Light scattering behavior to metasurfaces by diffusion model

Hay Lahav
ID: 308279918
Email: haylahav1@gmail.com

Nadav Marciano
ID: 305165698
Email: nadavmarci@gmail.com

October 2, 2023

Abstract

In this project, we tackle the problem of creating a deep learning generative diffusion model capable of discerning and generating optimal meta-surface configurations based on the observed light-scattering behaviors. The training of our model relies on a dataset that encompasses metasurface samples and their corresponding light-scattering behaviors. Our approach to solving this problem is to use a *conditioned diffusion model* [1] based on 2dUnet from Hugging Face and adjust it according to our data. In addition, we had to adjust and improve the model architecture in order to optimize its performance. As part of our course project, we have obtained a diverse dataset containing 25000 meta surface patterns and their corresponding light scattering to advance our research. Following model training and testing, we successfully obtained the predicted metasurface configuration corresponding to a given matrix of light properties.

1. Introduction

This project embarks on the challenge of harnessing the potential of diffusion models to navigate the complex landscape of meta-surface design. Our goal is to fit each light properties matrix into a suitable metasurface pattern by diffusion model. In the present era, metasurfaces offer precise manipulation of light properties like phase, amplitude, and wavefront [2]. The objective, when provided with a dataset comprising metasurfaces and their interactions with light, is to train a diffusion model capable of generating a suitable meta-surface for a given measurement of light**.** A diffusion model is a parametrized Markov chain trained using variational inference techniques to generate samples that closely resemble the original data within a finite timeframe [3]. In essence, transitions of this chain are learned to reverse a diffusion process, which is a Markov chain that gradually adds noise to the data in the opposite direction of sampling, ultimately obscuring an underlying signal.

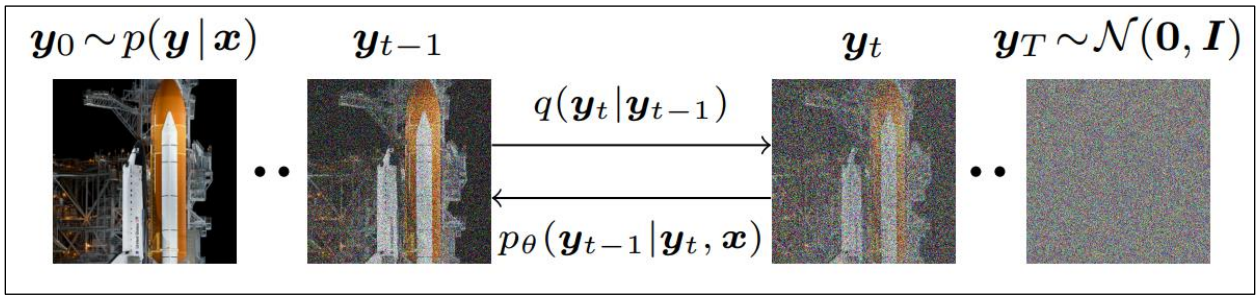

Figure 1: *The directed graphical model* [3].

This subject is important because diffusion models can be used in learning useful feature representations for scientific measurements and to generate a suitable output according to those measurements and the user demands. This area of study holds significant allure due to the versatile applications of diffusion models. They can serve various purposes, including generating images based on prompts or acquiring valuable feature representations. By adjusting the DDPM [5] to conditional image generation and using the Unet architecture from Hugging Face, we obtained after 50 epochs of training a satisfying output of a potential meta surface pattern given the data of the light matrices and their corresponding metasurface pattern.

