P} + \frac{1}{\varepsilon_0}\nabla(\nabla \cdot \mathbf{P}) \end{aligned} \tag{3.10}$$

In vacuum ($\mathbf{P} = 0$ and $\mathbf{j} = 0$), the right side of this equation vanishes and we are left with $\frac{1}{c^2}\frac{\partial^2}{\partial t^2}\mathbf{E} - \Delta \mathbf{E} = 0$ which is solved by the plane wave $\mathbf{E} = \mathbf{E}_0 e^{i(\mathbf{kr}-\omega t)}$ where $\frac{\omega}{k} = c$. This describes the propagation of light through empty space: a sinusoidal oscillation in time and space along the $\mathbf{k}$ direction where $\mathbf{E}$, $\mathbf{B}$ and $\mathbf{k}$ are all perpendicular to each other.

Light in homogeneous isotropic materials

The next question we can answer is how light propagates through a homogeneous and isotropic material. For us, the dielectric polarization is some linear function of the electric field so $\mathbf{P}(\mathbf{r},t) = \hat{\chi}(\omega,\mathbf{r})\mathbf{E}(\mathbf{r},t)$. An isotropic material behaves the same for all orientations of $\mathbf{E}$ therefor $\hat{\chi}(\omega,\mathbf{r})$ becomes a scalar function $\chi(\omega,\mathbf{r})$. If the material is additionally homogeneous, that is the same everywhere independent of $\mathbf{r}$, then $\boldsymbol{\nabla}\chi(\omega,\mathbf{r}) = 0$. With equation (3.7) this gives us $\boldsymbol{\nabla}\cdot\mathbf{P} = 0$ and the wave equation simplifies to

$$\begin{aligned} \frac{\varepsilon}{c^2}\frac{\partial^2}{\partial t^2}\mathbf{E} - \Delta\mathbf{E} = 0, \\ \text{where} \quad \varepsilon\,\mathbf{E} := (1+\chi)\mathbf{E} = \mathbf{E} + \mathbf{P}. \end{aligned} \tag{3.11}$$

This implies that light behaves in these materials exactly as it would in vacuum, we only have to account for a decreased speed of light $c = c_0/\sqrt{\varepsilon} =: c_0/n$ with the refractive index n. This is equivalent to a decreased wavelength $\lambda =: \lambda_0/n$ or an increased wave number $k = 2\pi/\lambda = n\,k_0$, where c_0, λ_0 and k_0 are the quantities in vacuum. The wave equation is also solved by a complex valued $n := \eta + i\kappa$ which describes the behavior of the exponentially decaying field commonly found in metals

$$\mathbf{E} = \mathbf{E}_0 e^{i(n\mathbf{k}\mathbf{r}-\omega t)} = \mathbf{E}_0 \underbrace{e^{-\kappa\mathbf{k}\mathbf{r}}}_{\text{decay}} \underbrace{e^{i(\eta\mathbf{k}\mathbf{r}-\omega t)}}_{\text{oscillation}} \tag{3.12}$$

Interfaces

The metasurface stacks we want to understand are obviously not one homogeneous material. Rather, they contain many interfaces between different materials and we can again use the maxwell equations to predict how light will behave at such an interface.

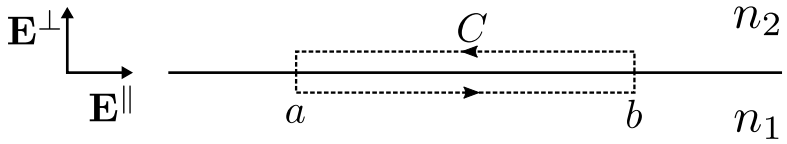


Figure 1: An interface of two materials with different refractive indices n_1 and n_2 and a closed contour C which is tangent to the interface between the points a and b. A field vector within the depicted plane can be decomposed into a tangential and a orthogonal component to the interface, $\mathbf{E}^{\parallel}$ and $\mathbf{E}^{\perp}$, respectively.

Let us consider a closed contour C around an interface as seen in Figure 1 and integrate the first maxwell equation (3.6) over the surface A enclosed by that contour

$$\begin{aligned} \int_A \boldsymbol{\nabla}\times\mathbf{E}(\mathbf{r},t)\,\mathrm{d}\mathbf{A} &= -\mu_0\frac{\partial}{\partial t}\int_A \mathbf{H}(\mathbf{r},t)\mathrm{d}\mathbf{A} \\ \overset{\text{Stokes}}{\Longleftrightarrow} \qquad \int_{C=\partial A} \mathbf{E}(\mathbf{r},t)\mathrm{d}\mathbf{r} &= -\mu_0\frac{\partial}{\partial t}\int_A \mathbf{H}(\mathbf{r},t)\mathrm{d}\mathbf{A} \end{aligned} \tag{3.13}$$

But now we can bring the contour infinitesimally close to the interface and thus reduce the right hand side of the equation, the total magnetic flux through the surface, to zero. That leaves us with

$$\begin{aligned} \int_a^b \mathbf{E}_1\,\mathrm{d}\mathbf{r} \;+ \int_b^a \mathbf{E}_2\,\mathrm{d}\mathbf{r} &= 0 \\ \Longleftrightarrow \qquad \int_a^b \mathbf{E}_1\,\mathrm{d}\mathbf{r} &= \int_a^b \mathbf{E}_2\,\mathrm{d}\mathbf{r} \end{aligned} \tag{3.14}$$

Because a and b were chosen arbitrarily, that means that the transverse field components along the path need to be continuous so $\mathbf{E}_1^{\parallel} = \mathbf{E}_2^{\parallel}$. The analog expression $\mathbf{H}_1^{\parallel} = \mathbf{H}_2^{\parallel}$ can be shown by starting with the third maxwell equation (3.8) instead. We can now use these continuity conditions to learn more about the behavior of light at these interfaces.

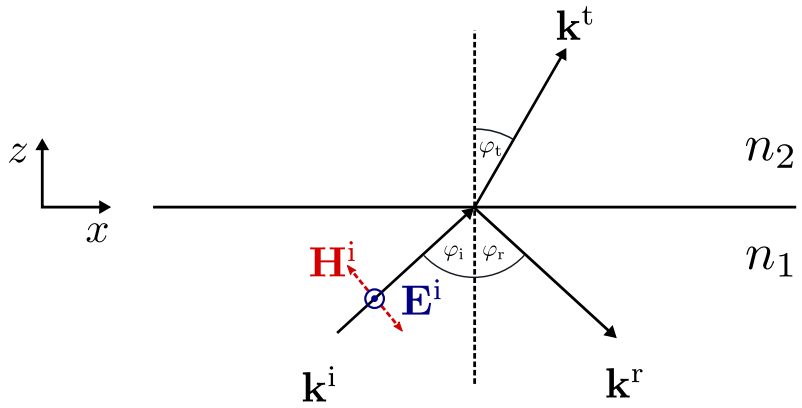


Figure 2: Interface between two materials of different refractive indices n_1 and n_2. The incident light has a wave vector $\mathbf{k}^{\mathrm{i}}$ and is partially transmitted and partially reflected. The fields are shown in transverse electric (TE) polarization where $\mathbf{H}$ is in the x-z plane and $\mathbf{E}$ oscillates in the y direction. In this polarization state the $\mathbf{E}$ field is fully tangential to the interface and therefor $\mathbf{E} = \mathbf{E}^{\parallel}$ and $\mathbf{E}^{\perp} = 0$. For the $\mathbf{H}$ field only the x component is tangential to the interface and the z component is orthogonal to it, so $\mathbf{H}_x = \mathbf{H}^{\parallel}$ and $\mathbf{H}_z = \mathbf{H}^{\perp}$.

Let us consider the same interface from before and have an incident field $\mathbf{E}^{\mathrm{i}}$ interact with it. From figure 2 we can see

$$\mathbf{k}^{\mathrm{i}} = k_0 n_1 \begin{pmatrix} -\sin\varphi_{\mathrm{i}} \\ 0 \\ -\cos\varphi_{\mathrm{i}} \end{pmatrix}, \quad \mathbf{k}^{\mathrm{r}} = k_0 n_1 \begin{pmatrix} \sin\varphi_{\mathrm{r}} \\ 0 \\ -\cos\varphi_{\mathrm{r}} \end{pmatrix} \quad \text{and} \quad \mathbf{k}^{\mathrm{t}} = k_0 n_2 \begin{pmatrix} \sin\varphi_{\mathrm{t}} \\ 0 \\ \cos\varphi_{\mathrm{t}} \end{pmatrix}. \tag{3.15}$$

For the derivation it is useful to decompose the fields into orthogonal polarization components. Here we are going to use the transversal electric (TE) and transversal magnetic (TM) polarizations. The TE component can be seen in figure 2 and in TM polarization $\mathbf{E}$ and $\mathbf{H}$ fields are simply interchanged. If we apply the continuity condition from before to the TE component, we get that $\mathbf{E}_1 = \mathbf{E}_2$ at $z = 0$ therefor

$$\begin{aligned} & \mathbf{E}^{\mathrm{i}} + \mathbf{E}^{\mathrm{r}} = \mathbf{E}^{\mathrm{t}} \\ \iff \quad & E^{\mathrm{i}} e^{i(k_x^{\mathrm{i}} x)}\, \vec{\mathbf{e}}_y + E^{\mathrm{r}} e^{i(k_x^{\mathrm{r}} x)}\, \vec{\mathbf{e}}_y = E^{\mathrm{t}} e^{i(k_x^{\mathrm{t}} x)}\, \vec{\mathbf{e}}_y, \end{aligned} \tag{3.16}$$

where $E^{\mathrm{i}}, E^{\mathrm{r}}, E^{\mathrm{t}} \in \mathbb{R}$ are the measurable real field magnitudes. This is only possible for all x if

$$E^{\mathrm{i}} + E^{\mathrm{r}} = E^{\mathrm{t}} \quad \text{and} \quad k_x^{\mathrm{i}} = k_x^{\mathrm{r}} = k_x^{\mathrm{t}}. \tag{3.17}$$

Two basic laws of optics lie in this relation: The law of reflection

$$k_x^{\mathrm{i}} = k_x^{\mathrm{r}} \quad \Rightarrow \quad k_0 n_1 \sin\varphi_i = k_0 n_1 \sin\varphi_r \quad \Rightarrow \quad \varphi_i = \varphi_r := \varphi_1 \tag{3.18}$$

and the Snells law of refraction

$$k_x^{\mathrm{i}} = k_x^{\mathrm{t}},\ \varphi_t := \varphi_2 \quad \Rightarrow \quad k_0 n_1 \sin\varphi_1 = k_0 n_2 \sin\varphi_2 \quad \Rightarrow \quad n_1 \sin\varphi_1 = n_2 \sin\varphi_2. \tag{3.19}$$

Fresnel Equations

To fully describe an interface we also need to know the transmitted and the reflected field amplitudes, $t := E^t/E^i$ and $r := E^r/E^i$, respectively. Again using the TE polarization shown in figure 2 we see

$$\mathbf{H}^{\mathrm{i}} = H^{\mathrm{i}} \begin{pmatrix} -\cos\varphi_{\mathrm{i}} \\ 0 \\ -\sin\varphi_{\mathrm{i}} \end{pmatrix}, \quad \mathbf{H}^{\mathrm{r}} = H^{\mathrm{r}} \begin{pmatrix} \cos\varphi_{\mathrm{r}} \\ 0 \\ \sin\varphi_{\mathrm{r}} \end{pmatrix} \quad \text{and} \quad \mathbf{H}^{\mathrm{t}} = H^{\mathrm{t}} \begin{pmatrix} -\cos\varphi_{\mathrm{t}} \\ 0 \\ \sin\varphi_{\mathrm{t}} \end{pmatrix}. \tag{3.20}$$

The H_x component is tangent to the interface, so it also needs to be continuous across the boundary

$$\begin{aligned} & H_x^{\mathrm{i}} + H_x^{\mathrm{r}} = H_x^{\mathrm{t}} \\ \iff \quad & H^{\mathrm{i}} \cos\varphi_1 - H^{\mathrm{r}} \cos\varphi_1 = H^{\mathrm{t}} \cos\varphi_2 \end{aligned} \tag{3.21}$$

If we assume the described plane waves to be $\mathbf{E} = \mathbf{E}_0 e^{i(n\mathbf{kr}-\omega t)}$ and $\mathbf{H} = \mathbf{H}_0 e^{i(n\mathbf{kr}-\omega t)}$, Maxwell's first equation (3.6) allows us to connect the magnitudes of $\mathbf{H}$ and $\mathbf{E}$ via the refractive index $H \sim n\,E$. This changes (3.21) to

$$n_1\, E^{\mathrm{i}} \cos\varphi_1 - n_1\, E^{\mathrm{r}} \cos\varphi_1 = n_2\, E^{\mathrm{t}} \cos\varphi_2. \tag{3.22}$$

Substituting E^{r} or E^{i} with (3.17) and rearranging we get

$$t = \frac{2n_1 \cos\varphi_1}{n_1 \cos\varphi_1 + n_2 \cos\varphi_2} \quad \text{and} \quad r = \frac{n_1 \cos\varphi_1 - n_2 \cos\varphi_2}{n_1 \cos\varphi_1 + n_2 \cos\varphi_2}. \tag{3.23}$$

These are called the Fresnel coefficients for the TE component. The TM component can be treated analog which yields

$$t = \frac{2n_1 \cos\varphi_1}{n_2 \cos\varphi_1 + n_1 \cos\varphi_2} \quad \text{and} \quad r = \frac{n_2 \cos\varphi_1 - n_1 \cos\varphi_2}{n_2 \cos\varphi_1 + n_1 \cos\varphi_2}. \tag{3.24}$$

For perpendicular incident light at $\varphi = 90°$ these should describe the same situation as we can no longer differentiate between TE and TM components and indeed for this angle they are equivalent if we consider that the TE coefficients describe the electric field and the TM coefficients describe the magnetic field. The TE coefficients become

$$t = \frac{2n_1}{n_1 + n_2} \quad \text{and} \quad r = \frac{n_1 - n_2}{n_1 + n_2}. \tag{3.25}$$

Stacking

We can take this one step further by including interfaces which have incident light from both sides. Let $\mathbf{E}_{\mathrm{out}}$ be the light coming out of the interface and $\mathbf{E}_{\mathrm{in}}$ be the light going into the interface as seen in figure 3. Using the factors from the Fresnel equations we get

$$E^b_{\mathrm{out}} = E^f_{\mathrm{in}}\, t^f + E^b_{\mathrm{in}}\, r^b \quad \text{and} \quad E^f_{\mathrm{out}} = E^b_{\mathrm{in}}\, t^b + E^f_{\mathrm{in}}\, r^f \tag{3.26}$$

and this can be expressed more concisely using matrices and vectors

$$\begin{pmatrix} E^f_{\mathrm{out}} \\ E^b_{\mathrm{out}} \end{pmatrix} = \underbrace{\begin{pmatrix} t^f & r^b \\ r^f & t^b \end{pmatrix}}_{=:\hat{S}} \begin{pmatrix} E^f_{\mathrm{in}} \\ E^b_{\mathrm{in}} \end{pmatrix}. \tag{3.27}$$

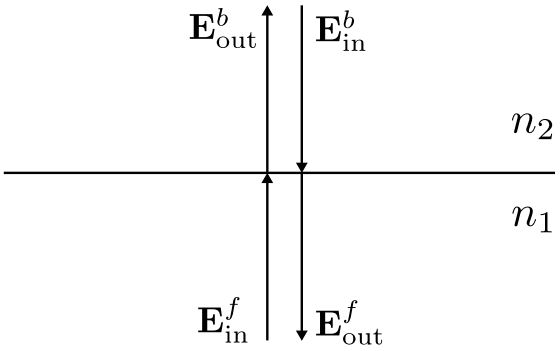


Figure 3: Interface between two materials of different refractive indices n_1 and n_2 where incident light is coming from the front and the back.

