A Quantitative Analysis of CNN Approaches to Image Reconstruction through Disordered Media

Matthew B. Robertson

A dissertation submitted in partial fulfilment
of the requirements for the degree of
Bachelor of Science
of the
University of Aberdeen.

Department of Computing Science

2025

Declaration

No portion of the work contained in this document has been submitted in support of an application for a degree or qualification of this or any other university or other institution of learning. All verbatim extracts have been distinguished by quotation marks, and all sources of information have been specifically acknowledged.

Signed: Matthew Robertson

Date: 2025

Abstract

When light propagates through a disordered medium, such as multimode optical fibers, water, or clouds, it undergoes complex scattering, resulting in scrambled images that are challenging to reconstruct. Traditional approaches to image reconstruction rely on computational methods to reverse these distortions, but they are computationally expensive. In contrast, convolutional neural networks (CNNs) / convolutional autoencoders have demonstrated promising capabilities in learning to recover original images from their distorted counterparts. This project presents a quantitative evaluation of different CNN architectures for image reconstruction through disordered media, focusing on ResNet [1], EfficientNet [2], UNet [3], and REDNet [4] architectures. Additionally, other factors such as loss functions, dataset size and complexity (Binary and Greyscale) are examined. Experiments are conducted using data acquired through a Digital Micromirror Device (DMD) and laser based optical setup. The findings provide insights into the trade-offs between different architectures and configurations, highlighting optimal approaches for image reconstruction in complex optical environments.

Acknowledgements

I would like to thank both of my honours project supervisors, Prof. Brian Logan and Prof. David McGloin, your unwavering support and faith in me have been vital to the success of my project.

I am also grateful to Cailean Mackay, Ilona Sadovska and last but not least Dr. Bing Yan, who I worked in the Laser lab with over these 13 weeks. You have all been very kind and I wish you luck in your future careers. I will miss our coffee's and walks through the botanic gardens.
I also would like to honour my late mother. I'm not sure I would be where I am if not for you.

Contents

Acronyms

Acronyms are in order of appearance.

NN Neural Network
CNN Convolutional Neural Network
MAE Mean Absolute Error
MSE Mean Squared Error
DMD Digital Micromirror Device
SLM Spatial Light Modulator
SSIM Structural Similarity Index Measure
LPIPS Learned Perceptual Image Patch
NPCC Negative Pearson Correlation Coefficient
D2NN Diffractive Neural Network
ROI Region of Interest
EMNIST Extended Modified National Institute of Standards and Technology
FPS Frames Per Second
CMOS Complementary Metal-Oxide-Semiconductor
SGD Stochastic Gradient Descent
ANOVA Analysis of Variance
RNN Recurrent Neural Network
PINN Physics Informed Neural Network

Chapter 1

Introduction

1.1 Motivation

The reconstruction of images transmitted through disordered media is an increasingly critical challenge in optics and computational imaging. Disordered media such as multimode optical fibers, biological tissue, and atmospheric conditions cause significant scattering and distortion of light, severely complicating the retrieval of clear and accurate images.

The scattering of this light can be defined by some deterministic transmission matrix $\mathbf{T}$, this transmission matrix characterizes the media such that if we know the input image, we know how the image will be altered by $\mathbf{T}$ and can be corrected. The challenge arises when $\mathbf{T}$ becomes variant with time ($d\mathbf{T}/dt$), temperature ($d\mathbf{T}/dT$) or some other external factor. Solving this analytically or computationally is slow and compute-intensive and as such cannot keep up with fast-paced changes to the environment.

This inherent flaw with analytical methods has seen data-driven methods such as AI and deep learning models get attention within this space, with a hope for reliable AI models to be developed that would allow real-time imaging through virtually any medium.
Recent advancements in machine learning, particularly convolutional neural networks (CNNs), present promising solutions to these limitations. CNNs have shown notable success in various image processing tasks, leveraging their ability to learn intricate patterns and correlations within data to achieve efficient and accurate reconstructions. However, there remains a need for a systematic and quantitative assessment of different CNN architectures and training approaches, particularly concerning their effectiveness under varying levels of scattering, dataset sizes, and loss functions.

1.2 Objectives

Motivated by these challenges, this dissertation aims to perform a comprehensive evaluation of CNN architectures for reconstructing images affected by disordered media. Specifically, the objectives of this research include:

- Evaluating the performance of different CNN architectures for image reconstruction through disordered media. Specifically hybrid autoencoder networks such as: EfficientNet-REDNet, EfficientNet-UNet, ResNet-REDNet, and ResNet-UNet.
- Investigating how various factors, such as the level of scattering (GRIT levels), dataset size, and choice of loss functions (L1 and NPCC), influence the reconstruction quality.
- Analyzing the discrepancies between different evaluation metrics, notably the Structural Similarity Index Measure (SSIM) and Learned Perceptual Image Patch Similarity (LPIPS), to identify their strengths and limitations.

Through achieving these objectives, this dissertation seeks to clarify best practices for CNN-based image reconstruction and address critical knowledge gaps within the field, paving the way for improved real-time imaging solutions in complex disordered environments.

Chapter 2

Background

2.1 Neural Networks

A Neural Network (NN) is a computational model inspired by the way neurons interact in the human brain. Despite the biological analogy, the fundamental concept is more straightforward mathematically: neural networks learn to approximate functions by adjusting internal parameters (weights and biases) in response to data. [5]
Each artificial neuron (in a hidden or output layer) performs two key operations:

1. **Weighted Sum and Bias:** A neuron receives one or more inputs x_i. Each input is multiplied by the corresponding weight w_i, and a bias term b is added:

$$z = \sum_i w_i x_i + b. \tag{2.1}$$

2. **Activation Function:** The result z is then passed through a non-linear activation function $\sigma(z)$. A common choice is the ReLU function for its simplicity:

$$\sigma(z) = \max(0, z). \tag{2.2}$$

Neurons are typically organised into layers:

- **Input Layer:** Receives the raw input data (for example, pixels of an image or the state of a card game).
- **Hidden Layers:** One or more layers in which neurons progressively transform inputs. Stacking more layers allows the network to capture increasingly abstract features.
- **Output Layer:** Produces the final prediction or classification (e.g., a class label, a continuous value, or a decision in a card game).

As data move through these layers, the outputs of each layer serve as inputs to the next. During training, the network parameters ($\{w_i, b_i\}$) are adjusted to minimise a chosen

loss function, guiding the network to produce the correct outputs for given inputs. This optimisation typically uses an algorithm such as backpropagation, which calculates the loss gradients with respect to each parameter.

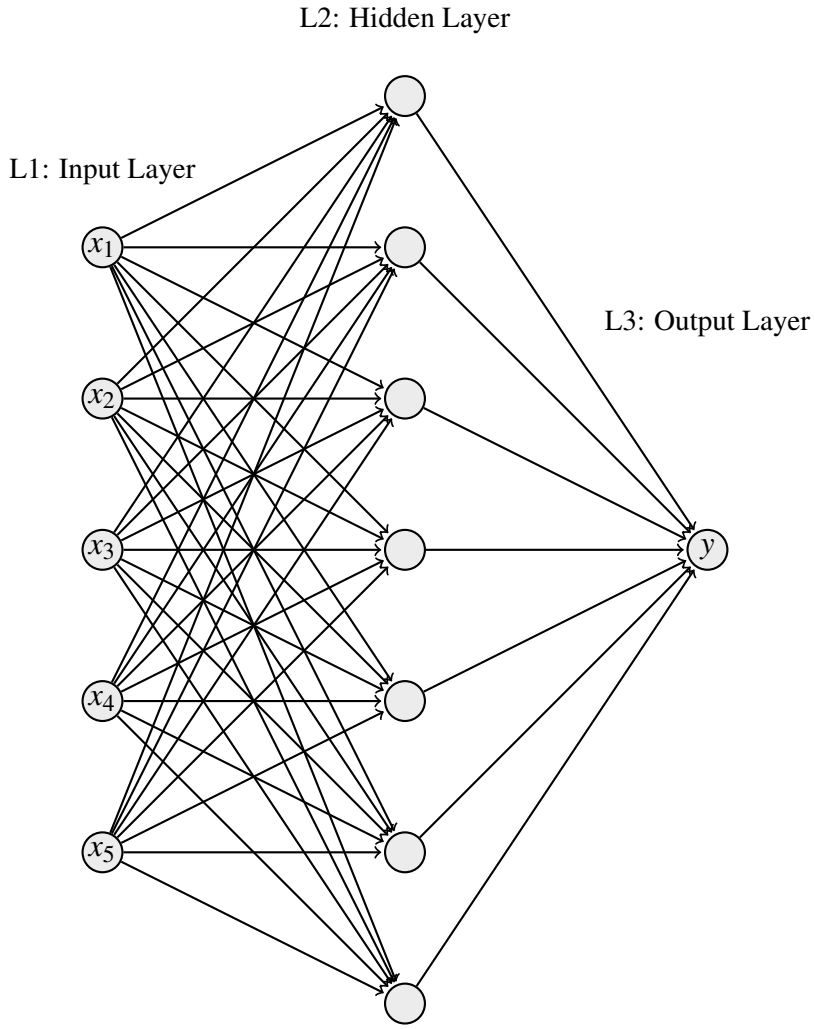

Figure 2.1: A simple fully-connected neural network with five inputs and one output.

However, when fully connected neural networks are presented with high-dimensional inputs (such as large images), the number of parameters and hidden-layer neurons grows rapidly, making them computationally expensive and prone to overfitting. [5] Convolutional Neural Networks (CNNs) address this problem by leveraging convolutional operations and weight sharing to handle image-like data more efficiently, reducing the parameter count, and improving performance.

2.2 Convolutional Neural Networks

The origin of Convolutional Neural Networks (CNNs) dates back to 1989, when LeCun et al. demonstrated the classification of handwritten US ZIP codes using a neural network trained with backpropagation. [6] Their approach introduced the concept of local receptive fields and weight sharing by applying convolutional filters to extract spatial features

from images.

CNNs are a type of deep learning algorithm designed to deal with images, unlike traditional neural networks, where each neuron may be fully connected to each input. CNNs introduce the concept of 'localised receptive fields'; these receptive fields are local in the sense that they are responsible for looking at a section of the original image. Essentially, they allow neurons in our CNN architecture to focus on very specific features of images such as edge detection. [7]

These receptive fields that represent subsections of an image are convolved with a learnable filter matrix (also known as kernels). This convolution operation is used to extract spatial details, such as patterns, edges, corners, or textures.
Although we call a CNN a convolutional neural network, we actually compute cross-correlation rather than true mathematical convolution. This difference is due to the better computational efficiency in cross-correlation [8]. Cross-correlation is an operation that involves computing the discrete summation of the product between two functions. Qualitatively, we can describe this operation as the 'effect' of one matrix (in our instance, the filter matrix representing a learnt feature) has on another (our local receptive field representing a region of our input image).

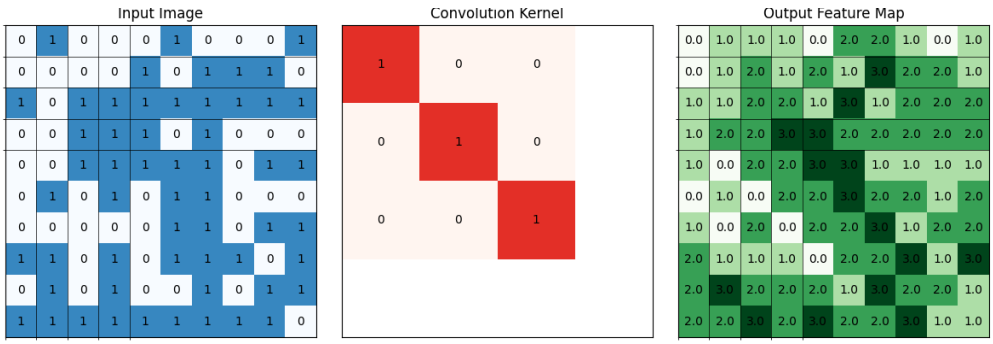

Figure 2.2: An example of a 3x3 Kernel applied to a 10x10 input matrix

As seen in Figure 2.2, the feature map has higher weights, where a pattern reminiscent of the kernel is found in the input matrix.
The learnable filter matrices slide across the input image, applying the same weights at each region within the image; this principle is known as 'weight sharing'. Weight sharing allows our network to generalise and reduce the number of parameters in our network, simplifying the complexity. The result of applying weight sharing is known as a 'feature map'. Feature maps highlight where certain learnt features are present in the image. After convolution, a non-linear activation function is applied to the feature map, allowing the model to identify more complex features. A pooling layer is also introduced to reduce the complexity of feature maps without removing important features captured by the network;

this is achieved using invariance, analogous to a lossless compression algorithm.[7]
As the depth of a CNN is increased, the localised receptive fields become less local, instead looking at a larger portion of the image, allowing feature maps to do much more complex detection, such as detecting actual objects.

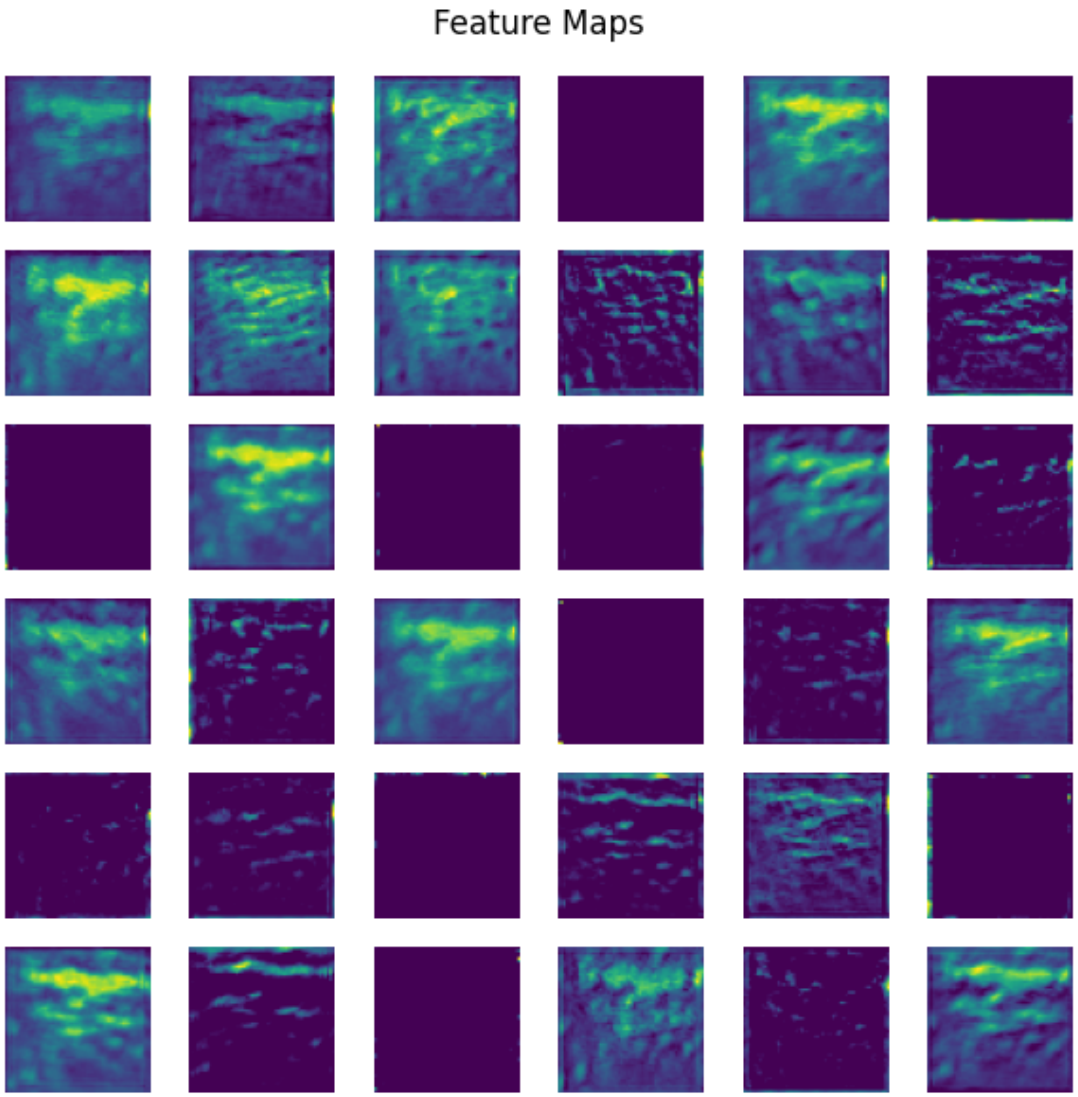

Figure 2.3: An example set of feature maps taken from an image from the EMNIST letters dataset.