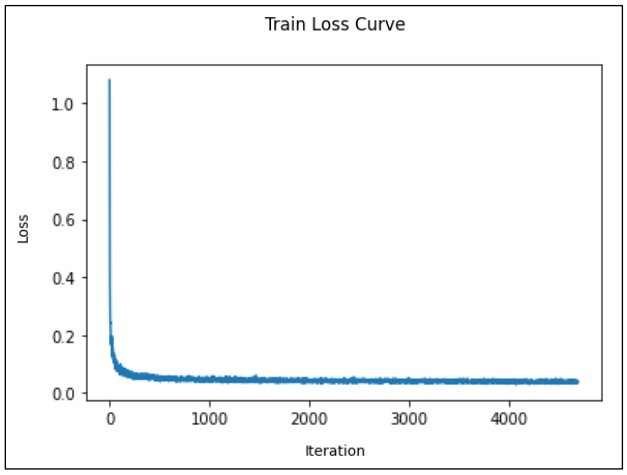

***Figure 2**: The training loss graph (batch =20, lr= 4e-3, MSELoss)*

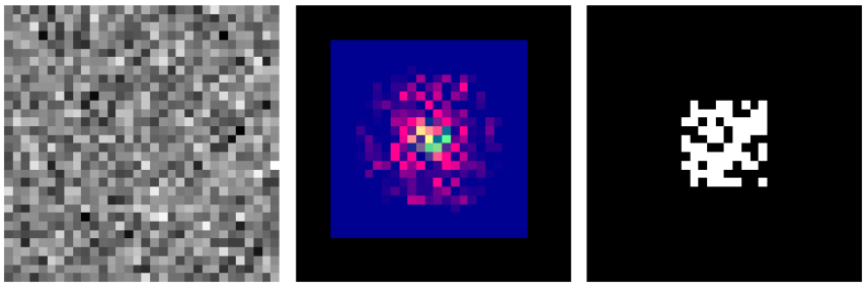

***Figure 3:** from right to left: model prediction, condition matrix, random noise matrix.*

Following the evaluation of our model on the test dataset, which included varying levels of noise added to the ground truth pattern, our findings indicate that the model accurately predicted only 12.4% of the patterns. Notably, the majority of accurate predictions occurred when the noise level was relatively low. However, in scenarios involving random noise or heavily corrupted patterns, the model exhibited limitations, generating new patterns that is not similar from the ground truth. Despite the fact that the model's predictions may not precisely match the ground truth, it is noteworthy that these predictions can still align with the light properties matrix observed in the laboratory. In certain situations, there may not be a singular solution, and the model's outputs can exhibit a meaningful correlation with the actual experimental conditions.

	Type	Count
0	Correct Predictions	248
1	Incorrect Predictions	1752
2	Overall Image-Level Accuracy	12.40%
3	Average Precision for Individual Images	0.7345
4	Average Recall for Individual Images	0.7177
5	Average unequal_pixels for Individual Images	30.4709

Overall Image-Level Accuracy: 12.40%

***Figure 4:** Model evaluation results.*

2. Related Work

There are no existing an identical implementation for our exact project, namely, fitting a meta-surface pattern in accordance with their corresponding light-scattering behaviors using diffusion models.

However, an alternative research approach, referred to as "RFDiffusion" [9] aligns with the idea of generating diverse protein structures based on a conditional input of amino acid sequences (text input). Unlike the previously mentioned method, RFDiffusion utilizes a diffusion model grounded in a pre-trained "RoseTTAFold" deep learning neural network. Achieving reliable results with this approach demands a significant time investment, often spanning several days, and requires multiple processing units. In our specific case, we lack a pre-trained model and the necessary hardware resources to tailor this method to meet our project requirements.

We had to develop a custom solution for this specific problem. Our search of relevant papers revealed that diffusion models are commonly used for segmentation and text-to-image generation tasks, such as Stable Diffusion model [4]. On a different note, we've come across classification papers employing the Generative Adversarial Network (GAN) model to address a comparable task of generating an artificial output closely resembling the ground truth target.

The Unit architecture and the noise scheduler for the diffusion model were taken from the Diffusers library of Hugging Face [1]. In the Hugging Face website, there is an example illustrating the construction of a conditional diffusion model using the MNIST dataset. This model is designed to regenerate images with similar figures as the conditional vector. The conditional vector is embedded and concatenated to the input data, ensuring that the denoised images portray the same number as the input images but exhibit different styles of handwriting.

Within our project, the objective is to create an appropriate pattern through the utilization of a conditional light properties matrix (image). This light matrix is concatenated with a suitable noised pattern to train the model on generating a suitable pattern corresponding to the given conditional light properties matrix. It is crucial to note that there may exist multiple correct patterns associated with each matrix, and their accuracy can be validated through measurements in the laboratory.

3. Data

The data we are working with is light scattering from silicon surfaces. The material being studied is silicone, and the wavelength used is 1550nm. Prof. Scheuer's nano photonics lab at Tel Aviv University provided the dataset, containing twenty-five thousand samples. Each sample includes the following data:

- xx_data: The material template is represented as a binary vector with a size of 100, arranged in a 10 x 10 matrix.
- h: The height of the template, which varies for each sample.
- Rm and Tm: Matrices representing the transmission and reflection of light, with vector lengths of 529, arranged in a 23 x 23 matrix.

The data was initially received in the form of MATLAB files. We loaded this dataset and converted it into a DataFrame, ensuring that each column contained the following information

for each sample: xx_data, h, Rm, and Tm. To match the UNET model architecture, we created two 32 x 32 matrices for each sample.

