Machine learning for the free-form inverse design in nanophotonics

Timo Gahlmann

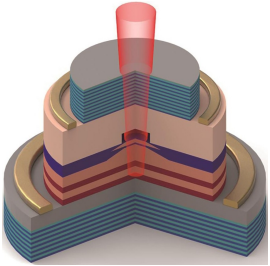
Amolf

21 October 2024

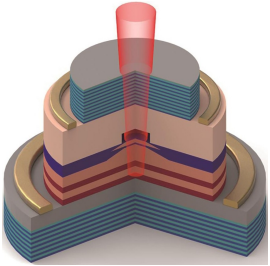
Why and how does one use ML to design metasurfaces?

Why and how does one use ML to design metasurfaces?

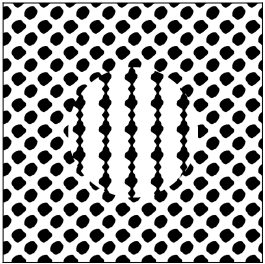
Why and how does one use ML to design metasurfaces?



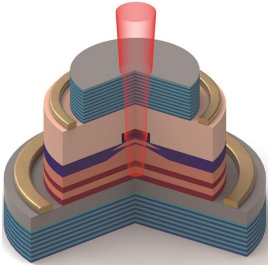
Why and how does one use ML to design metasurfaces?



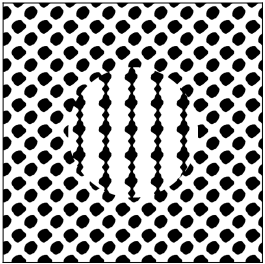
2D cross section of ZWFS metasurface

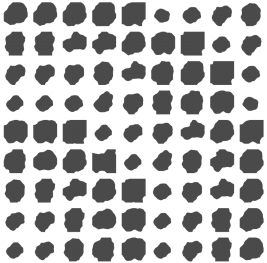


Why and how does one use ML to design metasurfaces?

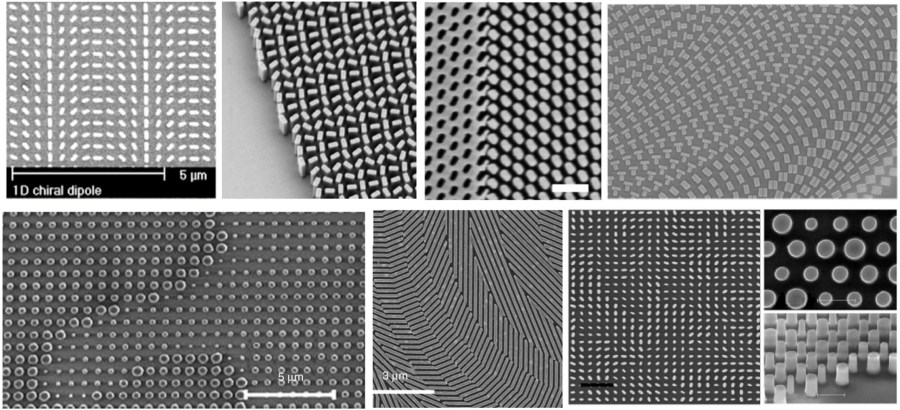


2D cross section of ZWFS metasurface





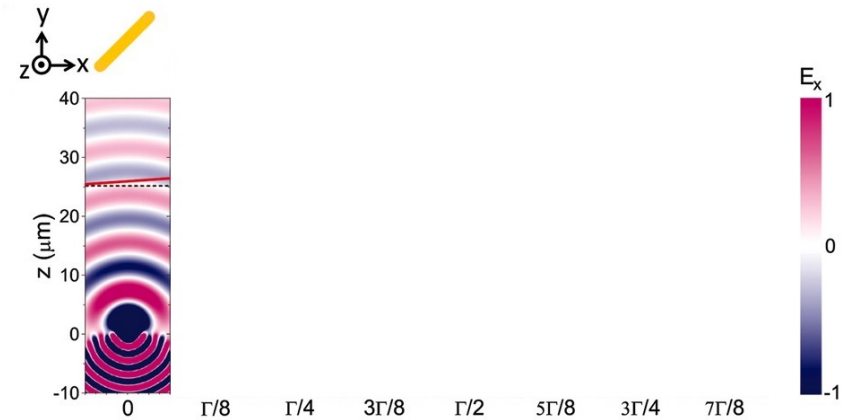
Metasurface examples



Dragomir Neshev, Igor Aharonovich "Optical metasurfaces: new generation building blocks for multi-functional optics," Light Sci Appl 7, 58 (2018).

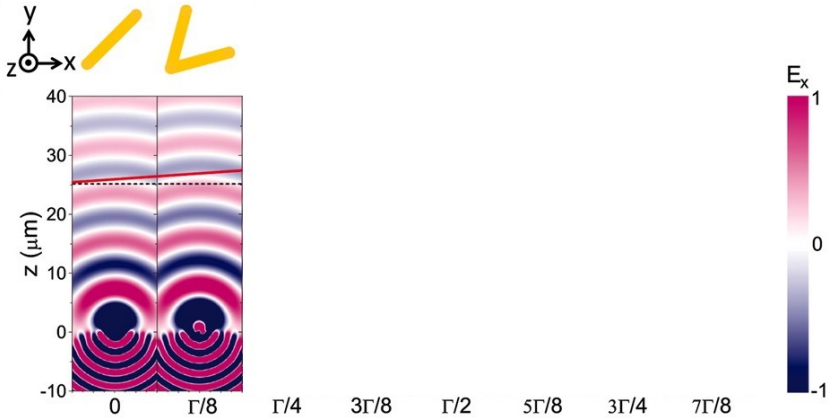
Ji Chen, et al. "Metamaterials: From fundamental physics to intelligent design," Interdisciplinary Materials, (2022).

Metasurfaces - unit cell/meta-atom



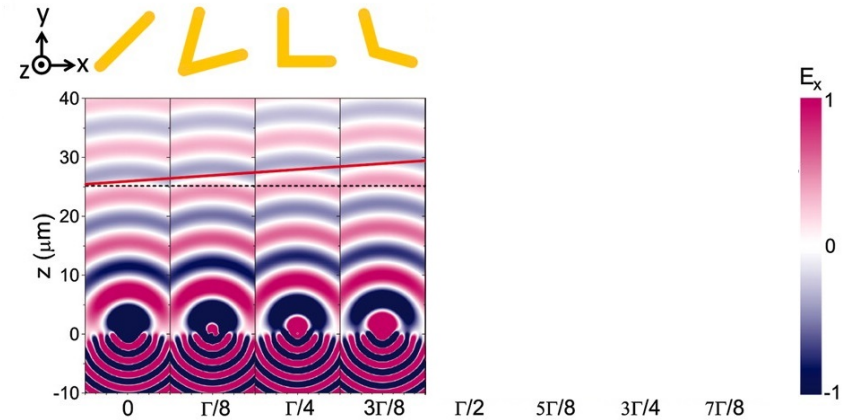
Nanfang Yu, Frederico Capasso, et al. "Light Propagation with Phase Discontinuities: Generalized Laws of Reflection and Refraction," Science334, 333-337 (2011).

Metasurfaces - unit cell/meta-atom



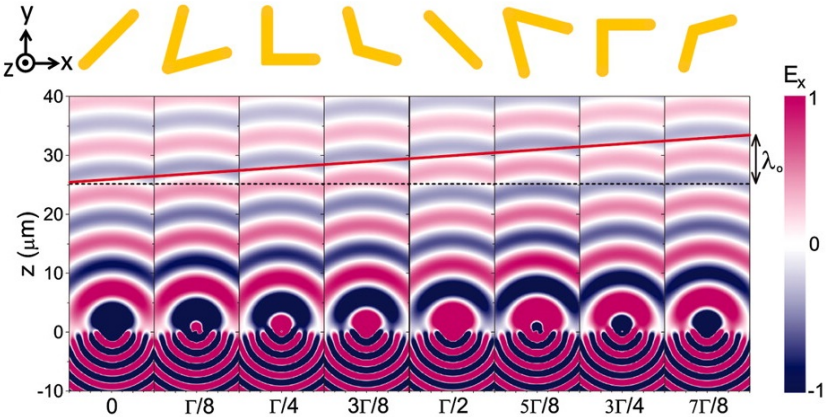
Nanfang Yu, Frederico Capasso, et al. "Light Propagation with Phase Discontinuities: Generalized Laws of Reflection and Refraction," Science334, 333-337 (2011).

Metasurfaces - unit cell/meta-atom



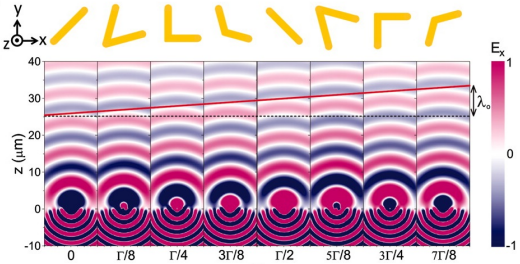
Nanfang Yu, Frederico Capasso, et al. "Light Propagation with Phase Discontinuities: Generalized Laws of Reflection and Refraction," Science334, 333-337 (2011).

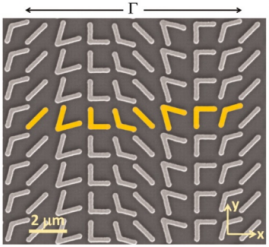
Metasurfaces - unit cell/meta-atom

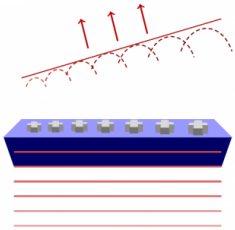


Nanfang Yu, Frederico **Capasso**, et al. "Light Propagation with Phase Discontinuities: Generalized Laws of Reflection and Refraction," Science334, 333-337 (2011).

