4 Algorithm

Before starting to implement anything we need to define which metasurface stacks the algorithm 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. Additionally to keep things simple we want the transmission spectra to be the same for every point of entry along the base of the stack. Transmission spectra which also depend on the point of entry so $I = I(\lambda, x, y)$ can also be done [7] but functions $I = I(\lambda)$ are already useful and can represent for example a custom filter (the directions x and y are shown in figure 16). With this in mind we are going to use a structure which is periodic in the x and y direction and consists of one repeating meta-atom. These kind of surfaces can be manufactured via ... **7.ask jan how these are made**.

As shown in section 3.2 paragraph Symmetries we cannot use meta-atoms of C_4 symmetry to produce different transmissions for the x and y polarization. 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 thats why we are going to limit the algorithm to two layer stacks even though theoretically arbitrary stacks could be used.

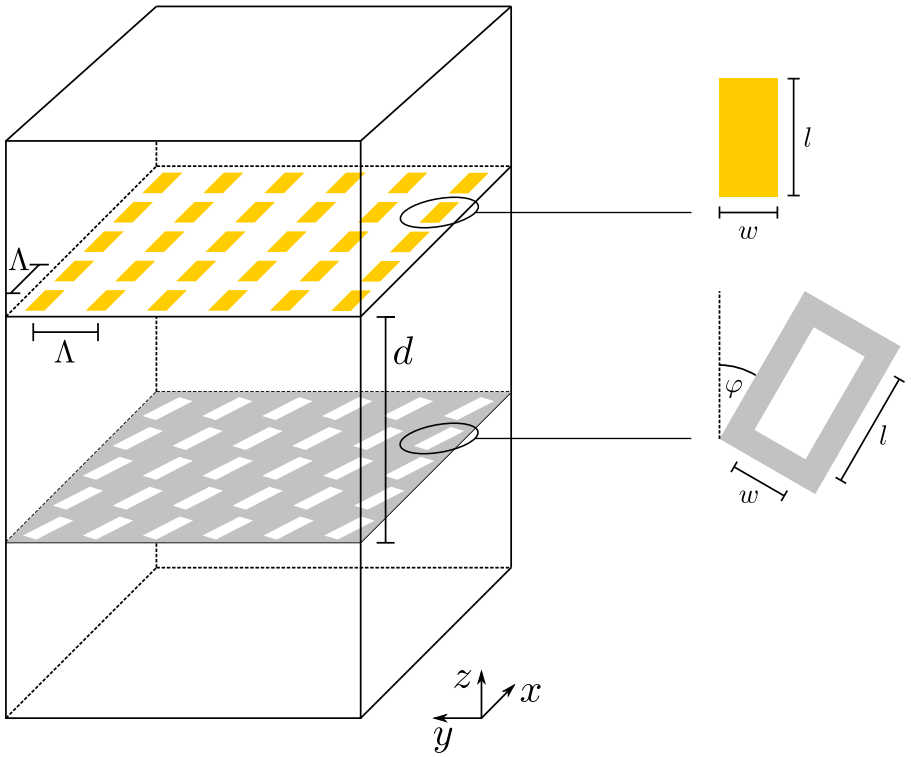


Figure 16: A two layer stack to visualize all the design parameters $\mathcal{D}$ which can be set by the algorithm. These include stack parameters $\mathcal{S}$ and two sets of layer parameters $\mathcal{L}_1$ and $\mathcal{L}_2$. So $\mathcal{D} = (\mathcal{S}, \mathcal{L}_1, \mathcal{L}_2)$. The stack 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 the choice about the material and geometry.

The last decision about the metasurfaces geometry is a little heuristic. A large variety of transmission spectra include some that are very absorbent everywhere except for specific wavelengths and some where almost all wavelengths are transmitted. A metal metasurface absorbs light when the electrons in the meta-atoms can oscillate in resonance. For rectangular meta-atoms this is the case for very specific wavelengths so metasurfaces made of these produce the former kind of spectrum. The opposite can be achieved when using rectangular holes. These metasurfaces have many different resonance frequencies

and only the wavelengths that would have "fit" into the holes are not absorbed. That means the hole geometry can be used to produce the latter kind of spectrum and we have to include both geometries to reach the greatest variety. As for the material we are going to limit the algorithm to Aluminium and Gold this is not physically motivated but we have to remember that every material we add multiplies the amount of metasurfaces that need to be simulated in preparation (check *curse of dimensionality*) and more materials are only useful when the corresponding parameter space is sufficiently dense more on that in the section 4.3. This ends the design considerations, a visualization of all parameters and the notation we are going to use referring to them can be found in figure 16.