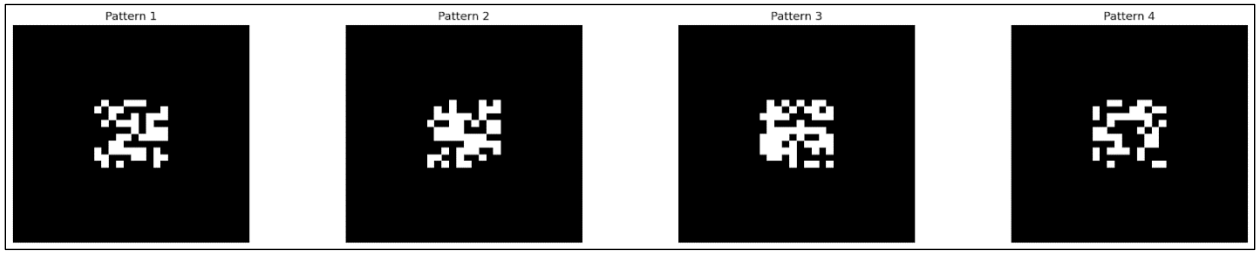

Figure 5: *An example of the metasurface patterns of the first four samples.*

The pattern is represented as a binary matrix with a single channel. To maintain a uniform size of 32 x 32, zeros are padded around the pattern:
The light matrix is a matrix with three channels:

Channel 1: Rm Matrix.
Channel 2: Tm matrix.
Channel 3: The height of the pattern. we normalized and padded the matrix with zeros to achieve a consistent size of 32 x 32.

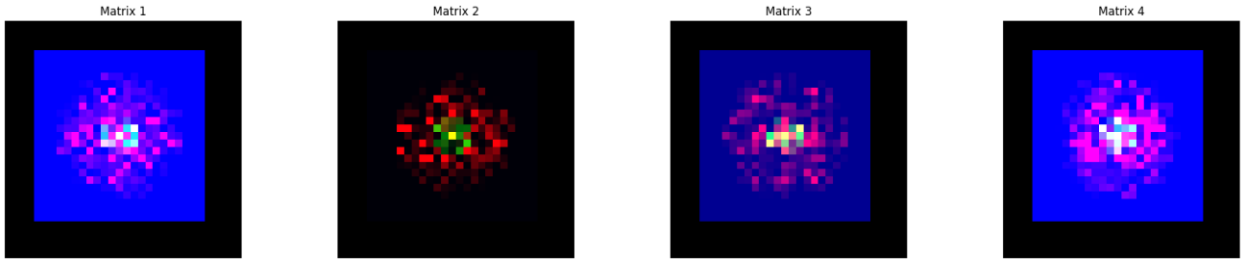

Figure 6: *An example of the light matrices after normalization and padding of the first four sample.*

We stored the data for each sample in a DataFrame, ensuring that each column contained the following information: the light matrix and xx_data (the pattern matrix). For further analysis, these matrices for each sample were transformed into tensors, and the dataset was divided into a training set consisting of 23 thousand samples and a test set consisting of 2 thousand samples.

4. Method

Diffusion models are a type of latent variable model, denoted as $p_\theta(y_0)$, defined through the integral of $p_\theta(y_{0:T})$ over $y_1, ..., y_T$, the latent variables of the same dimensionality as the data $y_0 \sim p(y_0)$. What sets diffusion models apart from other latent variable models is that the approximate posterior $q(y_{1:T}|y_0)$, known as the ***'forward process'*** is constrained to a Markov chain that gradually introduces Gaussian noise to the data following a predefined variance schedule $\beta_1, ..., \beta_T$. This β is defined for every time step according to some chosen schedule, and determines how much noise is added per timestep.:

$$q(y_T|y_0) \equiv \prod_{t=1}^{T} q(y_t|y_{t-1}), \qquad (1)$$

Where:

$$q(y_t|y_{t-1}) \equiv N(y_t; \sqrt{1-\beta_t}y_{t-1}, \beta_t I).$$

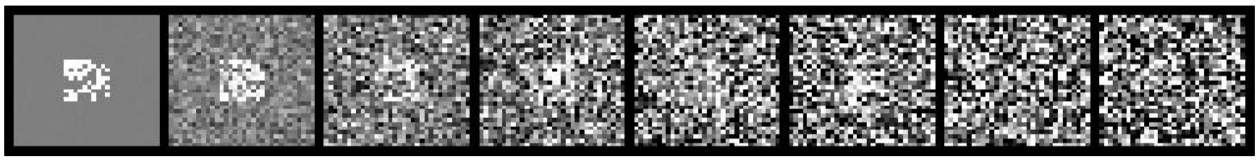

Figure 7: *The forward diffusion process q (left to right) gradually adds Gaussian noise to the target image.*