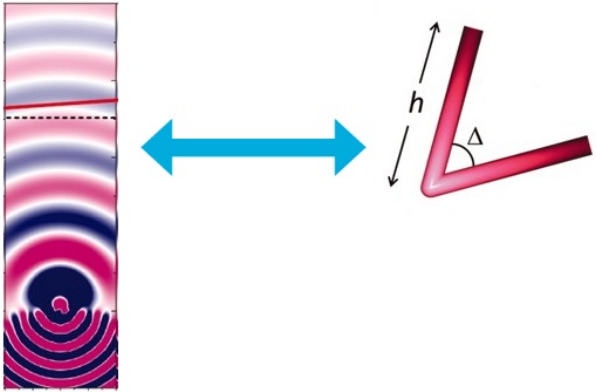
Huygens' principle - Metasurface phase gradient





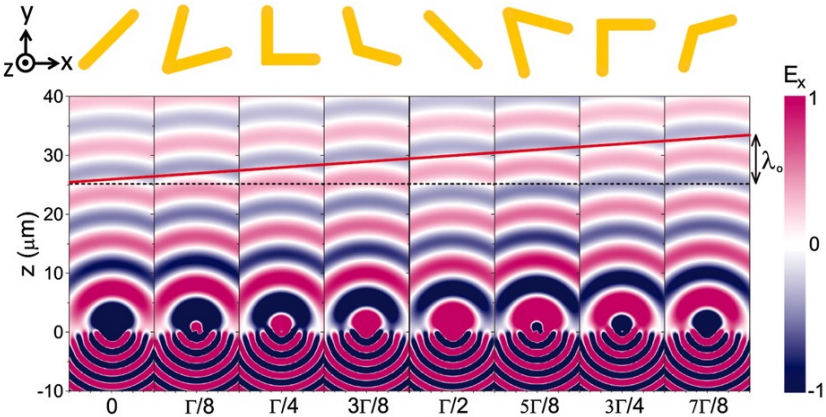


Inverse design - Parameterize geometry



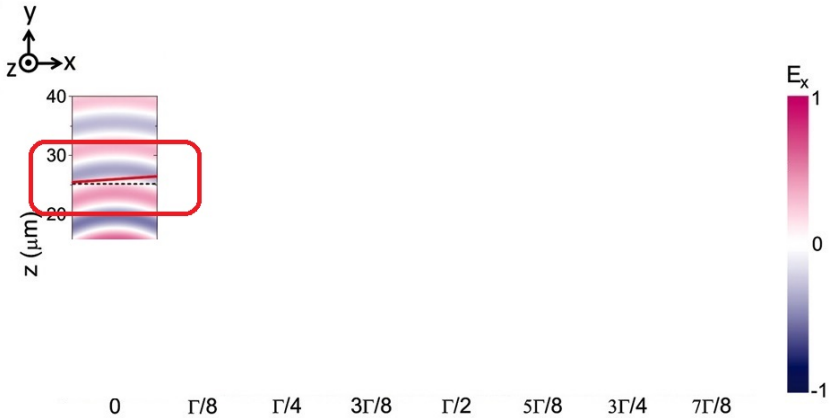
Nanfang Yu, Frederico Capasso, et al. "Light Propagation with Phase Discontinuities: Generalized Laws of Reflection and Refraction," Science334, 333-337 (2011).

Inverse design - Optical properties



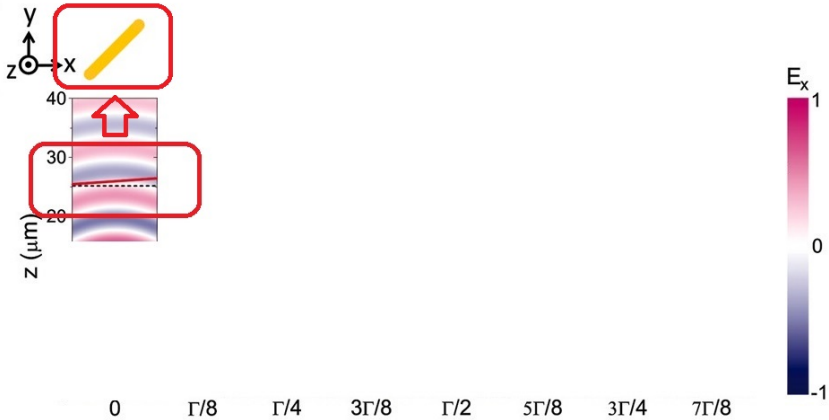
Nanfang Yu, Frederico **Capasso**, et al. "Light Propagation with Phase Discontinuities: Generalized Laws of Reflection and Refraction," Science334, 333-337 (2011).

Inverse design - Optical properties



Nanfang Yu, Frederico Capasso, et al. "Light Propagation with Phase Discontinuities: Generalized Laws of Reflection and Refraction," Science334, 333-337 (2011).

Inverse design - Optical properties → Parameterize geometry

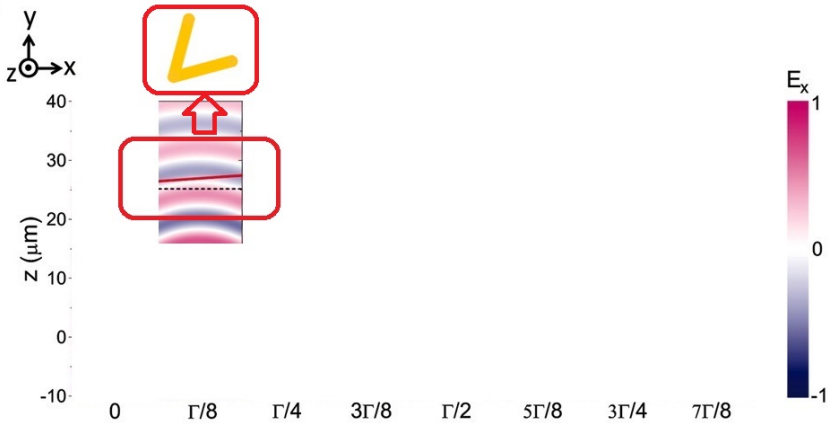


Nanfang Yu, Frederico Capasso, et al. "Light Propagation with Phase Discontinuities: Generalized Laws of Reflection and Refraction," Science334, 333-337 (2011).

Inverse design - Optical properties

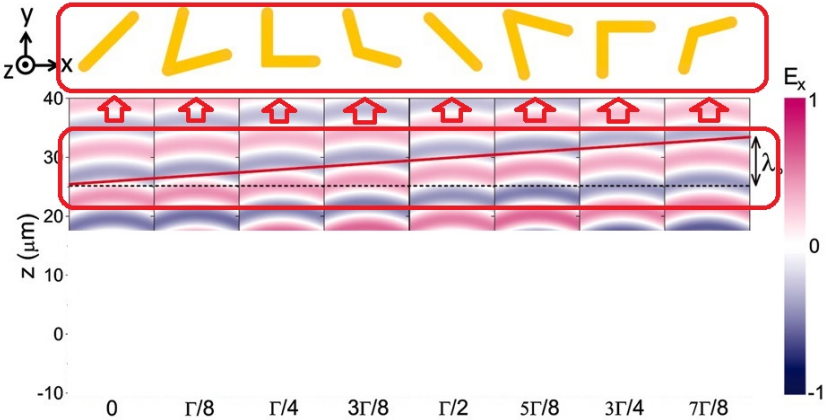
Nanfang Yu, Frederico **Capasso**, et al. "Light Propagation with Phase Discontinuities: Generalized Laws of Reflection and Refraction," Science334, 333-337 (2011).

Inverse design - Optical properties → Parameterize geometry



Nanfang Yu, Frederico Capasso, et al. "Light Propagation with Phase Discontinuities: Generalized Laws of Reflection and Refraction," Science334, 333-337 (2011).

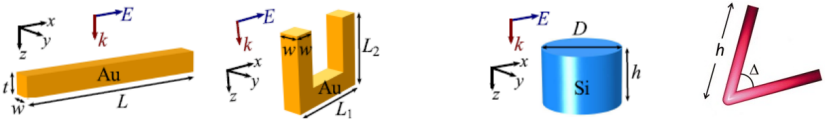
Inverse design - Optical properties → Parameterize geometries



Nanfang Yu, Frederico Capasso, et al. "Light Propagation with Phase Discontinuities: Generalized Laws of Reflection and Refraction," Science334, 333-337 (2011).

Inverse design - 1 to a few parameters

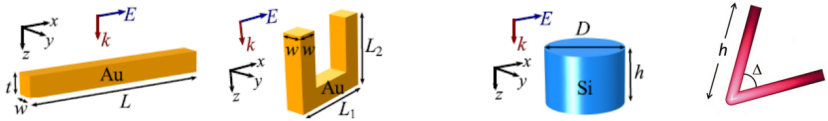
Inverse design - 1 to a few parameters



Fei Ding, Anders Pors and Sergey I. Bozhevolnyi "Gradient metasurfaces: a review of fundamentals and applications,"

https://arxiv.org/pdf/1704.03032.pdf

Inverse design - 1 to a few parameters

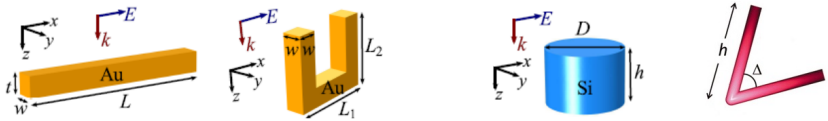


Fei Ding, Anders Pors and Sergey I. Bozhevolnyi "Gradient metasurfaces: a review of fundamentals and applications,"

https://arxiv.org/pdf/1704.03032.pdf

- **Little design freedom** → limited amount of optical responses accessible

Inverse design - 1 to a few parameters

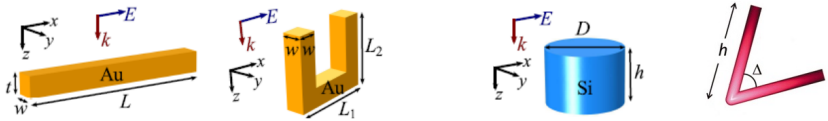


Fei Ding, Anders Pors and Sergey I. Bozhevolnyi "Gradient metasurfaces: a review of fundamentals and applications,"

https://arxiv.org/pdf/1704.03032.pdf

- **Little design freedom** → limited amount of optical responses accessible
- Solvable with:
  - Parameter scan

Inverse design - 1 to a few parameters



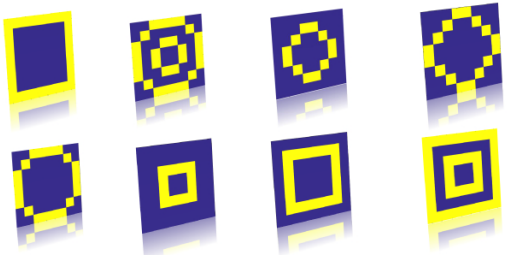
Fei Ding, Anders Pors and Sergey I. Bozhevolnyi "Gradient metasurfaces: a review of fundamentals and applications,"

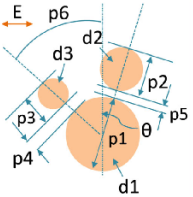
https://arxiv.org/pdf/1704.03032.pdf

- **Little design freedom** → limited amount of optical responses accessible
- Solvable with:
  - Parameter scan
  - Regression

Inverse design - few parameters to a few 10s of parameters

Inverse design - few parameters to a few 10s of parameters

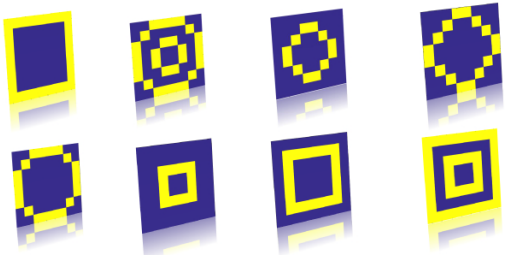


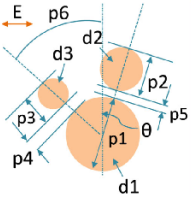


(left) Fardin Ghorbani, et al. "A deep learning approach for Inverse design of the metasurface for dual-polarized waves"