CNN's on their own however are used for classification, not reconstruction, so there is another key component that we must consider.

2.3 Autoencoders

Autoencoders are a class of neural networks designed to learn efficient data encoding in an unsupervised manner. Originally popularised by Hinton and Salakhutdinov [9], autoencoders perform dimensionality reduction (or feature learning) by compressing the

input into a latent representation (the 'bottleneck') and then reconstructing the input from that compressed representation. The autoencoder network consists of two main parts:

- **Encoder**: This portion of the network progressively reduces the dimensionality of the input, mapping it to a low-dimensional latent space.
- **Decoder**: The decoder reconstructs the original input from the low-dimensional latent representation by 'expanding' the compressed features back to the original input dimension.

When trained end-to-end, autoencoders learn to minimise a reconstruction loss that measures how well the decoder's output matches the original input. By doing so, the network implicitly learns a compact, meaningful representation of the data in its latent space; this is otherwise known as unsupervised learning pertaining to the fact that autoencoders do not require labels on the data that they are given. [10]

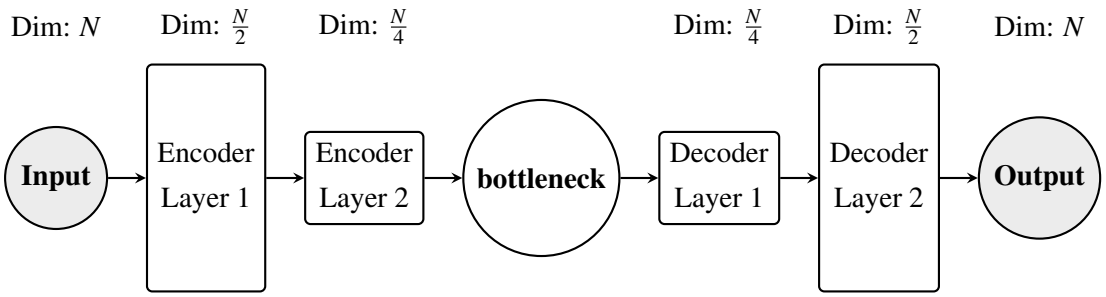

Figure 2.4: Schematic of a typical encoder-decoder architecture

Although traditional autoencoders often use fully connected layers, convolutional autoencoders leverage the strengths of CNNs for image-based tasks. Convolutional layers naturally exploit spatial correlations in images: they can progressively encode an image by learning hierarchical features through stacked convolutions and downsampling (pooling) operations, and then decode it by reversing these steps (e.g., using transposed convolutions or upsampling) to reconstruct the original image dimensions.

2.4 Skip Connections and Residuals

Skip connections are architectural features of neural networks that provide alternative paths for gradients to flow during training. By directly connecting non-consecutive layers, they alleviate problems such as vanishing gradients and help preserve information from earlier layers, improving training efficiency and convergence. [11] Skip connections reduce the number of parameters required, as the network does not need to learn redundant representations of already captured features.

Residual connections, a specific form of skip connection popularised by ResNet architectures [1], enable the network to learn residual functions with reference to the input of the

layer rather than direct transformations. Residual connections typically operate between layers of similar dimensions, creating local pathways for gradient propagation that make it easier to train deep networks.

Skip connections used in architectures like UNet [3] bridge the encoder and decoder sections of the network. These connections transfer detailed spatial information directly from the encoding phase to the corresponding decoding layers, counteracting the loss of spatial resolution caused by downsampling. This form of skip connection is essential for tasks that require precise spatial localisation, such as image reconstruction, so that the network can learn features of varying complexity, such as detailed edge detection at higher resolutions and character detection at lower resolutions with broader feature maps.

2.5 Loss Functions

The loss function is used during training for the model to evaluate how well it is performing during the training sequence, for regression-type problems such as image reconstruction. Loss functions such as L1 (Mean Absolute Error) and MSE (Mean Squared Error) are often preferred due to their ability to directly measure the pixel-wise difference between the reconstructed images and ground truth references [12; 13]. Mathematically, the L1 loss between a predicted image I_{pred} and its corresponding ground truth I_{true}, each containing N pixels, can be expressed as:

$$L_{\text{L1}}\left(I_{\text{pred}}, I_{\text{true}}\right) = \frac{1}{N}\sum_{i=1}^{N}\left|I_{\text{pred}}^{(i)} - I_{\text{true}}^{(i)}\right|, \tag{2.3}$$

while the MSE loss is defined as:

$$L_{\text{MSE}}\left(I_{\text{pred}}, I_{\text{true}}\right) = \frac{1}{N}\sum_{i=1}^{N}\left(I_{\text{pred}}^{(i)} - I_{\text{true}}^{(i)}\right)^2. \tag{2.4}$$

Although these pixel-wise approaches are widely used, more recent works have studied NPCC (Negative Pearson Correlation Coefficient) Loss, which focusses on the structural similarities or correlations within the images rather than raw pixel-wise differences [14]. The NPCC loss is typically given by:

$$L_{\text{NPCC}}\left(I_{\text{pred}}, I_{\text{true}}\right) = -\frac{\sum_{i=1}^{N}\left(I_{\text{pred}}^{(i)} - \overline{I_{\text{pred}}}\right)\left(I_{\text{true}}^{(i)} - \overline{I_{\text{true}}}\right)}{\sqrt{\sum_{i=1}^{N}\left(I_{\text{pred}}^{(i)} - \overline{I_{\text{pred}}}\right)^2}\sqrt{\sum_{i=1}^{N}\left(I_{\text{true}}^{(i)} - \overline{I_{\text{true}}}\right)^2}}, \tag{2.5}$$

where $\overline{I_{\text{pred}}}$ and $\overline{I_{\text{true}}}$ denote the mean intensities of I_{pred} and I_{true}, respectively. This correlation-based metric thereby captures the structural similarity between two images more directly than simple absolute or squared differences.

2.6 Normalization

Normalisation is a common technique in deep learning that helps stabilise and speed up training by ensuring that the distribution of activations remains controlled.[15] Several types of normalisation strategies exist:

- **Batch Normalisation (BN):** Normalises activations based on the mean and variance computed over the current minibatch. It is the most widely adopted method in many state-of-the-art architectures.
- **Layer Normalisation (LN):** Runs on each individual example by normalising across all features. It is less sensitive to batch size and is commonly used in sequence modelling tasks.
- **Group Normalisation (GN):** Splits channels into groups and normalises the features within each group. This approach tends to be more stable than BN for very small batch sizes, as it does not rely on batch-level statistics.
- **Instance Normalisation (IN):** Normalises each sample and each channel individually, making it particularly useful in style transfer tasks.

In this work, we employ only batch normalisation. Due to the time-consuming nature of our experiments, exploring multiple normalisation strategies was not feasible. Batch normalisation is commonly used in CNNs, so this felt like a sensible decision.

2.7 Attention

Attention mechanisms have become an essential component in modern deep learning architectures, particularly in tasks where the model must selectively focus on relevant parts of the input data. The concept of attention was popularised by Vaswani et al. in the landmark paper *'Attention is All You Need'* [16], which introduced the idea of dynamic content-dependent weighting of features.

In the context of encoder-decoder architectures for image reconstruction, we incorporate attention mechanisms along the skip connections between encoder and decoder layers. Rather than simply passing feature maps directly across, attention modules learn to weight different spatial regions of the features based on their relevance to the reconstruction task. This allows the network to prioritise important spatial details and suppress irrelevant or noisy information, improving the overall quality of the reconstruction.

Specifically, attention modules compute a learnt weighting for each feature channel or spatial location, enhancing salient features while diminishing less important ones. This is analogous to how human vision is focused on particular regions of a scene. The integration of attention into skip connections helps mitigate information bottlenecks and improves the network's ability to selectively fuse low-level (fine detail) and high-level (semantic) information across the encoder-decoder pathway.

2.8 Optics

2.8.1 Ground Glass Diffuser

A ground glass diffuser is an optical component that is designed to scatter light in a random manner. We can quantify a ground glass diffuser by a 'GRIT' level; a higher GRIT level indicates less scattering, whereas a lower GRIT level indicates more intense scattering.

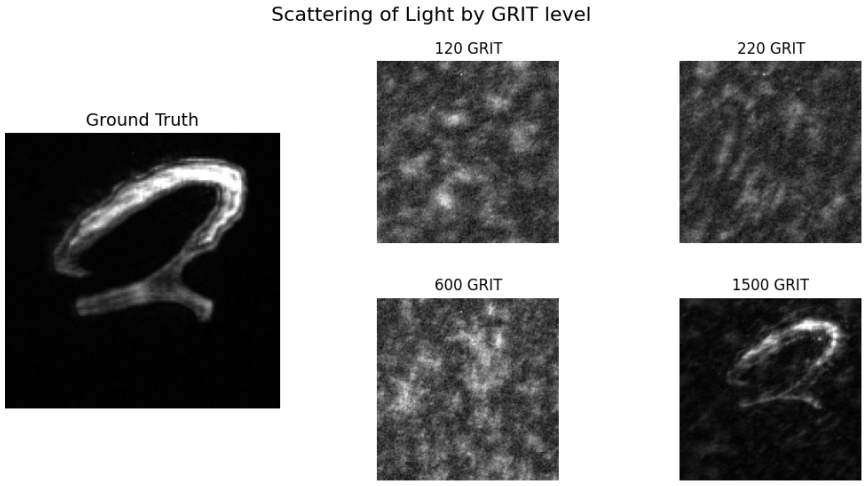

Figure 2.5: The scattering effects of different GRIT levels on a randomly sampled greyscale image

120, 220, 600 GRIT images are very scattered and no structure can be seen compared to the original ground-truth image. However, 1500 GRIT still retains the primary structure of the image, albeit noisier because the light is not scattered as much.

2.8.2 Digital Micromirror Device

A DMD (Digital Micromirror Device) is a type of Spatial Light Modulator (SLM) that we can use to project images in real space; they are commonly used in projectors to project digital images in physical space.

DMDs are composed of a 2D array of microscopic mirrors, each corresponding to a single pixel in the projected image. Each mirror can tilt between two angles: one directing light toward the projection path (ON state - 1) and the other directing light away (OFF state - 0). To display a digital image, the DMD processes the image into bitplanes, where each bitplane represents one bit of pixel intensity across the image. [17]

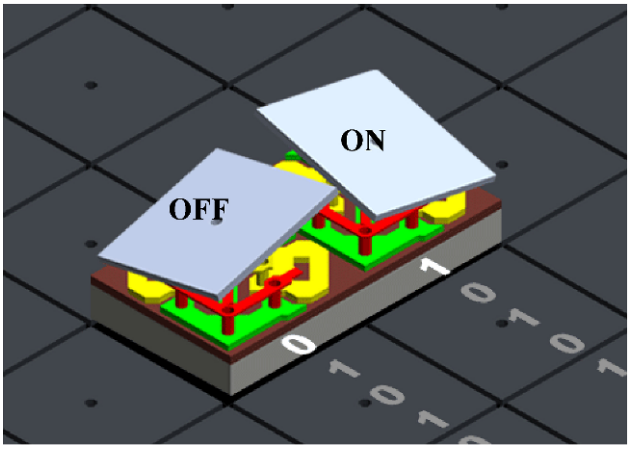

Figure 2.6: An example showing how the mirrors on the DMD tilt based on if a '1' or '0' is sent to that specific mirror. [18]

For binary images, there is only one bitplane, which means that a pixel can only be ON or OFF. For an 8 bit greyscale image, the image is decomposed into 8 binary bitplanes. The DMD rapidly cycles through these bitplanes, starting with the most significant bit. Each bitplane is displayed for a duration proportional to its bit weight (the bitplane for 128 intensity units is shown longer than the bitplane for 1 unit). For example, a pixel with an intensity of 193 (in 8-bit greyscale) will have an ON state in the 128,64 and 1 bitplanes and will have an OFF state in the 32,16,8,4 and 2 bitplanes. This temporal dithering exploits the integration of the eye over time to produce the perception of greyscale.

By precisely controlling the timing and synchronisation of the mirrors, the DMD can reconstruct full 8-bit greyscale images at high frame rates, despite each mirror being only binary (ON or OFF).

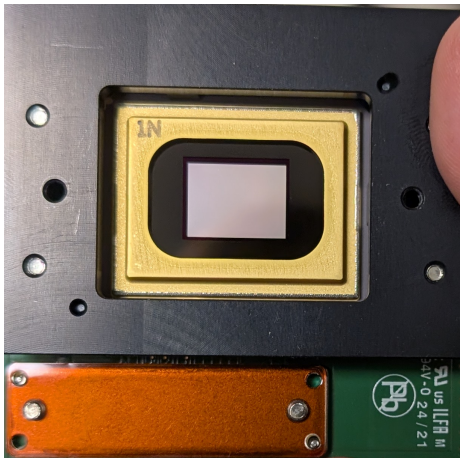

Figure 2.7: The DMD used for this project, the reflective surface in the center is the array of mirrors that reflect light into the camera.

2.9 Evaluation Metrics

2.9.1 SSIM

To assess the performance of image reconstruction models, appropriate evaluation metrics are required to quantify the similarity between the reconstructed output and the original undistorted image. In this project, the main metric used is the Structural Similarity Index Measure (SSIM) [19]. SSIM evaluates image quality based on perceived changes in structural information, brightness, and contrast. SSIM returns a value between -1 and 1, with 1 indicating perfect similarity. It is computed over local windows across the image and then averaged to produce a global score.