4.1 Implementation

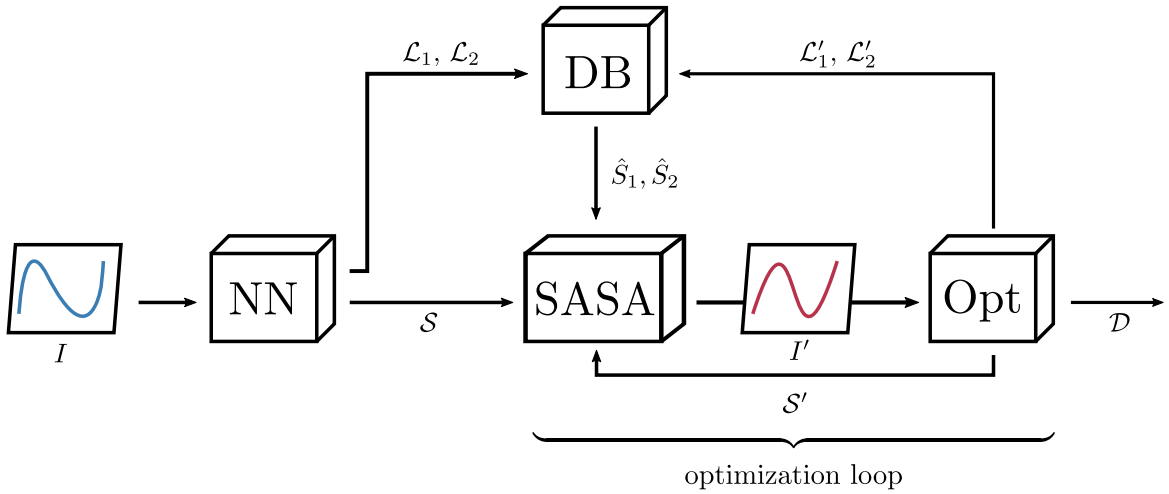


Figure 17: The algorithm tries to find to a given transmission spectrum I the design parameters $\mathcal{D}$ of a two layer meta surface stack which can reproduce this target. The input spectrum is passed to a convolutional neural network which outputs its guess for the stack and layer parameters. The 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 algorithm 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. Find the details to all parts of the algorithm in the sections below.

NN:	convolutional neural network trained to map spectra to stack and layer parameters
DB:	database of pre-simulated single layers
SASA:	algorithm 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 simple 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 3.3 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

$$C_\mathrm{mse}(\mathbf{y}, \mathbf{y}') = \sum_i \left(y_i - y_i'\right)^2 \tag{4.56}$$

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.

$$C_\mathrm{ce}(\mathbf{y}, \mathbf{y}') = -\sum_i y_i \log y_i', \tag{4.57}$$

This error treats false positives and false negatives differently. A false positive does not increase the overall cost as $y_i = 0$ anyway but for a false negative ($y_i = 1, y_i' = 0$) $C_\mathrm{ce} \rightarrow \infty$. This is wanted 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 is similar to the example given in figure 13 while meeting the above-mentioned constraints:

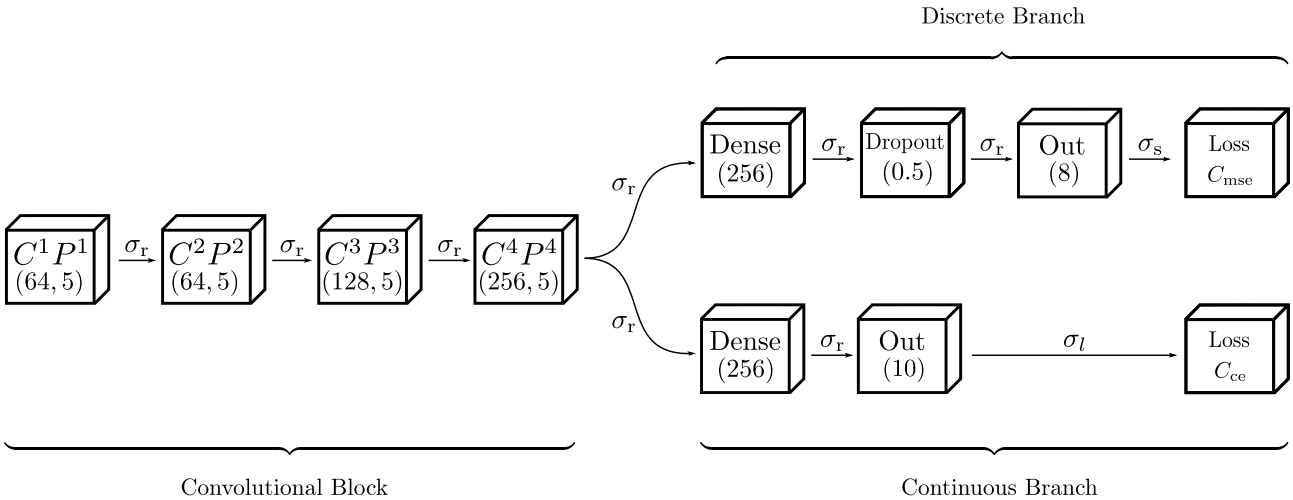
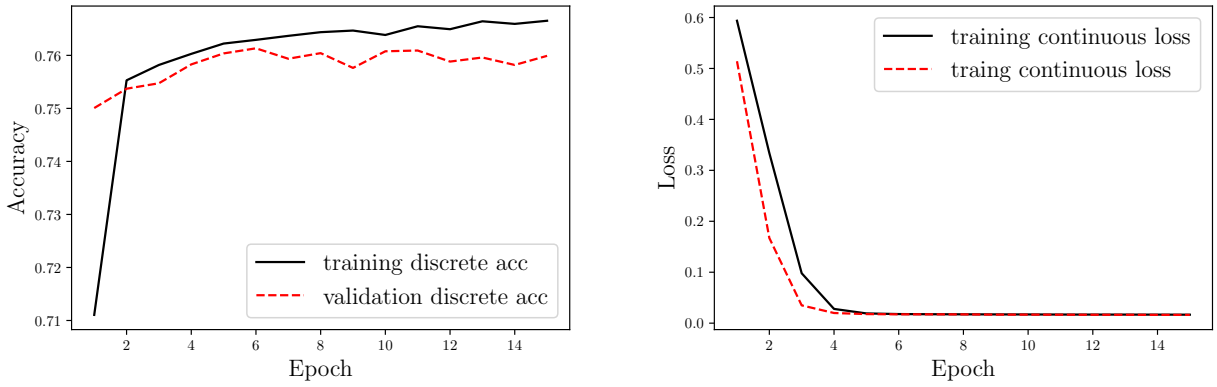


Figure 18: The network starts with 4 pairs of convolutional and pooling layers. The convolutions are characterized by (*number of kernels*, *kernel size*). The kernel size is always 5 and the number of kernels is gradually increased. Then the Network splits into a discrete and a continuous branch via 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. That 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 Aluminium and Gold. One epoch represents one loop over all training samples. After every epoch the network is validated on a dataset $(X_\mathrm{v}, Y_\mathrm{v})$ of samples it has not seen before. This is done to check weither the network actually learns something rather than just memorizing the input data. The results of this first training on the square geometry can be seen in figure 19:



(a) (b)

Figure 19: Training only with squares. (a) shows the accuracy of the discrete output that is what percentage of choices for material and geometry were correct. The performance of the continuous branch is evaluated directly through its cost/loss function C_mse.

Training and validation results are very similar which indicates that there is no overfitting/memorization. The discrete accuracy quickly reaches a maximum of $\sim 76\%$ but 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. That 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 the same for both directions. That means the data generation can result in two different stacks which produce the same spectrum. Consider a stack where one layer is Aluminium and the other is Gold as seen in figure ?? . As both of them produce the same spectrum one time the network is taught that the first layer is Gold and another time its 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.