Ronghui Lin, et al. "Inverse design of plasmonic metasurfaces by convolutional neural network" (right)

Inverse design - few parameters to a few 10s of parameters



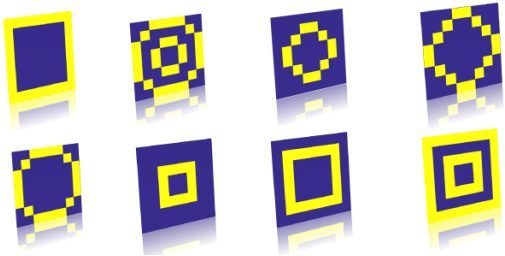


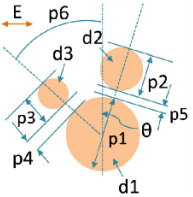
(left) Fardin Ghorbani, et al. "A deep learning approach for Inverse design of the metasurface for dual-polarized waves"

Ronghui Lin, et al. "Inverse design of plasmonic metasurfaces by convolutional neural network" (right)

- **Many (sequential) simulations**

Inverse design - few parameters to a few 10s of parameters



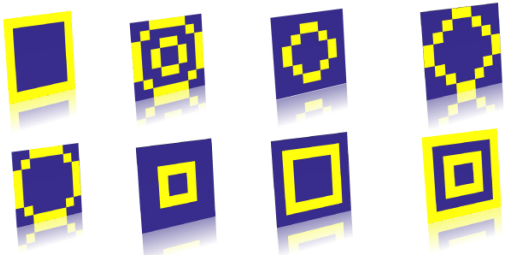


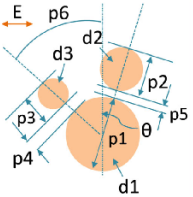
(left) Fardin Ghorbani, et al. "A deep learning approach for Inverse design of the metasurface for dual-polarized waves"

Ronghui Lin, et al. "Inverse design of plasmonic metasurfaces by convolutional neural network" (right)

- **Many (sequential) simulations**
- Solvable with:
  - Binary search

Inverse design - few parameters to a few 10s of parameters



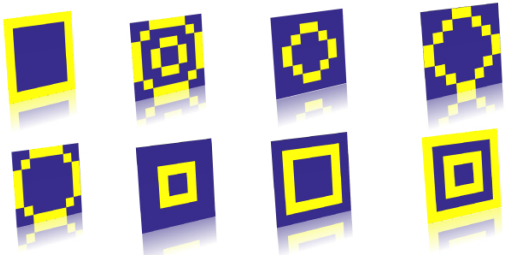


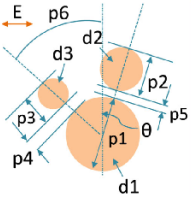
(left) Fardin Ghorbani, et al. "A deep learning approach for Inverse design of the metasurface for dual-polarized waves"

Ronghui Lin, et al. "Inverse design of plasmonic metasurfaces by convolutional neural network" (right)

- **Many (sequential) simulations**
- Solvable with:
  - Binary search
  - Genetic algorithms

Inverse design - few parameters to a few 10s of parameters



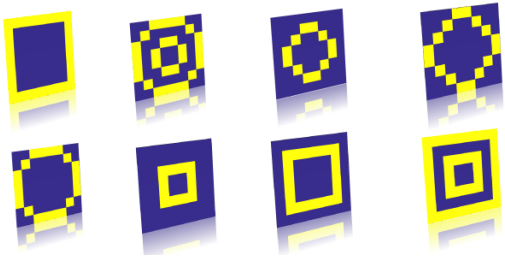


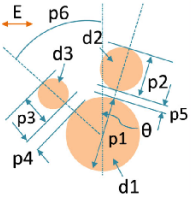
(left) Fardin Ghorbani, et al. "A deep learning approach for Inverse design of the metasurface for dual-polarized waves"

Ronghui Lin, et al. "Inverse design of plasmonic metasurfaces by convolutional neural network" (right)

- **Many (sequential) simulations**
- Solvable with:
  - Binary search
  - Genetic algorithms
  - Particle swarm optimization

Inverse design - few parameters to a few 10s of parameters





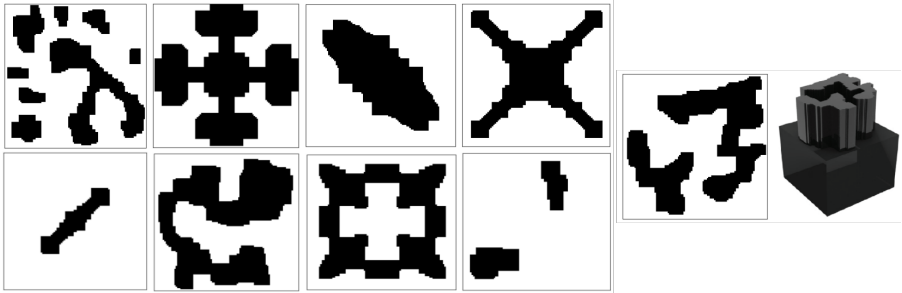
(left) Fardin Ghorbani, et al. "A deep learning approach for Inverse design of the metasurface for dual-polarized waves"

Ronghui Lin, et al. "Inverse design of plasmonic metasurfaces by convolutional neural network" (right)

- **Many (sequential) simulations**
- Solvable with:
  - Binary search
  - Genetic algorithms
  - Particle swarm optimization
  - Artificial neural networks

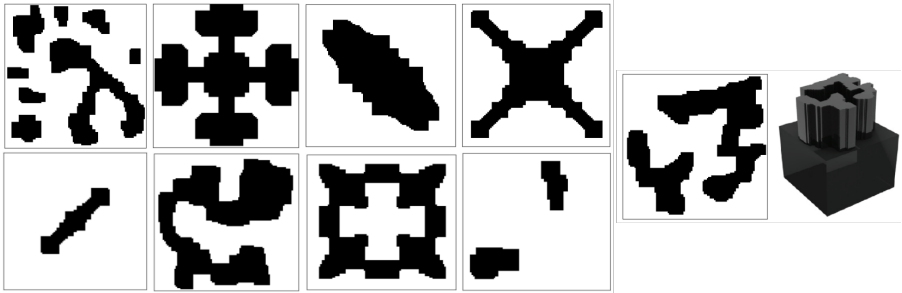
Inverse design - more then a few 10s of parameters

Inverse design - more then a few 10s of parameters



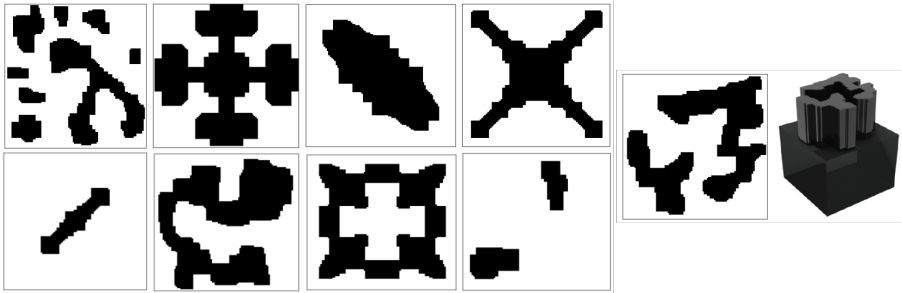
- **Free-form $\rightarrow$ High design freedom**

Inverse design - more then a few 10s of parameters



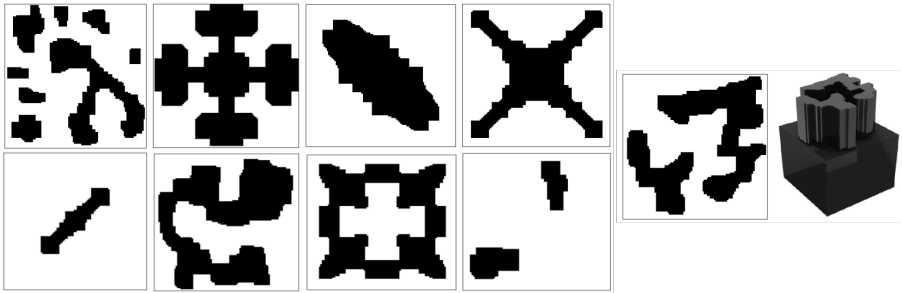
- **Free-form → High design freedom**
  - More optical responses accessible

Inverse design - more then a few 10s of parameters



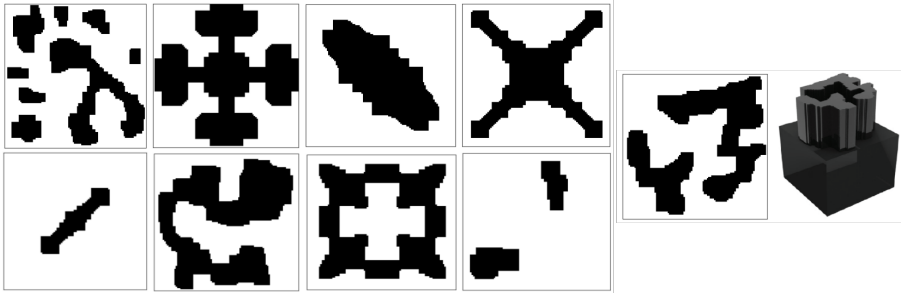
- **Free-form → High design freedom**
  - More optical responses accessible
  - Convergence issues

Inverse design - more then a few 10s of parameters



- **Free-form → High design freedom**
  - More optical responses accessible
  - Convergence issues
- Topological optimization

Inverse design - more then a few 10s of parameters



- **Free-form → High design freedom**
  - More optical responses accessible
  - Convergence issues
- Topological optimization
- Powerful algorithmic solution:
  Physics-informed gradient descent with adjoint method

Alternative solution: Artificial neural networks

Alternative solution: Artificial neural networks

Use **neural network** to **create a map** from **designs to optical properties**

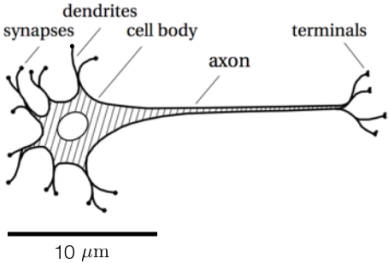
Alternative solution: Artificial neural networks - Inverse design

Use **neural network** to **create a map** from **optical properties to designs**

Neural Networks: Neuron

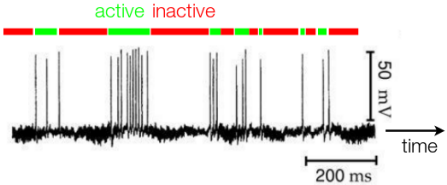
Schematic drawing of a neuron

input → process → output



Total length of dendrites up to $\sim$ cm .