SSIM is mathematically defined as:

$$\text{SSIM}(x,y) = \frac{(2\mu_x\mu_y + c_1)(2\sigma_{xy} + c_2)}{(\mu_x^2 + \mu_y^2 + c_1)(\sigma_x^2 + \sigma_y^2 + c_2)}[19] \tag{2.6}$$

where

- μ_x is the pixel sample mean of x
- μ_y is the pixel sample mean of y
- σ_x^2 is the sample variance of x
- σ_y^2 is the sample variance of y
- σ_{xy} is the sample covariance of x and y
- $c_1 = (k_1 L)^2$ & $c_2 = (k_2 L)^2$ are stability constants that prevent near-infinite values
- L is the range of the pixel values, for an 8-bit image this would be $2^8 - 1 = 255$
- $k_1 = 0.01$ and $k_2 = 0.03$ (by default)

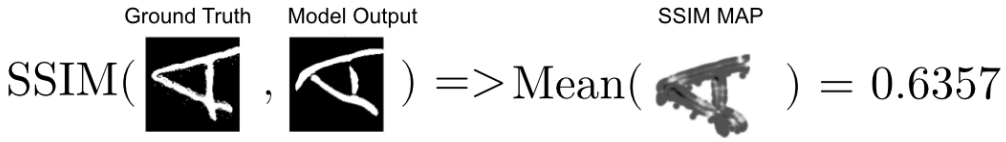

Figure 2.8: SSIM Visualisation generating SSIM Map and taking mean of that map.

2.9.2 LPIPS

The Learned Perceptual Image Patch Similarity (LPIPS) [20] is a perceptual metric used to quantitatively assess the similarity between two images. LPIPS uses a pre-trained convolutional neural network to capture perceptual differences based on human visual perception. It compares the activation patterns of the deep neural network layers generated by two images, providing a similarity score that attempts to align with human judgments

of image quality. This approach considers structural and textural features across various scales, making LPIPS particularly effective in evaluating high-level perceptual similarity rather than mere pixel-level differences. LPIPS is a minimising function meaning that on the LPIPS scale, 1 indicates a very poor score whilst 0 indicates a perfect score.

Chapter 3

Related Research

3.1 Computational Methods for Image Reconstruction

Traditionally, research for image reconstruction through disordered media was grounded within computational and analytical solutions, works such as [21; 22] use iterative methods to restore the original image, utilizing an SLM (2.8.2) to correct for the scattering of spatial information through the disordered media. Cizmar et al. [21] takes this work a step further and utilizes a similar iterative method as a way to create arbritrary intensity distributions (custom images) through a multi-mode fiber, Cizmar et al. achieves this by using the SLM to calibrate for all of the possible modes of the fiber. A mode of a fiber optic cable can be thought about as one of the valid paths light can take through the fiber. Once calibration is complete, a phase pattern is calculated for the SLM that accounts for the disordered effects of the fiber.

As mentioned previously in Section (1.1). The issue with these iterative methods is that they take time to compute based on the complexity of the media that is being imaged through and that the media's scattering characteristics change constantly. In the case of multi-mode fibers the way light scatters changes is very sensitive to temperature and bends in the fiber, meaning that calibration needs to be done constantly for time-varying systems.

This means that with current computation speeds, we are unable to image through scattering media in real-time and hence a need for deep learning approaches is required.

3.2 Neural Network approaches for Image Reconstruction

Many deep learning approaches have been developed for image reconstruction through disordered media. Rahmani et al. [12] use CNN's to reconstruct images travelling through multi-mode fiber's, not only do they reconstruct the original image, they also present work to reconstruct the given phase pattern that could be programmed onto a SLM to reconstruct the image optically, similar to the way iterative methods work.

Sun et al. [23] take a different approach, modifying the U-Net CNN architecture [3] to create their own network called CoreNet. The key innovation in CoreNet is the use of physics-based backpropagation to decompose a target image into its corresponding electric field, where the real part encodes amplitude and the imaginary part encodes phase information.

It is important to note that conventional cameras can only capture intensity, not the full complex optical field. The recorded intensity $I(x,y)$ is related to the electric field $E(x,y)$ by:

$$I(x,y) = |E(x,y)|^2 = \mathrm{Re}(E(x,y))^2 + \mathrm{Im}(E(x,y))^2 \tag{3.1}$$

Since both $\mathrm{Re}(E(x,y))^2$ and $\mathrm{Im}(E(x,y))^2$ are real-valued, the phase component is not directly observable in the captured intensity image. This is why cameras inherently lose phase information during acquisition.

Both components of the E-field are then fed into a U-Net architecture and are then concatenated back together for reconstruction. The final output of this network is a phased-array that is programmed onto an SLM which then reconstructs the image through the disordered media.

Li et al. [14] propose a densely connected convolutional neural network architecture, termed IDiffNet. The network is based on the DenseNet architecture, where each layer connects to every other layer within the same block. This dense connectivity enhances feature propagation, encourages feature reuse, and reduces the number of parameters, thereby improving generalization. An important contribution of this work is the introduction of the negative pearson correlation coefficient (NPCC) as a loss function, which the authors demonstrate is better suited than mean absolute error (MAE) for training on sparse datasets and under strong scattering conditions. IDiffNet is shown to outperform conventional denoising networks by learning the spatial correlations imposed by the diffuser, rather than just removing noise pixel-wise.

Mengu et al. [24] present a novel approach using diffractive optical neural networks (D2NNs) to perform image classification and reconstruction in an entirely optical domain. Unlike conventional neural networks that operate electronically, D2NNs physically implement computation through light diffraction across successive layers of passive transmissive surfaces. These layers are designed using deep learning and then fabricated using techniques such as 3D printing or photolithography.

In their setup, spatially overlapping, phase-only images—such as handwritten digits are encoded into the input plane. As light propagates through the trained diffractive layers, the optical field is modulated in a way that encodes class-specific features, allowing for all-optical classification and compression of the input information. This compressed optical

output is then optionally passed to a shallow electronic neural network to reconstruct the original phase images. A limitation of this approach however is that the trained diffractive layers are static, making them less suitable for scenarios involving dynamically varying transmission matrices.

The approaches described above highlight the substantial progress made by deep-learning and computational methods in image reconstruction through disordered media. Nevertheless, existing computational techniques remain limited by their inherent computational demands and inability to adapt swiftly to dynamic scattering conditions. Similarly, while recent neural-network-based methods demonstrate potential, current literature lacks comprehensive comparative analyses that systematically evaluate the individual impacts of different CNN architectures, loss functions, and training dataset characteristics. This thesis aims to address precisely this gap, providing a detailed and methodical comparison of these critical factors to identify optimal CNN-based strategies suited for real-time reconstruction in complex optical scenarios.

Chapter 4

Methodology

This chapter lays out the experiment, what decisions were made in relation to the experiment, and why that decision was taken.

4.1 Prerequisites

4.1.1 Optics

To project images, a Digital Micromirror Device (DMD) was selected as the spatial light modulator. DMDs offer extremely fast response times, allowing rapid switching between patterns and enabling efficient data collection. In this experiment, 60,000 images could be projected in approximately 15 minutes. Compared to other spatial light modulators, such as liquid crystal on silicon devices, DMDs are relatively inexpensive and robust, making them well-suited for high-volume runs.

To introduce diffusion, a ground glass diffuser was placed in the optical path. Ground glass was chosen because of its random scattering behaviour, which produces a complex speckle pattern. Unlike structured or engineered diffusers, the random nature of ground glass leads to transmission matrices that are highly sensitive and difficult to model analytically without calibration, an ideal condition for evaluating the performance of image reconstruction algorithms.

A mount was designed for the DMD so that it could be spatially manipulated to align the optics.

A simple 3D model was designed in SOLIDWORKS and subsequently 3D printed to adapt the DMD to our optical bench, as shown in Figure 4.1.

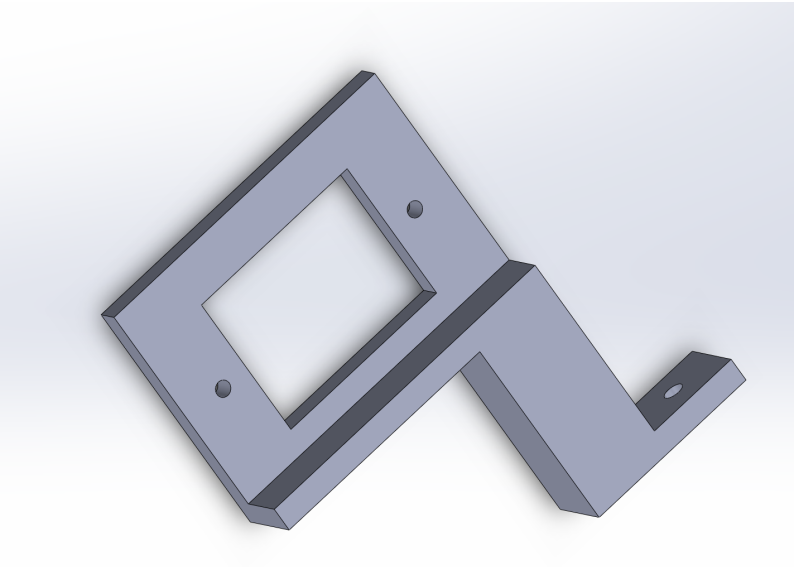

Figure 4.1: The SOLIDWORKS 3D model of the DMD mount used in the optical setup.

The optical system itself also had to be set up before any data collection for the experiment could begin; a helium-neon gas laser was used for the illumination of the DMD which would then be captured by a CMOS camera.

The final optical setup is illustrated in figure 4.2. While it would have been possible to simplify the setup by removing the adjustable mirror, there was a restriction on where we could place the DMD and camera on the optical workbench for connection to the PC. The final setup was found to be ideal.

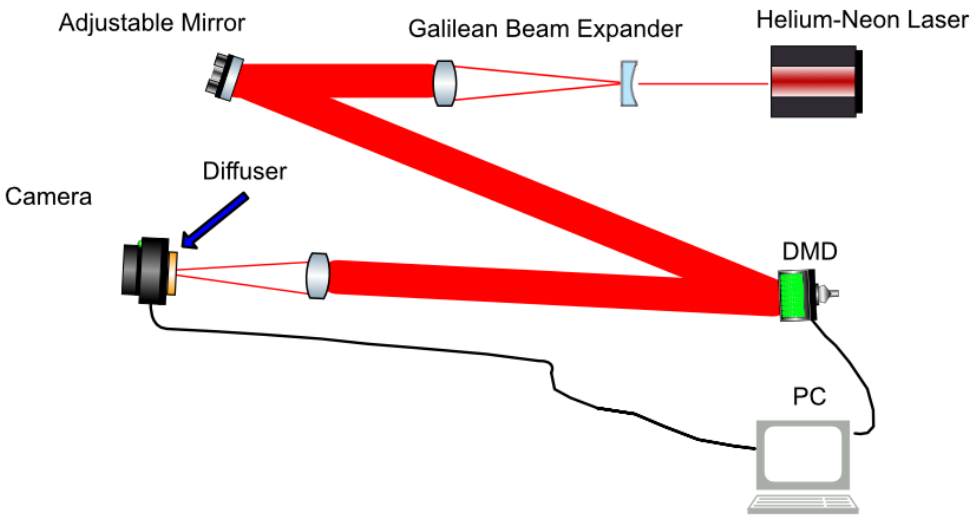

Figure 4.2: The final optical setup that was used to illuminate the DMD and capture the output.

4.1.2 Dataset

An important component of the experiment was selecting a dataset that was both appropriate for projection through the optical setup and sufficiently large to support effective training of deep neural networks. The EMNIST (Extended Modified National Institute of Standards and Technology) 'letters' dataset was deemed appropriate for this purpose.

EMNIST provides a large collection of handwritten character images, and its letters subset contains balanced samples across the English alphabet. This made it ideal for our task, as the character images are structured enough to be meaningfully reconstructed while still exhibiting sufficient variability to challenge the learning process.

After the initial data collection, a greyscale gradient filter was introduced to a second set of input images prior to projection. This filter added complexity to the input data in a controlled manner, allowing us to explore how increasing intensity variation affects model performance when passed through a diffuser. The implemented greyscale gradient filter is a matrix of values that scale from 1 to 0 vertically across the image. When we multiply this filter by our input, a gradient effect is applied to the input image.

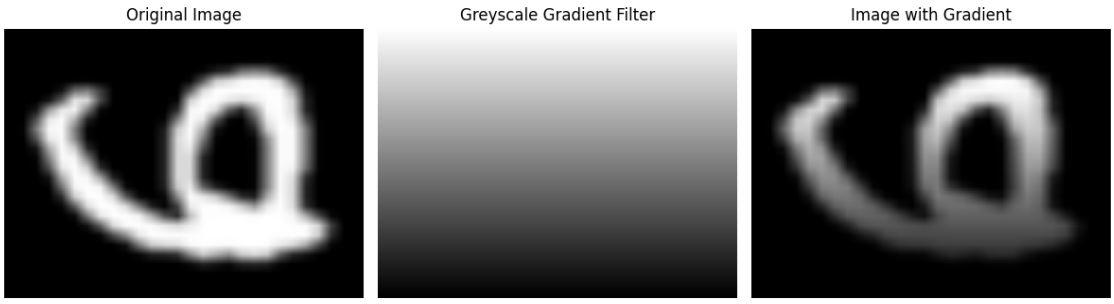

Figure 4.3: An example showing how an image is transformed by the gradient greyscale filter.

4.2 Programming Environment and Software Choices

Python 3.12 was selected as the main programming language for both data acquisition and model training, due to its strong ecosystem of scientific libraries and hardware control interfaces. Python offered a library for interfacing with the Thorlabs CMOS camera, enabling precise control over key parameters such as exposure time, region of interest (ROI), and frame rate.

Control of the Digital Micromirror Device (DMD) was implemented using an open-source Python library, originally developed by researchers from the CNRS in France. This library

provided a practical interface for projecting custom patterns, allowing fast and concurrent control of both the DMD and the camera.

Python also supported all aspects of the development of the NN models. PyTorch was used for implementing and training a range of CNN architectures (EfficientNet, ResNet, UNet, REDNet), while supporting libraries such as NumPy, PIL, and OpenCV enabled efficient manipulation of images and two-dimensional arrays.

The full list of Python packages used can be found in the Maintenance Manual A.2.3

All developed code was executed in a CLI environment on Windows 11 using Windows Command Prompt.
The machine that was used to train and evaluate the developed models had the following key hardware specifications.

- CPU - i7-12700H
- GPU - NVidia RTX 3070 TI (Mobile)
- RAM - 64GB

4.3 Overview of the Experiment

The experimental workflow can be broadly divided into three key stages, each representing a part of the image reconstruction pipeline.

1. **Data Collection**: EMNIST images were projected through a Digital Micromirror Device (DMD) and recorded using a CMOS camera after transmission through diffusers of varying GRIT ratings. The output images served as the diffused input for model training.

2. **Data Preparation & Model Training**: The collected diffused images were paired with their original clean counterparts and used to train various CNN architectures. The preprocessing steps included resizing and normalisation, followed by model training using PyTorch.

3. **Model Evaluation & Comparison**: Each trained model was evaluated using quantitative metrics such as SSIM and LPIPS to assess its quality of reconstruction. Comparisons were drawn between architectures, training regimes, and diffuser conditions.

4.4 Data Collection

The available ground glass diffusers in the laboratory were 1500 GRIT, 600 GRIT, 220 GRIT, and 120 GRIT; these are standard GRIT levels used for scattering in optics. 2.8.1
A custom Python script was developed to coordinate the operation of both the Digital Micromirror Device (DMD) and the camera. Simultaneous control was necessary to ensure precise synchronisation between pattern projection and image capture.
The images that were projected onto the DMD were also scaled down so that the image did not span across the whole of the DMD's surface; this was done for multiple reasons. Firstly, shrinking the image allowed us to shrink the ROI settings on the camera, allowing us to capture images quickly. Secondly, shrinking the image ensured uniform illumination across the image from the helium-neon gas laser onto the DMD, ensuring consistency in the data collection process. The camera's maximum framerate was significantly improved by applying a region of interest (ROI) setting. By limiting the ROI to 128×128 pixels, the achievable framerate increased from approximately 35 frames per second (fps) to 260 fps. The DMD, capable of high refresh rates, can project at 250 fps for 8-bit images and exceed 2000 fps for 1-bit binary images (2.8.2).
For most of the data collection, both the DMD and camera were configured to operate at 200 fps. A small delay was introduced in the script to ensure that the camera did not capture images prematurely, i.e., before the DMD had fully displayed the intended pattern.