The joint distribution $p(y_0{:}T)$ is referred to as the ***'reverse process'*** and is defined as a Markov chain with learned Gaussian transitions, starting with $p(y_t) = N(y_T; 0, I)$. In other words, Inference in our model involves a reverse process that operates in the opposite direction of the initial diffusion process. This reverse process starts from a Gaussian noise $\boldsymbol{y}_T$ and aims to recover the original noise-free image y_0.

$$p(y_0{:}T) \equiv p(y_T)\prod_{t-1}^{T} p_\theta(y_{t-1}|y_t), \qquad (2)$$

Where:

$$p_\theta(y_{t-1}|y_t) \equiv N(y_t; \mu_\theta(y_t, t), \Sigma_\theta(y_t, t))$$

$$and$$

$$p_\theta(y_{t-1}|y_t, x_0) = \mathcal{N}(y_{t-1}; \tilde{\mu}_t(x_t, x_0), \tilde{\tau}_t I).$$

4.1 Optimizing

The training process involves optimizing the standard variational bound on the negative log likelihood:

$$E[-logp_\theta(y_0)] \leq E_q\left[-log\left(\frac{p_\theta(y_0:T)}{q(y_1:T|y_0)}\right)\right] = E_q\left[-logp(y_T) - \Sigma_{t\geq 1} log\left(\frac{p_\theta(y_{t-1}|y_t)}{q(y_t|y_{t-1})}\right)\right] = L \qquad (3)$$

The forward process variances β_t can be either learned through reparameterization or set as hyperparameters. The expressive power of the reverse process is ensured, in part, by the choice of Gaussian conditionals in $p_\theta(y_{t-1}|y_t)$, especially when β_t values are small. A notable feature of the forward process is that it allows for the closed-form sampling of y_t at any timestep t, as equation (1) can be expressed by the following expression:

$$q(y_t|y_0) = N(y_t; \sqrt{\bar{\alpha}_t}y_0 + (1 - \bar{\alpha}_t)I) \qquad (4)$$

Where:

$$\alpha_t \equiv 1 - \beta_t \quad and \quad \bar{\alpha}_t \equiv \prod_{s-1}^{t} \alpha_s \qquad \tilde{y} = \sqrt{\bar{\alpha}_t}y_0, (1 - \bar{\alpha}_t)$$

Efficient training is achievable by optimizing random terms of L (3) using stochastic gradient descent. Further enhancements can be made by rewriting the negative log likelihood L in a form that leverages the Kullback-Leibler (KL) divergence:

$$\mathbb{E}_k \left[D_{KL(q(y_T|y_0)||p(y_T)} \right) + \sum_{t\geq 1} D_K L\left(q(y_{t-1}|y_t, y_0) \middle|\middle| p_\theta(y_{t-1}|y_t)\right) - log\ p_\theta(y_0|y_1)\right] \qquad (5)$$

In Equation, KL divergence is used to directly compare $p_\theta\ (y_{t-1} \mid y_t)$ against forward process posteriors, which become manageable when conditioned on x_0:

$$q(y_{t-1}|y_t, x_0) = \mathcal{N}(y_{t-1}; \tilde{\mu}_t(x_t, x_0), \tilde{\tau}_t I). \qquad (6)$$

Where:

$$\tilde{\mu}_t(y_t, y_0) \coloneqq \left(\frac{\sqrt{\alpha_{t-1}}\beta_t}{(1-\bar{\alpha}_t)}\right) y_0 + \left(\frac{\sqrt{\alpha_t}(1-\bar{\alpha}_{t-1})}{(1-\bar{\alpha}_t)}\right) y_t$$

$$and$$

$$\tilde{\tau}_t \coloneqq \frac{(1-\bar{\alpha}_{t-1})}{(1-\bar{\alpha}_t)}\beta_t .$$

As a result, all KL divergences in Equation (5) are comparisons between Gaussian distributions, allowing for their efficient calculation using closed-form expressions [3].

Our model training algorithm – train the model to estimate the noise step :

Algorithm 1 - Training a denoising model f_θ

1: **repeat**
2: $(\boldsymbol{x}, \boldsymbol{y}_0) \sim p(\boldsymbol{x}, \boldsymbol{y})$
3: $\bar{\alpha}_t \sim p(\bar{\alpha}_t)$
4: $\boldsymbol{\epsilon} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$
5: *Take a gradient descent step on*

$$\nabla\theta \left\| f_\theta(\boldsymbol{x}, \sqrt{\bar{\alpha}_t}\boldsymbol{y}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}, \bar{\alpha}_t) - \boldsymbol{\epsilon} \right\|_p^p$$

6: ***until*** *converged*

Inference via Iterative Refinement

Our sampling algorithm model is defined according to the following iterative algorithm:

Algorithm 2- Inference in T iterative refinement steps

1: $\boldsymbol{y}_T \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$
2: ***for*** $t = T, \ldots, 1$ ***do***
3: $\boldsymbol{z} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$ *if* $t > 1$, *else* $\boldsymbol{z} = \boldsymbol{0}$
4: $\boldsymbol{y}_{t-1} = \frac{1}{\sqrt{\alpha_t}}\left(\boldsymbol{y}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}} f_\theta(\boldsymbol{x}, \boldsymbol{y}_t, \bar{\alpha}_t)\right) + \sqrt{1-\alpha_t}\boldsymbol{z}$
5: ***end for***
6: ***return*** $\boldsymbol{y}_0$

This inference process is defined by using Gaussian conditional distributions, which provide insight into how the image changes at each step as noise is removed. Furthermore, by using Hugging face Unet2D conditioned diffusion model, we leverage our denoising model $\boldsymbol{f_\theta}$ to aid in reversing the diffusion process. This model takes a light properties image as a condition and concatenates it to a noisy pattern image $\boldsymbol{y}_t$ as input and attempts to recover the noise-free target image y_0 by predicting the noise $\boldsymbol{\epsilon}$ to reduce in each time step t.

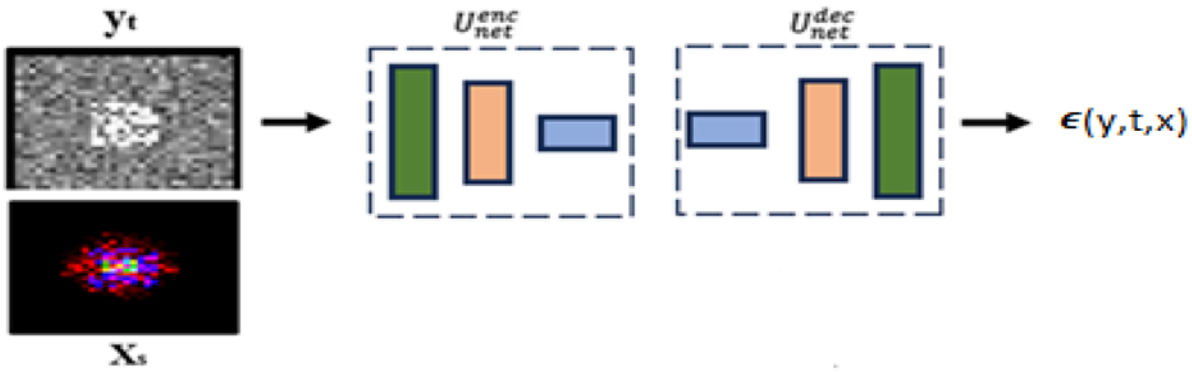

Figure 8: *Unet Model Architecture [8]. (y_t- noised pattern, X_S- light matrix (concatenated to y_t), $\boldsymbol{\epsilon}$ – predicted noise)*

The reverse process starts from the noisy image $\boldsymbol{y}_T$ and gradually denoises it until it reaches the initial image $\boldsymbol{y}_0$. The denoising model f_θ is trained to estimate the noise ϵ given the noisy image and the hyperparameters $\bar{\alpha}_t$and α_t. The hyperparameter $\bar{\alpha}_t$controls the amount of noise that is removed at each step of the reverse process. A higher value of $\bar{\alpha}_t$ signifies the removal of more noise, but it concurrently increases the risk of overfitting. This implies that the model may struggle when confronted with unfamiliar situations that fall outside its training, hence, we will obtain poor diversity of generated output images. The hyperparameter αt controls the variance of the Gaussian conditional distribution $p\theta(yt-1 \mid yt, x)$. A higher value of α_t means that the distribution is more concentrated around the mean, but it also makes the model more sensitive to the noise in the data. If the noise variance of the forward process steps is set as small as possible, i.e., $\alpha_{1:T} \approx 1$, the optimal reverse process $p(\boldsymbol{y}_{t-1} \mid \boldsymbol{y}_t, \boldsymbol{x})$ will be approximately Gaussian. Accordingly, the choice of Gaussian conditionals in the inference process can provide a reasonable fit to the true reverse process. Meanwhile, $1-\bar{\alpha}_t$ should be large enough so that $\boldsymbol{y}_T$ is approximately distributed according to the prior $p(\boldsymbol{y}_T) = \mathcal{N}(\boldsymbol{y}_T \mid \boldsymbol{0}, \boldsymbol{I})$, allowing the sampling process to start at pure Gaussian noise. The denoising model f_θ is trained to estimate $\boldsymbol{\epsilon}$, given any noisy image $\tilde{\boldsymbol{y}}$ including $\boldsymbol{y}_t$. Thus, given $\boldsymbol{y}_t$, we approximate y_0 by rearranging the terms in $\tilde{\boldsymbol{y}}$ (4) as

$$\hat{\boldsymbol{y}}_0 = \frac{1}{\sqrt{\bar{\alpha}_t}}\left(\boldsymbol{y}_t - \sqrt{1-\bar{\alpha}_t} f_\theta(\boldsymbol{x}, \boldsymbol{y}_t, \bar{\alpha}_t)\right).$$