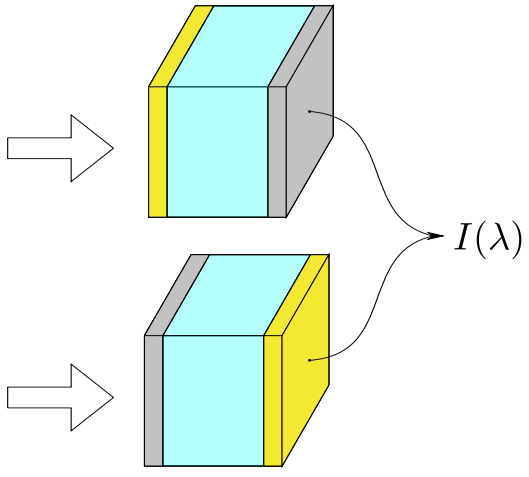
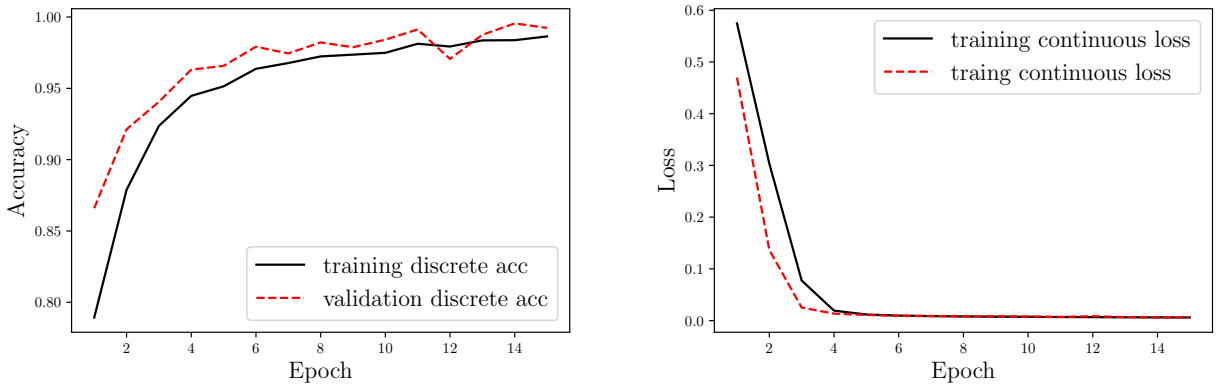


Figure 20: A stack of one Gold and one Aluminium layer separated by a glass spacer. Both stacks produce the same spectrum which leads to issues when using completely random stacks to train the network.

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 described in figure ?? simply by allowing only one of the orientations into the training data. By doing this the training results change as seen in figure 21:



(a) (b)

Figure 21: Training on the square geometry after only allowing one of the equivalent stacks shown in figure ?? into the training data.

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 19.

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 for example for the reflected electric field:

$$\mathbf{E}_\mathrm{r} = \sum_{m,n} \mathbf{R}_{mn} e^{i\mathbf{k}_{mn}\mathbf{r}} \tag{4.58}$$

and then applying the maxwell equations and continuity conditions described in section **??**. This 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. As (4.58) represents an infinite series the equation 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. Thats why the computational complexity of this method increases rapidly with the order. The method was first introduced here [8].

Interpolation

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

$$\bar{\mathcal{L}}_i = \frac{\mathcal{L}_i - \mathcal{L}_i^{\min}}{\mathcal{L}_i^{\max} - \mathcal{L}_i^{\min}},\ i \in 1...4 \quad \text{so that} \quad \bar{\mathcal{L}}_i \in [0,\, 1] \tag{4.59}$$

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:

$$d(\mathcal{L}^1,\, \mathcal{L}^2) := \sum_i \left|\mathcal{L}_i^1 - \mathcal{L}_i^2\right| \tag{4.60}$$

The output $\hat{S}(\mathcal{L})$ is calculated via Inverse Distance Weighting [9] 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 $(\hat{S}_1,\, \ldots,\, \hat{S}_n)$ then:

$$\begin{aligned} \hat{S}(\mathcal{L}) = \sum_{j=1}^{n} w_j\, \hat{S}_j \quad \text{where} \quad w_j = \frac{1/d_j^2}{\sum_i\ 1/d_i^2} \\ \text{so that} \quad \sum_j\ w_j = 1 \end{aligned} \tag{4.61}$$