Unwanted noise in the image was reduced by statistically combining the images. This was done using the following:

- Two frames were captured for each projection and the pixel-wise median was computed.
- For binary images projected without a diffuser, the output was binarized to reduce glare and background light.

In the case of greyscale projections, multiple frames needed to be captured and summed together. This was due to the way DMDs render greyscale images, by displaying a sequence of bitplanes in time (2.8.2). Capturing a single frame would not suffice, as it would only record a partial representation of the target image.
Although the EMNIST Letters dataset contains approximately 145,600 images, a maximum training set size of 60,000 images was selected. This limitation was imposed due to GPU memory constraints encountered during model training at higher dataset scales. The final data set configuration included the following.

- **Training images**: 60,000 binary and 60,000 greyscale
- **Testing images**: 5,000 binary and 5,000 greyscale

- **Diffuser conditions**: RAW (no diffuser), 1500 GRIT, 600 GRIT, 220 GRIT, 120 GRIT

This results in a total of 610,000 captured images under all conditions.

Before starting data collection, Thorcam software was used to verify that the camera was properly aligned with the DMD and that the projected image was within the camera's field of view.

After each set of data was collected, the ground glass diffuser was swapped out until we covered all the available diffusers. The camera was also recalibrated to ensure continued data consistency.

The pseudocode for the data collection process can be seen in Algorithm 1

Algorithm 1 Image Projection and Acquisition with DMD and Camera

```
function MAIN
    Load EMNIST dataset
    Initialize DMD and camera devices
    for each EMNIST image index do
        Format EMNIST image (resize, center, optional invert/gradient)
        Project image onto DMD
        Trigger camera and capture multiple frames
        if greyscale mode then
            Sum frames and clip to 8-bit
        else
            Compute pixel-wise median across frames
        end if
        Mirror processed image
        Save image to disk
        Halt DMD sequence
    end for
    Release DMD and camera resources
end function
function FORMATIMAGE(index, DMD, dataset, options)
    Load EMNIST image, scale to 8-bit
    Optionally invert and apply gradient
    Resize and center on DMD canvas
    return formatted image
end function
```

4.5 NN Architectures & Model Training

The networks selected for this experiment - ResNet, EfficientNet, UNet, and REDNet - were chosen based on their widespread popularity, proven effectiveness in literature, and suitability for image reconstruction tasks. ResNet was included as a canonical deep CNN architecture that introduced residual connections, which are particularly valuable for stabilizing the training of deeper networks and preserving spatial features across layers. EfficientNet was selected for its lightweight design and efficient scaling properties, offering strong feature extraction performance with fewer parameters. For decoders, UNet and REDNet were chosen due to their extensive success in various image restoration and reconstruction problems. UNet's symmetric architecture with skip connections is well-suited for preserving spatial detail, while REDNet's residual encoder-decoder framework is specifically designed to enhance image restoration by learning residual mappings.

Other popular architectures, such as VGGNet [25], were considered but ultimately not included due to time constraints within the experimental timeline. Overall, the chosen architectures represent a balanced selection of modern, well-established CNN models capable of addressing the challenges posed by reconstructing images diffused through random scattering media.

4.5.1 Encoder: ResNet

ResNet (Residual Network) [1] was introduced to address the degradation problem that occurs in very deep networks. It uses residual connections which allow gradients to flow more easily through the network during training. These connections enable the model to learn residual mappings rather than full transformations, making it easier to optimize deeper architectures. In image reconstruction, ResNet is valued for its ability to maintain spatial structure and depth without suffering from vanishing gradients.

4.5.2 Encoder: EfficientNet

EfficientNet [2] is a family of convolutional neural networks designed to achieve strong performance with fewer parameters by scaling the network's depth, width, and resolution in a balanced way. It is built around a lightweight and modular design, allowing it to extract meaningful features while remaining computationally efficient.

4.5.3 Decoder: UNet

UNet [3] is a symmetric encoder-decoder architecture originally designed for biomedical image segmentation. It features skip connections that directly transfer spatial information from encoder layers to corresponding decoder layers, preserving fine-grained details lost during downsampling. This makes UNet highly suitable for image reconstruction tasks, as it helps the network recover high-resolution outputs from low-resolution feature maps.

4.5.4 Decoder: REDNet

REDNet (Residual Encoder-Decoder Network) [4] is designed for image restoration tasks like denoising or deblurring. It combines the encoder-decoder structure of UNet with residual learning at multiple stages. These residual connections help the model learn the difference between the input and output more effectively.

All CNN models follow a convolutional autoencoder framework composed of an encoder and a decoder. UNet is a typical example of a convolutional autoencoder architecture and is often used in the literature for image reconstruction [12; 13], denoising, and computer vision. We experimented with hybrid configurations that combine the strengths of residual networks, attention mechanisms, and skip connections to see how they might compare against typical singular architectures such as UNet or REDNet. All network weights were initialised from scratch rather than using pretrained weights; pretrained weights were not suited to our task and worsened results. All attention mechanisms 2.7 implemented in the decoders are implemented similarly to [26]. Where encoder and decoder feature maps are combined using learnt 1x1 convolutions and a gating operation is applied locally to each pixel, this creates a matrix of weights across the feature map, representing how important individual pixels are to reconstructing the image, helping the network learn what areas to prioritise when looking at new diffused images.

4.5.5 ResNet-UNet and ResNet-REDNet

We implemented two hybrid models using a ResNet34-based encoder [1] adapted for single-channel (greyscale) input. The encoder extracts five hierarchical feature maps in decreasing spatial resolutions. These outputs were passed to either a UNet decoder or a REDNet decoder:

- **ResNet-UNet** combines this encoder with a decoder that performs upsampling using transposed convolutions and attention-gated skip connections. Intermediate features are concatenated during decoding to retain spatial detail.
- **ResNet-REDNet** pairs the same encoder with a REDNet-style decoder that takes advantage of dilated residual blocks and attention-weighted additive skip connections, batch normalisation is applied after each convolutional layer.

4.5.6 EfficientNet-UNet and EfficientNet-REDNet

EfficientNet-B3 was selected as an alternative encoder backbone due to its efficient depth-width-resolution scaling and strong performance in vision tasks. Since the original EfficientNet expects a 3-channel RGB input, we modified the first convolutional layer to accept a single channel. The encoder outputs five feature maps from stages with channel depths ranging from 40 to 384. Although a larger variant such as EfficientNet-B7

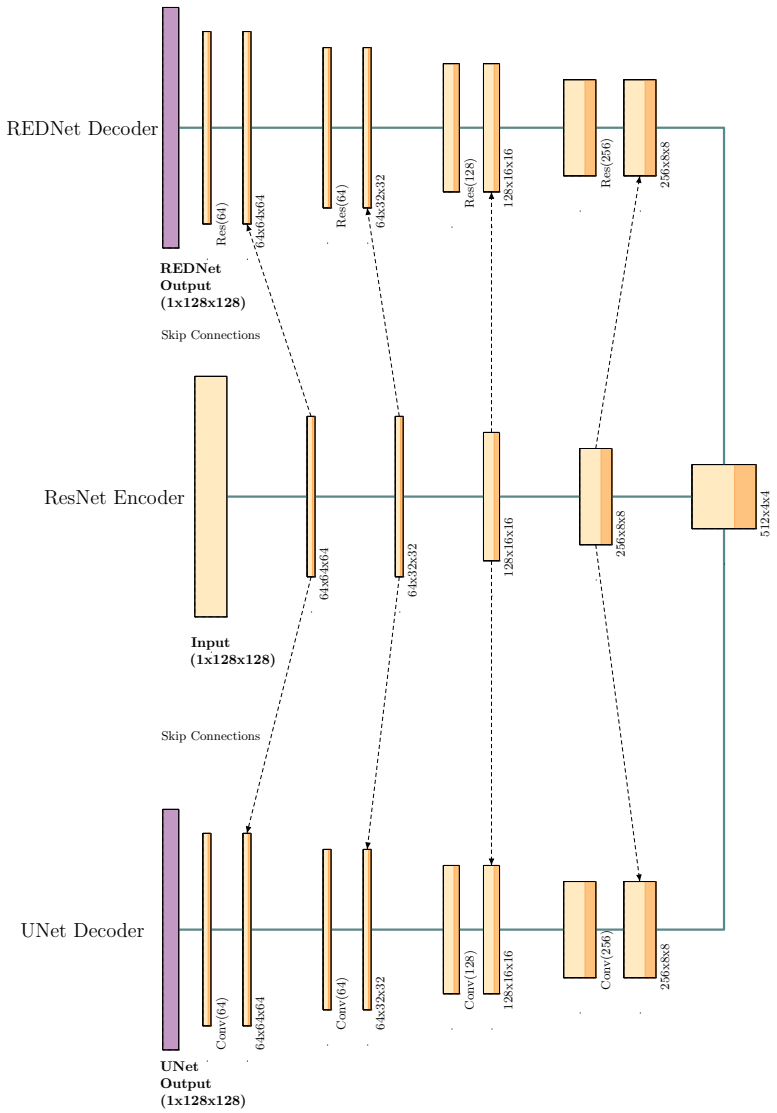

Figure 4.4: ResNet Encoder structure showing connection to either REDNet or UNet.

[2] could increase performance, it would also require significantly more computational resources, making EfficientNet-B3 the more suitable choice in this context.

- **EfficientNet-UNet** follows the same decoding logic as the ResNet-UNet variant, using concatenation and attention at each decoding stage.
- **EfficientNet-REDNet** instead uses the REDNet-style residual decoder with element-wise additive attention gates and deeper residual fusion blocks, batch normalisation is applied after each convolutional layer.

These EfficientNet-based models emphasise lightweight encoding to assess the effect of architectural efficiency on the quality of the reconstruction.

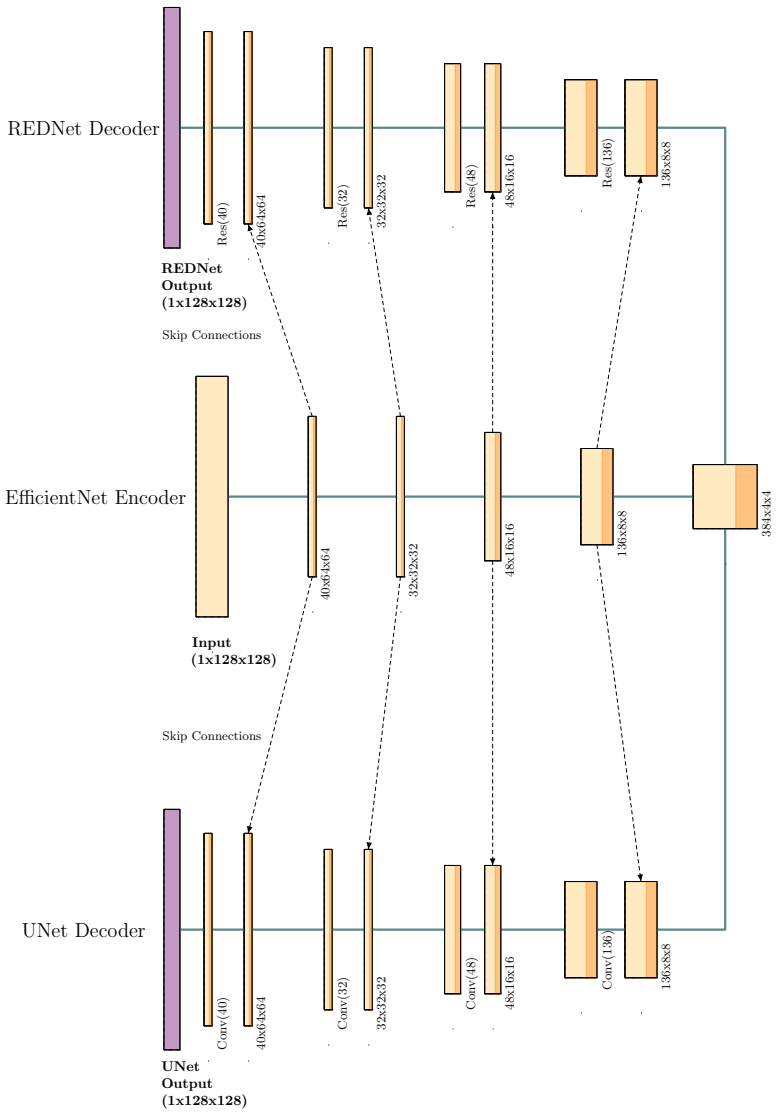

Figure 4.5: EfficientNet Encoder structure showing connection to either REDNet or UNet.

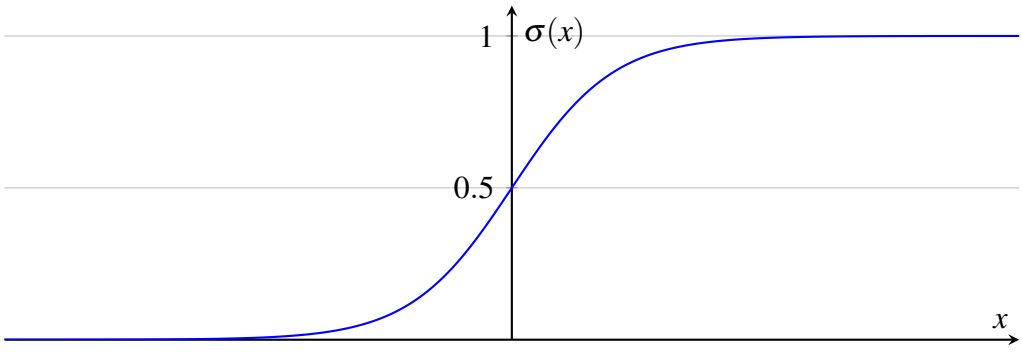

Figure 4.6: Sigmoid function $\sigma(x) = \frac{1}{1+e^{-x}}$

4.5.7 Activation Function

A final sigmoid activation function was applied at the decoder output layer in all models. As the objective was to reconstruct arbitrary continuous intensity distributions, rather

than discrete class labels, the use of the sigmoid function allowed the pixel intensities to be mapped into the [0,1] interval. This facilitated a smooth gradient flow during training, better reflecting the continuum of real-valued optical intensities being measured. The sigmoid function is visualised in figure 4.6

4.5.8 Optuna

Optuna is a Python library that automates hyperparameter optimisation for training deep neural networks. We used it to fine-tune key parameters such as the learning rate, number of epochs, optimiser choice, and batch size. For each diffuser grit value (1500, 600, 220, 120), we created an Optuna "study" that iteratively trained the CNN model, using the performance of previous trials to guide subsequent hyperparameter selections. After a fixed number of trials, we recorded the best-performing hyperparameter configuration, which was then used in the main training phase. Performance for Optuna studies was evaluated using the L1 loss function (MAE) (2.5) with lower values indicating better performance. Optuna studies also have a 'search space' which dictates valid ranges of values for given hyperparameters; these were as follows:

Hyperparameter	**Minimum**	**Maximum**
Learning Rate	1×10^{-5}	1×10^{-3}
Optimizer	Adam or SGD	Adam or SGD
Batch Size	4	12

Table 4.1: The search space used in the optuna case studies.