Output of a neuron: *spike train*



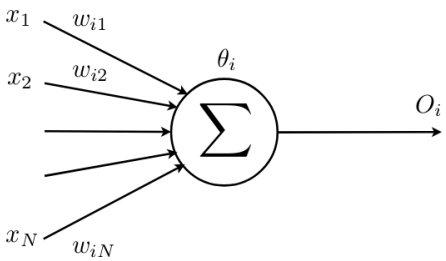
Spike train in electrosensory pyramidal neuron in fish (*Eigenmannia)*

Gabbiani & Metzner, J. Exp. Biol. **202** (1999) 1267

from: Machine Learning with Neural Networks, B Mehlig

Artificial Neuron

Simple model for a neuron:



Incoming signals x_j $j=1,\ldots,N$

Weights w_{ij} (synaptic couplings)

Threshold θ_i
Neuron number i

Output O_i

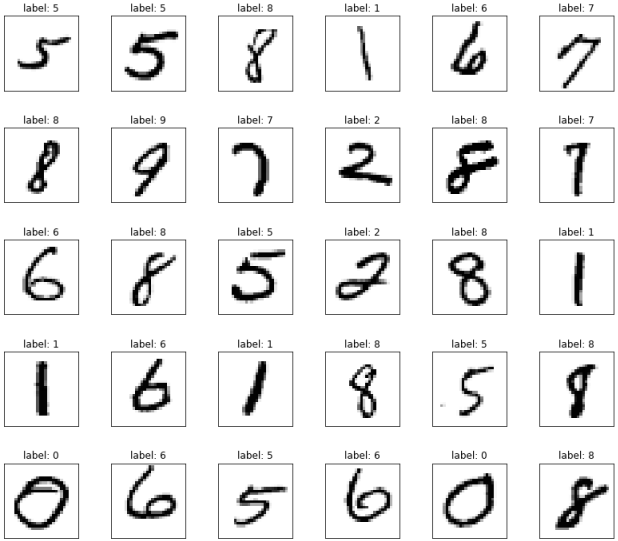
Signal processing: weighted sum of inputs
Activation function $g(b_i)$

$$O_i = g\Big(\underbrace{\sum_{j=1}^{N} w_{ij}x_j - \theta_i}_{= b_i \text{ (local field)}}\Big)$$

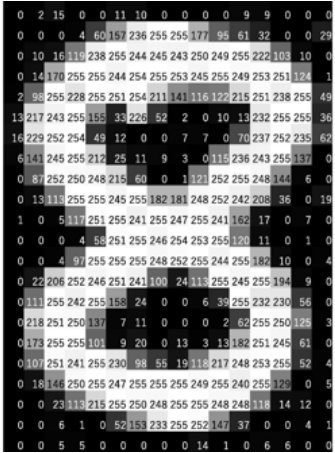
$$= g(w_{i1}x_1 + \ldots + w_{iN}x_N - \theta_i)$$

from: Machine Learning with Neural Networks, B Mehlig

Mnist dataset



Mnist dataset



0	2	15	0	0	11	10	0	0	0	0	9	9	0	0	0
0	0	0	4	60	157	236	255	255	177	95	61	32	0	0	29
0	10	16	119	238	255	244	245	243	250	249	255	222	103	10	0
0	14	170	255	255	244	254	255	253	245	255	249	253	251	124	1
2	98	255	228	255	251	254	211	141	116	122	215	251	238	255	49
13	217	243	255	155	33	226	52	2	0	10	13	232	255	255	36
16	229	252	254	49	12	0	0	7	7	0	70	237	252	235	62
6	141	245	255	212	25	11	9	3	0	115	236	243	255	137	0
0	87	252	250	248	215	60	0	1	121	252	255	248	144	6	0
0	13	113	255	255	245	255	182	181	248	252	242	208	36	0	19
1	0	5	117	251	255	241	255	247	255	241	162	17	0	7	0
0	0	0	4	58	251	255	246	254	253	255	120	11	0	1	0
0	0	4	97	255	255	255	248	252	255	244	255	182	10	0	4
0	22	206	252	246	251	241	100	24	113	255	245	255	194	9	0
0	111	255	242	255	158	24	0	0	6	39	255	232	230	56	0
0	218	251	250	137	7	11	0	0	0	2	62	255	250	125	3
0	173	255	255	101	9	20	0	13	3	13	182	251	245	61	0
0	107	251	241	255	230	98	55	19	118	217	248	253	255	52	4
0	18	146	250	255	247	255	255	249	255	240	255	129	0	5	
0	0	23	113	215	255	250	248	255	255	248	248	118	14	12	0
0	0	6	1	0	52	153	233	255	252	147	37	0	0	4	1
0	0	5	5	0	0	0	0	0	14	1	0	6	6	0	0

0	2	15	0	0	11	10	0	0	0	0	9	9	0	0	0
0	0	0	4	60	157	236	255	255	177	95	61	32	0	0	29
0	10	16	119	238	255	244	245	243	250	249	255	222	103	10	0
0	14	170	255	255	244	254	255	253	245	255	249	253	251	124	1
2	98	255	228	255	251	254	211	141	116	122	215	251	238	255	49
13	217	243	255	155	33	226	52	2	0	10	13	232	255	255	36
16	229	252	254	49	12	0	0	7	7	0	70	237	252	235	62
6	141	245	255	212	25	11	9	3	0	115	236	243	255	137	0
0	87	252	250	248	215	60	0	1	121	252	255	248	144	6	0
0	13	113	255	255	245	255	182	181	248	252	242	208	36	0	19
1	0	5	117	251	255	241	255	247	255	241	162	17	0	7	0
0	0	0	4	58	251	255	246	254	253	255	120	11	0	1	0
0	0	4	97	255	255	255	248	252	255	244	255	182	10	0	4
0	22	206	252	246	251	241	100	24	113	255	245	255	194	9	0
0	111	255	242	255	158	24	0	0	6	39	255	232	230	56	0
0	218	251	250	137	7	11	0	0	0	2	62	255	250	125	3
0	173	255	255	101	9	20	0	13	3	13	182	251	245	61	0
0	107	251	241	255	230	98	55	19	118	217	248	253	255	52	4
0	18	146	250	255	247	255	255	249	255	240	255	129	0	5	
0	0	23	113	215	255	250	248	255	255	248	248	118	14	12	0
0	0	6	1	0	52	153	233	255	252	147	37	0	0	4	1
0	0	5	5	0	0	0	0	0	14	1	0	6	6	0	0

Mnist dataset

ChatGPT 4o with canvas

Create a tensorflow classifier of the mnist dataset. Include loading the dataset, training the classifier and testing it.

Sure thing! Here's a simple TensorFlow classifier for the MNIST dataset. This script includes loading the dataset, training the classifier, and evaluating its performance on the test set.

Mnist Tensorflow Classifier

This code will train a simple Convolutional Neural Network (CNN) to classify handwritten digits from the MNIST dataset. If you'd like to modify or add complexity to the network, feel free to let me know!

Mnist Tensorflow Classifier

```python
import tensorflow as tf
from tensorflow.keras import layers, models
import numpy as np

# Load the MNIST dataset (handwritten digits 0-9)
(x_train, y_train), (x_test, y_test) = tf.keras.datasets.mnist.load_data()

# Normalize the images to the range [0, 1]
x_train, x_test = x_train / 255.0, x_test / 255.0

# Expand the dimensions to add a channel for compatibility with Conv2D
x_train = np.expand_dims(x_train, -1)
x_test = np.expand_dims(x_test, -1)

# Build a simple Convolutional Neural Network (CNN) model
model = models.Sequential([
    layers.Input(shape=(28, 28, 1)),
    layers.Conv2D(32, kernel_size=(3, 3), activation='relu'),
    layers.MaxPooling2D(pool_size=(2, 2)),
    layers.Conv2D(64, kernel_size=(3, 3), activation='relu'),
    layers.MaxPooling2D(pool_size=(2, 2)),
    layers.Flatten(),
    layers.Dense(128, activation='relu'),
    layers.Dense(10, activation='softmax')
])

# Compile the model
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

# Train the model
model.fit(x_train, y_train, epochs=5, batch_size=64, validation_split=0.1)

# Evaluate the model on the test set
test_loss, test_accuracy = model.evaluate(x_test, y_test)
print(f"Test accuracy: {test_accuracy:.4f}")
```

Mnist dataset

ChatGPT 4o with canvas

Create a tensorflow classifier of the mnist dataset. Include loading the dataset, training the classifier and testing it.

Mnist Tf Classifier

Here's a simple TensorFlow classifier for the MNIST dataset. It includes loading the dataset, defining and training the model, and evaluating its performance. Let me know if you need any modifications or additional explanations!

do the same in pytorch

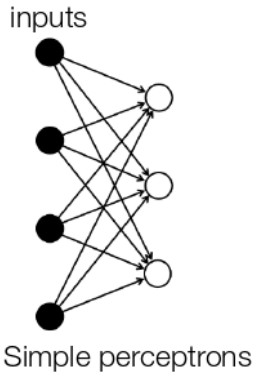
Edited

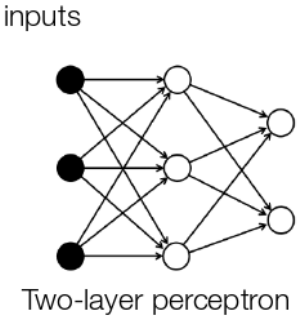
I've converted the TensorFlow code to PyTorch, including data loading, model definition, training, and evaluation. Let me know if you need further modifications or additional explanations about the changes!