We call these matrices, that map field-in to field-out, scattering matrices or short S-matrices. They can be used to describe more than just interfaces. As shown in the paragraph Light in materials when light propagates through homogeneous isotropic material just the phase changes by a factor of e^{ik_0nd}, where d is the distance traveled and $\mathbf{k}_0$ the wave vector in vacuum. We can express this propagation by allowing complex valued t and r

$$\hat{S}_{n,d} = \begin{pmatrix} e^{ik_0nd} & 0 \\ 0 & e^{ik_0nd} \end{pmatrix} \tag{3.28}$$

and analog the S-matrix for an interface from n_1 to n_2 is

$$\hat{S}_{n_1,n_2} = \begin{pmatrix} \frac{2n_1}{n_1+n_2} & \frac{n_2-n_1}{n_1+n_2} \\ \frac{n_1-n_2}{n_1+n_2} & \frac{2n_1}{n_1+n_2} \end{pmatrix}. \tag{3.29}$$

Say we have a stack of different homogeneous isotropic materials. We are now able to write down an S-matrix for every part of the stack, that is for every interface and every propagation between interfaces as seen in figure 4, but we cannot predict the behavior of the stack as a whole.

The combined S-matrix $\hat{S}$ consisting of $\hat{S}_1$ and $\hat{S}_2$ cannot be obtained by simply using the matrix product. The result would be

$$\hat{S}_1\,\hat{S}_2 = \begin{pmatrix} t_1^f\,t_2^f + r_1^b\,r_2^f & t_1^f\,r_2^b + r_1^b\,t_2^b \\ r_1^f\,t_2^f + t_1^b\,r_2^f & r_1^f\,r_2^b + t_1^b\,t_2^b \end{pmatrix} \neq \begin{pmatrix} t^f & r^b \\ r^f & t^b \end{pmatrix} = \hat{S} \tag{3.30}$$

and this would imply that the transmission from the front t^f increases when the internal reflections of the system r_1^b and r_2^f increase which is the opposite of the behavior we expect. In the 1960s Redheffer [8] developed a linear transformation called *star product* $\star$ which yields the correct combined S-matrix

$$\hat{S}_1 \star \hat{S}_2 := \begin{pmatrix} t_2^f(1-r_1^br_2^f)^{-1}t_1^f & r_2^b + t_2^fr_1^b(1-r_2^fr_1^b)^{-1}t_2^b \\ r_1^f + t_1^br_2^f(1-r_1^br_2^f)^{-1}t_1^f & t_1^b(1-r_2^fr_1^b)^{-1}t_2^b \end{pmatrix}. \tag{3.31}$$

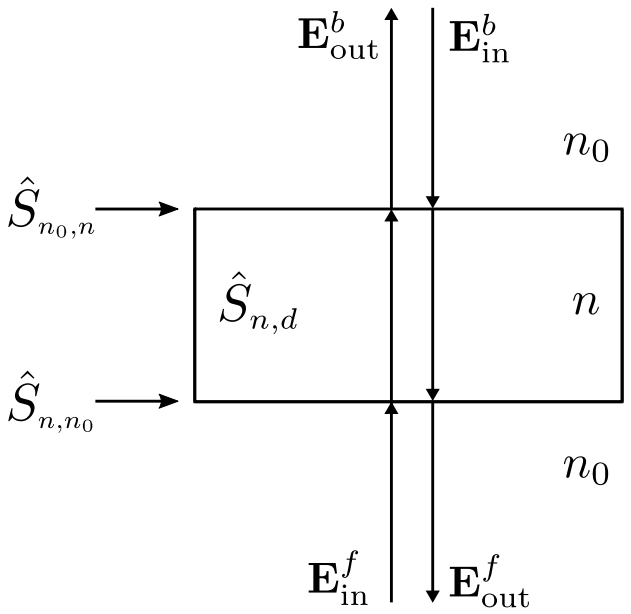


Figure 4: A slab of one homogeneous isotropic material with thickness d in air (n_0). This setup is fully described by two interface S-matrices $\hat{S}_{n_0,n}$, $\hat{S}_{n,n_0}$ and one propagation S-matrices $\hat{S}_{n,d}$.

For this operation to produce physically valid results, certain conditions have to be met. We will discuss these in detail in the paragraph Conditions of section 3.2. Applied to the example of figure 4, the star product gives

$$\hat{S} = \hat{S}_{n_0,n} \star \hat{S}_{n,d} \star \hat{S}_{n,n_0}. \tag{3.32}$$

Polarization

Up to this point the S-matrix is made of the four complex numbers t^f, r^b, r^f and t^b. This means when a S-matrix is applied to the input vector containing the total incoming light $\begin{pmatrix} E^f_{\text{in}} \\ E^b_{\text{in}} \end{pmatrix}$, E^f_{in} and E^b_{in} are also restricted to the complex plane and can only describe linear polarized light in one direction. The last generalization we want to make is to extend the scattering matrix calculus to all polarizations. As shown in the paragraph Light in materials a planar light wave propagating along the z axis through a homogeneous isotropic material can be described as

$$\mathbf{E} = \begin{pmatrix} E_x e^{i(kz-\omega t+\varphi_x)} \\ E_y e^{i(kz-\omega t+\varphi_y)} \\ 0 \end{pmatrix} = \left(E_x\, e^{i\varphi_x}\, \vec{\mathbf{e_x}} + E_y\, e^{i\varphi_y}\, \vec{\mathbf{e_y}}\right) e^{i(kz-\omega t)}. \tag{3.33}$$

The waves polarization is determined by the scaling factors of $\vec{\mathbf{e_x}}$ and $\vec{\mathbf{e_y}}$ and can be expressed as a *Jones Vector* $\mathbf{j} \in \mathbb{C}^2$

$$\mathbf{j} = \frac{1}{\sqrt{E_x^2 + E_y^2}} \begin{pmatrix} E_x \\ E_y\, e^{i\delta} \end{pmatrix}, \quad \text{with} \quad \delta := \varphi_y - \varphi_x. \tag{3.34}$$

Now all linear operations on the polarization are matrices $\hat{M} \in \mathbb{C}^{2\times 2}$. This means all passive components have a corresponding matrix. Examples for components aligned with the x axis are

$$\begin{aligned} \text{linear polarizer:} \quad & \hat{M} = \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix}, \\ \lambda/4 \text{ plate:} \quad & \hat{M} = \begin{pmatrix} 1 & 0 \\ 0 & i \end{pmatrix} e^{-\frac{i\pi}{4}} \quad \text{and} \\ \lambda/2 \text{ plate:} \quad & \hat{M} = \begin{pmatrix} -i & 0 \\ 0 & i \end{pmatrix}. \end{aligned} \tag{3.35}$$

We can now generalize the S-matrix calculus simply by allowing Jones matrices in place of the Fresnel coefficients: $t \to \hat{T}$ and $r \to \hat{R}$. The star product of two S-matrices becomes

$$\hat{S}_1 \star \hat{S}_2 = \begin{pmatrix} \hat{T}_1^f & \hat{R}_1^b \\ \hat{R}_1^f & \hat{T}_1^b \end{pmatrix} \star \begin{pmatrix} \hat{T}_2^f & \hat{R}_2^b \\ \hat{R}_2^f & \hat{T}_2^b \end{pmatrix} = \begin{pmatrix} \hat{T}_2^f (\mathbb{1} - \hat{R}_1^b \hat{R}_2^f)^{-1} \hat{T}_1^f & \hat{R}_2^b + \hat{T}_2^f \hat{R}_1^b (\mathbb{1} - \hat{R}_2^f \hat{R}_1^b)^{-1} \hat{T}_2^b \\ \hat{R}_1^f + \hat{T}_1^b \hat{R}_2^f (\mathbb{1} - \hat{R}_1^b \hat{R}_2^f)^{-1} \hat{T}_1^f & \hat{T}_1^b (\mathbb{1} - \hat{R}_2^f \hat{R}_1^b)^{-1} \hat{T}_2^b. \end{pmatrix} \tag{3.36}$$

So the generalized S-matrix is a 2×2 block matrix of four Jones matrices and the input vector $\begin{pmatrix} \mathbf{E}^f_{\text{in}} \\ \mathbf{E}^b_{\text{in}} \end{pmatrix}$ is a 2×1 block vector of two Jones vectors.

3.2. SASA

The S-matrix calculus enables us to calculate the optical behavior of a stack if we know the optical behavior of all its layers in the form of their S-matrices. In section 3.1 we derived the S-matrices of propagation through and interfaces between homogeneous isotropic materials. This calculation is based on the Fresnel equations and is therefore analytic. However, we cannot calculate the S-matrix of a metasurface analytically. This results in a semi-analytic Stacking Algorithm where the S-matrices of the metasurfaces need to be provided but the remaining S-matrices can be calculated based on geometric parameters.

A core idea of this algorithm is the fact that geometric transformations like rotation and mirroring can be applied directly to Jones matrices. For example, let $\hat{M}$ be the Jones matrix of an optical component, then the Jones matrix of the rotated component $\hat{M}_\varphi$ is calculated as

$$\hat{M}_\varphi = \hat{\Theta}(-\varphi)\,\hat{M}\,\hat{\Theta}(\varphi), \quad \text{where} \quad \hat{\Theta}(\varphi) = \begin{pmatrix} \cos\varphi & \sin\varphi \\ -\sin\varphi & \cos\varphi \end{pmatrix}. \tag{3.37}$$

On this basis we can rotate S-matrices in a similar manner and

$$\hat{S}_\varphi = \begin{pmatrix} \hat{\Theta}(-\varphi)\hat{T}^f\hat{\Theta}(\varphi) & \hat{\Theta}(-\varphi)\hat{R}^b\hat{\Theta}(\varphi) \\ \hat{\Theta}(-\varphi)\hat{R}^f\hat{\Theta}(\varphi) & \hat{\Theta}(-\varphi)\hat{T}^b\hat{\Theta}(\varphi) \end{pmatrix}. \tag{3.38}$$

Flipping and mirroring of S-matrices is done analogously. With all mathematical operations well defined we can write down SASA in pseudo code.

Input: Stack = [Layer 0, Layer 1, ...]
Output: Stack S-matrix
S-mats = [ ]
for $i = 0$; $i <$ len(Stack); $i = i + 1$; **do**
  if *Stack*[i] **is** *metasurface* **then**
    add layer $\hat{S}$ to S-mats
  else
    calculate propagation S-matrix $\hat{S}_{n_i,d_i}$
    add $\hat{S}_{n_i,d_i}$ to S-mats
  end
  apply geometric transformations to added S-matrix
  calculate interface S-matrix $\hat{S}_{n_i,n_{i+1}}$
  add $\hat{S}_{n_i,n_{i+1}}$ to S-mats
end
$\hat{S} = \mathbb{1}$
for $i = 0$; $i <$ len(*S-mats*); $i = i + 1$; **do**
  $\hat{S} = \hat{S} \star S$-mats[$i$]
end
return $\hat{S}$

SASA pseudo code: A Layer variable holds all the information of that layer, that is the S-matrix if its a metasurface and the refractive index n and thickness d otherwise.

Conditions

For this algorithm to produce physical valid output, some conditions need to be met. Equations that are boundary conditions to the algorithm will be marked bold. It has been shown [9] that the S-matrix calculus can only be used when the meta materials in the stack do not interact with each other via the near field. That is, only the zeroth diffraction order of a metasurface can be non-evanescent and higher orders need to have decayed enough when they reach the next metasurface.

First, we need to ensure that only the zeroth diffraction order is non evanescent by requiring

$$\lambda > \max(n^f, n^b) \cdot \Lambda. \tag{3.39}$$

Additionally, the distance between metasurfaces needs to be large enough so that all higher order modes have decayed. It was shown in [3] that if we demand

$$e^{\mathrm{Im}(k_z)d} < e^{-2} \approx 0.14, \quad \text{then} \quad d > \frac{\Lambda}{\pi\sqrt{1 - \frac{\Lambda^2 n^2}{\lambda^2}}}. \tag{3.40}$$

The factor e^{-2} was chosen arbitrarily. A smaller value increases the agreement of the S-matrix calculus and more rigorous numerical methods which also take into account the near field. One of these, the Fourier Modal Method will be described in more detail in section 4.4.

Symmetries

We can use geometric transformations to examine the S-matrices of metasurfaces which satisfy certain symmetries. We will start with mirror symmetry. The Jones matrix $\hat{M}'$ of a mirrored component $\hat{M}$ can be calculated as

$$\hat{M}' = \hat{F}^{-1}\hat{M}\hat{F} \quad \text{where} \quad \hat{F} = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}. \tag{3.41}$$

If a metasurface has mirror symmetry, the S-matrix needs to satisfy $\hat{S} = \hat{S}'$ and therefore all the contained Jones matrices need to satisfy this condition too.

$$\begin{gathered} \text{Let} \quad \hat{T}^f = \begin{pmatrix} A & B \\ C & D \end{pmatrix}, \quad \text{then} \\ \left(\hat{T}^f\right)' = \hat{F}^{-1}\hat{T}^f\hat{F} = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}\begin{pmatrix} A & B \\ C & D \end{pmatrix}\begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \overset{!}{=} \begin{pmatrix} A & B \\ C & D \end{pmatrix} = \hat{T}^f \\ \Rightarrow \qquad \hat{T}^f = \begin{pmatrix} A & 0 \\ 0 & D \end{pmatrix}. \end{gathered} \tag{3.42}$$

We can also examine rotational symmetry with the rotation matrix $\hat{\Theta}(\varphi)$. A metasurface of C_4 symmetry, for example square shaped meta particles, maps onto itself if rotated 90°. That means the S-matrix should also remain equal, so $\hat{S}_{\pi/2} \overset{!}{=} \hat{S}$ and

$$\begin{gathered} \hat{T}^f_{\pi/2} = \hat{\Theta}(-\pi/2)\,\hat{T}^f\,\hat{\Theta}(\pi/2) \overset{!}{=} \hat{T}^f, \quad \hat{\Theta}(\pi/2) = \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix} \\ \Rightarrow \qquad \hat{T}^f = \begin{pmatrix} A & B \\ -B & A \end{pmatrix}. \end{gathered} \tag{3.43}$$

So if a metasurface has both mirror and C_4 symmetry, its $\hat{T}^f$ matrix has the form

$$\hat{T}^f = \begin{pmatrix} A & 0 \\ 0 & A \end{pmatrix}. \tag{3.44}$$

In this case the star product is commutative and therefor the stack will behave the same for light coming from the top and bottom. It should be noted that these considerations apply to all Jones matrices and not only to $\hat{T}^f$.

3.3. Plasmonic Metasurfaces

As stated in the introduction, metasurfaces are 2D materials with some sub-wavelength structure. Their properties are determined by the geometry of this structure which is usually a simple shape, like a rectangle or oval, on a periodic grid. The shape is called metaatom and in the case of plasmonic metasurfaces it is made of metal. With the S-matrix calculus and SASA we can fully describe stacks of homogeneous isotropic materials but we still have no understanding of what happens when we add a metasurface to a stack. To build some intuition for how a specific metasurface will affect the transmission spectrum of a stack, we will need a little more theory on optics in materials. As stated before, all optical properties of a material a captured by the $\mathbf{D}$ and $\mathbf{H}$ fields and in our context

$$\mathbf{D} = \epsilon_0 \mathbf{E} + \mathbf{P} = \epsilon_0 \underbrace{(1+\chi)}_{:=\epsilon} \mathbf{E} \tag{3.45}$$

so the $\mathbf{D}$ field is determined by the $\mathbf{E}$ field and dielectric function ϵ. In the following sections we will discuss which materials have which kind of dielectric functions and how we can use these to gain some understanding of the optical mechanisms in a metasurface.