Given the search space in our Optuna study, the ideal hyperparameters found for each GRIT level were as follows:

GRIT Level	**Learning Rate**	**Optimizer**	**Batch Size**
120	4.4×10^{-4}	Adam	4
220	1.3×10^{-4}	Adam	4
600	1.97×10^{-4}	Adam	12
1500	9.3×10^{-4}	Adam	8

Table 4.2: Training hyperparameters used for each GRIT level.

4.5.9 Training Procedure

The size of the training set was increased in increments of 10,000 images (from 10,000 up to 60,000), for each architecture. This yielded multiple trained models per architecture-dataset combination." or combine the sentence with the previous one for grammatical completeness. All input images were normalised from [0, 255] to [0, 1].

Training primarily used the L1 loss function for binary and greyscale images, and a negative pearson correlation coefficient (NPCC) loss was additionally explored for the maximum dataset size considered (60,000 images) to compare the best performances between L1 and NPCC. NPCC was only evaluated at 60,000 images due to the increased computation time that came with opting for the NPCC loss function over the L1 loss function. The Adam optimiser was selected because of its constant superior performance compared to other optimisers such as SGD (Stochastic Gradient Descent) in the Optuna tests. Two strategies, 'early stopping' and 'reduce on plateau', were employed to mitigate overfitting and improve convergence. 'Early stopping' halts training if validation performance does not improve for a specified number of epochs (we used a patience value of 8), thereby preventing unnecessary computation once progress stalls. 'Reduce on plateau' lowers the learning rate after a set number of unimproved epochs (we used a patience value of 3), allowing the optimiser to settle into a potentially better local minimum. At the end of each epoch, the model was tested on the loss function on which it was trained, and the epoch that produced the lowest validation loss was saved. Training was capped at a maximum of 20 epochs to limit the computation time for other models to run.

The pseudocode for the training procedure can be seen in Algorithm 2

4.6 Testing Models

The model evaluation was carried out by comparing the reconstructed images of each network with the corresponding ground-truth targets using the Structural Similarity Index Measure (SSIM) and the Learned Perceptual Image Patch Similarity (LPIPS).

The Structural Similarity Index Measure (SSIM) was initially chosen because it offers an interpretable standalone measure of how closely two images match in terms of luminance, contrast, and structure, independent of any specific loss function used during training. SSIM was attractive for its relative simplicity, popularity in image quality assessment, and its ability to highlight structural differences between reconstructions and ground truth targets through the SSIM map. (See Subsection 2.9.1)

However, SSIM alone was insufficient to capture perceptual quality, particularly in cases where minor structural distortions did not align with human visual judgment. To address this, the Learned Perceptual Image Patch Similarity (LPIPS) metric was introduced as a secondary evaluation metric. LPIPS uses deep neural network features to assess perceptual similarity, providing a second opinion more attuned to human visual perception. The AlexNet variant of LPIPS was selected for its faster computation time while maintaining robust perceptual scoring. The selection of AlexNet for the evaluation of the LPIPS does not influence the validity of our metric, despite the fact that AlexNet itself is not used

Algorithm 2 CNN Model Training

```
1:  function MAIN
2:      Parse input arguments (architecture, loss type, dataset settings)
3:      Initialize dataset paths and data loaders
4:      Instantiate model and loss function
5:      Setup optimizer and learning rate scheduler
6:      Train model with TRAINMODEL
7:      Save best-performing model
8:  end function
9:  function TRAINMODEL(model, train_loader, test_loader, optimizer, loss, device)
10:     Initialize training history and early stopping counters
11:     for each epoch do
12:         Train model on training set
13:         Evaluate model on validation set
14:         Update learning rate if needed
15:         if validation loss improves then
16:             Save current model state
17:             Reset early stopping counter
18:         else
19:             Increment early stopping counter
20:             if patience exceeded then
21:                 Stop training early
22:             end if
23:         end if
24:     end for
25:     Save loss curves and metrics to files
26: end function
27: function VALIDATEMODEL(model, dataloader, loss, device)
28:     Evaluate model on validation set without gradient computation
29:     Return average validation loss
30: end function
```

in our trained models. LPIPS, as a generalised perceptual metric, remains robust and meaningful regardless of the specific CNN architectures used during model training. (See Subsection 2.9.2)

Alternative metrics such as the Peak Signal-to-Noise Ratio (PSNR) and the Mean Squared Error (MSE) were considered, but ultimately were not chosen. These metrics heavily penalise minor pixel errors and do not correlate well with perceived image quality, particularly in the context of diffused or partially reconstructed images. SSIM and LPIPS together were judged to provide a balanced evaluation, measuring both the structural precision and the perceptual realism of the reconstructions.

To prevent SSIM from rewarding the background of our images, a mask was implemented

to focus exclusively on the ground-truth image and the reconstructed image, thus rewarding accurate reconstruction and penalising background artefacts.

For each model, testing was performed on a subset of 1,000 EMNIST images that were held for evaluation. These test samples underwent the same preprocessing pipeline as during training, including normalisation from [0,255] to [0,1]. LPIPS separately expects a range of values between [-1,1], when using LPIPS as an evaluation metric, the data were scaled accordingly for this calculation.

The pseudocode for the evaluation process can be seen in Algorithm 3

Algorithm 3 Evaluate CNN Models

```
Input: Directory of models, dataset root paths, LPIPS model, device (CPU/GPU)
Output: CSV file containing average SSIM and LPIPS per model
function MAIN
    Load LPIPS model with AlexNet backbone
    Set output path and create evaluation CSV file
    for all model files in the directory do
        if file is a .pth model file then
            Parse architecture, GRIT value, and greyscale flag from filename
            Call EVALUATEMODEL(grit_value, greyscale, model_path, lpips_alex, device)
            if evaluation returned valid metrics then
                Write average SSIM and LPIPS to CSV
            end if
        end if
    end for
end function
function EVALUATEMODEL(grit_value, greyscale, model_path, lpips_alex, device)
    Select appropriate dataset directories based on GRIT and greyscale
    Parse architecture and instantiate corresponding model
    Load model weights and set model to evaluation mode
    Create dataset and dataloader
    Initialize total_ssim and total_lpips to 0
    for all (diffused, clean) pairs in dataloader do
        Run model on diffused input to get output
        Create binary masks from clean and output using a fixed threshold
        Combine masks using logical OR for evaluation region
        Compute SSIM on combined mask using SSIM(output, clean, mask)
        Compute LPIPS using COMPUTELPIPS(lpips_alex, output, clean)
        Accumulate SSIM and LPIPS scores
    end for
    Compute averages over all samples
    return architecture, average SSIM, average LPIPS
end function
function SSIM(img1, img2, mask)
    Create Gaussian window
    Convolve window to get local means and variances
    Compute SSIM map from luminance, contrast, and structure components
    Apply mask and return mean SSIM
end function
function COMPUTELPIPS(model, img1, img2)
    Convert grayscale images to 3-channel format
    Compute LPIPS distance and return mean score
end function
```

Chapter 5

Results

The results presented will firstly go over reconstruction results, what the best architectures are for each grit and image type, how dataset size affects reconstruction, and how loss function affects reconstruction. after that statistical analysis such as ANOVA and correlation analysis is performed. Finally, some statements will be made on the evaluation metrics based on the prior analyses.
A total of 224 (112 binary, 112 greyscale) different models are evaluated each with different architectures, loss functions, and dataset sizes.
Note that LPIPS is minimised (0 is best), whereas SSIM is maximised (1 is best).

5.1 Reconstruction Results

5.1.1 Best Models

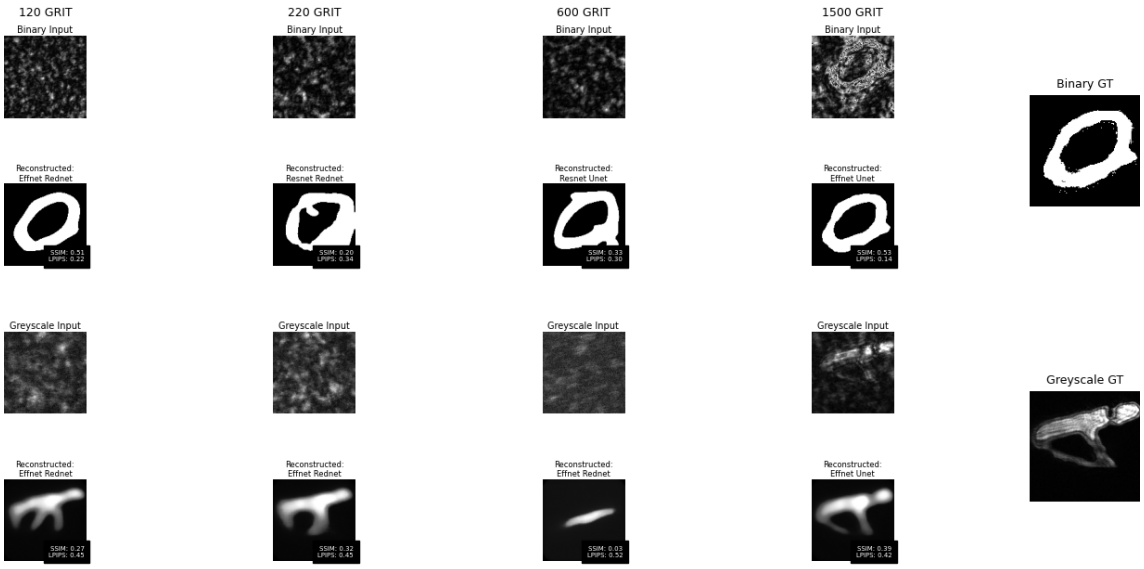

Figure 5.1: The best reconstructions and their architectures that we got over the experiment for both greyscale and binary images.

In general, it seems that **EffNetREDNet** is the best model out of the four that we have used to reconstruct images through disordered media. It can be clearly seen that at 1500

GRIT, there is a significant increase in the performance of the reconstructions. This is because the image still retains some structure even after travelling through the 1500 GRIT diffuser. The other GRIT diffusers produce similar results, although it seems that 120 GRIT produces the best result after 1500 GRIT. This is counter-intuitive, from a physics perspective we would assume that less scattering (i.e. higher GRIT levels) would mean better reconstruction but this observation does not reflect that. It would seem that once the spatial structure that reflects the ground-truth image like in 1500 GRIT has gone, then there is a degradation in performance for all GRIT levels less than that point. The improved performance at 120 GRIT may also be due to variations in the optical elements themselves, for example, the 120 GRIT diffuser may be cleaner or have fewer particulates than the others, which could lead to better input quality and, consequently, better reconstructions. There is also a severe lack of decent reconstruction for greyscale at 600 GRIT, this is due to the models not learning at all for this dataset when training. This may be due to data inconsistency when collecting the data for this dataset or some other fault with the data collection process.

5.1.2 Variation of Dataset Size

In order to see how effectively our models were able to reconstruct images, the models were trained on a varying dataset size and performance evaluated.

Figure 5.2: Comparison of reconstructed images of binary images across four CNN architectures trained on varying dataset sizes using the L1 loss function and a randomly sampled 120 GRIT diffused image from the testing set. Each cell shows the reconstructed image along with its corresponding SSIM and LPIPS metrics. The input diffused image (left) and the clean ground truth (right) are shown for visual reference.

As shown in Figure 5.2, all models exhibit improved reconstruction performance as the dataset size increases, which is consistent with expectations. At 10,000 images, the reconstructed outputs show some structural features, but do not resemble the actual characters.

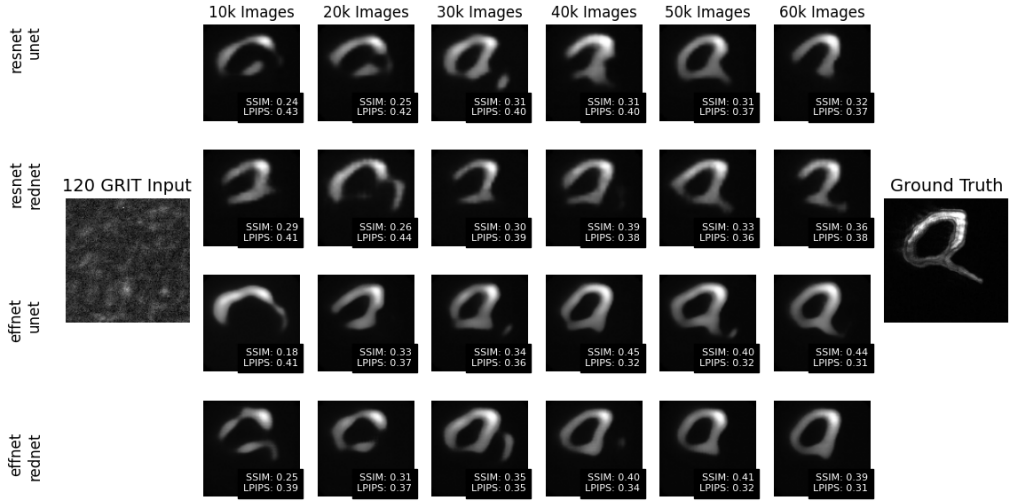

Figure 5.3: Comparison of reconstructed images of greyscale images across four CNN architectures trained on varying dataset sizes using the L1 loss function and a randomly sampled 120 GRIT diffused image from the testing set. Each cell shows the reconstructed image along with its corresponding SSIM and LPIPS metrics. The input diffused image (left) and the clean ground truth (right) are shown for visual reference.

By 40,000 images, the reconstructions begin to look more like the ground-truth image, suggesting that increased data volume enhances the models' ability to learn complex structures. Notably, visual clarity improves significantly with 60,000 training samples, particularly in models like EffNetREDNet.

Similarly, Figure 5.3 illustrates a visual improvement in the reconstruction of greyscale images as the size of the dataset increases. Among the models, EffNetUNet clearly produces the most accurate visual output at 60,000 images, outperforming ResNetUNet, although the metric differences between them remain surprisingly narrow. Interestingly, at 40,000 images the SSIM score for EffNetUNet is 0.45 which is higher than the result at 60,000 images. However, the LPIPS is slightly higher (worse performance) for that reconstruction compared to the final model, indicating that the evaluation metrics do not always agree with each other.

Another interesting point is that the spread of LPIPS values is smaller than in SSIM values, suggesting that SSIM performance increases faster by increasing the dataset size than LPIPS performance.

5.1.3 Variation of Loss Function

We compared the L1 loss function to the NPCC loss function using a dataset size of 60,000 images and reconstruction scores in tables being averaged over the 1000 image testing dataset. We wanted to see how much of an impact the loss function can have at producing an accurate reconstruction of the image using the LPIPS and SSIM evaluation

metrics 2.9 when given the same set of conditions.