Mnist Tf Classifier

```python
import torch
import torch.nn as nn
import torch.optim as optim
import torch.nn.functional as F
from torchvision import datasets, transforms
from torch.utils.data import DataLoader

# Load MNIST dataset with torchvision transforms
transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.5,), (0.5,))
])

train_dataset = datasets.MNIST(root='./data', train=True, download=True, transform=transform)
test_dataset = datasets.MNIST(root='./data', train=False, download=True, transform=transform)

train_loader = DataLoader(train_dataset, batch_size=64, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=1000, shuffle=False)

# Define the model
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(1, 32, kernel_size=3)
        self.conv2 = nn.Conv2d(32, 64, kernel_size=3)
        self.pool = nn.MaxPool2d(2, 2)
        self.fc1 = nn.Linear(64 * 5 * 5, 128)
        self.fc2 = nn.Linear(128, 10)

    def forward(self, x):
        x = F.relu(self.conv1(x))
        x = self.pool(F.relu(self.conv2(x)))
        x = x.view(-1, 64 * 5 * 5)
        x = F.relu(self.fc1(x))
        x = self.fc2(x)
        return F.log_softmax(x, dim=1)

model = Net()

# Define the optimizer and loss function
optimizer = optim.Adam(model.parameters(), lr=0.001)
```

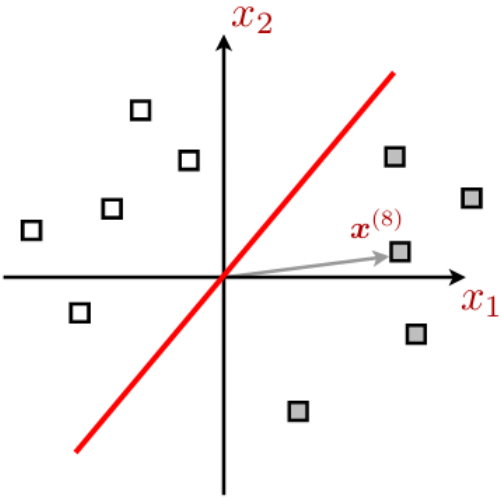
First code example





from: Machine Learning with Neural Networks, B Mehlig

Simple Classification



Back Propagation / Gradient Decent

- **Start with random weights**

Back Propagation / Gradient Decent

- **Start with random weights**
- Let ANN perform on subset of data (batch)

Back Propagation / Gradient Decent

- **Start with random weights**
- Let ANN perform on subset of data (batch)
- **Calculate error** through loss function (similar to Hamilton)

Back Propagation / Gradient Decent

- **Start with random weights**
- Let ANN perform on subset of data (batch)
- **Calculate error** through loss function (similar to Hamilton)
- Find **gradient** in multi dimensional landscape

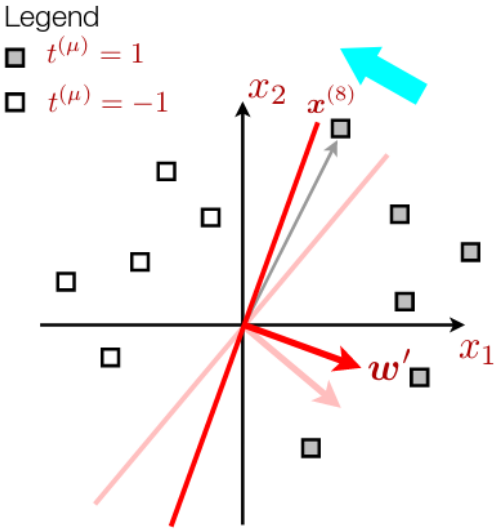
Back Propagation / Gradient Decent

- **Start with random weights**
- Let ANN perform on subset of data (batch)
- **Calculate error** through loss function (similar to Hamilton)
- Find **gradient** in multi dimensional landscape
- **Adjust weights and biases** to minimize error

Back Propagation / Gradient Decent

- **Start with random weights**
- Let ANN perform on subset of data (batch)
- **Calculate error** through loss function (similar to Hamilton)
- Find **gradient** in multi dimensional landscape
- **Adjust weights and biases** to minimize error
- **Repeat** until (hopefully global) minimum is found

Gradient Decent



Neural Networks

Simple building blocks in wide variety of **complex networks** perform numerous tasks:

Neural Networks

Simple building blocks in wide variety of **complex networks** perform numerous tasks:

- **Function approximation**, regression analysis, time series prediction

Neural Networks

Simple building blocks in wide variety of **complex networks** perform numerous tasks:

- **Function approximation**, regression analysis, time series prediction
- **Classification**, including pattern and sequence recognition

Neural Networks

Simple building blocks in wide variety of **complex networks** perform numerous tasks:

- **Function approximation**, regression analysis, time series prediction
- **Classification**, including pattern and sequence recognition
- **Data processing**, filtering, clustering and compression

Neural Networks

Simple building blocks in wide variety of **complex networks** perform numerous tasks:

- **Function approximation**, regression analysis, time series prediction
- **Classification**, including pattern and sequence recognition
- **Data processing**, filtering, clustering and compression
- Robotics and control

Neural Networks

Simple building blocks in wide variety of **complex networks** perform numerous tasks:

- **Function approximation**, regression analysis, time series prediction
- **Classification**, including pattern and sequence recognition
- **Data processing**, filtering, clustering and compression
- Robotics and control
- **Creative AI**

Important to note

Limitations of Neural networks:

Important to note

Limitations of Neural networks:

- Solutions are **never a proof**

Important to note

Limitations of Neural networks:

- Solutions are **never a proof**
- **Good interpolation**
  poor extrapolation

Important to note

Limitations of Neural networks:

- Solutions are **never a proof**
- **Good interpolation**
  poor extrapolation
- **Convergence not guaranteed**

Important to note

Limitations of Neural networks:

- Solutions are **never a proof**
- **Good interpolation**
  poor extrapolation
- **Convergence not guaranteed**

Approaches:

Important to note

Limitations of Neural networks:

- Solutions are **never a proof**
- **Good interpolation**
  poor extrapolation
- **Convergence not guaranteed**

Approaches:

- Optimize **hyperparameters**

Important to note

Limitations of Neural networks:

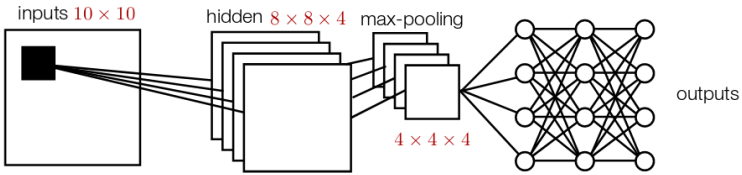
- Solutions are **never a proof**
- **Good interpolation**
  poor extrapolation
- **Convergence not guaranteed**

Approaches:

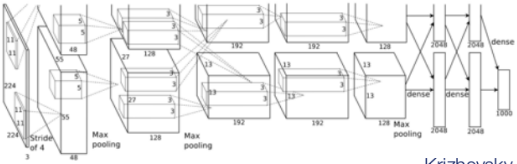
- Optimize **hyperparameters**
- Optimize **training data**

Deep ANN: Image Classification

- *max-pooling*: take maximum of V_{mn} over small region (2×2) to reduce # of parameters



- add fully connected layers to learn more abstract features

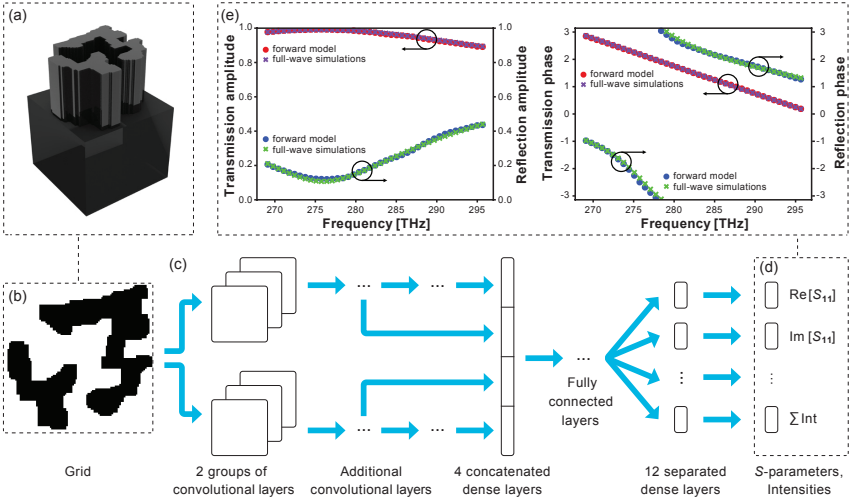


Krizhevsky, Sutskever & Hinton (2012)

But not new!

Patterns detected by convolutional net in 1986 Rumelhart, Hinton & Williams (1986)

Forward model: Unit cell (2D grid) - Optical properties (float table)



Training data set - Input: Random grid creation

Training data set - Input: Random grid creation

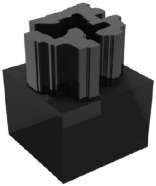
Python algorithm:

- **Binary 2D array** [100*100]

Training data set - Input: Random grid creation

Python algorithm:

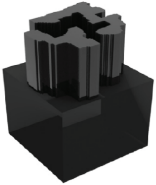
- **Binary 2D array** [100*100]



Training data set - Input: Random grid creation

Python algorithm:

- **Binary 2D array** [100*100]

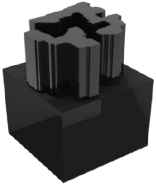


- 0: No material (white)
- 1: Material (black)

Training data set - Input: Random grid creation

Python algorithm:

- **Binary 2D array** [100*100]

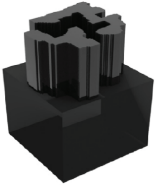


- 0: No material (white)
- 1: Material (black)
- **Free form unit cell**

Training data set - Input: Random grid creation

Python algorithm:

- **Binary 2D array** [100*100]

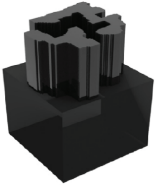


- 0: No material (white)
- 1: Material (black)
- **Free form unit cell**
- **Correct grids** for increasing length scales

Training data set - Input: Random grid creation

Python algorithm:

- **Binary 2D array** [100*100]

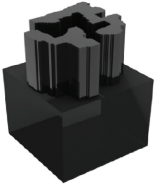


- 0: No material (white)
- 1: Material (black)
- **Free form unit cell**
- **Correct grids** for increasing length scales
  - Fill holes

Training data set - Input: Random grid creation

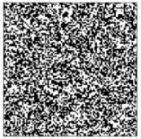
Python algorithm:

- **Binary 2D array** [100*100]



- 0: No material (white)
- 1: Material (black)
- **Free form unit cell**
- **Correct grids** for increasing length scales
  - Fill holes
  - Remove islands

Algorithm visualisation



(a)

Algorithm visualisation

(a)

(b)

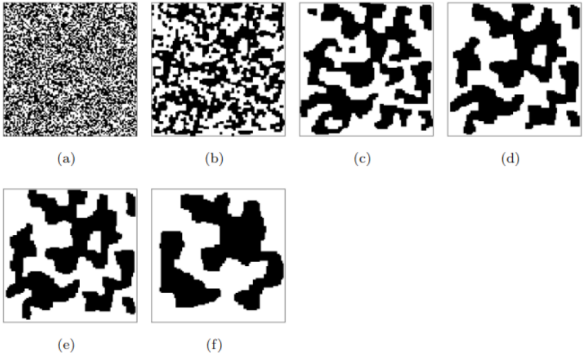
Algorithm visualisation

(a) (b) (c)