Dielectric Function

In the simplest model we can describe electrons in a material as an ensemble of harmonic oscillators

$$m\ddot{\mathbf{x}} + m\gamma\dot{\mathbf{x}} + m\omega_0^2\mathbf{x} = -e\mathbf{E}, \tag{3.46}$$

with the displacement $\mathbf{x}$, electron mass m, electron charge e, dampening factor γ and resonance frequency ω_0. In this model the macroscopic polarization $\mathbf{P}$ is directly caused by this displacement $\mathbf{x}$ through

$$\mathbf{P} = -\rho e\mathbf{x}, \tag{3.47}$$

where ρ is the density of electrons. If we assume a harmonic time dependency $\mathbf{E}(t) = \mathbf{E}_0 e^{i\omega t}$ equation (3.46) is solved by

$$\mathbf{x}(t) = \frac{e}{m(\omega^2 + i\gamma\omega - \omega_0^2)}\mathbf{E}(t) \tag{3.48}$$

and using equation (3.45) and (3.47) this results in the $\mathbf{D}$ field

$$\mathbf{D} = \epsilon_0 \underbrace{\left(1 - \frac{f}{\omega^2 + i\gamma\omega - \omega_0^2}\right)}_{=\epsilon} \mathbf{E}, \tag{3.49}$$

where $f = \rho e^2/\epsilon_0 m$ is the oscillator strength. In a real material there are always multiple resonance frequencies ω_m but in good approximation these do not influence each other and the total dielectric function can be obtained simply by summing over all m [10]

$$\epsilon(\omega) := \epsilon'(\omega) + i\epsilon''(\omega) = 1 + \sum_m \frac{f_m}{\omega_m^2 - \omega_0^2 - i\gamma\omega} \tag{3.50}$$

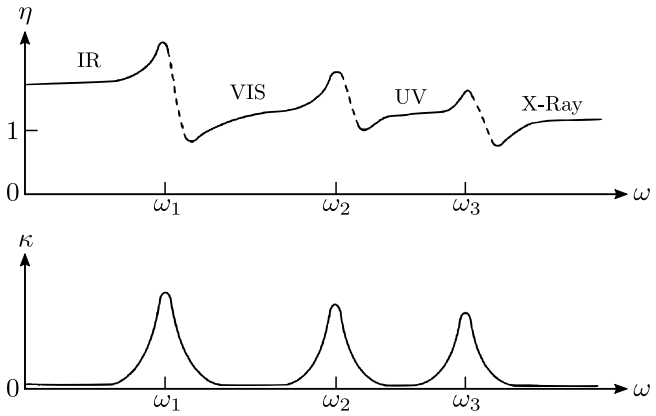


Figure 5: Example plots for the real and imaginary part of the refractive index $n = \eta + i\kappa$ as calculated by the multiple resonance Lorentz model. [10]

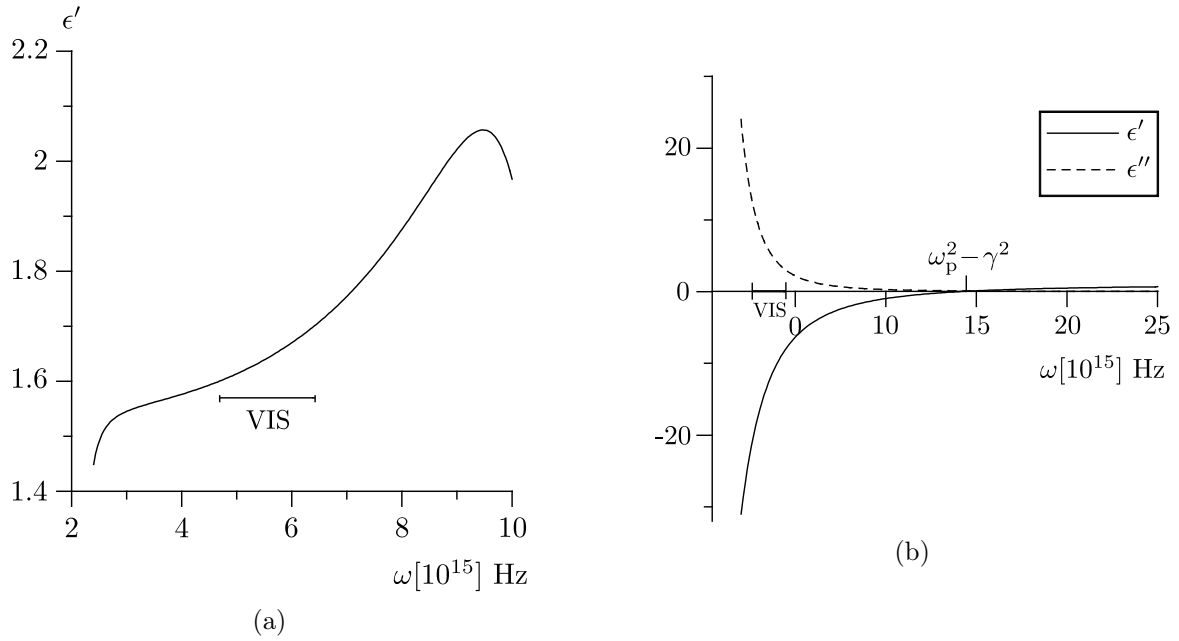


Figure 6: Imaginary and real part of the dielectric function $\epsilon = \epsilon' + i\epsilon''$ for dielectrics (a) and metals (b) in the visible spectrum. [10]

In section 3.1 we have already discussed how the dielectric function is related to the complex refractive index by

$$\epsilon = n^2 = (\eta + i\kappa)^2 \tag{3.51}$$

and this relationship is used to generate figure 5. We observe high κ, so high absorption, at the resonance frequencies in the from of Lorentz peaks. Between resonance frequencies generally $\partial\eta/\partial\omega > 0$. This is called normal dispersion and explains, for example, why red light is diffracted stronger than blue light. Figure 5 is just a general example. To understand metasurfaces we need to discuss how the resonance frequencies are distributed in real materials.

Dielectrics in the visible spectrum are well described by two resonance frequencies. One in the IR and one in the UV range. That means in the visible spectrum we have a vanishing imaginary part ϵ'' of the dielectric function and normal dispersion as seen in figure 6a. Metals on the other hand are characterized by their high availability in free charges, thereby eliminating the restoring force ($\omega_0 = 0$) and reducing the dielectric function to

$$\epsilon(\omega) = 1 - \frac{\omega_p^2}{\omega^2 + i\gamma\omega}\,, \tag{3.52}$$

with the plasma frequency $\omega_p^2 = f = \rho e^2/\epsilon_0 m$. This results in a large negative ϵ' and a positive ϵ'' as seen in figure 6b. The negative ϵ' is responsible for the reflection of visible light at metallic surfaces.

Surface Plasmon Polaritons

Finally, we can discuss the mechanism behind the optical properties of plasmonic metasurfaces. These work by allowing the excitation of a special electromagnetic wave at a metal dielectric interface called Surface Plasmon Polariton (SPP). This wave is confined to and travels along the interface until its energy is lost via absorption or scattering. That means it enables some photons to be absorbed by coupling into the interface. At which wavelengths this conversion occurs depends on the metasurface geometry and can thus be tailored for a specific optical target.

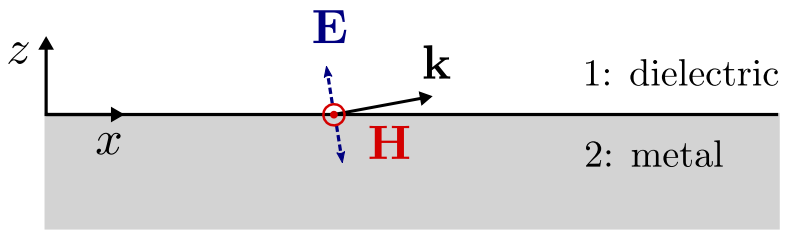


Figure 7: Metal dielectric interface with light in a TM mode.

Suppose the described wave exists as a TM mode, as can be seen in figure 7, and has the form

$$\begin{aligned} E_x &= E_0 e^{ik_x x - k_z z} \\ E_z &= E_0 \frac{k_x}{k_z} e^{ik_x x - k_z z} \\ H_y &= H_0 e^{ik_x x - k_z z} \end{aligned} \tag{3.53}$$

That is, propagating along the x axis and evanescently decaying along the z axis. This wave does indeed satisfy Maxwell's equation and the continuity conditions discussed in section 3.1 if and only if [11]

$$\frac{\epsilon_1}{k_{z1}} = -\frac{\epsilon_2}{k_{z2}} \tag{3.54}$$

and

$$k_x^2 + k_{zn}^2 = \epsilon_n \frac{\omega^2}{c^2} \quad \text{for} \quad n = 1, 2. \tag{3.55}$$

Equation (3.54) explains why SPP's are only possible at metal dielectric interfaces because ϵ_1' and ϵ_2' need to have different signs and this is fulfilled precisely for metals and dielectrics as can be seen in figure 6. If we solve equation (3.54) and (3.55), we can obtain the dispersion relation for SPP's

$$k_x = \frac{\omega}{c} \left(\frac{\epsilon_1 \epsilon_2}{\epsilon_1 + \epsilon_2} \right)^{1/2}. \tag{3.56}$$

This dispersion relation is shown in figure 8. For small wave vectors SPP's behave like light but then they start to have increasingly lower energy $E = \hbar\omega$ compared to a free photon at the same momentum $\mathbf{p} = \hbar\mathbf{k}$. However, for a conversion Photon $\rightarrow$ SPP both energy and momentum have to be conserved. Thus excitation of SPP's is not possible at a simple smooth metallic surface, but rather the wave vector of an incident photon has to be matched to that of an SPP of the same energy by the surface geometry.

To summarize, photons can couple into a plasmonic metasurface by exiting a special electromagnetic wave called Surface Plasmon Polariton confined to the interface. The exact wavelength at which this conversion is possible depends on the metaatom geometry and can thus be easily tuned to create different optical behaviors.

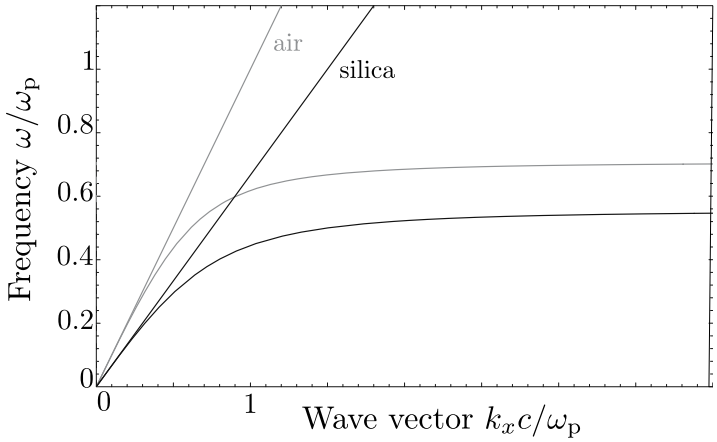


Figure 8: Dispersion relation for light and Surface Plasmon Polaritons in air (grey) and silica (black). Frequency and wave vector are normalized to the plasma frequency ω_p [11].

3.4. Neural Networks

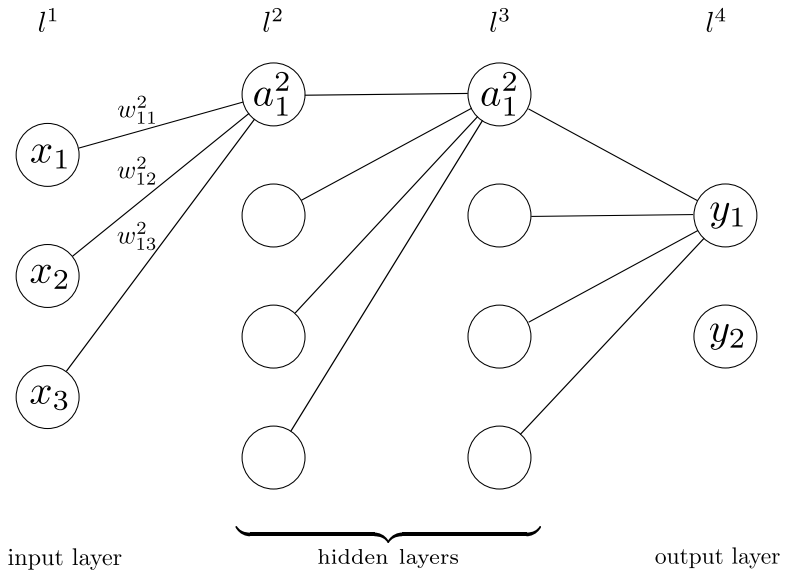


Figure 9: The most simple kind of NN is called densely connected or multilayer perceptron. For clarity only connections to the top most node of each layer are shown.

Artificial Neural Networks (ANN's or short NN's) are a kind of data structure inspired by the biological neurons found in nature. They can be used to find a wide range of input output relations. One classic example is mapping pictures of hand written digits to the actual digits. Rather than explicitly programmed, NN's are trained on a dataset (X, Y) of correct input output pairs.

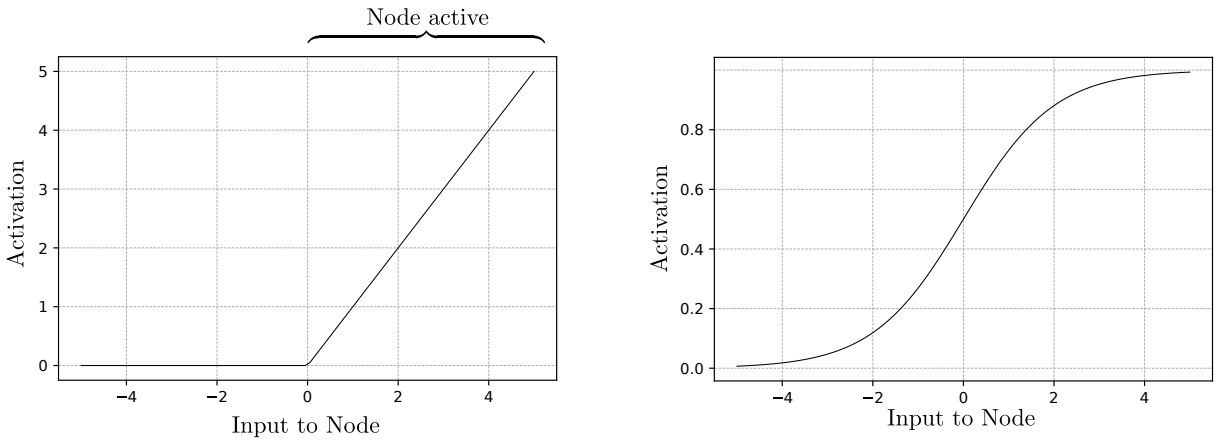
Multilayer Perceptron This kind of classic NN consist of single nodes or neurons which are organized into layers. The terms node and neuron can be used interchangeably. Every node is connected to all the nodes of the previous and the next layer. For this reason the network is called dense or densely connected. Each node holds a value called activation a where the activation to the first layer is the input to the network, here: (x_1, x_2, x_3). The nodes are connected by weights w which specify how much one node should influence the next and every node has a bias b to control at what total input activation the node itself should become active. To calculate the activation of a node, one has to multiply all the activations of the previous layer with their respective weights w, add the bias b and finally apply a non-linear activation function σ. In order to describe this process mathematically, we are going to use the usual index notation where superscripts specify the layer and subscripts the node. So a^2_1 is the activation of the first node in the second layer. To characterize each weight two subscripts are needed, for the end and beginning of the connection. For the example in figure 9 that gives

$$a^2_1 = \sigma\left(\sum_i w^2_{1i}\, x_i + b^2_1\right) \tag{3.57}$$

However, it is more convenient to stop considering every node individually and to view the involved quantities as vectors and matrices. So that (3.57) can be written as

$$\mathbf{a}^l = \sigma\underbrace{\left(\hat{\mathbf{w}}^l \mathbf{a}^{l-1} + \mathbf{b}^l\right)}_{:=\,\mathbf{z}^l}. \tag{3.58}$$

That means the activation function σ maps the total input of a neuron $\mathbf{z}^l$ to the output/activation of that neuron $\mathbf{a}^l$. Two examples for activation functions can be seen in Figure 10.