Table 5.1: L1 - Binary

Architecture	EffNetREDNet		EffNetUNet		ResNetREDNet		ResNetUNet	
Metric	LPIPS	SSIM	LPIPS	SSIM	LPIPS	SSIM	LPIPS	SSIM
GRIT								
120	0.223	0.357	0.239	0.334	0.227	0.350	0.229	0.346
220	0.334	0.265	0.359	0.212	0.354	0.230	0.372	0.216
600	0.315	0.280	0.302	0.283	0.304	0.287	0.299	0.292
1500	0.149	0.556	0.147	0.549	0.163	0.507	0.154	0.541

Table 5.2: NPCC - Binary

Architecture	EffNetREDNet		EffNetUNet		ResNetREDNet		ResNetUNet	
Metric	LPIPS	SSIM	LPIPS	SSIM	LPIPS	SSIM	LPIPS	SSIM
GRIT								
120	0.664	0.164	0.689	0.172	0.669	0.180	0.691	0.164
220	0.714	0.119	0.797	0.068	0.720	0.128	0.823	0.063
600	0.681	0.163	0.696	0.165	0.697	0.167	0.763	0.077
1500	0.654	0.224	0.681	0.155	0.664	0.224	0.675	0.203

For binary image the L1 loss function performs consistently better than the NPCC loss function across all conditions, both architecture differences and GRIT differences according to the evaluation metrics used.

The spread or variance of the evaluation metric values is noticeably smaller for L1 than for NPCC, suggesting more stable and predictable performance.

For L1 loss, performance is relatively architecture invariant. Similar results are achieved regardless of the network used.

The ResNetREDNet architecture consistently produces better results in terms of the SSIM for NPCC loss whereas EffNetREDNet architecture seems to produce better results in terms of the LPIPS for NPCC, it would seem from this result that the REDNet decoder atleast produces minor improvements over UNet for reconstructing these diffused images when using the NPCC loss.

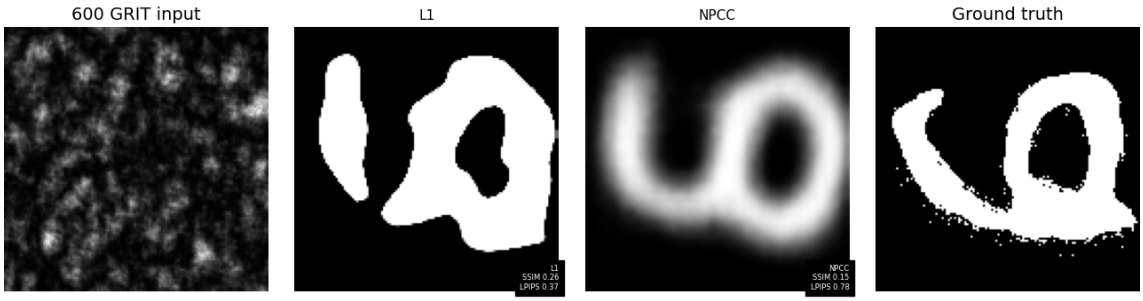

Figure 5.4: Comparison of an NPCC vs L1 image reconstruction on a randomly selected 600 GRIT binary image.

As shown in Figure 5.4, the L1 loss produces sharper, more defined reconstructions, while NPCC outputs are noticeably blurrier. Interestingly, while NPCC visually resembles the ground truth more closely in shape and topology, L1 achieves better SSIM and LPIPS values - likely due to its stronger, more confident edge predictions.

This illustrates the complexity of evaluating image reconstruction: metrics like SSIM and LPIPS do not always align with perceptual accuracy, especially when comparing sharp vs. soft reconstructions.

Table 5.3: L1 - Greyscale

Architecture	EffNetREDNet		EffNetUNet		ResNetREDNet		ResNetUNet	
Metric	LPIPS	SSIM	LPIPS	SSIM	LPIPS	SSIM	LPIPS	SSIM
GRIT								
120	0.357	0.351	0.364	0.356	0.401	0.349	0.407	0.344
220	0.350	0.355	0.369	0.338	0.394	0.358	0.405	0.351
600	0.466	0.064	0.484	0.068	0.507	0.069	0.512	0.070
1500	0.343	0.345	0.341	0.390	0.391	0.312	0.392	0.313

Table 5.4: NPCC - Greyscale

Architecture	EffNetREDNet		EffNetUNet		ResNetREDNet		ResNetUNet	
Metric	LPIPS	SSIM	LPIPS	SSIM	LPIPS	SSIM	LPIPS	SSIM
GRIT								
120	0.692	0.160	0.670	0.197	0.501	0.300	0.511	0.313
220	0.561	0.270	0.719	0.146	0.492	0.315	0.722	0.149
600	0.761	0.119	0.776	0.125	0.550	0.116	0.855	0.096
1500	0.556	0.284	0.667	0.201	0.477	0.359	0.490	0.350

Again, according to the SSIM and LPIPS metrics, the L1 loss function performs much better than the NPCC loss function across all architectures and GRIT levels for greyscale images.

It is noticeable that the architectures struggled to reconstruct images at 600 GRIT across both loss functions, as mentioned previously in Subsection 5.1.1. This may be indicative of an inconsistency in data collection or that the dataset was too small for the networks to learn.

Like binary it seems that for NPCC loss, that ResNetREDNet performs atleast slightly better than other architectures, for L1 loss the architecture does not seem to make a major difference in image reconstruction.

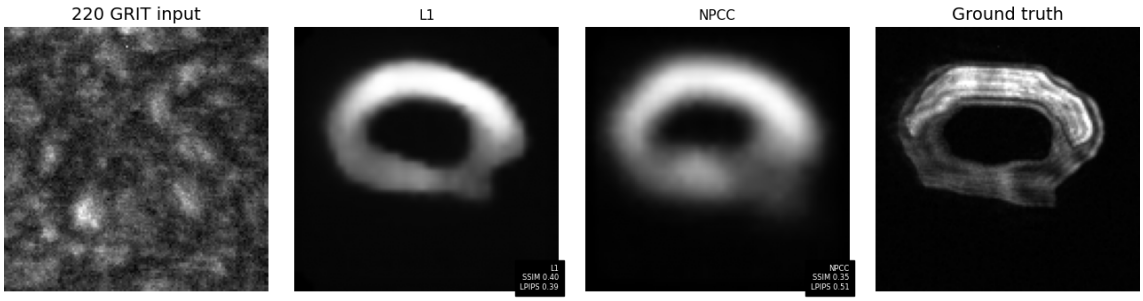

Figure 5.5: Comparison of an NPCC vs L1 image reconstruction on a randomly selected 120 GRIT greyscale image.

Figure 5.5 shows that the reconstructed image using the L1 loss function is sharp and clearly defined, closely matching the edges and stroke density of the ground truth. In contrast, the NPCC loss function reconstruction appears smoother and blurrier, with weaker

edge definition. Although NPCC loss may visually appear more fluid or natural at first glance, it lacks the confident edges necessary to achieve high SSIM and low LPIPS scores, like the binary images, meaning this applies to any image no matter the complexity.

5.2 ANOVA

In order to estimate how much various parameters affect the final SSIM and LPIPS values ANOVA (Analysis of Variance) [27] can be used. This allows us to provide a statistical justification for the trends observed in Section 5.1

For the ANOVA statistics, we define a null hypothesis as follows:
There is no significant of the model parameters (architecture, loss function, GRIT level, dataset size) on the SSIM and LPIPS values.
With an alternate hypothesis of:
At least one of the parameters that we are testing will have a significant impact on the SSIM and LPIPS performance values.
Using the **statsmodel** and **pandas** package for python, we can easily parse a CSV file and run an ANOVA study on the data collected when evaluating our models.
In order to determine whether a factor is statistically significant, we will define a threshold p-value of $p < 0.05$, this value indicates that there is a less than 5% chance that the result is due to random variation.

Source	Sum Sq	df	F	p-value
C(Architecture)	0.0011	3	0.039	9.90e-01
C(Dataset Size)	0.1549	5	3.277	8.47e-03
C(Image Type)	0.0283	1	2.993	8.64e-02
C(Training Time)	6.3832	111	6.081	1.91e-19
C(Loss Function)	0.2491	1	26.346	1.23e-06
C(GRIT)	1.2828	3	45.219	3.59e-19
Residual	1.0496	111	—	—

Table 5.5: ANOVA table for SSIM results showing the affect of various factors and their interactions.

For SSIM, the following parameters have $p < 0.05$:

- Dataset Size - Dataset size has an affect on SSIM performance
- Training Time - Time taken to train severely affects SSIM performance
- Loss Function - Chosen Loss function greatly affects SSIM performance
- GRIT level - The GRIT level chosen has a severe affect on SSIM performance

Source	Sum Sq	df	F	p-value
C(Architecture)	0.0383	3	1.854	1.42e-01
C(Dataset Size)	0.0432	5	1.252	2.90e-01
C(Image Type)	0.1127	1	16.346	9.77e-05
C(Training Time)	6.0497	111	7.905	3.33e-24
C(Loss Function)	1.8093	1	262.437	5.14e-31
C(GRIT)	0.6595	3	31.888	6.06e-15
Residual	0.7653	111	—	—

Table 5.6: ANOVA table for LPIPS results showing the affect of various factors and their interactions.

For LPIPS, the following parameters have $p < 0.05$:

- Image Type - Different image types (Binary or Greyscale) has an affect on LPIPS Performance
- Training Time - Time taken to train severely affects LPIPS performance
- Loss Function - Chosen Loss function severely affects LPIPS performance
- GRIT level - The GRIT level chosen has a severe affect on LPIPS performance

For the most part, SSIM and LPIPS agree with each other using ANOVA, which is a good indicator that the evaluation metrics are working as intended. Furthermore, the ANOVA study agrees with the impact that the loss function has and the impact that the GRIT level has on the SSIM and LPIPS performance, which further ensures the results retrieved.

Both evaluation metrics (SSIM and LPIPS) agree that the architecture used is not statistically significant for their performance metrics. However, it should be noted that the tested architectures share similar encoder structures (ResNet and EfficientNet backbones), which may reduce observable differences. This suggests that within the limits of this study and the specific architectures evaluated, different designs produce broadly comparable results.

In particular, the ANOVA analysis reveals that for LPIPS, the size of the data set does not exert a statistically significant impact on performance. This finding contradicts the conventional assumption that increasing the size of the dataset would enhance performance. As discussed in Section (5.1.2), the variance of LPIPS is observed to be less than that of SSIM when the size of the dataset varies, thus validating the ANOVA result obtained.

The relatively low residual sum of squares per degree of freedom for both SSIM and LPIPS ($\frac{1.0496}{111} = 0.009$ and $\frac{0.7653}{111} = 0.007$, respectively) indicates that the ANOVA model captures a substantial proportion of the variance, confirming the explanatory power of the selected factors.

5.3 Correlation Analysis

A correlation analysis was also performed on the results; the intention behind the correlation analysis is to determine whether adding more, or less of a certain parameter such as the size of the dataset will correlate to an increase or decrease in performance.

A concise way to visualise correlation in a multi-parameter system is through the use of a heatmap, the parameters we are looking to correlate are placed on the X and Y axes, and we change the colour of a tile representing those two parameters to represent some relationship. In our correlation heatmaps, a 1 represents a direct correlation such that $y \propto x$, a 0 represents no relationship between the parameters, and a -1 represents an inverse relationship such that $y \propto \frac{1}{x}$.

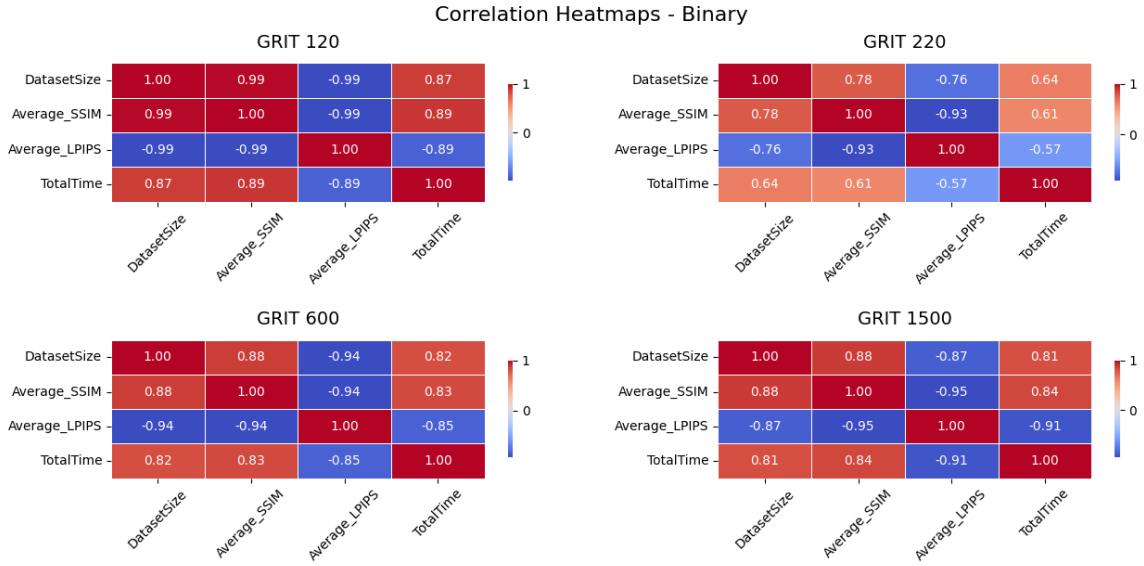

Figure 5.6: A Heatmap describing the Correlation between various parameters used for Binary image reconstruction. (1 indicates a direct correlation, 0 indicates no correlation and -1 indicates an inverse correlation)

Binary images (Figure 5.6) show strong correlations at all GRIT levels. In particular, data set size shows a consistent positive correlation with SSIM and a strong negative correlation with LPIPS. This suggests that increasing the number of training samples reliably improves the quality of the reconstruction as measured by both metrics.

In contrast, the correlations for greyscale images (Figure 5.7) are weaker and less consistent. Although there is some positive relationship between DatasetSize and SSIM, particularly at lower GRITs, this relationship weakens significantly at 600 GRIT and 1500 GRIT. This could suggest diminishing returns or delayed learning curves in the case of greyscale data. However, as discussed in Section 5.1, it is more likely that these weaker correlations reflect limitations in the SSIM and LPIPS metrics rather than the model performance. A particular interesting point is 600 GRIT, where almost no relationship is shown; this is because this model struggled to reconstruct images well at all at this GRIT

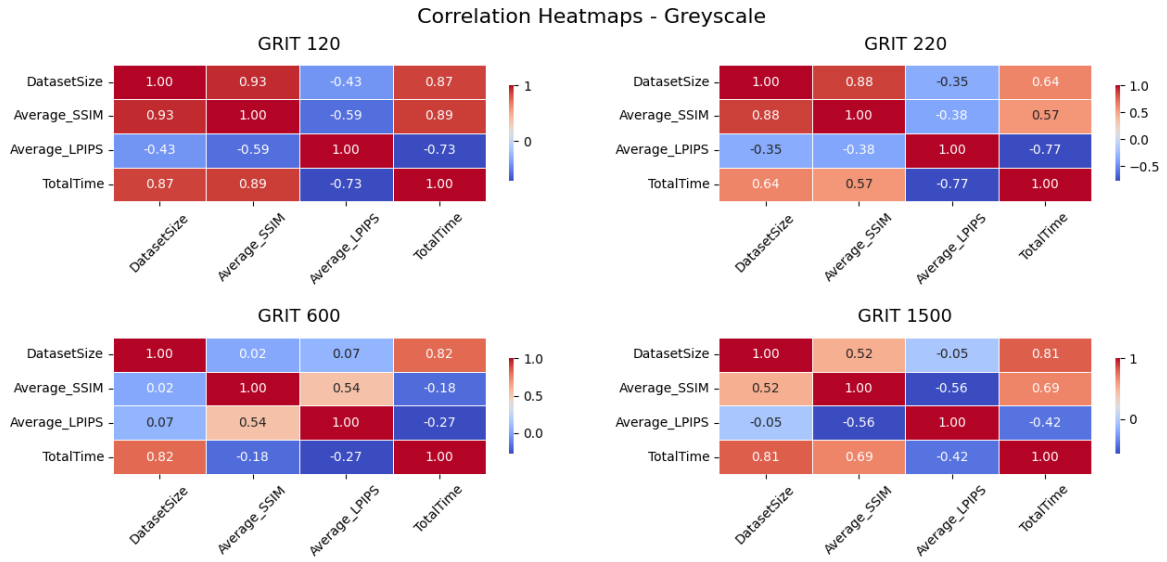

Figure 5.7: A Heatmap describing the Correlation between various parameters used for greyscale image reconstruction. (1 indicates a direct correlation, 0 indicates no correlation and -1 indicates an inverse correlation)

level for greyscale images, as mentioned in Section 5.1, meaning that no matter how much we increased the dataset size, performance was not affected.

