APPROACHING PHASE RETRIEVAL WITH DEEP LEARNING

Dissertation

zur Erlangung des Grades eines

Doktors der Naturwissenschaften

der Technischen Universität Dortmund
an der Fakultät für Informatik

von

TOBIAS UELWER

Dortmund
2023

Tag der mündlichen Prüfung: 12.07.2023
Dekan: Prof. Dr. Gernot Fink
Gutachter: Prof. Dr. Stefan Harmeling
Prof. Dr. Timo Dickscheid

Abstract

Phase retrieval is the process of reconstructing images from only magnitude measurements. The problem is particularly challenging as most of the information about the image is contained in the missing phase. An important phase retrieval problem is Fourier phase retrieval, where the magnitudes of the Fourier transform are given. This problem is relevant in many areas of science, e.g., in X-ray crystallography, astronomy, microscopy, array imaging, and optics. In addition to Fourier phase retrieval, we also take a closer look at two additional phase retrieval problems: Fourier phase retrieval with a reference image and compressive Gaussian phase retrieval.

Most methods for phase retrieval, e.g., the error-reduction algorithm or Fienup's hybrid-input output algorithms are optimization-based algorithms which solely minimize an error-function to reconstruct the image. These methods usually make strong assumptions about the measured magnitudes which do not always hold in practice. Thus, they only work reliably for easy instances of the phase retrieval problems but fail drastically for difficult instances.

With the recent advances in the development of graphics processing units (GPUs), deep neural networks (DNNs) have become fashionable again and have led to breakthroughs in many research areas. In this thesis, we show how DNNs can be applied to solve the more difficult instances of phase retrieval problems when training data is available. On the one hand, we show how supervised learning can be used to greatly improve the reconstruction quality when training images and their corresponding measurements are available. We analyze the ability of these methods to generalize to out-of-distribution data. On the other hand, we take a closer look at an existing unsupervised method that relies on generative models. Unsupervised methods are agnostic toward the measurement process which is particularly useful for Gaussian phase retrieval. We apply this method to the Fourier phase retrieval problem and demonstrate how the reconstruction performance can be further improved with different initialization schemes. Furthermore, we demonstrate how optimizing intermediate representations of the underlying generative model can help overcoming the limited

range of the model and, thus, can help to reach better solutions. Finally, we show how backpropagation can be used to learn reference images using a modification of the well-established error-reduction algorithm and discuss whether learning a reference image is always efficient. As it is common in machine learning research, we evaluate all methods on benchmark image datasets as it allows for easy reproducibility of the experiments and comparability to related methods. To better understand how the methods work, we perform extensive ablation experiments, and also analyze the influence of measurement noise and missing measurements.

Acknowledgements

Writing this thesis would not have been possible without the help of many people.

First and foremost, I would like to thank Prof. Dr. Stefan Harmeling for supervising my thesis. I would like to thank him for his support, mentorship, and inspiration. I am grateful for the countless hours of brain-storming and the many writing sessions spent together. It was a great pleasure for me to work under his supervision. I would also like to thank Prof. Dr. Timo Dickscheid for his interest in my work and for reviewing my thesis. Many thanks to JProf. Dr. Thomas Liebig and Prof. Dr. Mario Botsch for agreeing to be part of my thesis committee.

Special thanks to my co-authors Sebastian Konietzny, Alexander Oberstraß, Nick Rucks, Tobias Hoffmann, and Stefan Harmeling for the fruitful collaboration, support with the experiments, and for allowing me to use parts of our work for this thesis.

Furthermore, I would like to express my gratitude to my colleagues Maike Behrendt, Stefan Wagner, Jan Robine, Oliver De Candido, Joseph Adams, Marc Höftmann, and Felix Michels for proof-reading parts of my thesis. I am also thankful for successful collaborations with Oliver De Candido and Felix Michels. I would like to thank my colleague Thomas Germer for his detailed code reviews and exciting projects that we did together. I am also very thankful for the administrative support provided by Claudia Forstinger and Angela Rennwanz at the Heinrich Heine University Düsseldorf and by Alla Stankjawitschene at the Technical University of Dortmund. Additionally, I would like to thank Guido Königstein for maintaining our GPU servers.

I am thankful for my girlfriend Ashley-Jane du Plessis for her love, encouragement, and support. Finally, I thank my parents Norbert Uelwer and Petra Uelwer, and my sister Annika Uelwer for being the most wonderful family I could have wished for.