(a) rectified linear unit (ReLu)

(b) sigmoid

Figure 10: Two examples of activation functions σ. Especially the ReLu function has a destinct on and off state similar to a biological neurons.

Training

During training the networks output $\mathrm{NN}(\mathbf{x}) := \mathbf{y}'$ is calculated through repeated use of (3.58) and is then compared with the known correct output $\mathbf{y}$ by a cost function $C = C(\mathbf{y}, \mathbf{y}')$. The goal of the training is to minimize this function C. The cost function might simply be the mean squared difference between $\mathbf{y}$ and $\mathbf{y}'$

$$C_{\mathrm{mse}}(\mathbf{y}, \mathbf{y}') = \sum_i \left(y_i - y_i'\right)^2 \tag{3.59}$$

but there are different cost functions for different kind of outputs. For example, a network which predicts continuous values needs a different cost function than one predicting categories. More on this in section 4.2. Now we can quantify how well the NN is performing and are able to use this information to train the network.

As stated before, the goal of the training is to minimize the cost function C by changing the weights and biases. This means we are looking for the partial derivatives $\partial C/\partial \hat{w}^l_{j,k}$ and $\partial C/\partial b^l_j$. To find these we need the concept of the *error of a single neuron* δ^l_j, where

$$\delta^l_j := \frac{\partial C}{\partial z^l_j}. \tag{3.60}$$

This property expresses how much the final cost is affected by a change to the input of neuron j in layer l. For the last layer L, we can find a simple expression for this property by using the chain rule

$$\delta^L_j = \frac{\partial C}{\partial z^L_j} = \frac{\partial C}{\partial a^L_j}\,\frac{\partial a^L_j}{\partial z^L_j} \overset{(3.58)}{=} \frac{\partial C}{\partial a^L_j}\,\sigma'(z^L_j). \tag{3.61}$$

Remember, z^l_j is the input to a neuron and a^l_j is the output. That means δ^L_j is determined by a combination of how much the last output changes the cost function and how much the last input changes the last output. Notice how both these terms are easily accessible. z^l_j was already calculated on the forward pass and the derivatives of the cost and activation functions can be found analytically. By using the gradient operator $\boldsymbol{\nabla}$ and the Hadamard product $\odot$, where

$$\mathbf{a} \odot \mathbf{b} := \begin{pmatrix} a_1\, b_1 \\ a_2\, b_2 \\ \vdots \end{pmatrix}, \tag{3.62}$$

we can return to the more convenient vector notation and write

$$\boldsymbol{\delta}^L = \boldsymbol{\nabla}_a C \odot \sigma'(\mathbf{z}^L). \tag{3.63}$$

Now we know $\boldsymbol{\delta}$ for the last layer but not for the rest of the network. However, we can express the error of an arbitrary layer $\boldsymbol{\delta}^l$ by the error in the next layer $\boldsymbol{\delta}^{l+1}$ through

$$\boldsymbol{\delta}^l = \left[(\hat{w}^{l+1})^\top \boldsymbol{\delta}^{l+1}\right] \odot \sigma'(\mathbf{z}^l). \tag{3.64}$$

This equation can be intuitively understood as moving the error $\boldsymbol{\delta}^{l+1}$ back one layer by applying $(\hat{w}^{l+1})^\top$ and then through the activation function of layer l by applying σ'. In a sense its again "simply" a chain rule. With these two equations, (3.63) and (3.64), all the errors $\boldsymbol{\delta}$ in the network are known. Just start with the last layer and work your way backwards. This idea of moving the error backwards is the reason why the algorithm is called *Backpropagation*.

The last thing to do is to relate $\boldsymbol{\delta}$ back to the original derivatives $\partial C/\partial \hat{w}^l_{j,k}$ and $\partial C/\partial b^l_j$. Using equation (3.58) gives

$$\partial C/\partial b^l_j = \delta^l_j \quad \text{and} \tag{3.65}$$

$$\partial C/\partial \hat{w}^l_{j,k} = a^{l-1}_k \, \delta^l_j. \tag{3.66}$$

Now we could modify the weights after every forward pass by

$$\hat{w}^l_{j,k} \leftarrow \hat{w}^l_{j,k} - \eta \, \partial C/\partial \hat{w}^l_{j,k}\,, \tag{3.67}$$

with the learning rate η . This method is called *stochastic gradient descent* and because the weights and biases are updated on every training sample the cost function can oscillate strongly. This is why in practice updates are performed on small *mini-batches* of training samples. This concludes the training of a Multilayer Perceptron. Other Neural Networks are trained in a similar manner. An excellent comprehensive explanation of Backpropergation was published by Nielsen [12].

Convolutional Neural Networks

An area where NNs have been very successful is image recognition or more general computer vision but the described multilayer perceptron has a number of weaknesses for this kind of task. Let's say our input is a n by n gray scale image. This can be expressed as a $n \times n$ matrix, flattened and fed into the input layer as seen in figure 11. But now the number of weights to the next layer $\hat{\mathbf{w}}^2$ is $n \cdot n \cdot l^2$ which soon becomes unfeasible. As described in the section on Notation, l^2 is here the number of nodes in the second layer and not l squared.

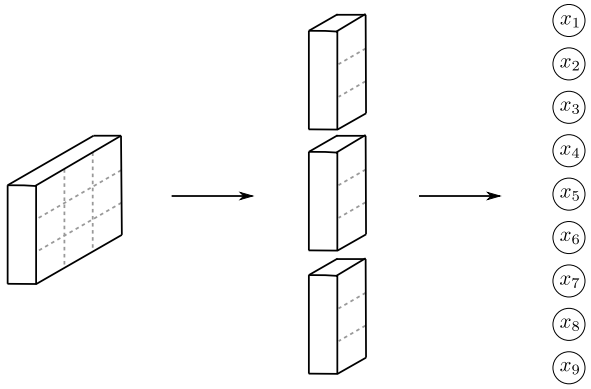


Figure 11: Flattening of a 3×3 matrix to fit the input of a multilayer perceptron.

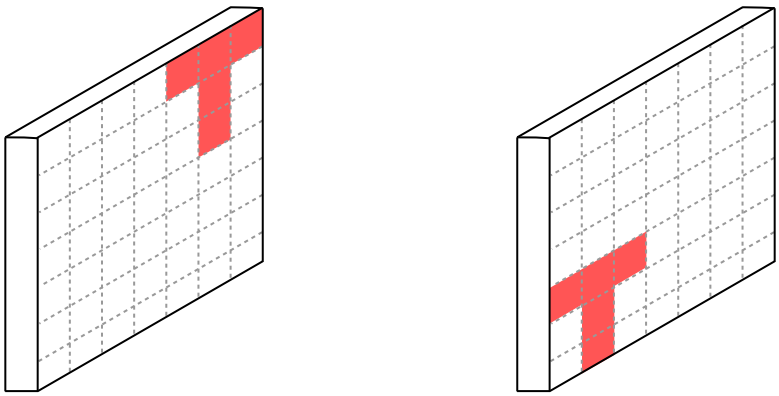


Figure 12: Two pictures of a T at different positions where the red color signifies a high value in the grayscale image. After the flatten operation seen in figure 11 very different nodes are active.

Computational limits aside, there is another problem. Imagine an image with the letter T in the top right corner. If this letter moves to a different position, as in figure 12, the networks reaction will be completely different because the weights and biases involved are completely different. So the NN cannot learn the concept "letter T" independent of its position in the picture. The information about the distance between pixels is lost. These problems led to the development of a new kind of layer called *Convolution*. A fixed size squared matrix called *kernel* is shifted over the matrix and at every position the point wise product between kernel and matrix is calculated and summed as shown in figure 13.

The result of this operation, called *feature map* of that kernel, is shown in figure 14. Notice how the greatest value of the feature map is at the position of the letter T. So with only a small number of weights the convolution is able to detect the T independent of its position in the image. This is still slightly misleading because this "T kernel" was intentionally constructed to find the T. In a real convolutional layer the kernel values are trained via Backpropagation similar to the weights of a Multilayer Perceptron as described in the paragraph Training.

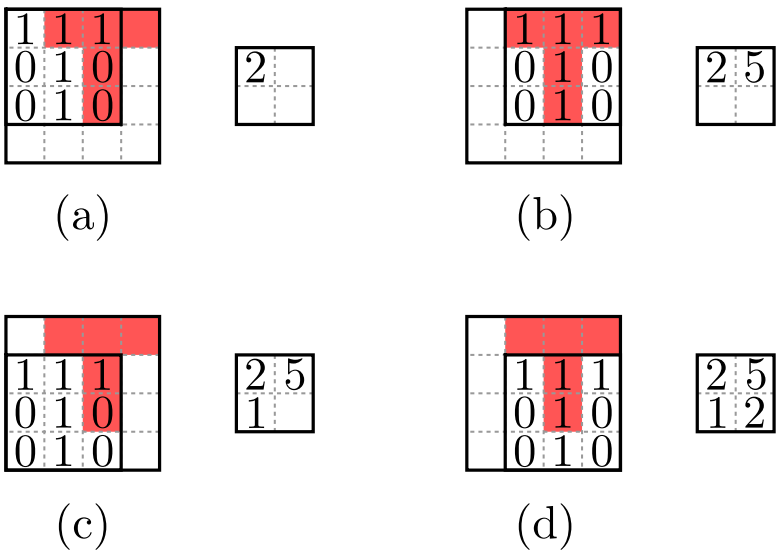


Figure 13: Example of a convolution. The 3×3 kernel is shifted over the image one step at the time. The red color in the image represents a pixel value of 1. For example in picture (a) the point wise product between kernel and image is zero everywhere except at two positions where a one in the kernel meets a one in the image. Because there are less valid positions for the kernel than pixels in the image the result is smaller in size.

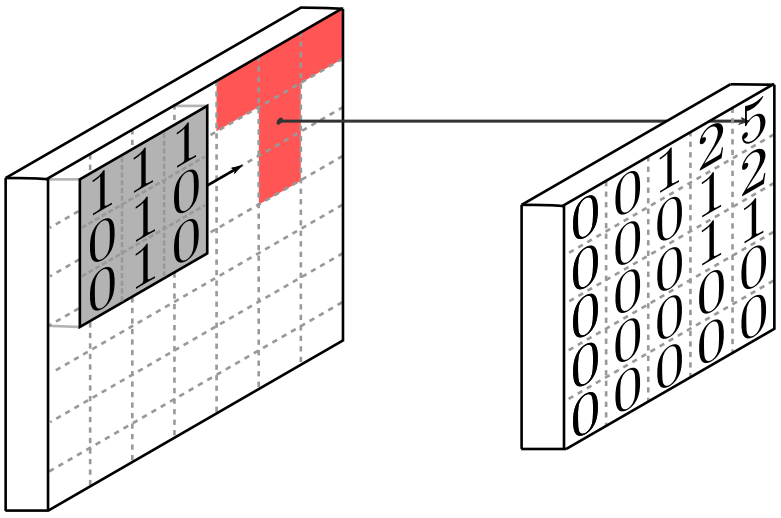


Figure 14: Example of a convolution where white pixel are 0 and red pixel are 1. The 3×3 kernel is shifted over the image and the point wise product between kernel and image is calculated at every step as described in figure 13. The result is the greatest when the kernel is directly over the letter T.

One convolutional layer contains not only one but a number of different kernels k. The resulting k feature maps are stacked in the "z direction" so that the shape of the $n \times n$ matrix transforms to $(n-2) \times (n-2) \times k$ when convolved with a 3×3 kernel. In a convolutional network, short ConvNet, multiple of these layers are used so that it can find "patterns in patterns". For the letter detection example, one could imagine the first layer to detect various edges and the next layer to detect letters in the position of these edges.

Pooling Layers

For a big image and a large number of kernels the output shape of a convolutional layers is still $\mathcal{O}\big((n)^2 \cdot k\big)$, so quite large. Also, notice how in figure 14 the "T kernel's" feature map is not only active at the exact position of the T but in the general region. The solution to this is to downsample the output with a *Pooling Layer*. Here a smaller kernel, usually 2×2 is shifted over the matrix two steps at a time and at every position an operation is performed to reduce the number of values to one. This could be taking the maximum or the average of that 2×2 region. This operation reduces the matrix in the x and y dimension by a factor of 2 as shown in figure 15.

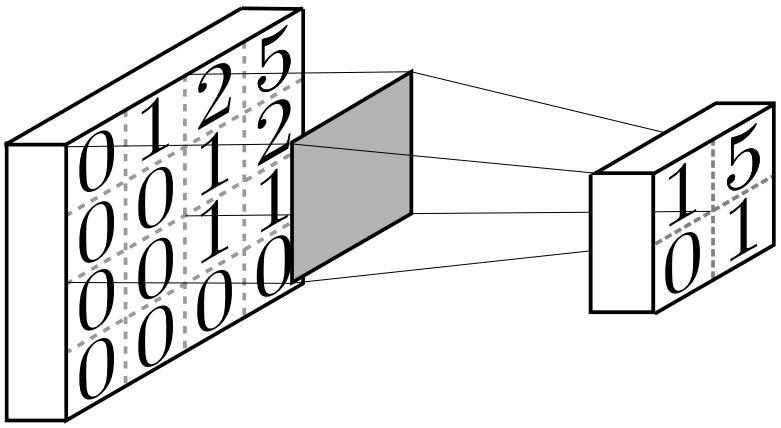


Figure 15: Example of a Max Pooling Layer. For every 2 by 2 field the maximum is calculated. After applying first the convolution and then the pooling layer the information "T in the top right corner" is still there and size of the resulting matrix is very manageable.

Example Network Architecture

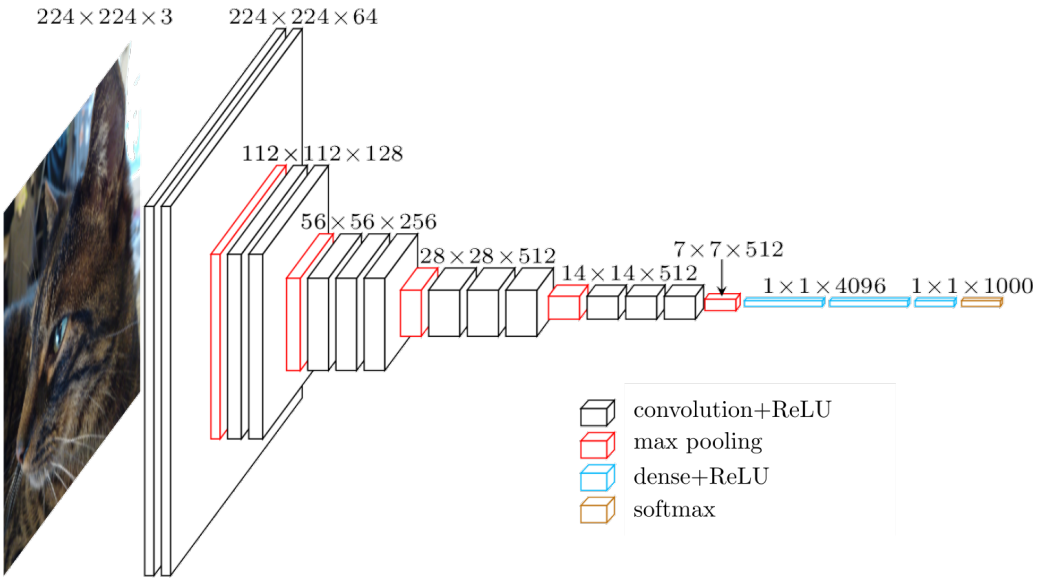


Figure 16: Example for a complete ConvNet. Note that the input is an RGB image so there are three layers in the z dimension corresponding to the different colors. In this case the first layer has 64 kernels of size $3 \times 3 \times 3$. The next convolution then has 128 kernels of size $3 \times 3 \times 64$. The final softmax activation rescales the output so that its sum is 1. [13]

Now all the building blocks for a complete ConvNet are available. Repeatedly alternating convolution and pooling layers changes the input from wide in x and y dimension and narrow in z to a long z-strip. At the very end this strip is fed into one densely connected layer which is in turn connected to the output neurons. An example architecture of this kind is shown in figure 16. This network could for example choose one of 1000 labels for the input image like car, human, cat...