5.4 Evaluation Method Discrepancies

Although SSIM and LPIPS are widely used and are often reliable, several discrepancies were observed between their quantitative output and the perceived quality of reconstructions. This is particularly apparent in cases involving the NPCC loss function, where images appear visually smoother but blurrier, but receive significantly lower SSIM scores and higher LPIPS values compared to their L1 counterparts (see Section 5.1).

These discrepancies are most pronounced in greyscale reconstructions, where certain architectures such as EffNetUNet produce reconstructions that are visually more complete (see Figure 5.3), despite achieving metric scores comparable to models that fail to reconstruct parts of the image. This suggests that SSIM and LPIPS may not fully capture certain structural continuity.

As such, while these metrics are useful, care must be taken when interpreting them in isolation. They should not be trusted blindly.

Chapter 6

Summary & Conclusion

In conclusion, this dissertation has comprehensively evaluated convolutional neural network (CNN) architectures for reconstructing images distorted by disordered media, providing significant insight into optimal methodologies and architectures. Specifically, EfficientNet-REDNet emerged as the superior architecture for *most* use cases. Particularly at higher GRIT levels meaning less disordered scattering (e.g. 1500 GRIT), confirming its suitability for handling complex image reconstructions. Although it is important to note that EfficientNet-REDNet is only slightly better than its competitors, as shown in 5.2, the variation of architecture is statistically insignificant when it comes to varying the SSIM and LPIPS metrics.

The findings highlight critical dependencies on the size of the data set, the level of GRIT, and the chosen loss function, each significantly contributing to the quality of reconstruction. In particular, larger datasets improved model performance consistently across conditions, emphasising the need for extensive training data to achieve optimal CNN performance.

An important observation was the discrepancy between evaluation metrics, SSIM and LPIPS. SSIM and LPIPS tended to reward sharp, high-contrast reconstructions. This divergence was particularly apparent when comparing reconstructions derived from L1 and NPCC loss functions. The L1 loss function consistently produced sharper and more visually distinct reconstructions, aligning more closely with the SSIM evaluations and scoring lower LPIPS scores. In contrast, NPCC loss, although resulting in visually more complete reconstructions, typically scored lower on SSIM and higher on LPIPS metrics, although appearing to be more similar when compared empirically. Highlighting limitations within these metrics and suggesting caution in their isolated interpretation.

Furthermore, the ANOVA analysis underscored the statistical significance of all the parameters evaluated, notably the substantial impacts of the GRIT level and the choice of the loss function. Correlation analysis further reinforced these findings, illustrating clear relationships between dataset size and performance metrics, especially in binary image reconstructions.

The main challenge faced with a project like this is estimating the time it will take to fulfil all of the components within the project, maintaining consistency between images during data collection can be difficult, often requiring tweaks to the collection process to improve consistency across samples. Not to mention the time it takes to train the models themselves; care was taken to maximise efficiency, training binary models while simultaneously collecting greyscale images. The total time taken for training for all 224 models from the collected CSV files was 903 hours and 11 minutes or 37 **days**.

At the start of the project work was also done to try and implement a PINN (Physics Informed Neural Network) variant of a CNN for the model to train off of, however, this idea was abandoned due to issues getting sensible image reconstructions and due to the time constraints and computation issues.

Some other work was done with a 'Fourier Transform' loss function, using the fast fourier transform function (**fft2**) in **pytorch** to evaluate the frequency domain between the ground truth and the attempted reconstruction at each epoch. However, this also did not result in any intelligible reconstructions of any value, which led this to be not included in the dissertation.

Future research should address the observed metric discrepancies, perhaps exploring alternative or hybrid evaluation methods that better align with human visual perception. Additionally, further investigation into the inconsistencies observed at the 600 GRIT level could clarify underlying causes and optimise data collection procedures by validating optical alignment. It would also be of interest to investigate other deep learning techniques for image reconstruction, such as RNN's (Recurrent Neural Networks) or a successful implementation of some sort of PINN (Physics-Informed Neural Network).

Overall, this dissertation contributes valuable empirical evidence that guides future CNN implementations for image reconstruction through disordered media, setting a foundation for continued research and practical application in computational optics and related fields.

Appendix A

Appendices

A.1 User Manual

A.1.1 Installing Prerequisite software.

For data collection the **Thorcam**, **ALP4.3** and **Python 3.12** and software must be installed onto the target computer. For ease of use, the links to download these software applications have been provided by clicking on the respective linked text. The links for these are also provided within the **Installers** folder within the code submission.
The Thorcam software is used for camera control and camera calibration, and the ALP4.3 software is used for DMD control.

A.1.2 Installing Packages

In order to install the required packages to run all of the Python code you can run the batch file provided. **InstallDependencies.bat**
Normally a **requirements.txt** could be used, and for most of the packages this is the case. However, the Thorcam library for controlling Python has to be installed from a .zip file locally. Python's pip only supports absolute paths for locally installed packages, in order to get around this we use **%CD%** in batch language to get the current directory and append the relative path to this. The batch file solution should work for all Windows installations.
It is also important to note that the **ALP4.3** Python package looks for a fixed path in the ALP4.3 Driver installation folder, the folder called **ALP-4.3 API** MUST be renamed to **ALP-4.3 high-speed API** in order for the Python package to pick up the driver API.

A.1.3 Running the Data Collection Code

In order to run the Data Collection code, both the camera and DMD must be connected to the target PC.
The data collection code resides in the **Data Collection** folder. **CollectData.py** should be executed when collecting binary images, and **CollectDataGreyscale.py** should be executed when collecting greyscale images. The amount of images taken can be manipulated

by manipulating the bounds of the for loop within the respective Python files, up to the size of the EMNIST letters database (124,800 images).

A.1.4 Training a Network

The model training code resides in the **NNArchitectures** folder. A CNN model can be trained using the **TrainModel.py** Python file. The argparse Python package is used for modifying parameters such as the network architecture, loss function, and initial learning rate of the training procedure a summary of the available arguments are given in table A.1

Argument	Typical Value	Description
`--grit_value`	600	The GRIT value to train on.
`--cap`	10000	The amount of images to train on.
`--lr`	1e-4	The initial learning rate of training.
`--epochs`	5	The amount of epochs to train for.
`--batchsz`	4	The batch size used for training.
`--architecture`	effnet_unet	The architecture we are training on.
`--loss`	NPCC	The loss function to train on.
`--greyscale`	True/False	Whether we train on binary or greyscale images.

Table A.1: Summary of argparse arguments for TrainModel.py and typical values for these parameters.

From the root of the codebase submission, the model training code can be run with the following example arguments:

python ./NNArchitectures/TrainModel.py --grit_value 1500 --cap 10 --lr 1e-5 --epochs 20 --batchsz 16 --architecture "effnet_unet" --loss "NPCC" --greyscale

```
C:\Users\admin\OneDrive - University of Aberdeen\University Notes
\4th Year\Sem 2\Honours\Programming\DeepImage>python ./NNArchitec
tures/TrainModel.py --grit_value 1500 --cap 10 --lr 1e-5 --epochs
 20 --batchsz 16 --architecture "effnet_unet" --loss "NPCC" --gre
yscale
Loading training dataset...
Loading testing dataset...
Beginning training...
Epoch [1/20], Train Loss: 0.4602, Test Loss: 0.5462
```

Figure A.1: Initial output from example training run.

Once a model has been trained, it will be saved to the root of the directory with the following naming convention for binary models:

architecture_grit_cap_lossmethod.pth

and for greyscale models:

architecture_grit_cap_lossmethod_greyscale.pth

Various metrics such as training time and performance scores as a function of epoch are saved to a csv file using the same naming schema as the model in the **csv** folder.

A.1.5 Evaluating a Network

All evaluations use the metrics defined in Section 2.9.

To evaluate a single model, the Python script **TestModel.py** in the **NNArchitectures** folder can be used. To evaluate all models in the root directory that have been saved using the filename schema used in A.1.4, the **EvalModels.py** file can be executed in the **Eval** folder.

EvalModels.py requires no arguments, since it incrementally evaluates all the models found, the results of the evaluation are saved to the **evaluation_results.csv** file in the **csveval** folder.

TestModel.py supports the following input arguments:

Argument	Typical Value	Description
`--grit_value`	1500	The GRIT value.
`--model_path`	./resnet_unet_1500_l1_greyscale.pth	The path to the model.
`--greyscale`	True/False	Binary or Greyscale Images.
`--architecture`	resnet_unet	The architecture of the model.

Table A.2: Summary of argparse arguments for TestModel.py and typical values for these parameters.

```
Batch 980: SSIM = 0.250517, LPIPS = 0.368756
Batch 981: SSIM = 0.301127, LPIPS = 0.458030
Batch 982: SSIM = 0.322960, LPIPS = 0.404592
Batch 983: SSIM = 0.347103, LPIPS = 0.401694
Batch 984: SSIM = 0.301004, LPIPS = 0.426209
Batch 985: SSIM = 0.191260, LPIPS = 0.315119
Batch 986: SSIM = 0.284919, LPIPS = 0.385401
Batch 987: SSIM = 0.225147, LPIPS = 0.351480
Batch 988: SSIM = 0.175553, LPIPS = 0.344367
Batch 989: SSIM = 0.275623, LPIPS = 0.335557
Batch 990: SSIM = 0.250252, LPIPS = 0.331881
Batch 991: SSIM = 0.308080, LPIPS = 0.426492
Batch 992: SSIM = 0.336896, LPIPS = 0.378442
Batch 993: SSIM = 0.277370, LPIPS = 0.425430
Batch 994: SSIM = 0.354892, LPIPS = 0.380155
Batch 995: SSIM = 0.280823, LPIPS = 0.380230
Batch 996: SSIM = 0.202577, LPIPS = 0.307662
Batch 997: SSIM = 0.218532, LPIPS = 0.427972
Batch 998: SSIM = 0.183456, LPIPS = 0.398656
Batch 999: SSIM = 0.265726, LPIPS = 0.393459
Validation Completed.
Average SSIM:  0.300847
Average LPIPS: 0.392398  (lower = better)
```

Figure A.2: Initial output from example evaluation run.

As an example, you can run an evaluation as follows:

python ./NNArchitectures/TestModel.py –grit_value 1500 –model_path

"resnet_unet_1500_60000_l1_greyscale.pth" –greyscale –architecture "resnet_unet"

Figure A.2 shows the output from the **TestModel.py** code.

A.2 Maintenance Manual

A.2.1 Recommended Hardware Requirements

In order to train models, we recommend certain hardware in order to keep computation time to a minimum, the hardware recommended is as follows:

- A dedicated graphics card (e.g Nvidia RTX 2080)
- 16GB RAM

A.2.2 Folder Structure & Source Code Summaries

The file tree given in Figure A.3 shows how the project is composed and gives descriptions on key files and folders.

A.2.3 Software & Python Package Dependencies

A full list of the required software and Python packages is given below.

- **Python** - 3.12
  - **PyTorch** - 2.6.0
  - **Torchvision** - 0.21.0
  - **NumPy** - 2.1.3
  - **OpenCV** - 4.11.0.86
  - **Pillow (PIL)** - 11.1.0
  - **Matplotlib** - 3.10.1
  - **ALP4Lib** - 1.0.2 (for DMD control in Python)
  - **Scikit-Image** - 0.25.2
  - **SciPy** - 1.15.2
  - **Thorlabs Camera SDK** - *(No specified version)* (for CMOS camera control; manual installation required, see user manual subsection A.1.1)
  - **Optuna** - 4.3.0
  - **Pandas** - 2.2.3
  - **Seaborn** - 0.13.2 (for graph creation)

  - **Statsmodels** - 0.14.4 (for statistical analysis)

- **Thorcam** - (for camera preview and calibration prior to data collection)
- **ALP-4.3 Drivers** - (for DMD drivers)

A.3 Risk Assessment

In order to gain access and use the laser, a risk assessment had to be signed from the 'Lab supervisor' and myself. Figure A.4 shows the front page of the risk assessment.

A.4 Laser Safety Training

Before I could work with a laser within the university, I had to attend and pass a test on laser safety. Figure A.5 shows my certificate for working with lasers.

A.5 Induction

Finally, a health and safety induction had to be performed on my first day in the lab. Figure A.6 shows my induction form.

```
DeepImage/
├── csv/
│   └── *.csv                # Training CSV Data
├── csveval/
│   └── evaluation\_results.csv
├── Data Collection/         # Python scripts used for Data Collection
│   ├── archive/             # Various old testing or data collection scripts
│   │   ├── camera\_stream.py
│   │   ├── camera\_stream2.py
│   │   ├── camera\_test.py
│   │   ├── collect\_data\_animated\_rotation.py
│   │   ├── collect\_data\_animated\_translation.py
│   │   ├── collect\_data\_greyscale.py
│   │   ├── get\_data.py
│   │   └── get\_data\_combine.py
│   ├── dlls/                # Various Dynamic Link Libraries referenced by the Thorcam Pip Package
│   │   └── 64\_lib/
│   │       └── *.dll        # Dlls for Thorcam Pip Package
│   ├── x64/
│   │   └── alp4395.dll      # Dlls for ALP4FL Pip Package
│   ├── CameraSetup.py       # Sets up PATH for the Thorcam DLLS
│   ├── CollectData.py       # Collect Binary Images
│   ├── CollectDataGreyscale.py # Collect Greyscale Images
│   ├── DMDTest.py           # Test DMD Operation
│   └── LoadEMNIST.py        # Load EMNIST images
├── Eval/                    # Evaluation Code, creates various figures used in disseration
│   ├── Visualisations/      # Visualisations for Figures
│   │   ├── visualize\_bitplanes.py
│   │   ├── visualize\_cnn.py
│   │   ├── visualize\_conv.py
│   │   ├── visualize\_gradient.py
│   │   └── visualize\_SSIM.py
│   ├── ANOVA.py
│   ├── BestModelComparison.py
│   ├── CreateDatasetGraphs.py
│   ├── CreateImageSample.py
│   ├── CreateL1vsNPCCGraphs.py
│   ├── EvalModels.py        # Main Python Evaluation Code for Models
│   ├── GRITDemo.py
│   ├── Heatmap.py
│   ├── NPCCvsL1Reconstruction.py
│   └── TrainTime.py
├── Figures/                 # Holds Matplotlib Training performance figures for binary images as a function of epoch
│   └── *.png
├── FiguresGreyscale/        # Holds Matplotlib Training performance figures for greyscale images as a function of epoch
│   └── *.png
├── Installers/              # Installers for various required software and some notes.
│   ├── ALP4.3 Software Driver Install.url
│   ├── ALP4LIB Python Lib.txt
│   ├── Python 3.12 Install.url
│   ├── Thorcam Software Install.url
│   └── thorlabs\_tsi\_camera\_python\_sdk\_package.zip
├── NNArchitectures/
│   ├── Archive/             # Unused NN Architectures or Loss Functions used for testing.
│   │   ├── BCEDice.py
│   │   ├── CLAHETransform.py
│   │   ├── CombinedLoss.py
│   │   ├── DataFeeder.py
│   │   ├── evaluate.py
│   │   ├── FourierLoss.py
│   │   ├── HybridVGGNetUNetModel.py
│   │   ├── PICNNModel.py
│   │   ├── PICNNModelHelmholtz.py
│   │   ├── reconstruct\_video.py
│   │   ├── ResNetUNetDebug.py
│   │   ├── SSIML1.py
│   │   └── VGGNetModel.py
│   ├── Decoders/            # Decoder Network Definitions
│   │   ├── REDNet.py
│   │   └── UNet.py
│   ├── Encoders/            # Encoder Network Definitions
│   │   ├── EfficientNet.py
│   │   └── ResNet.py
│   ├── Optuna/              # Optuna Study script and outputs
│   │   ├── Hyperparameter.py
│   │   ├── Optuna - 10000.txt
│   │   └── optuna.txt
│   ├── DiffusionDataset.py   # Custom Class to store Images for Model Training
│   ├── HybridEfficientNetREDNetModel.py  # Combines Encoders and Decoders
│   ├── HybridEfficientNetUNetModel.py
│   ├── HybridResNetREDNetModel.py
│   ├── HybridResNetUNetModel.py
│   ├── NPCCLoss.py          # Define NPCC Loss
│   ├── TestModel.py         # Test single model script
│   └── TrainModel.py        # Train model script
├── Utils/                   # Various utility python scripts used at the start of the project before data collection
│   ├── image\_diffuser.py
│   ├── image\_to\_array.py
│   ├── load\_emnist.py
│   ├── mm\_scrambler.py
│   ├── mode\_scrambling.py
│   ├── move\_files.py
│   ├── scatter.py
│   ├── speckled\_scrambling.py
│   └── transmission\_matrix.py
├── .gitignore
├── CS165MU.json
├── CS165MU\_Greyscale.json
└── *.pth                    # Models saved from TrainModel.py
```