Contents

1
Introduction

In many physical measurement processes it is only possible to capture the Fourier magnitude of an image. This is particularly difficult as the most information about the image is contained in the (missing) Fourier phase. Phase retrieval is the process of reconstructing the phase (and by doing so, the image) while solely given the magnitude measurements. This is a relevant problem that shows up in different forms in many different research areas, e.g., in X-ray crystallography [64], astronomical imaging [21], optics [96], array imaging [7], or microscopy [108]. Although the first mention dates back almost 60 years already, the problem is still not fully solved and there are interesting instances of this problem.

Applications

Phase retrieval problems are often approached with optimization-based methods that usually rely on alternating projections onto convex and non-convex sets. These methods work reasonable well for well-posed instances of the phase retrieval problem, but fail for difficult ones, which are also relevant in practice.

In this thesis we focus on three instances of phase retrieval: (i) the general non-oversampled Fourier phase retrieval problem, (ii) the Fourier phase retrieval problem with reference image and (iii) the compressive Gaussian phase retrieval problem. While instances (i) and (ii) are relevant in practical applications, instance (iii) is more of theoretical nature and mainly included for comparison.

Three instances

In the following chapters we describe how deep learning techniques can be employed to solve the phase retrieval problem when a training dataset of images is available that are similar to the image of interest.

1.1 Research Questions

There are several open research questions, which we aim to discuss in this thesis:

How can supervised and unsupervised learning be applied to solve phase retrieval problems? We formulate the problem of phase retrieval as a learning task and show how modern deep neural network architectures can be used to solve phase retrieval problems in a specific domain, i.e., we restrict the solution space to images that are similar to a given dataset and demonstrate excellent performance in this domain.

How do these learning-based methods generalize to unseen data or data that follows a different distribution? By training the neural network on a given dataset it specializes on that domain. We try to understand how well the network generalizes to images coming from a different distribution. How well does the trained neural network perform on out-of-distribution samples?

How do unsupervised methods compare to supervised methods? We develop and analyze the performance of methods that rely on supervised and unsupervised learning, where in the context of phase retrieval supervised means that we have a dataset consisting of images and their corresponding magnitude measurements. In contrast to that, unsupervised means that we have only access to images without using their magnitude measurements during training.

How is the performance of learning-based methods influenced by measurement noise? Practical measurement setups are only able to capture noisy magnitudes. We simulate such a setup by adding Poisson and Gaussian noise onto the measurements and show that the discussed learning-based models are robust up to some degree.

Is learning a reference image for reference-based phase retrieval necessary? While in existing works the reference is usually hand-crafted, Hyder et al. [36] have shown that gradient descent can be used to learn a reference image from a dataset. We discuss whether learning a reference image always makes sense and propose a simpler approach that gives similar results.

1.2 Contributions

The contributions of this thesis can be summarized as follows:

1. We show how conditional generative adversarial networks can be used to solve phase retrieval problems using supervised learning.

2. We demonstrate how the performance of end-to-end learned networks can be improved by stacking multiple networks to a cascade.

3. By applying the concept of intermediate layer optimization in combination with additional modifications and initialization schemes we show how the performance of generative modelling-based methods for phase retrieval can be drastically improved.

4. For the problem of Fourier phase retrieval with a learned reference, we discuss how the error reduction (ER) algorithm can be unrolled in order to learn such a reference image. Furthermore, we provide further insights about the benefit of a learned reference.

1.3 Corresponding Publications

Parts of this thesis have been published as:

1. Tobias Uelwer, Alexander Oberstraß, and Stefan Harmeling. Phase retrieval using conditional generative adversarial networks. In *2020 25th International Conference on Pattern Recognition (ICPR)*, pages 731–738. IEEE, 2021. DOI: 10.1109/ICPR48806.2021.9412523.

2. Tobias Uelwer, Tobias Hoffmann, and Stefan Harmeling. Non-iterative phase retrieval with cascaded neural networks. In *Artificial Neural Networks and Machine Learning–ICANN 2021: 30th International Conference on Artificial Neural Networks, Bratislava, Slovakia, September 14–17, 2021, Proceedings, Part II 30*, pages 295–306. Springer, 2021. DOI: 10.1007/978-3-030-86340-1_24.

3. Tobias Uelwer, Sebastian Konietzny, and Stefan Harmeling. Optimizing intermediate representations of generative models for phase retrieval. *Transactions on Machine Learning Research*, 2022. ISSN 2835-8856. The first two authors contributed equally.

4. Tobias Uelwer, Nick Rucks, and Stefan Harmeling. A closer look at reference learning for fourier phase retrieval. In *NeurIPS 2021 Workshop on Deep Learning and Inverse Problems*, 2021.