1D ConvNets

The input to the desired algorithm is a target spectrum to which the Network should output some parameters (more on this in the section 4.2). This input data is a function $I(\lambda)$, so only one dimensional, in contrast to an image which is a function $f(x, y)$ but all the same ideas apply. Convolutional kernels are sized $1 \times 3 \times z$ and pooling kernels are $1 \times 2 \times z$. Both are only shifted in one direction as see in figure 17. These 1D convolutions might detect features like rising and falling edges and the later layers might combine these features into concepts like peaks and troughs. However, as always in machine learning what the network actually does to reach its objective is not controlled by the programmer.

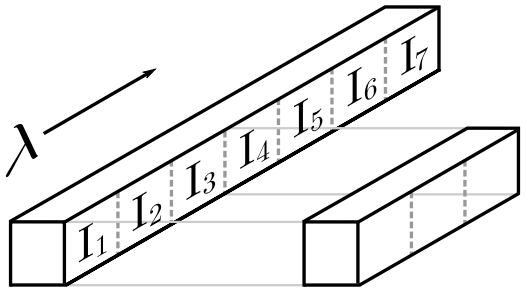
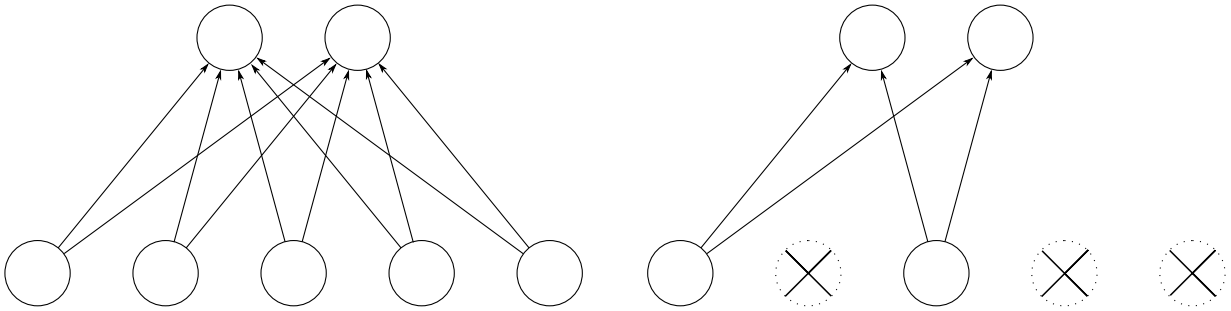


Figure 17: Example of a 1D convolution. A 1×3 kernel is shifted over a spectrum I discretized at 7 wavelengths

Dropout Layer

In 2014 Srivastava et al.[14] presented a method to prevent overfitting and speed up the training process of large Neural Networks. During training, for each step, they randomly drop a number of neurons in a layer along with all connections to and from these neurons. This prevents the neurons from co-adapting [14] and because there are less weights and biases to tune for each step the training becomes overall faster. These layers are characterized by their *drop rate* that determines what fraction of neurons should be dropped. The process of dropping neurons is shown in figure 18.



(a) no dropout (b) with dropout

Figure 18: Example of a dropout applied to the bottom layer. Only two of the neurons remain active and only their weights and biases are modified during this training step. [14]

4. Algorithm

Before starting to implement the Algorithm, we need to define which metasurface stacks the it is allowed to produce. This was already partly motivated in the introduction: We want easy to manufacture metasurfaces which can produce a high variety of transmission spectra when stacked. Physically, we consider stacks which are homogeneously illuminated by plane waves and produce transmission spectra $I = I(\lambda) \in [0, 1]$. We are going to use the homogeneous plasmonic metasurfaces discussed in section 3.3. These surfaces can be manufactured via electron beam lithography.

We want to enable the algorithm to produce custom polarizers. This means we need to be able to control the x and y polarizations separately. As shown in section 3.2 paragraph Symmetries, we cannot use meta-atoms of C_4 symmetry for this task. The next simplest geometry is the rectangle. Another limitation given by the manufacturing process is the number of metasurface layers in a stack. Stacks with more than two layers become increasingly hard to manufacture this is why we are going to limit the algorithm to two-layer stacks even though theoretically arbitrary stacks could be used.

To reach the greatest variety of transmission spectra, we will have to expand the possible geometries slightly. As described in section 3.3, rectangular metaatoms tend to produce a minima in the transmission at a wavelength where incident photons can couple efficiently into SPP's. However, many transmission spectra contain both minima and maxima and at this point we are not able to produce the maxima. For this reason, we will expand the possible geometries to rectangular holes as seen in the bottom layer of figure 19. This type of metaatom has a low transmission for most wavelengths and only the photons which would have fit the resonance condition of the hole geometry can transmit freely. In a sense, the rectangular hole acts as an inverse to the rectangle in the type of transmission spectra they produce.

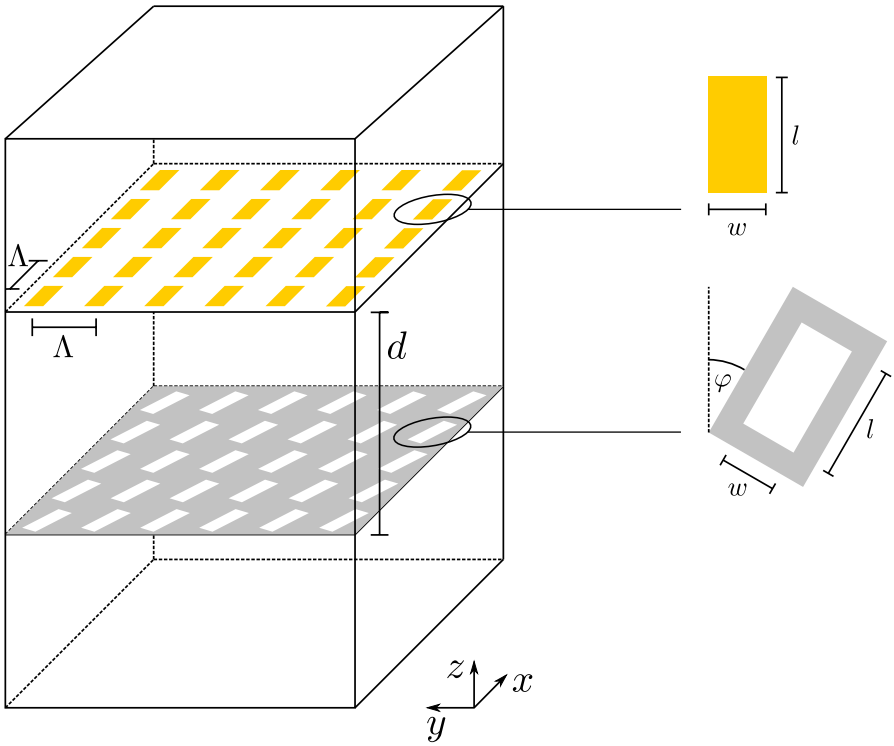


Figure 19: Schematic of a two layer metasurface stack to visualize all the design parameters $\mathcal{D}$ which can be set by the algorithm. These include stack parameters $\mathcal{S}$ and two sets of layer parameters $\mathcal{L}_1$ and $\mathcal{L}_2$. So $\mathcal{D} = (\mathcal{S}, \mathcal{L}_1, \mathcal{L}_2)$. The stack parameters consist of the distance between the metasurfaces d and their rotation angle φ so $\mathcal{S} = (d, \varphi)$. The layer parameters are the width w, length l and thickness t of the meta-atom, the period of meta-atoms Λ the material of the layer m and the kind of geometry g. So $\mathcal{L} = (w, l, t, \Lambda, m, g)$. All of the parameters are continuous except for the choice about the material and geometry.

As for the material, we are going to limit the algorithm to Aluminum and Gold. Here we have to remember that every material we add multiplies the amount of metasurfaces which have to be simulated in preparation which is often referred to as the *curse of dimensionality*. This is because more materials are only useful when the corresponding parameter space is sufficiently dense. This ends the design considerations, a visualization of all parameters and the notation we are going to use can be found in figure 19.

4.1. Implementation

Having done the necessary preliminary considerations we can now start implementing the algorithm. It is organized into separate software modules defined by their inputs and outputs. The composition of these modules to the complete algorithm is shown in figure 20 and the individual modules will be discussed in the following sections. Each section will start with the input and output to that module. The sections are ordered as their corresponding modules appear in the algorithm, with the exception of the SASA module, which was already discussed in the background section 3.2.

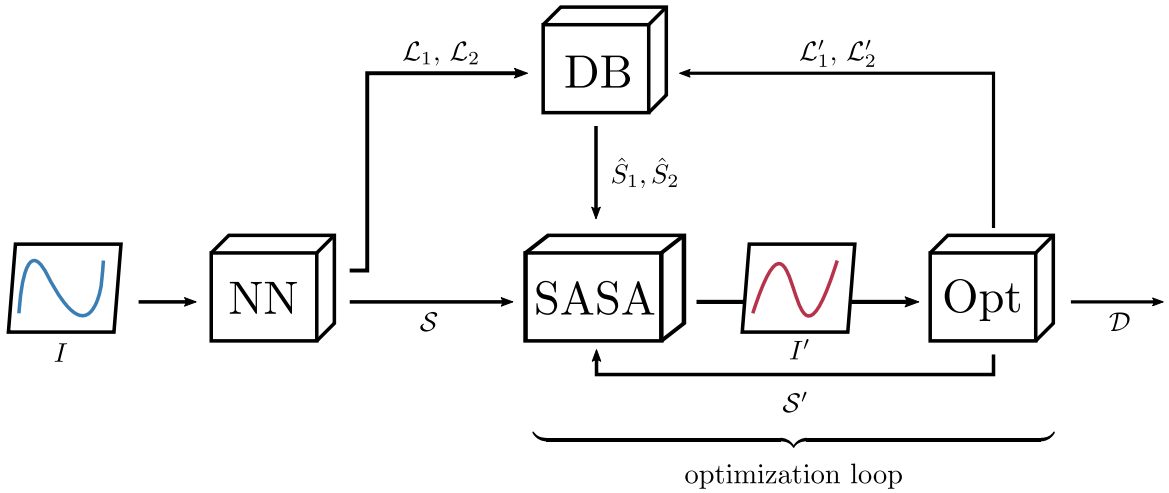


Figure 20: The algorithm tries to find to a given transmission spectrum I the design parameters $\mathcal{D}$ of a two layer meta surface stack which can reproduce this target. The input spectrum is passed to a convolutional neural network which outputs its guess for the stack and layer parameters. The Database module looks at the two sets of layer parameters and interpolates between stored S-matrices to give an estimate for the two S-matrices describing the layers. The SASA module calculates the resulting current transmission spectrum and passes it to the optimizer. This last module compares it to the target spectrum and adjusts the continuous parameters to minimize the difference between the two spectra.

NN:	convolutional neural network trained to map spectra to stack and layer parameters
DB:	database of pre-simulated single layers
SASA:	module calculating $\hat{S}_{\text{stack}} = \hat{S}_{\text{stack}}(\hat{S}_1, \hat{S}_2, \varphi, h)$
Opt:	optimizer changing the continuous parameters to minimize the difference between the current and target spectrum
$\hat{S}_1, \hat{S}_2$	S-matrices of the top and bottom layer
$\mathcal{L}_1, \mathcal{L}_2$	two sets of layer parameters where $\mathcal{L} = (w, l, t, \Lambda, m, g)$ w...width, l...length, t...thickness, Λ...Period, m...material, g...geometry
$\mathcal{S}$	stack parameters $\mathcal{S} = (d, \varphi)$ where d...distance between layers, φ...rotation angle
optimization loop	this loop is repeated until the target accuracy is reached
I	input transmission spectrum
$\mathcal{D}$	output design parameters

4.2. Network

Input:	transmission spectrum $I = (I_\mathrm{x}, I_\mathrm{y})$ a $\lambda \times 2$ array
	$I_{\mathrm{x/y}}$ ... X- and Y-transmission spectra, λ ... number of wavelengths
Output:	two sets layer parameters $\mathcal{L}_1$ and $\mathcal{L}_2$, stack parameters $\mathcal{S}$

Network Architecture This module is a 1D Convolutional Neural Network instead of the basic Multi Layer Perceptron. It was chosen to utilize the translational invariance of ConvNets. For example the concept "peak" should be learned independent of its position in the spectrum. As described in section 3.4, a ConvNet provides this functionality. Another constraint on the network architecture arises from the different kind of outputs. Most of the outputs are continuous but the choices about material m and geometry g are discrete/categorical. These need different activation functions σ to reach the different value ranges. The continuous outputs are mostly bounded by physical constraints and m, $g \in [0, 1]$ as they are *one hot encoded*, meaning $1 \rightarrow$ "The layer has this property" and $0 \rightarrow$ "The layer does not have this property".

The different outputs also need different cost functions $C(y, y')$ during training where y' is the networks output and y is the known solution. For the continuous output one can simply use the mean squared error

$$C_\mathrm{mse}(\mathbf{y}, \mathbf{y}') = \sum_i \left(y_i - y_i'\right)^2 \tag{4.1}$$

as all outputs are equally important and the cost function should be indifferent on whether the networks prediction is over or under target. For the categorical output the network learns quicker with the *Categorical Cross-Entropy* error

$$C_\mathrm{ce}(\mathbf{y}, \mathbf{y}') = -\sum_i y_i \log y_i'. \tag{4.2}$$

This error treats false positives ($y_i = 0$, $y_i' = 1$) and false negatives ($y_i = 1$, $y_i' = 0$) differently. A false positive does not increase the overall cost as $y_i = 0 \Rightarrow C_\mathrm{ce} = 0$ but for a false negative $C_\mathrm{ce} \rightarrow \infty$. This is favorable behavior because it does not matter if the network outputs some probability for a wrong class as long as it outputs a higher probability for the correct class. The final architecture, seen in figure 21, is similar to the example given in figure 16 while meeting the above-mentioned constraints.