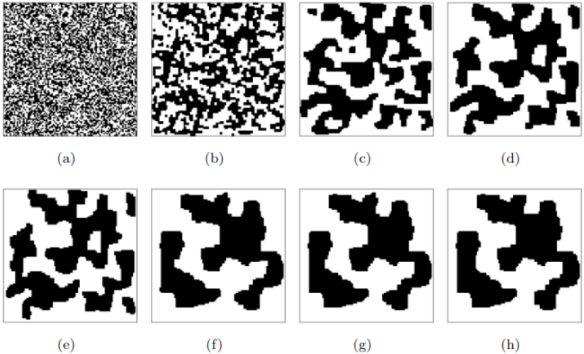
Algorithm visualisation

(a) (b) (c) (d)

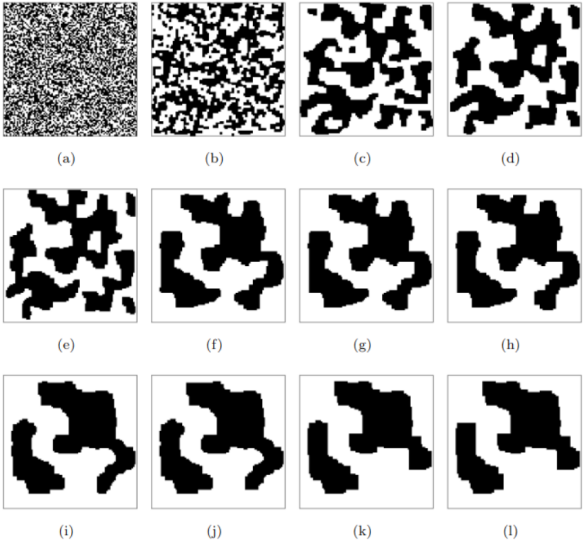
Algorithm visualisation



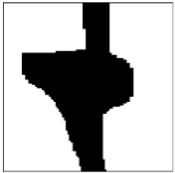
Algorithm visualisation



Algorithm visualisation

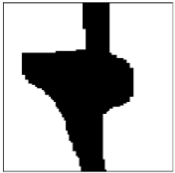


Training data set - Output: Comsol simulation

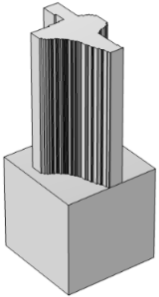


(a)

Training data set - Output: Comsol simulation

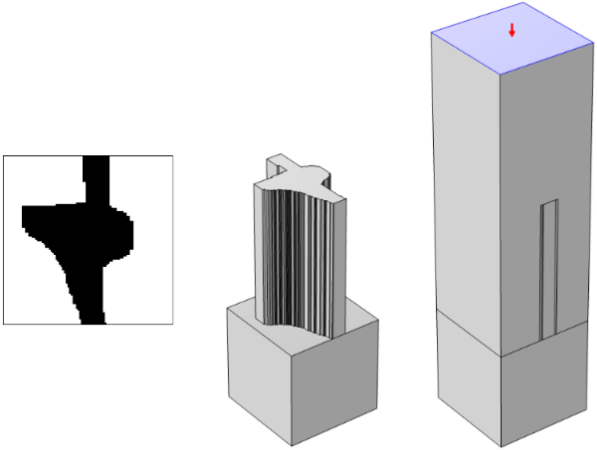


(a)



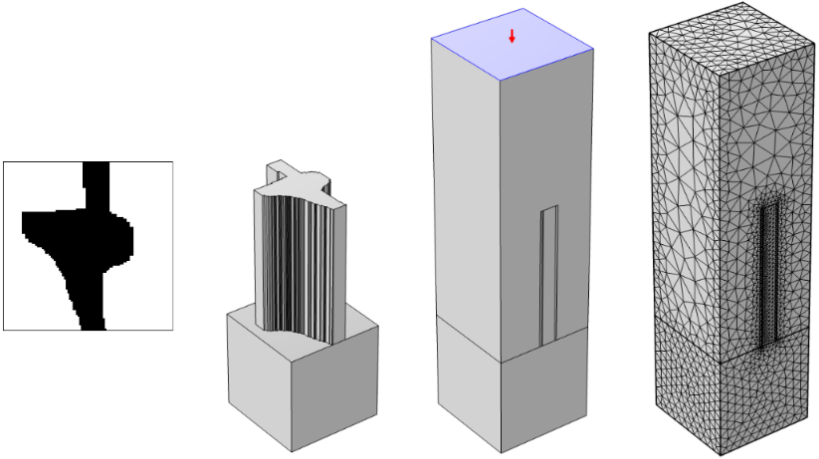
(b)

Training data set - Output: Comsol simulation



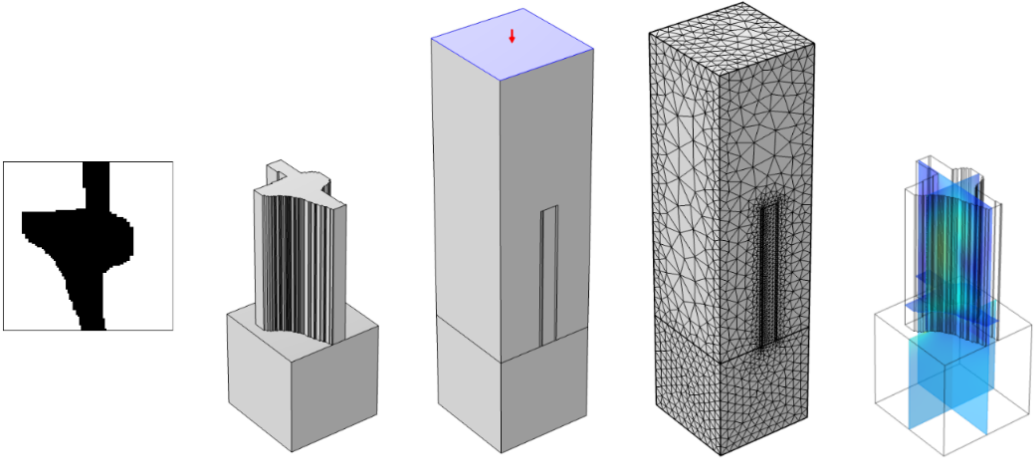
(a) (b) (c)

Training data set - Output: Comsol simulation



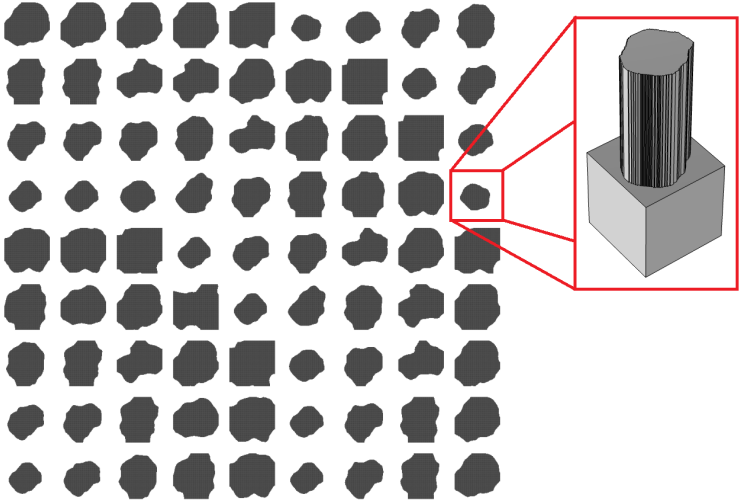
(a) (b) (c) (d)

Training data set - Output: Comsol simulation

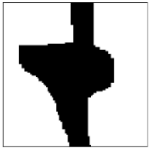


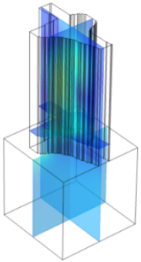
(a) (b) (c) (d) (e)

Training data set - Output: Comsol simulation

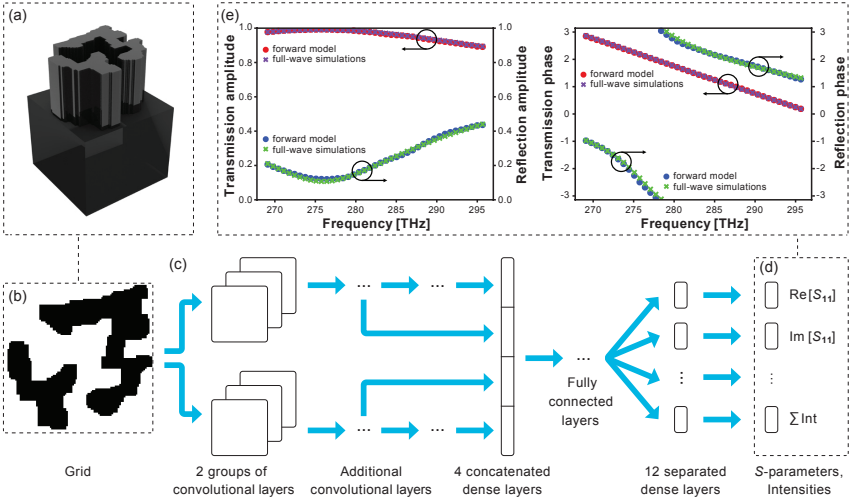


Forward model: Unit cell (2D grid) - Optical properties (float table)





Forward model: Unit cell (2D grid) - Optical properties (float table)



ML techniques

what are useful additional ml learning techniques to improve the performance of the trained network?

To improve the performance of the trained network, you could implement a variety of machine learning techniques. Here are some useful ideas:

1. **Data Augmentation**:
   - Increase the diversity of training data using transformations like random rotations, flips, translations, scaling, and Gaussian noise.
   - In PyTorch, you can use the `torchvision.transforms` library to apply augmentations to the training data.
2. **Batch Normalization**:
   - Add Batch Normalization layers after each convolutional or fully connected layer to stabilize and speed up training. This can help the network generalize better.
   - Example: `nn.BatchNorm2d(num_features)` can be used for 2D convolutional layers.
3. **Dropout Regularization**:
   - Add Dropout layers to reduce overfitting by randomly dropping units during training. Common values for dropout rate are between 0.2 and 0.5.
   - Example: `nn.Dropout(p=0.5)` to add dropout after fully connected layers.
4. **Learning Rate Scheduling**:
   - Use a learning rate scheduler to decrease the learning rate during training, allowing the model to converge more smoothly.
   - In PyTorch, you can use `torch.optim.lr_scheduler.StepLR` or `torch.optim.lr_scheduler.ReduceLR` ↓ `lateau`.
7. **Increase Model Complexity**:
   - Increase the number of convolutional layers or the number of filters in each layer to allow the model to learn more complex features. However, be careful not to overfit.
8. **Gradient Clipping**:
   - Prevent exploding gradients by clipping them during backpropagation. In PyTorch, you can use `torch.nn.utils.clip_grad_norm_()`.
9. **Label Smoothing**:
   - Apply label smoothing to make the model less confident in predictions, which can help with generalization.
10. **Optimizer Tuning**:
    - Experiment with different optimizers like `RMSprop` or `SGD with momentum` instead of Adam, which sometimes performs better depending on the dataset.
    - You could also try tweaking optimizer hyperparameters like learning rate, momentum, or using adaptive learning optimizers like `AdamW`.
11. **Early Stopping**:
    - Monitor the validation loss, and stop training if it doesn't improve for several epochs. This helps to avoid overfitting.
12. **Ensemble Methods**:
    - Train multiple models and average their predictions. This reduces model variance and often yields better performance.
13. **Residual Connections**:
    - If your network is deeper, consider adding residual connections (like in ResNet) to allow gradients to flow more easily and mitigate the vanishing gradient problem.