We substitute our estimate $\hat{\boldsymbol{y}}_0$ into the posterior distribution of $q(\boldsymbol{y}_{t-1} \mid \boldsymbol{y}_0, \boldsymbol{y}_t)$ in (6) to parameterize the mean of $p_\theta(\boldsymbol{y}_{t-1} \mid \boldsymbol{y}_t, \boldsymbol{x})$ as

$$\mu_\theta(\boldsymbol{x}, \boldsymbol{y}_t, \bar{\alpha}_t) = \frac{1}{\sqrt{\alpha_t}}\left(\boldsymbol{y}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}} f_\theta(\boldsymbol{x}, \boldsymbol{y}_t, \bar{\alpha}_t)\right),$$

and we set the variance of $p_\theta(\boldsymbol{y}_{t-1} \mid \boldsymbol{y}_t, \boldsymbol{x})$ to $(1-\alpha_t)$, a default given by the variance of the forward process.

4.2 The strengths of our approach

After reviewing the relevant academic papers and literature, we found that the main alternative to our approach is generative adversarial networks (GANs). We assume that our approach is the right thing to do due to the following advantages [7] of diffusion models over GANs:

- More stable training: Diffusion models are generally easier to train than GANs. This is because diffusion models do not require an adversarial training process, which can be unstable and difficult to converge.
- Less prone to mode collapse: Mode collapse is a problem that can occur in GANs, where the model learns to generate only a small number of output images. Diffusion models are less prone to mode collapse because they do not rely on an adversarial training process.
- Better diversity of output images: Diffusion models can generate more diverse images than GANs. This is because diffusion models start with a high-entropy latent representation and gradually add noise to it, which allows the model to explore a wider range of possible images.

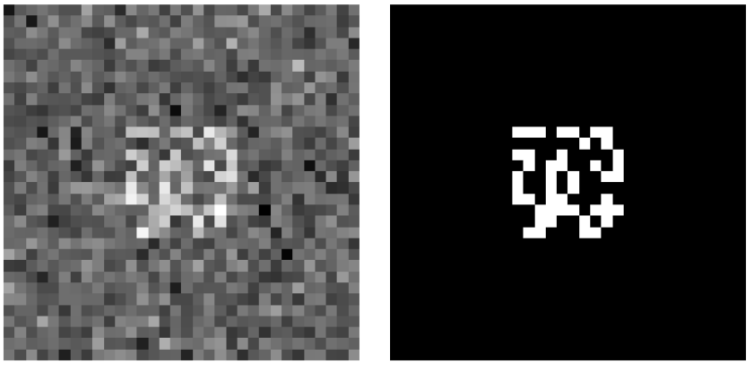

Figure 9: *Pattern prediction after 10 epochs*

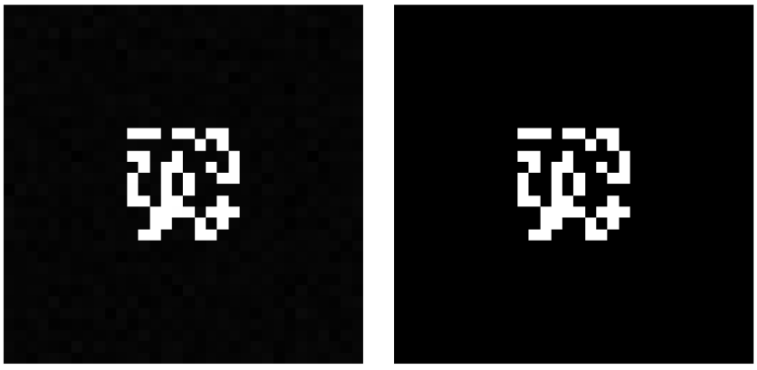

Figure 10: *Pattern prediction after 50 epochs*

5. Experiments

5.1 Validation of Problem-Solving Approaches

In our experiments, we chose to use Hugging Face Condition Unet Architecture to fit a suitable meta surface pattern light matrix to its light properties matrix. We tried to solve this problem with different approaches and inputs:

1. The Unet model received a 32x32x1 pattern as its primary input. Simultaneously, we introduced a 32x32x3 light matrix as a class conditional input, encompassing transmitted light, reflected light, and the height.

2. In the second approach, we fed a 32x32x4 noised pattern concatenated with a light matrix into the Unet model. Additionally, the metasurface height was normalized according to the maximal and minimal value of the other parameters.