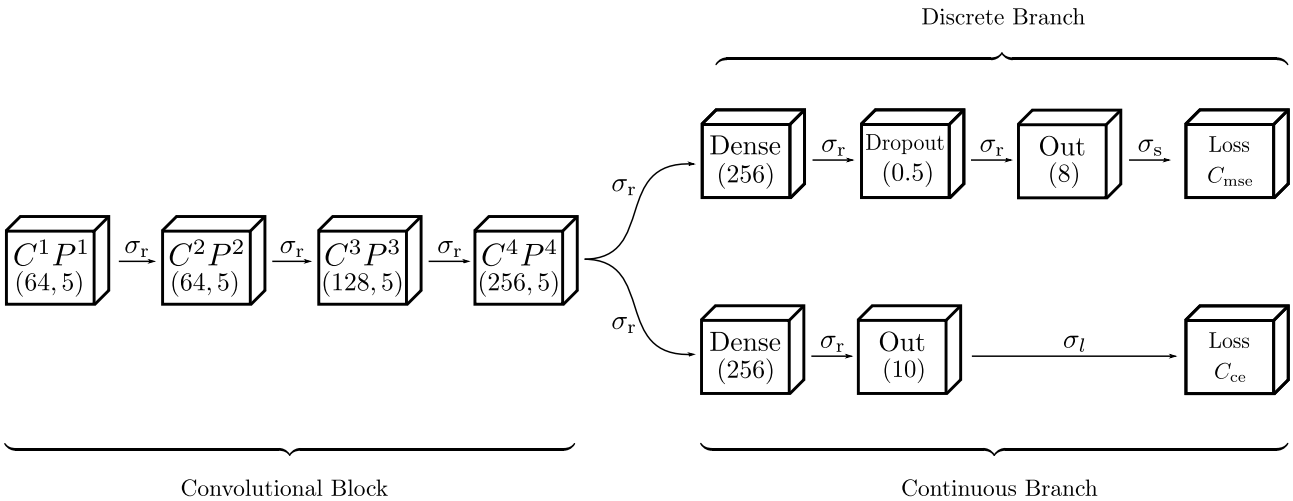


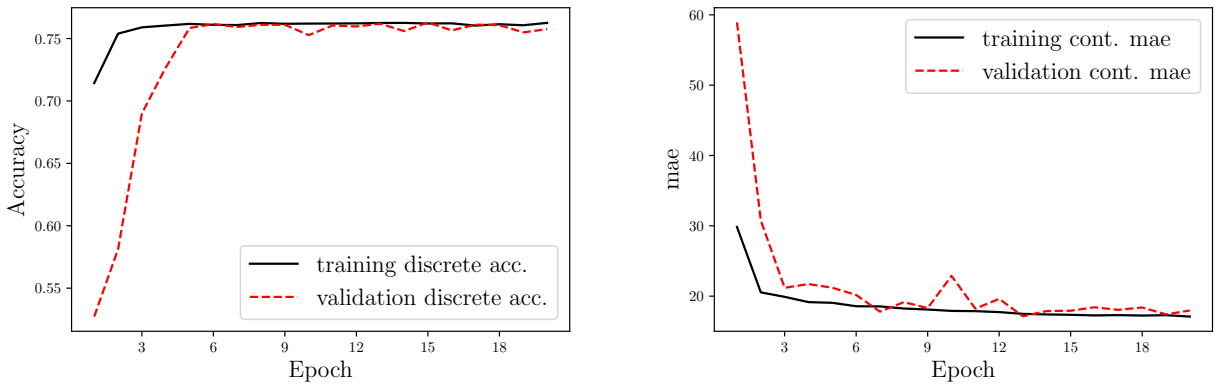
Figure 21: The network starts with 4 pairs of convolutional and pooling layers C^iP^i. The convolutions are characterized by (*number of kernels*, *kernel size*). The kernel size is always 5 and the number of kernels is gradually increased. Then the Network splits into a discrete and a continuous branch via two Dense layers with (*number of neurons*). In the discrete branch a dropout is applied to the dense layer where (0.5) is (*fraction of neurons to drop*). All the internal activations σ_r are ReLu's and the final activations σ_s and σ_l are a sigmoid and a linear function.

Network Training

To train a Neural Network, one needs a training set (X, Y) of known input output pairs. In this case they are generated using the pre-simulated single layers in the database which are randomly combined into stacks. Then the stacks X- and Y-transmission spectra (I_x, I_y) are calculated via SASA. This means $I = (I_x, I_y) \in X$ are the networks input and the random design parameters $\mathcal{D} = (\mathcal{S}, \mathcal{L}_1, \mathcal{L}_2) \in Y$ are the output. For the first test we used squares and square holes of Aluminum and Gold. During the training one epoch is defined as one loop over all training samples. After every epoch the network is validated on a data set $(X_\mathrm{v}, Y_\mathrm{v})$ of samples it has not seen before. This is done to check wether the network actually learns something rather than just memorizing the input data. It is common to rely on the cost function for training but use a separate function called *metric* to evaluate the Network. This is done because there are occasions when the cost of a network drops but the performance does not increase. The metric should be a function where the output can be easily understood by the human evaluating the network. For discrete outputs we are going to use the metric *accuracy*, that is what fraction of choices were correct and for the continuous outputs we will use the *mean absolute error*

$$M_\mathrm{mae}(\mathbf{y}, \mathbf{y}') = \sum_i |y_i - y_i'|. \tag{4.3}$$

The results of this first training on the square geometry can be seen in figure 22. Training and validation results are very similar which indicates that there is no overfitting or memorization. The discrete accuracy quickly reaches a maximum of $\sim 76\%$ which is less than expected and also the speed at which this value is reached is suspicious. The network does not improve much after the fourth epoch and this does not change by tuning the network architecture. This is because the issue lies not within the architecture but in the training data. In section 3.2 we have shown that for the used two layer stacks the transmission spectrum is reciprocal, that is the same for both directions. Therefor the data generation can result in two different stacks which produce the same spectrum. Consider a stack where one layer is Aluminum and the other is Gold as seen in figure 23. As both of them produce the same spectrum, one time the network is taught that the first layer is Gold and the second is Aluminum but another time it is taught the complete opposite. Actually, if the network is trained this way it only ever predicts stacks with layers of equal materials because this is the only setup it can get right.



(a) (b)

Figure 22: Training only with squares. (a) shows the accuracy of the discrete branch. That is, what percentage of choices for material and geometry were correct. In (b) we can see the performance of the continuous branch evaluated using the mean absolute error 4.3.

This is a well known problem when trying to solve an inverse problem. In this case the function $\mathcal{D} \to I$ is well defined as one stack can only produce one spectrum but for the inverse problem we are trying to solve $I \to \mathcal{D}$ and there might be multiple designs which produce the same spectrum. We can solve the issue simply by allowing only one of the orientations into the training data. By doing this the training results change as seen in figure 24. The discrete accuracy is much better at $\sim 98\%$ and the training curve looks more natural in the sense that improvements diminish over time but do not hit a sudden barrier as they did in figure 22. Having resolved this issue, we can train the model on the more complex rectangle geometry reaching 94% accuracy and 11 mae.

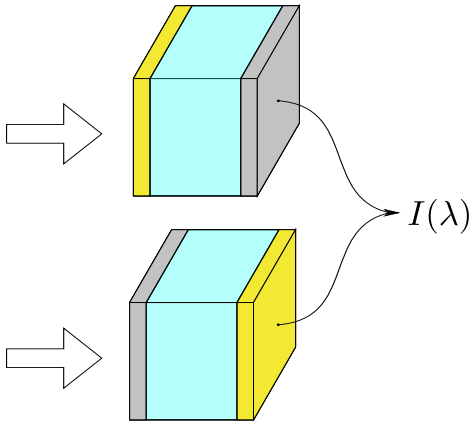
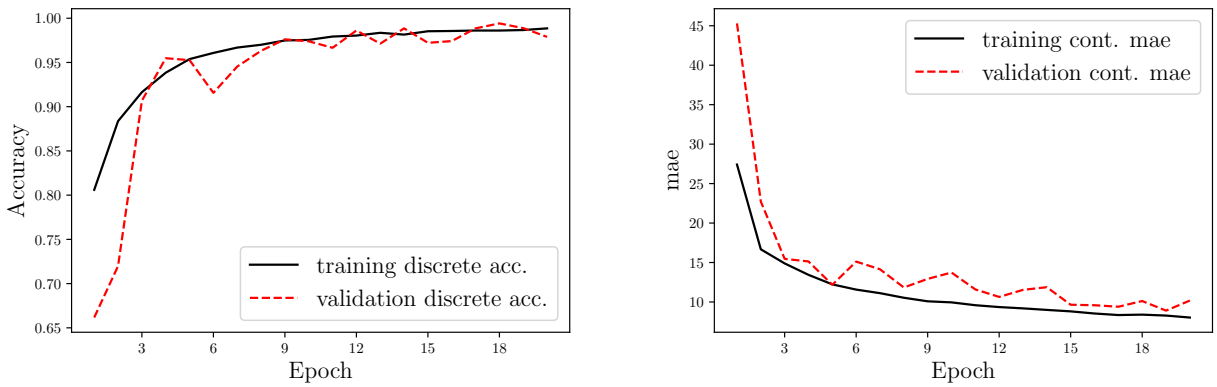


Figure 23: A stack of one Gold and one Aluminium layer separated by a glass spacer. Both stacks produce the same spectrum which leads to issues when using completely random stacks to train the network.



(a) (b)

Figure 24: Training on the square geometry after only allowing one of the equivalent stacks shown in figure 23 into the training data. Again, (a) shows the accuracy of the discrete branch and (b) the average cost of the continuous branch. The discrete accuracy has improved significantly compared to figure 22 but the continuous loss remains similar.

4.3. Database

Input:	Layer parameters $\mathcal{L} = (w, l, t, \Lambda, m, g)$
Output:	Interpolation of the S-matrix for this layer $\hat{S} = \hat{S}(\mathcal{L})$

Fourier Modal Method

The database consist of $\sim$ 5000 S-matrices of single layers which were simulated with the Fourier Modal Method (FMM) on a compute cluster. This method is applicable to all surface structures which are periodic in x and y direction and constant in z direction. It works by expanding the involved fields into their diffraction orders via a Fourier expansion. For example, for the reflected electric field is

$$\mathbf{E}_\mathrm{r} = \sum_{m,n} \mathbf{R}_{mn} e^{i\mathbf{k}_{mn}\mathbf{r}}. \tag{4.4}$$

Applying Maxwell's equations and the continuity conditions described in section 3.1 results in an eigenvalue problem which can be reformulated into a linear set of equations. The unknown properties like R_{mn} are found by solving this set of equations. Computationally, a Fourier series like (4.4) has to be truncated at some order. This order determines the accuracy of the FMM and the matrix which represents the set of linear equations is sized order $\times$ order. Because of this the computational complexity of this method increases rapidly with the order. The method was first introduced by Noponen and Turunen [15].

Interpolation

To find the S-matrix to layer parameters $\mathcal{L}$, that are not already stored in the database, this module has to interpolate between the pre simulated S-matrices. First, it looks for the n closest neighbors of $\mathcal{L}$. To do this, the continuous input is normalized through

$$\bar{\mathcal{L}}_i = \frac{\mathcal{L}_i - \mathcal{L}_i^\mathrm{min}}{\mathcal{L}_i^\mathrm{max} - \mathcal{L}_i^\mathrm{min}}, \; i \in 1...4 \quad \text{so that} \quad \bar{\mathcal{L}}_i \in [0, 1]. \tag{4.5}$$

Now the distance d to every entry in the database satisfying the material geometry combination is calculated and the n entries with the smallest distance are selected where

$$d(\mathcal{L}^1, \mathcal{L}^2) := \sum_{i=1}^{4} \left| \mathcal{L}_i^1 - \mathcal{L}_i^2 \right| \tag{4.6}$$

The interpolation $\hat{S}(\mathcal{L})$ is calculated via Inverse Distance Weighting [16] so that more distant entries have a smaller effect on the result. Let $(\mathcal{L}^1, \ldots, \mathcal{L}^n)$ be the n closest neighbors to $\mathcal{L}$ with stored S-matrices $(\hat{S}_1, \ldots, \hat{S}_n)$ and $d_j = d(\mathcal{L}, \mathcal{L}^j)$ then

$$\hat{S}(\mathcal{L}) = \sum_{j=1}^{n} w_j \hat{S}_j, \quad \text{where} \quad w_j = \frac{1/d_j^2}{\sum_i 1/d_i^2} \qquad \text{so that} \quad \sum_j w_j = 1. \tag{4.7}$$

The exact parameters of the simulated S-matrices can be found at https://github.com/TimLucaTuran/sasa_db/blob/master/data/NN_smats.xlsx.

4.4. Optimizer

Input:	Current spectrum I', Current design parameters $\mathcal{D}$
Output:	Improved design parameters $\mathcal{D}'$

The optimizer is at the core a Downhill-Simplex [17] tuning the continuous design parameters to minimize the mean-squared-difference between the current spectrum I' and target spectrum I we defined as $C_{\text{mse}}(I, I')$. However, the standard approach is unable to follow the physical constraints discussed in section 3.2 and has to be modified in that regard. To achieve this, we can introduce a distance to the boundary D. Let p be a single continuous parameter with lower bound p^l and upper bound p^u, then

$$D\left(p, p^l, p^u\right) = \begin{cases} p^l - p, & \text{for } p < p^l \\ 0, & \text{for } p^l \leq p \leq p^u \\ p - p^u, & \text{for } p^u < p \end{cases}. \tag{4.8}$$

In this way one can penalize the simplex for stepping over the set boundaries by using a total loss L which depends on the sum of all distances D_i by

$$L(I_c, I_t, p) = \underbrace{C_{\text{mse}}(I_c, I_t)}_{\text{find target}} + \underbrace{\left[1 + \sum_i D\left(p_i, p_i^l, p_i^u\right)\right]^3}_{\text{stay within bounds}}. \tag{4.9}$$

The choice of exponent, 3 in this case, depends on how much the simplex should be penalized for stepping over a boundary and the $+1$ was added so that even $\mathcal{D} < 1$ are increased by the exponentiation. All our conditions are requirements for physical approximations and these approximations do no break completely when a boundary is only slightly violated. This justifies this approach where the simplex might step over a boundary but is then "pushed back" in the right direction. In figure 25 we can see the optimizer in operation.

In the very left graph of figure 25 at step 0 the optimizer has not jet tuned any parameters and the current and target spectrum are quite different. The blue target spectrum is in this case produced by a random stack from the validation data. This means there is a set of parameters which could reproduce this spectrum perfectly. By step 250 the optimizer has found a design which comes close to the original.

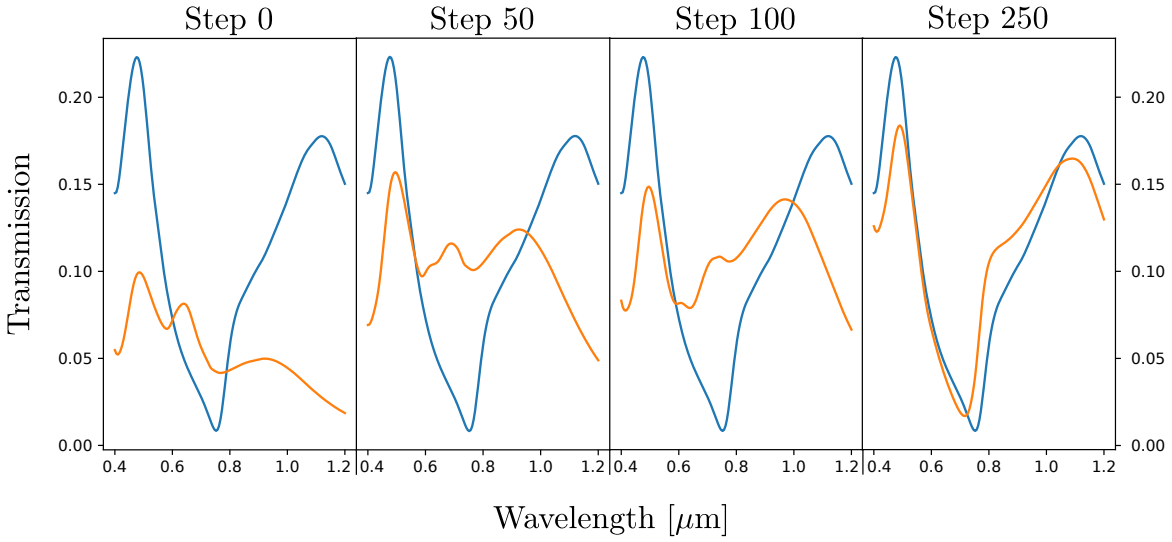


Figure 25: The fraction of transmitted light polarized in the x direction over the wavelength in μm at optimization steps $(0, 50, 100, 250)$ left to right. Blue is the target spectrum I and orange is the current spectrum I'.