Inverse design model

Inverse model is **not easy** because:

Inverse design model

Inverse model is **not easy** because:

- **One to many relation**:
  Different metasurfaces can have the same optical properties

Inverse design model

Inverse model is **not easy** because:

- **One to many relation**:
  Different metasurfaces can have the same optical properties
  $\Rightarrow$ **Problem for convergence**

Inverse design model

Inverse model is **not easy** because:

- **One to many relation**:
  Different metasurfaces can have the same optical properties
  $\Rightarrow$ **Problem for convergence**
  Naive inversion of forward network **fails**

Inverse design model

Inverse model is **not easy** because:

- **One to many relation**:
  Different metasurfaces can have the same optical properties
  $\Rightarrow$ **Problem for convergence**
  Naive inversion of forward network **fails**
- **Solutions**:

Inverse design model

Inverse model is **not easy** because:

- **One to many relation**:
  Different metasurfaces can have the same optical properties
  $\Rightarrow$ **Problem for convergence**
  Naive inversion of forward network **fails**
- **Solutions**:
  - Tandem network

Inverse design model

Inverse model is **not easy** because:

- **One to many relation**:
  Different metasurfaces can have the same optical properties
  $\Rightarrow$ **Problem for convergence**
  Naive inversion of forward network **fails**
- **Solutions**:
  - Tandem network
  - Conditional Variational Autoencoders (CVAE)

Inverse design model

Inverse model is **not easy** because:

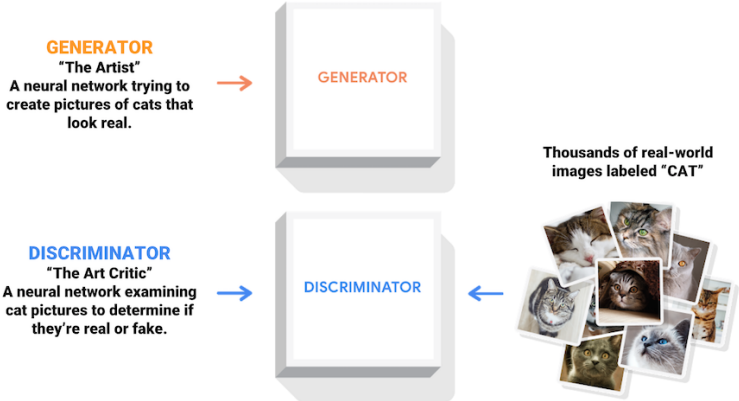
- **One to many relation**:
  Different metasurfaces can have the same optical properties
  $\Rightarrow$ **Problem for convergence**
  Naive inversion of forward network **fails**
- **Solutions**:
  - Tandem network
  - Conditional Variational Autoencoders (CVAE)
  - **Conditional Generative Adversarial Net (CGAN)**

Inverse design model

Inverse model is **not easy** because:

- **One to many relation**:
  Different metasurfaces can have the same optical properties
  $\Rightarrow$ **Problem for convergence**
  Naive inversion of forward network **fails**
- **Solutions**:
  - Tandem network
  - Conditional Variational Autoencoders (CVAE)
  - **Conditional Generative Adversarial Net (CGAN)**
  - Diffusion models

Generative Adversarial Networks (GAN)



https://www.tensorflow.org/tutorials/generative/dcgan

Generative Adversarial Networks (GAN)

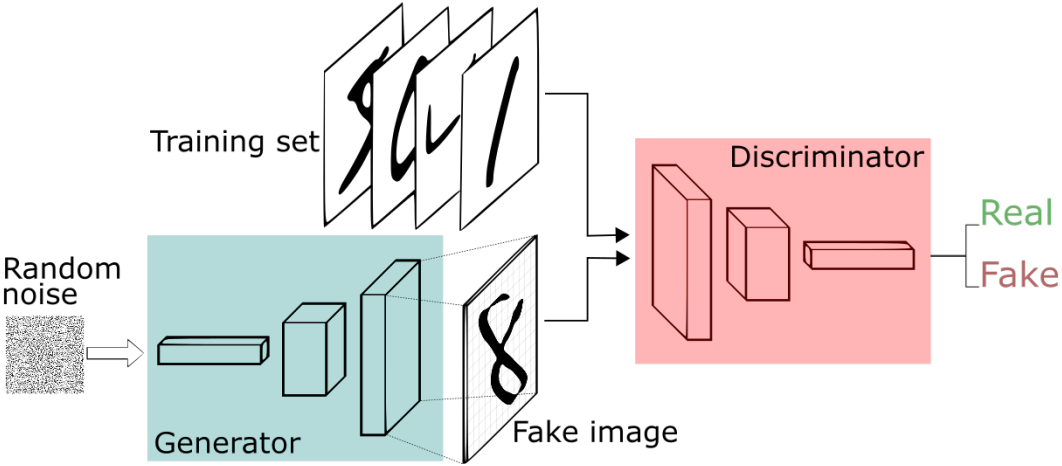
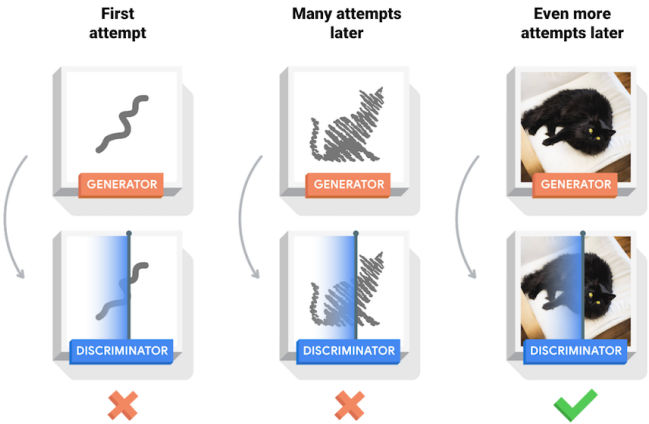


Image credit: Thalles Silva

Generative Adversarial Networks (GAN)

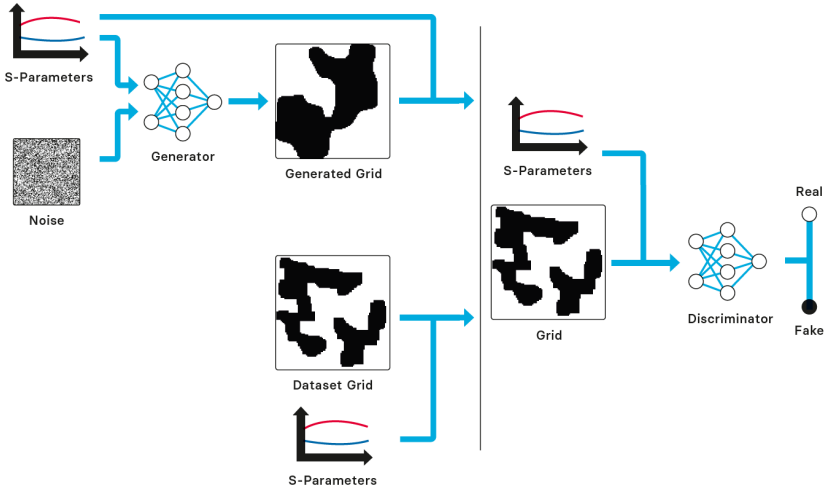


https://www.tensorflow.org/tutorials/generative/dcgan

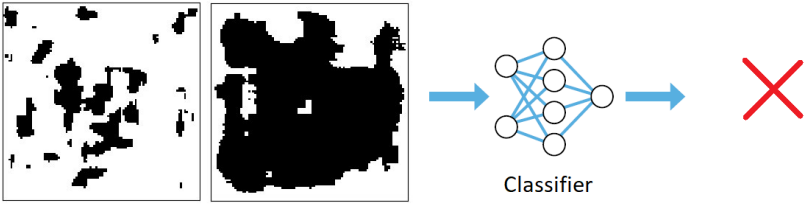
Style-GAN

Team at NVIDIA, "A Style-Based Generator Architecture for Generative Adversarial Networks," 2019

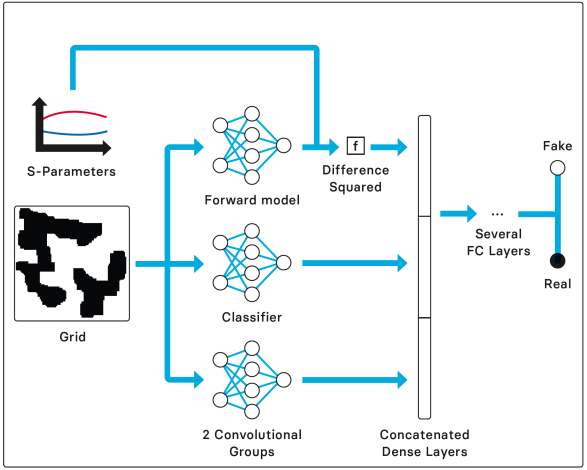
Conditional-GAN (CGAN)