3. In an alternative approach, we fed a 32x32x4 noised pattern concatenated with a light matrix into the Unet model. Additionally, the metasurface height was normalized according to the maximal and minimal value of the other parameters.

After a thorough evaluation, we chose the third approach. The second approach was less accurate after 50 epochs compared to the first, and the first approach doesn't solve the case when an unfamiliar light properties matrix will be inserted to the model as a condition..

5.2 Results Visualization

To enhance the clarity of our experimental findings, we have incorporated graphs, tables, and visualizations throughout the following sections. These visual aids are intended to offer a comprehensive understanding of the model's performance. Our initial analysis executed the model with varying batch sizes of 16, 20, 24, and 32 across 50 epochs.
Each run of the model corresponding to one of the batch sizes, encompassed the application of three distinct loss functions: MSELoss, L1 Loss, and HuberLoss. by the following table

batch size	16			20			24			32		
Loss Function	MSELoss()	L1Loss()	HuberLoss()	MSELoss()	L1Loss()	HuberLoss()	MSELoss()	L1Loss()	HuberLoss()	MSELoss()	L1Loss()	HuberLoss()
Learning Rate												
0.0100	0.046664	0.053940	0.023332	0.047459	0.051601	0.023732	0.046662	0.050402	0.023332	0.048438	0.052113	0.024217
0.2000	0.047020	0.113538	0.023508	0.047689	0.062936	0.023836	0.045205	0.059244	0.023331	0.048520	0.048239	0.024259
0.0003	0.050000	0.030300	0.002000	0.011000	0.070500	0.003000	0.011000	0.046300	0.007000	0.050000	0.054900	0.002000
0.0040	0.036000	0.031949	0.002800	0.060000	0.057990	0.001300	0.074000	0.036567	0.002700	0.018100	0.028004	0.006300

Table 1: *learning rate loss function for each batch*

we can notice that the best accuracy is found under the Huber loss column with leaning rate $3e^{-04}$ and batch 20. After conducting model testing, we observed that the predictions generated by the model using the Huber Loss exhibit were noisy compared to those obtained with MSE loss, despite achieving a lower training loss with the Huber Loss and utilizing the same set of parameters. We assume it happened due to the overfitting of the training model.

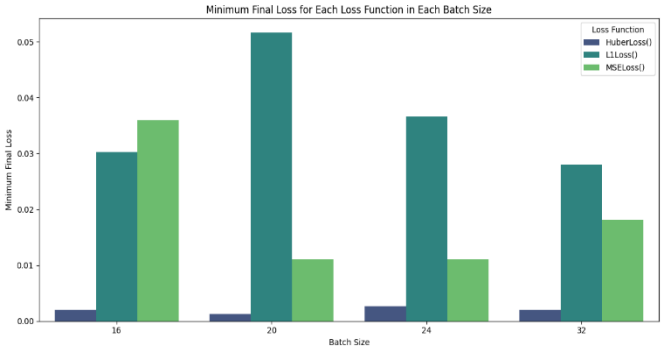

Table 2: *Minimum Final Loss for Each Loss Function in Each Batch Size*

By Table 1, we notice that the optimal batches for our model are batches 20 and 24 because these contains the lowest MSE losses over each one of the loss batch sizes. Although there is no

directly comparable classification project for benchmarking our results, we can compare its performance in comparation with the performance of the same model architecture on an image generation with MNIST database on the Hugging Face website.

5.3 Addressing Failure Modes in Our Model

- Overfitting: By using the Huber Loss function during the training of the model, we obtained the lowest loss values with a required predicted pattern. However, after testing the model, the predicted pattern was relatively blurred in comparison to the MSE Loss function. We assume that the reason for the blurred output is due to overfitting which is a common problem in machine learning and statistical modeling that occurs when a model learns the training data too well.

- Limited Denoising Performance under Heavy Noise: Our observations reveal a relative decline in denoising efficacy, especially when the input is heavily noised but not entirely corrupted. It's known that when the input noise exceeds a certain threshold, the model's predictions may exhibit reduced accuracy due to potential overfitting issues. However, our anticipation was that the model's output would still closely resemble the ground truth even in such challenging scenarios. In comparison to analogous research projects, we acknowledge that enhancing performance without modifying the model requires an extended training duration with a substantially larger dataset. This approach is likely to yield improved results by allowing the model to better generalize and adapt to diverse noise distributions.

- Inadequate Data Preparation: When we inserted the input data into the model, we observed that the height values were too high relative to the transmitted light, and reflected light values thus the representation of them in the light matrix was barely noticeable and had negligible influence in the training model. We solved this problem by normalizing the height parameter.