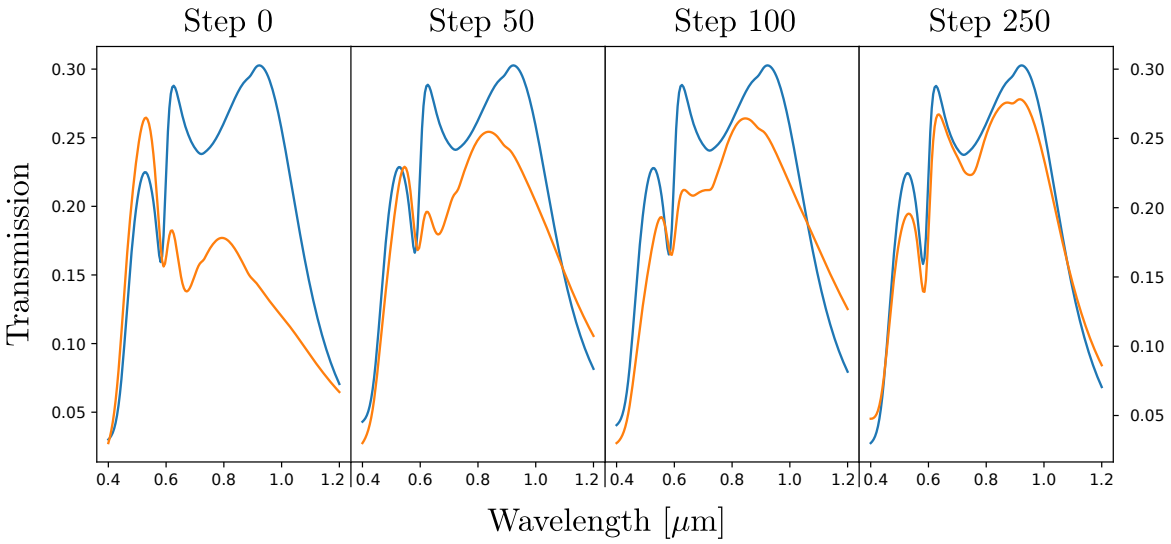


Figure 26: The fraction of transmitted light polarized in the y direction over the wavelength in μm. The optimization of the y polarization is done simultaneous to the x polarization shown in figure 25.

To gain more insight, we can consider how the parameters changed during the optimization as seen in figure 27. As designed, the optimizer did not change the material and geometry parameters and they were chosen correctly by the Neural Network. This is in accordance with the high discrete accuracy seen in section 4.2. Some of the continuous parameters are close to the original but others like the rotation angle φ, second from the bottom, are completely different. This shows that even though we have removed the equivalent stacks described in figure 23 there are still pairs of designs $\mathcal{D}_1, \mathcal{D}_2$ with different continuous parameters which produce very similar spectra $I(\mathcal{D}_1) \simeq I(\mathcal{D}_2)$ and these make it impossible for a network of this architecture to learn about the continuous parameters with good accuracy.

Step 0	Step 250	True Parameters
Layer 1: material: Au holes: holes width: 170 length: 105 thickness: 24 periode: 235 Layer 2: material: Al holes: holes width: 156 length: 90 thickness: 31 periode: 178 Stack spacer_h: 0.18 angle: 36 loss: 2.61	Layer 1: material: Au holes: holes width: 186 length: 94 thickness: 21 periode: 221 Layer 2: material: Al holes: holes width: 158 length: 90 thickness: 29 periode: 204 Stack spacer_h: 0.23 angle: 14 loss: 0.15	Layer 1: material: Au holes: holes width: 180 length: 80 thickness: 20 periode: 250 Layer 2: material: Al holes: holes width: 150 length: 90 thickness: 20 periode: 200 Stack spacer_h: 0.23 angle: 0 loss: 0.00

Figure 27: The design parameters $\mathcal{D}$ to the optimization shown in figure 25 and 26 at the beginning and end compared to the true parameters which produced the target spectrum.

4.5. Evaluation

The resulting algorithm is generally able to reproduce spectra from stacks generated based on the database. This is a good first test because we know these spectra are possible and if the algorithm could not reproduce them something would be wrong. However, the goal was not to reproduce known spectra but to be able to find design parameters for new ones. Here the evaluation has to be more nuanced. Yes, some new transmission spectra, such as Gauß peaks, dips and sinus functions, are possible and can be seen in Appendix B. But, the first limitation becomes apparent and is illustrated in figure 28. The algorithm can successfully reproduce the sine with a maximum transmission of 0.7 but not the sine with a maximum transmission of 1. In general, for all fits, the maximum transmission has to stay below ≈ 0.7 to be approximated successfully. This can be understood by the ohmic losses in plasmonic metasurfaces. Another limitation can be seen in figure 34a. Even when the target function is approximated successfully there might be additional unwanted features in the spectrum.

Interestingly, some targets that would be trivial for a human are very challenging for the algorithm. If a human was tasked to create the maximum possible transmission at all wavelengths, he or she would probably use zero thickness metasurfaces or metasurfaces which only consist of holes. When the algorithm is given a transmission spectrum that is one everywhere it gets confused to a point of predicting negative parameters (figure 35). They are chosen despite being completely out of bounds and thus very negatively affecting the simplexes loss function through equation (4.9). However, even this behavior can be explained because there are no zero thickness or hole only metasurfaces in the database we cannot expect the simplex to know about them. Additionally, the less reachable a task is, the higher is the total loss and the less important is the boundary term explaining the nonsensical negative parameters.

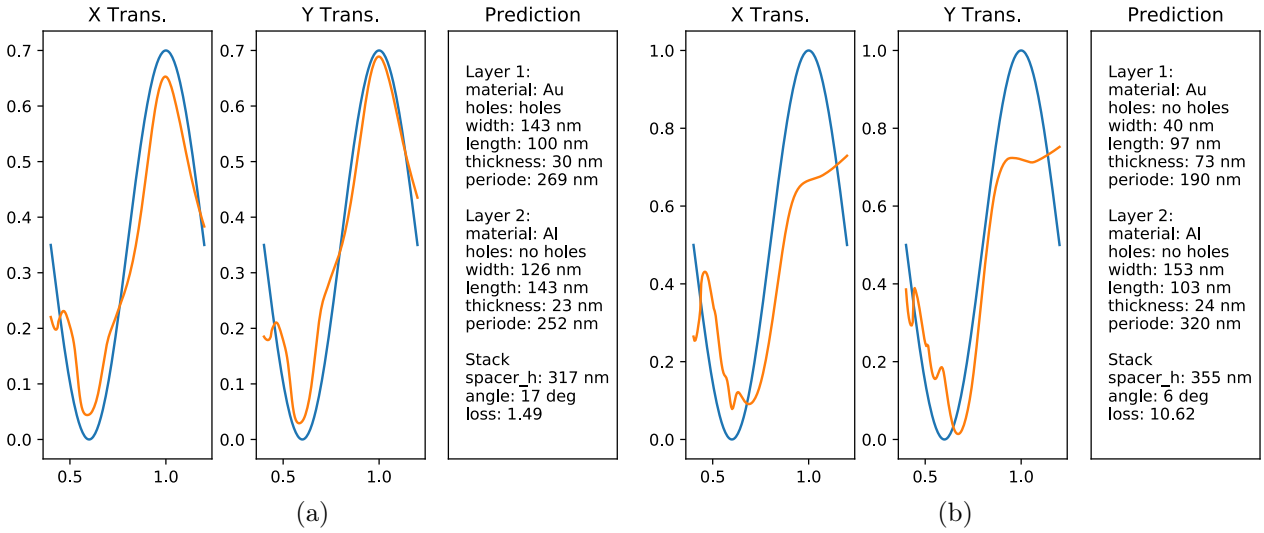


Figure 28: Two fits of sine functions. Transmission in dependence of wave length. 0.4 to 1.2 μm. (a), the sine with a maximum transmission of 0.7, could be reproduced successfully but (b), the sine with a maximum transmission of 1, could not.

5. Conclusion

To summarize, we started off by discussing the physics behind the chosen optical system of stacked plasmonic metasurfaces. We learned how Maxwell's equations govern the behavior of light at the interfaces between layers and how to use the S-matrix calculus and the semi analytic stacking algorithm to describe stacks as a whole. This calculus came with a number of boundary conditions the algorithm needed to obey, such as the minium distance between layers and the maximum period of metaatoms. Then, we turned to neural networks and discussed how they are build and trained. We settled on Convolutional Neural Networks as they would be able to efficiently use the spatial information in a transmission spectrum.

The first step in implementing the algorithm was training a network on spectra generated by SASA. Here we encountered the many-to-one problem for the first time, where multiple designs mapped to a single spectrum. For instance, certain metasurface stacks would produce the same spectrum independent of the layer order. This resulted in multiple designs mapping to a single spectrum and impacted network training negatively. After removing these equivalent stacks the network could be trained successfully. It was very accurate in finding the discrete design parameters to a spectrum but the predictions for the continuous parameters were only good enough as an initial guess. This is why the another optimization step was added. A conventional simplex further tunes the continuous parameters and at every step the performance of the new stack is evaluated with the semi analytic stacking algorithm.

In the second part of these conclusions, some possibilities are shown how the algorithm could be improved. In section 4 we have seen that it is very difficult or even impossible to solve the inverse problem (spectrum $I \rightarrow$ design $\mathcal{D}$) directly with a neural network because of multiple designs mapping to a single spectrum. For the algorithm, we solved this by adding a conventional optimization method after the network to tune the parameters the network was not able to set correctly. This had some success as described in 4.5 but this approach is also a concession because we loose many of the advantages neural networks bring. Conventional optimization methods are more computationally expensive than a single forward pass in a neural network. More importantly, a simplex combined with interpolation is not able to generalize beyond the database. Having one network being able to solve $I \rightarrow \mathcal{D}$ directly would be the best solution.

As this is a well known problem numerous solutions have been proposed. A very interesting route was taken by Liu et al. [18]. They approached the inverse problem by first solving the forward problem in our case $\mathcal{D} \rightarrow I$ and then building a combined network in a kind of tandem structure $I \rightarrow \mathcal{D} \rightarrow I'$ where the cost function is $C = C(I, I')$. This means during training the network is given a target spectrum I and predicts the corresponding design parameters $\mathcal{D}$. This design is then fed into the forward model and produces a spectrum I' and the cost function depends on the difference of I and I'. This approach completely circumvents the issue of multiple designs mapping to a single spectrum because it only depends on the differences between the target and the produced spectrum. This approach was tried but could ultimately not be implemented successfully due to overfitting issues in the combined model as described in appendix A. However, it should be possible with the right combination of network architecture, regularization and hardware.

The larger a project becomes, the more *hyperparmeters* have to be set. Hyperparmeters are a term from machine learning and traditionally include parameters such as the learning rate or the drop rate in Dropout layers. We could also include the network architecture, that is how many layers or how many neurons per layer the network should have. One could even include the question, of how many different materials to use and how densely to simulate the parameters space to each material. At some point it is not enough to set these parameters based on trial and error and intuition. One has to start to use more thorough methods. There are many options including *Grid Search* or *Random Search* [19] that could be tried.

Lastly, the initial numerical simulation of metasurfaces were done on a compute cluster but all other computations were done on the CPU of a reasonably modern laptop. That includes the training of neural networks and the simplex optimization. However, due to the algebraic nature of the calculations when training a neural network, these are best done on at least one GPU. Better Hardware could improve performance by enabling the usage of bigger models and more training data.

A. Tandem Model

It would be beneficial to train the inverse model $\mathcal{D} \to I$ in a tandem structure $I \to \mathcal{D} \to I'$ with a pre trained forward model. This would circumvent the many-to-one problem and the network could even be trained on arbitrary functions as knowledge of the design parameters to a training spectrum are not necessary. This combined model can easily be implemented as all the necessary modules are already there. A possible implementation is shown in figure 29. However, this model can not be trained. To understand why we need to go back to the concept of Backpropagation. During training the gradient of the cost function needs to propagate backwards through the network. To capture this mathematically we will identify the forward model in figure 29 as the last layer L in a combined network. Equation (3.61) tells us we need to calculate

$$\delta_j^L = \frac{\partial C}{\partial a_j^L} \frac{\partial a_j^L}{\partial z_j^L} . \tag{1.1}$$

The first part $\partial C/\partial a_j^L$ is simple and only depends on the cost function. For example using $C_{\text{mse}} = \sum_j \left(a_j^L - y_j\right)^2$ the derivative is $\partial C/\partial a_j^L = 2\left(a_j^L - y_j\right)$. However, the second part $\partial a_j^L/\partial z_j^L$, the output of the last layer derived by the input to the last layer, is not easily accessible. Maybe not accessible at all if we consider the calls to the database and interpolation that happen during this step. The only way to train this combined tandem model is by replacing the forward model with something where we can access the gradient. This means training a second Neural Network to solve the forward problem.

The Network has to generate the right spectrum to an input Design. Once again we will use convolutional layers, as the output neurons of this Network are spatially related. Another indication to try a convolutional architecture is that the NN's used to generate photorealistic Images, like Nvidea's StyleGan [20], also rely on convolutional layers and we can think of a transmission spectrum as a 1D image.

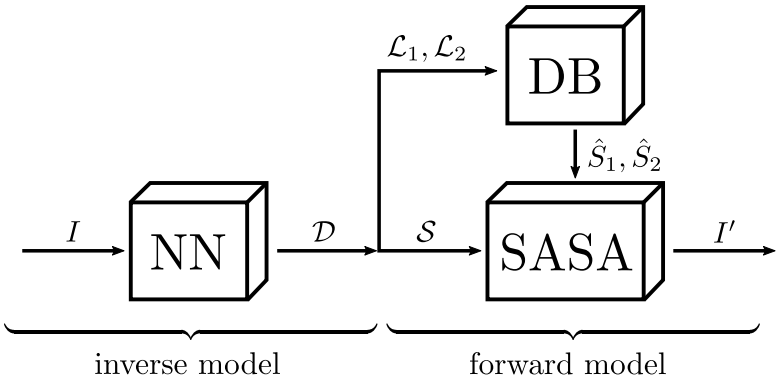


Figure 29: A part of the algorithm described in section 4 can be understood as a tandem model $I \to \mathcal{D} \to I'$. This combined model receives a target spectrum and outputs its best reproduction of that spectrum I'. The notation is the same as in figure 20.

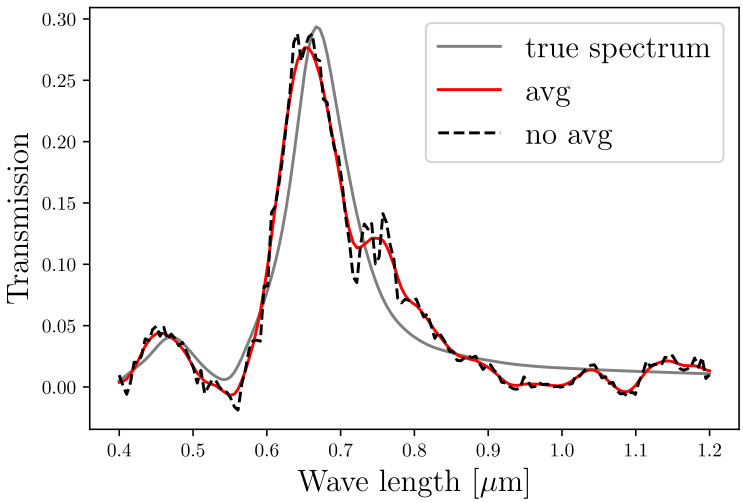


Figure 30: Example spectrum to viziulaize the effect of a running averge layer at the end of the forward model.

Network Architecture

The network architecture we used for the inverse problem contained pooling layers to gradually reduce the many parameters of a spectrum I to a few design parameters $\mathcal{D}$. Here we have to go the opposite way and upsample $\mathcal{D}$ to I. For this we will use upsampling layers which double the size of their input, e.g. $(1, 3) \to (1, 1, 3, 3)$. The convolution is then applied to this upsampled output to "fill in the details" [**?**]. This combination of upsampling and convolution is called *transposed convolution* in literature [21]. The full Network architecture is shown in figure 31 and contains two more new layers. Batch Normalization (BN) and a running average filter (Avg).