Experimental feasibility classifier

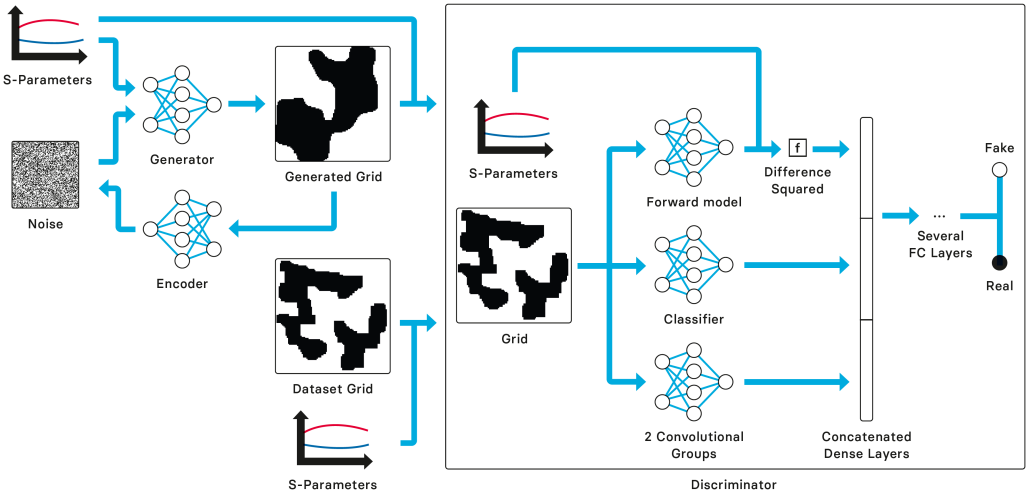


Advisor-Discriminator

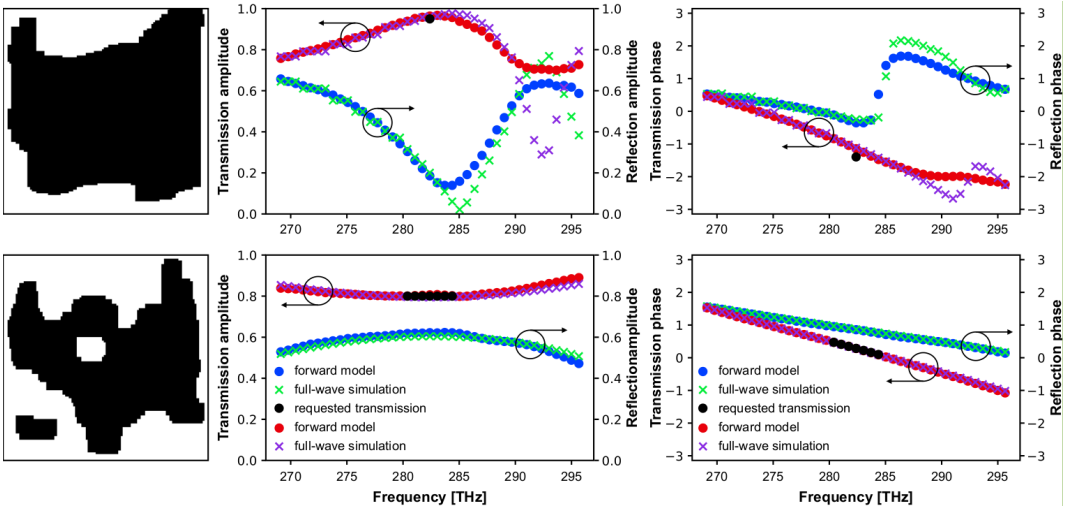


Discriminator

Full Conditional Generative Adversarial Network

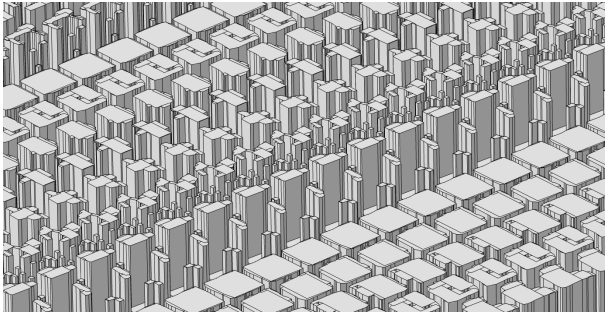


Generator output

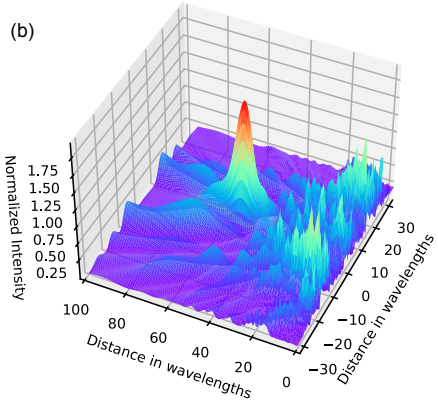


Cylindrical lens

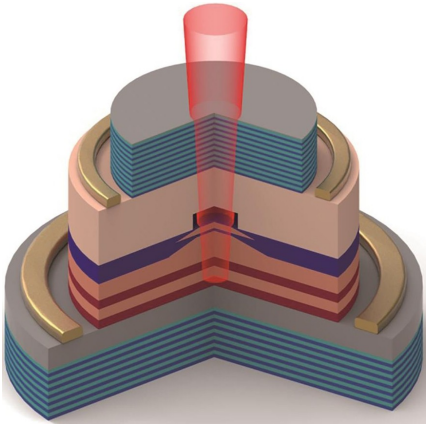
(a)

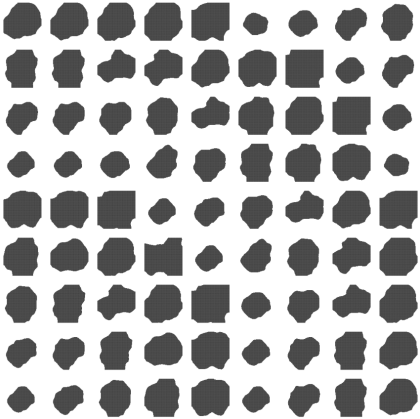


(b)



Polarization converting lens on a VCSEL





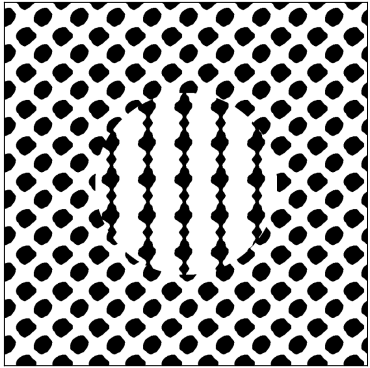
(left) Schematic VCSEL, and part of etching mask (right)

Courtesy of Erik Strandberg

Phase mask for a ZWFS in a telescope

Interdependent optical properties of the two regions

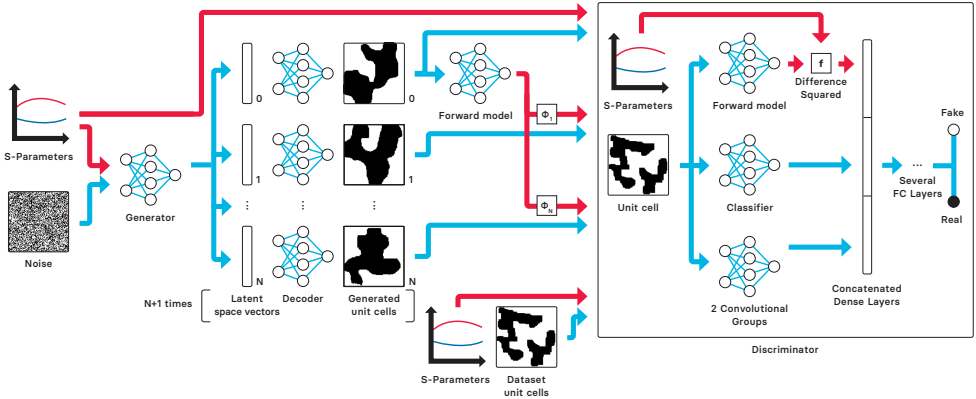
2D cross section of ZWFS metasurface



Schematic: meta-atoms not to scale

Maximize the **signal** to noise ratio in **telescopes**

CGAN interdependent properties

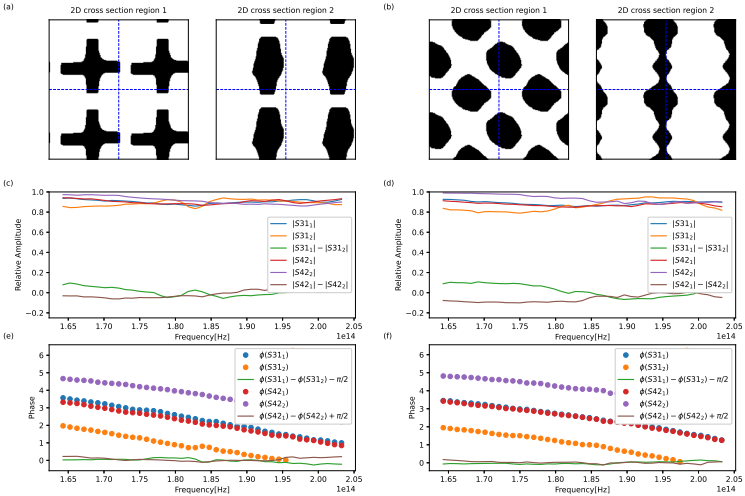


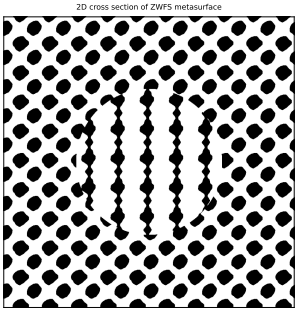
CGAN graph for unit cells with **interdependent $\mathcal{S}$ parameters**.

Red arrows = $\mathcal{S}$ parameter, blue arrows = rest of the graph.

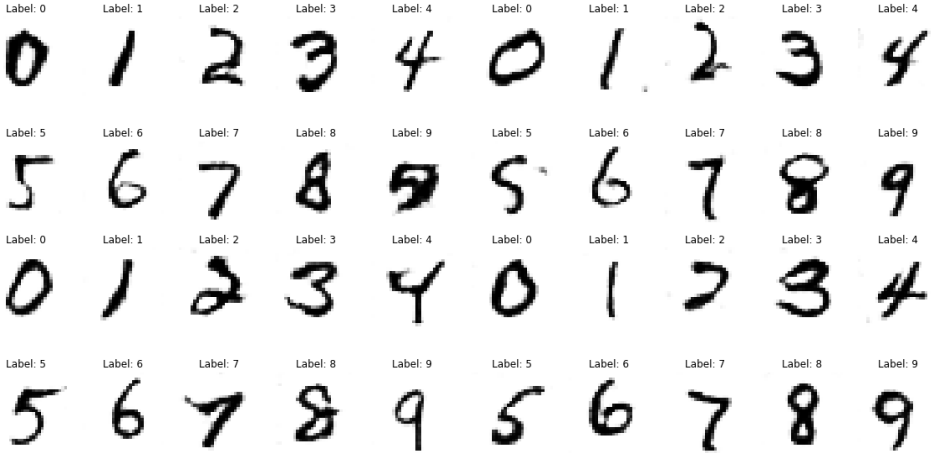
Courtesy of Jonas Rose

Phase mask for a ZWFS - Free-form design





Mnist CGAN


Label: 0
Label: 1
Label: 2
Label: 3
Label: 4
Label: 0
Label: 1
Label: 2
Label: 3
Label: 4
Label: 5
Label: 6
Label: 7
Label: 8
Label: 9
Label: 5
Label: 6
Label: 7
Label: 8
Label: 9
Label: 0
Label: 1
Label: 2
Label: 3
Label: 4
Label: 0
Label: 1
Label: 2
Label: 3
Label: 4
Label: 5
Label: 6
Label: 7
Label: 8
Label: 9
Label: 5
Label: 6
Label: 7
Label: 8
Label: 9

Thank you for your attention!