To have this interpolated approximation $\hat{S}(p)$ be close to the result of rigoros simulation FMM(p) the simulated grid of parameters needs to be sufficiently dense. **8.Insert some calculation of what is sufficiently dense**

4.4 Optimizer

Input:	current spectrum I_c, current design parameters $\mathcal{D}$
Output:	improved design parameters $\mathcal{D}'$

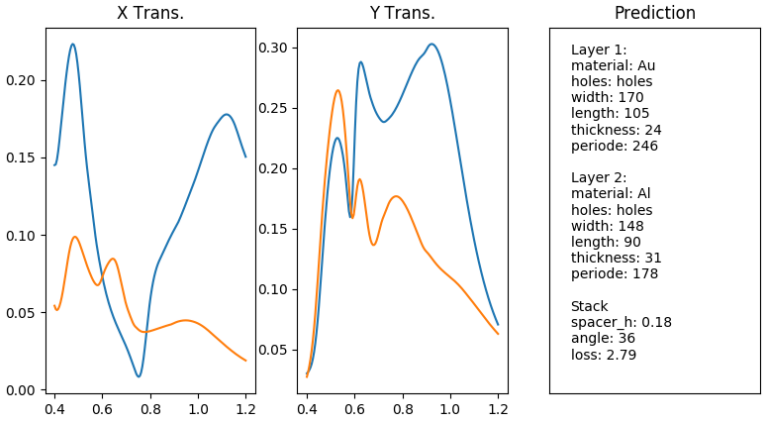
The optimizer is at the core a Downhill-Simplex [10] tuning the continuous design parameters to minimize the mean-squared-difference between the current spectrum I_c and target spectrum I_t we defined as $C_{mse}(I_c(p), I_t)$. The standard method is however unable to follow the physical constraints discussed in section **??** and has to be modified in that regard. To achieve this one 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:

$$D\Big(p, p^l, p^u\Big) = \begin{cases} p^l - p, & \text{for } p < p^l \\ 0, & \text{for } p^l \leq p \leq p^u \\ p - p^u, & \text{for } p^u < p \end{cases} \tag{4.62}$$

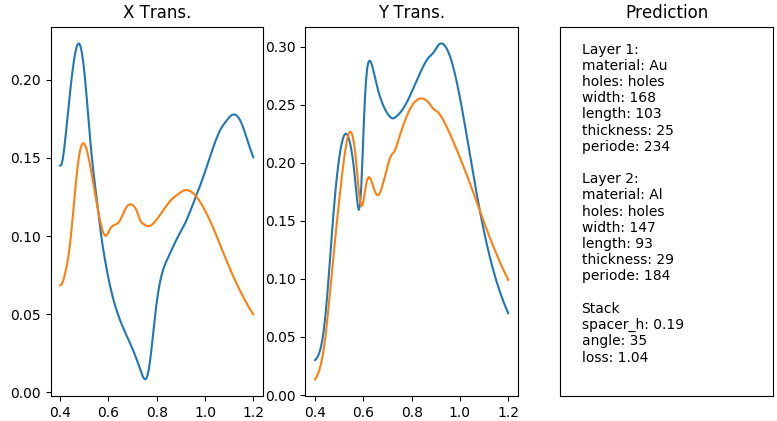
In this way one can penalize the simplex for stepping over the set boundaries by using a total loss L which depends on the sum of all distances D_i:

$$L(I_c, I_t, p) = \underbrace{C_{mse}(I_c, I_t)}_{\text{find target}} + \underbrace{\left[\sum_i D\Big(p_i, p_i^l, p_i^u\Big)\right]^2}_{\text{stay within bounds}} \tag{4.63}$$