Figure A.3: File tree of the project code, brief descriptions are given to key code files.

School of Engineering Risk Assessment Page 1 of 3

Risk Assessment

PROCEDURE:
- Complete risk assessment in consultation with PI/Supervisor and technical staff as appropriate.
- Risk assessment checked and signed by PI/Supervisor
- A copy or scan of the signed document to be given to the lab technician, School Safety Adviser and PI/Supervisor.

NOTES:
- No laboratory work is to commence without a risk assessment signed by the PI/Supervisor.
- The risk assessment must be reviewed when any changes are made to the equipment, materials, procedure or personnel.
- Technical staff can stop work if no risk assessment is in place or if, in their opinion, there is a risk to safety.
- Examples of how to complete this form are available at www.hse.gov.uk/risk/casestudies/

Project name:	Optical beam shaping through disordered media		
Location of work:	FN Laser Lab		
Principal Investigator/Supervisor:	Prof. David McGloin	Signed:	Date: 27/1/25
Assessment Prepared by:	Matthew Robertson	Signed:	Date:28/01/25
Outline description of the work:	Use of a DMD SLM combined with AI to control the DMD to “unscramble” an image going through disordered media. The work will initially begin on preparatory steps such as: alignment process between a HeNe laser, DMD and CMOS camera, and interfacing with the DMD and CMOS camera. Once these steps have been completed, most of the work will be in software, developing the Neural Network, and collating data.		
Names of persons carrying out the work:	Matthew Robertson, David McGloin, Yan Bing		

16 April 2013

Figure A.4: Front Page of the Risk Assessment filled out to gain access to the laser lab

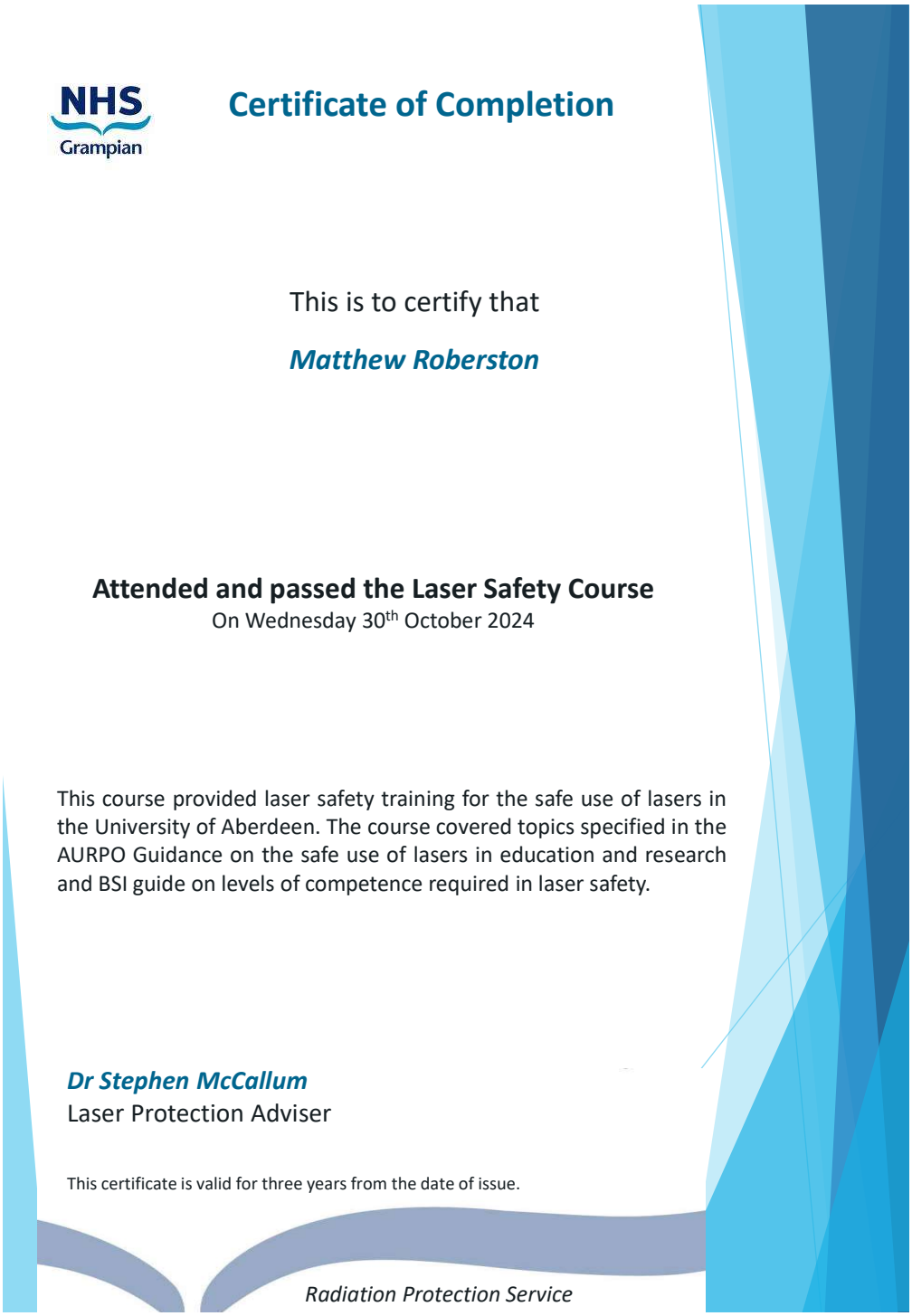

Figure A.5: My Laser Safety Training Certificate

SCHOOL of ENGINEERING

HEALTH AND SAFETY INDUCTION (LABORATORY)

Inductee Name	Matthew Robertson
Name of Supervisor	DAVID MCCLOY
Laboratory Name	OPTICS LAB

Reference may be made to the Safety Handbook	√
Safety Handbook Where it is kept in Lab. (http://www.abdn.ac.uk/engineering/about/safety-281.php) - online	☑
Risk Assessment (RA) These must be completed prior to commencing laboratory based work.	☑
Personal & Protective Equipment (PPE) Eye, ear, face, feet, body.	☑
Out of hours working in laboratory Not normally permitted consult supervisor. Include in risk assessment.	
Lone working in laboratories Not normally permitted consult supervisor. Include in risk assessment.	
Good Laboratory Practices Clean tidy work space.	☑
Spill Management (water, oils, chemicals) Contact technician, TRO or Spills Management Team.	☑
Chemicals Use PPE. Storage, Handling, MSDS, Ventilation, Disposal.	☐
Dust or Fumes Use PPE and/or LEV.	☑
Air lines, Pressurised Gasses and Pressure Equipment Training is required before use.	
Noise Use PPE. Contact TRO for Db tests.	☐
Machinery & Power Hand Tools Training is required before use. Use guards provided.	☐
Laser Equipment Training is required before use. Use PPE.	√
Electrical Equipment Inspect for damage before use and report problems. Equipment to be regularly tested for safety.	☑
Lab specific List of items specific to this laboratory.	☑
Fire Drill What to do, where to go	☑
Accidents/ Emergencies What to do, where to go	☑
	☐
	☐
	☐

Carried out by	DAVID MCCLOY	28/1/25
Inductee Signature		

Induction form.docx

Figure A.6: My Health and Safety induction for working in the laser lab.

Bibliography

[1] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *arXiv preprint arXiv:1512.03385*, 2015. URL: https://arxiv.org/abs/1512.03385.

[2] M. Tan and Q. V. Le. Efficientnet: Rethinking model scaling for convolutional neural networks. *arXiv preprint arXiv:1905.11946*, 2019. URL: https://arxiv.org/abs/1905.11946.

[3] O. Ronneberger, P. Fischer, and T. Brox. U-net: Convolutional networks for biomedical image segmentation. *arXiv preprint arXiv:1505.04597*, 2015. URL: https://arxiv.org/abs/1505.04597.

[4] X. Mao, C. Shen, and Y. Yang. Image restoration using very deep convolutional encoder-decoder networks with symmetric skip connections. *arXiv preprint arXiv:1603.09056*, 2016. URL: https://arxiv.org/abs/1603.09056.

[5] Roheen Qamar and Baqar Ali Zardari. Artificial neural networks: An overview. *Mesopotamian Journal of Computer Science*, August 2023. URL: https://www.researchgate.net/publication/373700317_Artificial_Neural_Networks_An_Overview, doi:10.58496/MJCSC/2023/015.

[6] Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel. Backpropagation applied to handwritten zip code recognition. 1989. URL: https://galileo-unbound.blog/wp-content/uploads/2025/02/lecun.neco_.1989.1.4.541.pdf.

[7] IBM. Convolutional neural networks. https://www.ibm.com/think/topics/convolutional-neural-networks. Accessed: 25 March 2025.

[8] Anuj Saini. Convolution and cross-correlation in cnn. https://www.geeksforgeeks.org/convolution-and-cross-correlation-in-cnn/, 2021.

[9] Geoffrey E. Hinton and Ruslan R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 2006. URL: https://www.science.org/doi/epdf/10.1126/science.1127647.

[10] Umberto Michelucci. An introduction to autoencoders. arXiv preprint arXiv:2201.03898, 2022. URL: http://arxiv.org/abs/2201.03898.

[11] Abouzar Choubineh, Jie Chen, Frans Coenen, and Fei Ma. A quantitative insight into the role of skip connections in deep neural networks of low complexity: A

case study directed at fluid flow modeling. *Journal of Computing and Information Science in Engineering*, 2023. URL: https://livrepository.liverpool.ac.uk/3158295/1/abouzarJCISE-2022.pdf, doi:10.1115/1.4054868.

[12] Babak Rahmani, Damien Loterie, Georgia Konstantinou, Demetri Psaltis, and Christophe Moser. Multimode optical fiber transmission with a deep learning network. *Light: Science Applications*, 2018. URL: https://www.nature.com/articles/s41377-018-0074-1.

[13] Changyan Zhu, Eng Aik Chan, You Wang, Weina Peng, Ruixiang Guo, Baile Zhang, Cesare Soci, and Yidong Chong. Image reconstruction through a multimode fiber with a simple neural network architecture. *Scientific Reports*, 2021. URL: https://www.nature.com/articles/s41598-020-79646-8.

[14] Shuai Li, Mo Deng, Justin Lee, Ayan Sinha, and George Barbastathis. Imaging through glass diffusers using densely connected convolutional networks. *Optica*, 2018. URL: https://doi.org/10.1364/OPTICA.5.000803.

[15] Nilesh Vijayrania. Different normalization layers in deep learning, 2020. URL: https://towardsdatascience.com/different-normalization-layers-in-deep-learning-1a7214ff71d6/.

[16] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2023. URL: https://arxiv.org/abs/1706.03762, arXiv:1706.03762.

[17] Taerim Yoon, Chang-Seok Kim, Kyujung Kim, and Jong-ryul Choi. Emerging applications of digital micromirror devices in biophotonic fields. *Optics and Laser Technology*, February 2018. doi:10.1016/j.optlastec.2018.02.005.

[18] Cuiling Gong and Tim Hogan. Cmos compatible fabrication processes for the digital micromirror device. *IEEE Journal of the Electron Devices Society*, 2014. doi:10.1109/JEDS.2014.2309129.

[19] J. Nilsson and T. Akenine-Möller. Understanding SSIM. *arXiv preprint arXiv:2006.13846*, 2020. URL: https://arxiv.org/abs/2006.13846.

[20] Richard Zhang, Phillip Isola, Alexei A. Efros, Eli Shechtman, and Oliver Wang. The unreasonable effectiveness of deep features as a perceptual metric. 2018. URL: https://arxiv.org/abs/1801.03924.

[21] Tomáš Čižmár and Kishan Dholakia. Shaping the light transmission through a multimode optical fibre: complex transformation analysis and applications in biophotonics. *Opt. Express*, 2011. URL: https://opg.optica.org/oe/fulltext.cfm?uri=oe-19-20-18871&id=222508.

[22] James R. Fienup. Phase retrieval algorithms: a comparison. *Applied Optics*, 1982. doi:10.1364/AO.21.002758.

[23] Jiawei Sun, Jiachen Wu, Nektarios Koukourakis, Liangcai Cao, Robert Kuschmierz, and Juergen Czarske. Real-time complex light field generation through a multi-core

fiber with deep learning. *Scientific Reports*, 2022. URL: https://www.nature.com/articles/s41598-022-11803-7.

[24] Deniz Mengu, Muhammed Veli, Yair Rivenson, and Aydogan Ozcan. Classification and reconstruction of spatially overlapping phase images using diffractive optical networks. *Scientific Reports*, 2022. URL: https://www.nature.com/articles/s41598-022-12020-y.

[25] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014. URL: https://arxiv.org/abs/1409.1556.

[26] Ozan Oktay, Jo Schlemper, Loic Le Folgoc, Matthew Lee, Mattias Heinrich, Kazunari Misawa, Kensaku Mori, Steven McDonagh, Nils Y. Hammerla, Bernhard Kainz, Ben Glocker, and Daniel Rueckert. Attention u-net: Learning where to look for the pancreas. *arXiv preprint arXiv:1804.03999*, 2018.

[27] Muhammad Hassan. Anova (analysis of variance). https://researchmethod.net/anova/.