Batch Normalization was introduced by Ioffe and Szegedy in 2015 [22]. This layer calculates the mean $\mu_\mathcal{B}$ and variance $\sigma_\mathcal{B}$ over every mini-batch $\mathcal{B}$. Then it normalizes the output of the previous layer $\mathbf{x}$ to

$$\bar{x}_i = \frac{x_i - \mu_\mathcal{B}}{\sigma_\mathcal{B}} \quad \text{and rescales it to} \quad y_i = \gamma \bar{x}_i + \beta \,, \tag{1.2}$$

where γ and β are trainable parameters. This way the output of the previous layer has a mean of zero and a variance of one which has advantages for the networks stability and training speed [22]. (Batch Normalization was later also added to the inverse model.) Additionally, the running average filters were added because the output oscillated too much on small wavelength differences. This way the fact, that transmission spectra do not change too much on a per wavelength scale, is directly contained in the network architecture.

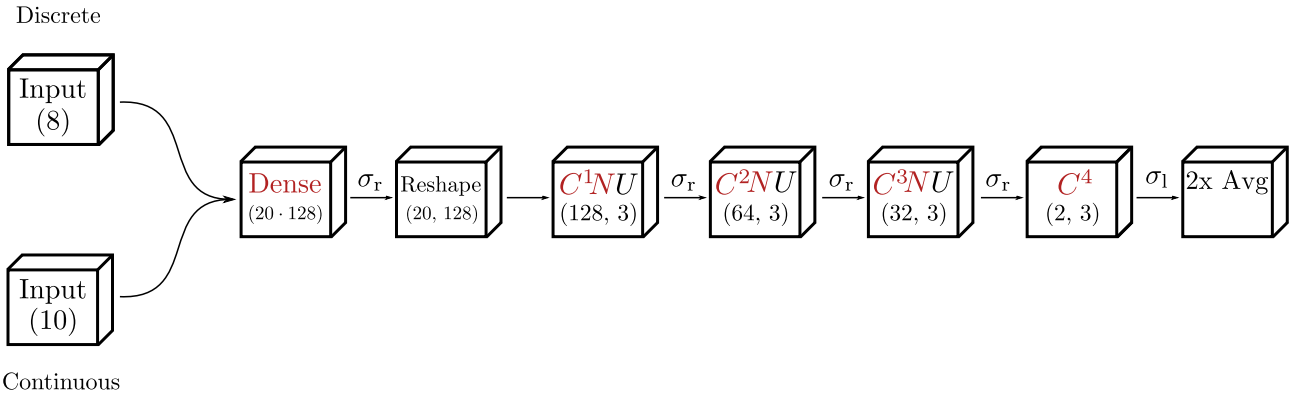


Figure 31: Network architecture of the forward model. First the discrete and continuous inputs are combined and connected to a Dense layer with $(20 \cdot 128)$ neurons. The number 20 will later become 160 discretized wavelength after the three upscale operations are applied. As discussed in section 3.4 convolutional layers operate on stacked feature maps. This is why the $20 \cdot 128$ flat neurons need to be reshaped into a 20×128 array. Next there are three transposed convolutional layers CNU. Each consisting of a basic convolution C, a Batch Normalization N and an upscale layer U. The convolution is characterized by the parameters (*number of kernels, kernel size*). The final convolution C^4 reduces the number stacked feature maps to 2. One each for the x and y polarization. Finally, two running average filters (Avg) are applied to smooth the output. All the internal activations σ_r are ReLu's and the final activations σ_l is a linear function. Layers which contain trainable parameters are marked red.

Network Training

The forward model is trained like the inverse model just the inputs and outputs are exchanged. That is, the training data (X, Y) now consists of tuples $(\mathcal{D}, I)$ instead of $(I, \mathcal{D})$. This allows us to use the same underlying data of pre simulated stacks, we already have, to also train the forward model. The mean absolute error (mae) per spectrum drops to 0.04 over 20 epochs as shown in figure 32. I consists of 160 discretized wavelengths for two polarizations. This means the transmission at every wavelength is on average $4/320 = 1/30$ percent points off. However, the forward model is not perfect. As can be seen in figure 36 it generates small troughs and peaks where there are none in the original spectrum.

Now we can combine the inverse model $\mathcal{D} \to I$ and the forward model $I \to \mathcal{D}$ into a combined model $I \to \mathcal{D} \to I'$ and directly use the loss function $C_{\text{mse}}(I, I')$. However, even trying different regularization techniques, this model could not be trained successfully as seen in figure 33.

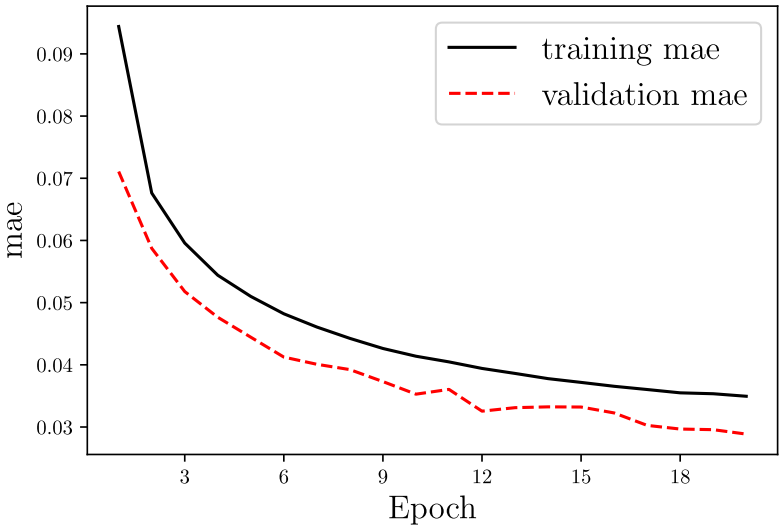


Figure 32: Training of the forward model. The Network performance is evaluated based on the mean absolute error 4.3.

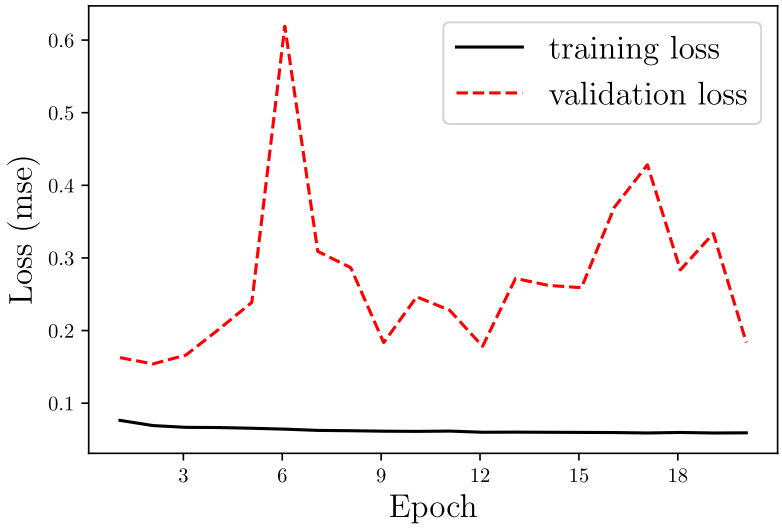
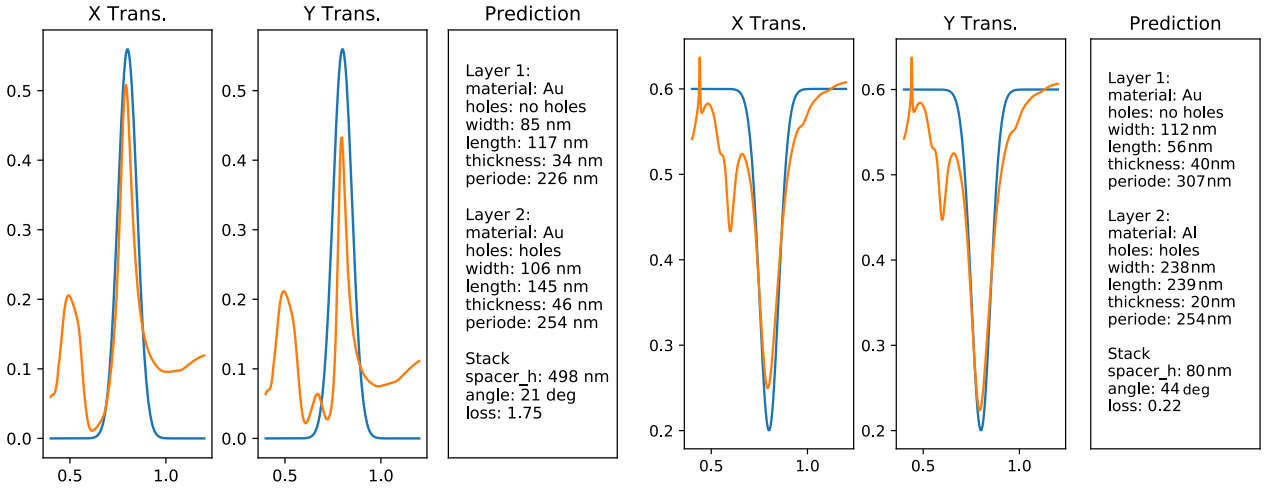


Figure 33: Training of the combined model. Validation loss oscillates wildly.

B. Example Fits

Some example fits. All figures show the transmission over the wave length in μm from 0.4 μm to 1.2 μm.



(a) Gauß Peak (b) Gauß Dip

Figure 34

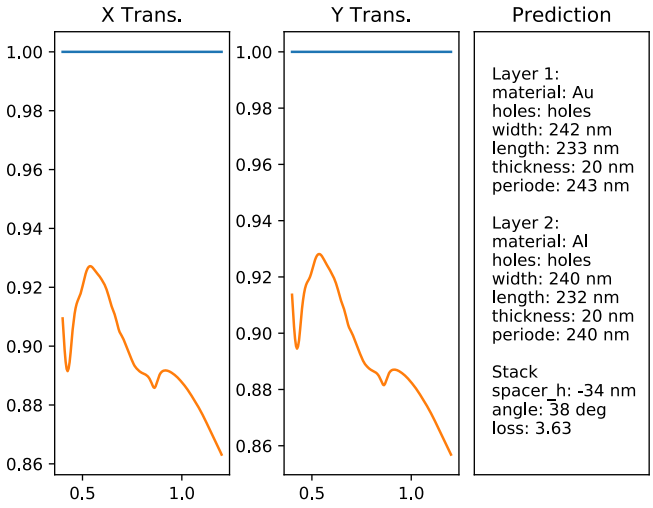


Figure 35: Failed fit on an all one transmission spectrum. The Spacer height should not be negative.

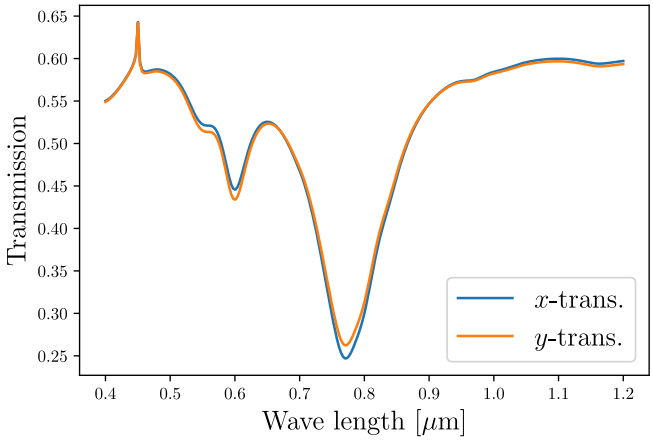


Figure 36: Here, the two sets of layer parameters from figure 34b were directly simulated via FMM and then the transmission spectrum was calculated via SASA. The agreement of this transmission spectrum and figure 34b indecates that the interpoation of S-matrices is working correctly.

References

[1] R. A. Shelby. Experimental verification of a negative index of refraction. *Science*, 292(5514):77–79, apr 2001.

[2] Nanfang Yu and Federico Capasso. Flat optics with designer metasurfaces. *Nature Materials*, 13(2):139–150, jan 2014.

[3] C. Menzel, J. Sperrhake, and T. Pertsch. Efficient treatment of stacked metasurfaces for optimizing and enhancing the range of accessible optical functionalities. *Physical Review A*, 93(6), jun 2016.

[4] Julie Dequaire, Dushyant Rao, Peter Ondruska, Dominic Wang, and Ingmar Posner. Deep tracking on the move: Learning to track the world from a moving vehicle using recurrent neural networks.

[5] Paul Covington, Jay Adams, and Emre Sargin. Deep neural networks for youtube recommendations. In *Proceedings of the 10th ACM Conference on Recommender Systems*, RecSys '16, page 191–198, New York, NY, USA, 2016. Association for Computing Machinery.

[6] J. A. Fan. Generating high performance, topologically-complex metasurfaces with neural networks. In *2019 Conference on Lasers and Electro-Optics (CLEO)*, pages 1–2, 2019.

[7] Wei Ma, Feng Cheng, and Yongmin Liu. Deep-learning-enabled on-demand design of chiral metamaterials. *ACS Nano*, 12(6):6326–6334, jun 2018.

[8] R. M. Redheffer. On a certain linear fractional transformation. *Journal of Mathematics and Physics*, 39(1-4):269–286, apr 1960.

[9] Constantin R. Simovski and Sergei A. Tretyakov. Local constitutive parameters of metamaterials from an effective-medium perspective. *Physical Review B*, 75(19), may 2007.

[10] Thomas Pertsch. Fundermentals of modern optics. Lecture Script, November 2016.

[11] Stefan Alexander Maier. *Plasmonics*. Springer-Verlag GmbH, 2007.

[12] Michael Nielsen. http://neuralnetworksanddeeplearning.com/chap2.html, December 2019.

[13] Matthieu Cord Arthur Douillard, Yifu Chen. Reeseaux convolutionnels pour l'image. https://arthurdouillard.com/files/rdfia_resources/tp6-7.pdf, November 2019.

[14] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.*, 15(1):1929–1958, jan 2014.

[15] Eero Noponen and Jari Turunen. Eigenmode method for electromagnetic synthesis of diffractive elements with three-dimensional profiles. *Journal of the Optical Society of America A*, 11(9):2494, sep 1994.

[16] Donald Shepard. A two-dimensional interpolation function for irregularly-spaced data. In *Proceedings of the 1968 23rd ACM national conference*. ACM Press, 1968.

[17] R. Mead J. A. Nelder. A simplex method for function minimization. *The Computer Journal*, 8(1):27–27, apr 1965.

[18] Dianjing Liu, Yixuan Tan, Erfan Khoram, and Zongfu Yu. Training deep neural networks for the inverse design of nanophotonic structures. *ACS Photonics*, 5(4):1365–1369, feb 2018.

[19] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13(10):281–305, 2012.

[20] Tero Karras, Samuli Laine, and Timo Aila. A style-based generator architecture for generative adversarial networks.

[21] Vincent Dumoulin and Francesco Visin. A guide to convolution arithmetic for deep learning.

[22] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift.

„Ich erkläre hiermit eidesstattlich, dass ich die vorliegende Arbeit selbständig angefertigt habe. Die aus fremden Quellen übernommenen Gedanken sind als solche kenntlich gemacht. Die Arbeit wurde bisher keiner anderen Prüfungsbehörde vorgelegt und auch nicht veröffentlicht. Ich bin mir bewusst, dass eine unwahre Erklärung rechtliche Folgen haben kann.“

Ich möchte mich ganz herzlich bedanken bei Jan Sperrhake für die tolle inhaltliche und formale Unterstützung und bei Johannes Nicklaus für das Korrekturlesen.