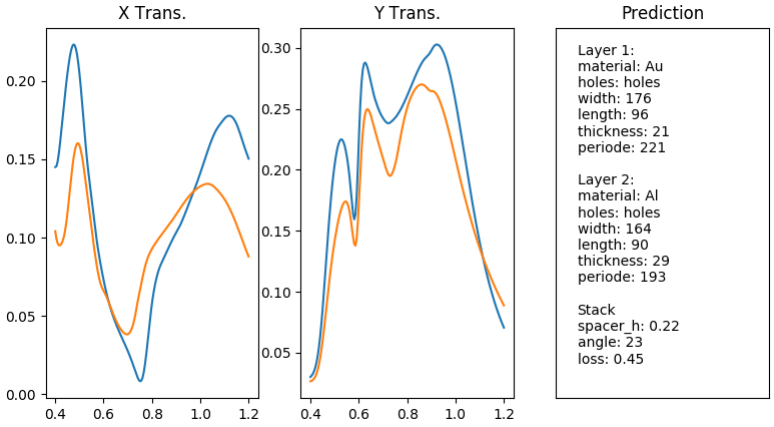
The choice of power depends on how much the simplex should be penalized for stepping over a boundary. All our conditions are requirements for physical approximations and these approximations do no break completely when a boundary is only slightly violated. That justifies this approach where the simplex might step over a boundary but is then "pushed back" in the right direction.



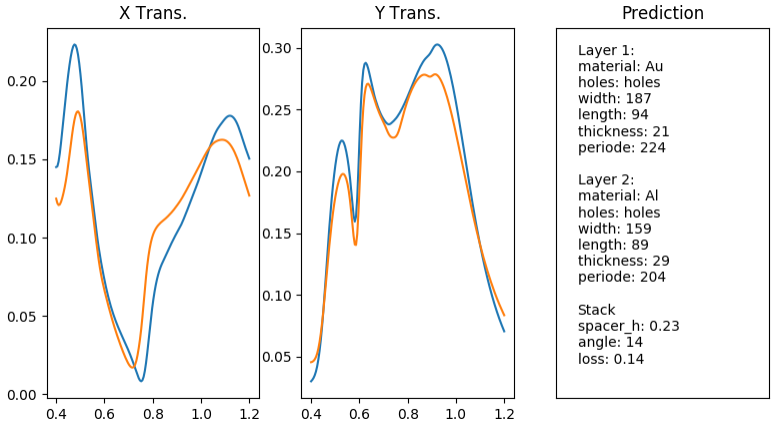
Step 0:
This is the userinterface of the algorithm. In blue we can see the target spectrum I_t which is in this case a random stack of the validation data and the current spectrum I_c is shown in orange. In the right window all the current stack parameters are shown and at the very bottom we can see the current loss. This is the situation right after the network has made its prediction and the optimizer has not changed any parameters.



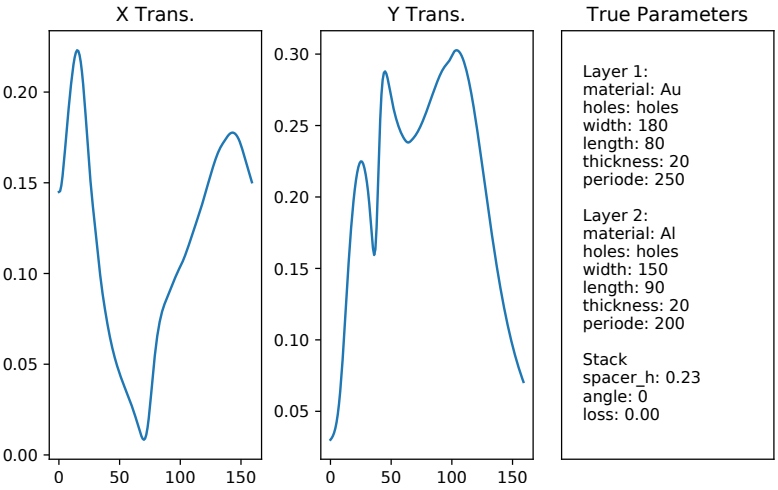
Step 50:
After 50 steps we can see the optimizer has tuned the continuous parameters and left the material and geometry unchanged. The loss has halved and the target and current spectrum look visually more similar.



Step 150:
As the target spectrum is in this cased produced by a know stack the optimizer should be able to reproduce it perfectly.



Step 250:
This is the final result. The optimizer has gotten close to the target but there are still some slight differences. We can compare the result to the true parameters used for the generation of this spectrum seen in the figure below.



6 Sources

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

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

[6] 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.

[7] J. P. Balthasar Mueller, Noah A. Rubin, Robert C. Devlin, Benedikt Groever, and Federico Capasso. Metasurface polarization optics: Independent phase control of arbitrary orthogonal states of polarization. *Physical Review Letters*, 118(11), mar 2017.

[8] 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.

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

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