- Evaluation: Throughout the project, we gained insights into the model's behavior when exposed to noisy inputs. It became evident that introducing noise could lead to the generation of diverse patterns that might not precisely align with the ground truth. Recognizing the need for optimization, we realized that regularly testing the model outputs in the laboratory and adjusting the model based on the specific requirements of researchers is essential. This iterative process allows to prioritize practical applicability over strict adherence to a predefined ground truth pattern.

6. Conclusions

Through experimentation, we determined that setting the parameters as follows - a learning rate of $3e^{-04}$, a batch size of 20, and employing the Mean Squared Error (MSE) Loss function, The model yielded variant predictions of patterns for the given light properties matrix after approximately 50 training epochs. The reliability of the generated patterns for matrix outside the training dataset can be verified solely by examination in the laboratory.

6.1 Observations

After the training and testing of our diffusion model, we learned the following:

Leveraging Pre-built Models: Throughout the project, we transitioned from the practice of constructing models from the ground up to utilizing pre-existing models from Hugging Face. This shift in approach allowed us to streamline our development process and leverage state-of-the-art resources effectively.

Conditional Input in Diffusion Models: An intriguing aspect of our project involved utilizing a conditional input, represented by the light matrix in our case, to generate a suitable pattern from the noisy input according to conditional light properties matrix. This approach was new to us as, it seems that diffusion models are primarily associated with tasks like segmentation, restoration, and high-definition image generation, rather than fitting tasks. This experience expanded our understanding of the versatility of diffusion models and their applicability in diverse problem domains.

6.2 Ideas for future extensions or new applications

- In this project we used the Google Colab and cloud service to process the diffusion model. A possible future extension can be deploying the model on hardware accelerators to ensure fast and efficient predictions.

- Interactive User Interface:
  We suggest developing an interactive user interface such as a smartphone application that allows users to insert different parameters and visualize the predicted metasurface patterns in real time.

- Fine-tuning the model parameters to achieve even greater optimization and identify a more efficient optimizer, aiming to accelerate the training process significantly and improving the accuracy of the metasurface pattern prediction.

- Multi-Modal Learning:
  Exploring the integration of multiple modalities into the diffusion model to have more specific results. For example, incorporate additional types of data such as thermal or infrared imaging to enhance the model's ability to predict metasurface patterns under various conditions.

- Visit the laboratory - verify the results more frequently in the lab and consult with the experts in the field.

7. Appendix

[1] Hugging Face Diffusion Models Course: https://github.com/huggingface/diffusion-models-class/tree/main

[2] *Mahmoud M. R. Elsawy, Stéphane Lanteri, Régis Duvigneau, Jonathan A. Fan, and Patrice Genevet.* **Numerical Optimization Methods for Metasurfaces: Laser Photonics,** Rev. 2020, 14, 1900445.

[3] *Chitwan Saharia†, Jonathan Ho, William Chan, Tim Salimans, David J. Fleet, Mohammad Norouzi.* **Image Super-Resolution via Iterative Refinement**, arXiv:2104.07636v2 30 Jun 2021.

[4] *Florinel-Alin Croitoru, Vlad Hondru, Radu Tudor Ionescu, and Mubarak Shah.* **Diffusion Models in Vision: A Survey**, arXiv:2209.04747v2 6 Oct 2022.

[5] *Jonathan Ho, Ajay Jain, Pieter Abbeel.* **Denoising Diffusion Probabilistic Models**, arXiv:2104.07636v2 30 Jun 2021.

[6] *Ling Yang Zhilong Zhang, Yang Song, Shenda Hong, Runsheng Xu, Yue Zhao, Wentao Zhang, Bin Cui, Ming-Hsuan Yang.* **Diffusion Models: A Comprehensive Survey of Methods and Applications**, arXiv:2209.00796v10 23 Mar 2023.

[7] *Prafulla Dhariwal and Alex Nicho*l. **Diffusion Models Beat GANs on Image Synthesis**, arXiv:2105.05233v4 1 Jun 2021.

[8] *Ankan Kumar Bhunia, Salman Khan, Hisham Cholakkal, Rao Muhammad Anwer,Jorma Laaksonen,Mubarak Shah, Fahad Shahbaz Khan, Mohamed bin Zayed.* **Person Image Synthesis via Denoising Diffusion Model**, arXiv:2211.12500v2 28 Feb 2023.

[9] *Joseph L.Watson and David Baker*. **Broadly applicable and accurate protein design by integrating structure prediction networks and diffusion generative models**, biorxiv:2022.12.09.519842v1 10 Dec 2012

[10] Our code: https://github.com/HayLahav/Deep_Learning_TAU/blob/main/Diffusion_model_project/Final_Diffusion_Model_Project.ipynb