1.4 Outline

This thesis is structured as follows: Chapter 2 gives a brief introduction into the field of deep learning. Chapter 3 reviews mathematical fundamentals, the Fourier transform, different instances of the phase retrieval problem, and existing projection-based phase retrieval algorithms. Chapter 4 presents our work on conditional generative adversarial networks for Fourier phase retrieval and (compressive) Gaussian phase retrieval. Chapter 5 discusses a cascaded neural network architecture for Fourier phase retrieval that is purely based on learning and does not involve optimization. Chapter 6 shows how phase retrieval

based on generative models can be improved by also optimizing the intermediate representations learned by the generator. We evaluate this method on Fourier phase retrieval and compressive Gaussian phase retrieval. In Chapter 7 we discuss the need for reference images and modify the error-reduction algorithm [22] to also leverage a reference image. Furthermore, we show how algorithm unrolling and backpropagation can be used to learn a reference image from data. In Chapter 8 we summarize our findings and draw a conclusion.

1.5 Notation

In the following we use squared brackets to index into a vector, matrix, or tensor, e.g., we use $x[k, l]$ to refer to the entry with index (k, l) in matrix $x \in \mathbb{C}^{m \times n}$. We make the convention that the index of the first entry is 0 if not mentioned otherwise. We use

$$\|x\|_1 = \sum_{k=0}^{n-1} |x[k]| \tag{1.1}$$

to denote the ℓ_1-norm of a vector $x \in \mathbb{C}^n$ and

$$\|x\|_2 = \sqrt{\sum_{k=0}^{n-1} x[k]^2} \tag{1.2}$$

denote the ℓ_2-norm/Euclidean norm of a vector $x \in \mathbb{C}^n$. Note that we also use this notation for matrices and tensors, e.g.,

$$\|x\|_1 = \sum_{k=0}^{m-1} \sum_{l=0}^{n-1} \sum_{j=0}^{p-1} |x[k, l, j]| \tag{1.3}$$

and

$$\|x\|_2 = \sqrt{\sum_{k=0}^{m-1} \sum_{l=0}^{n-1} \sum_{j=0}^{p-1} x[k, l, j]^2} \tag{1.4}$$

for tensor $x \in \mathbb{C}^{m \times n \times p}$. For a matrix $x \in \mathbb{R}^{m \times n}$ the norm $\|x\|_2$ is also known as Frobenius norm. Throughout this thesis, we will *never* use the column-sum or spectral matrix norm which are usually also denoted by $\|x\|_1$ and $\|x\|_2$ for $x \in \mathbb{C}^{m \times n}$, respectively. We write $D = [x_1, \ldots, x_n]$ to refer to a dataset of n vectors, matrices or tensors. For matrix or tensor x we write $\mathrm{vec}(x)$ to refer to a vector that consists of the entries of x.

2
Deep Learning

With the recent advances in hardware development, Deep neural networks (DNNs) have come back into fashion. To give a short overview: A neural network $H_\theta = H_k \circ \cdots \circ H_1$ is a concatenation of k layers $H_1, \ldots, H_k$, where some of the layers have learnable parameters. We denote the set of all learnable parameters of H_θ as θ. The learnable parameters θ are optimized, usually, using a stochastic gradient descent (SGD) algorithm, such that a loss function $\mathcal{L}$ is minimized for a given training dataset. One distinguishes two classes of deep learning: In supervised learning the datasets $\mathcal{D} = [(x_1, y_1), \ldots, (x_n, y_n)]$ consist of paired inputs and outputs (for example, input images and class labels in the classification setting), whereas in the unsupervised setting the training datasets only consist of inputs $\mathcal{D} = [x_1, \ldots, x_n]$. In the latter case, the task is usually to learn the distribution of the dataset (generative modelling), e.g., by learning a sampler that can generate data similar to the training dataset. This chapter introduces concepts of deep learning that are used in this thesis and loosely follows the textbook written by Goodfellow et al. [26].

Neural networks

Supervised learning

Unsupervised learning

2.1 *Neural Network Layers*

In the following, we discuss the most important layers that are the building blocks of modern DNNs.

2.1.1 *Fully-Connected Layers*

Fully-connected layers are at the heart of multilayer perceptrons (MLPs) which are also sometimes called fully-connected networks. For vector-valued inputs $x \in \mathbb{R}^n$ a fully-connected layer with m outputs can be written as

$$H_i(x) = W_i x + b_i, \tag{2.1}$$

where $W_i \in \mathbb{R}^{m \times n}$ is the learnable weight matrix and $b_i \in \mathbb{R}^m$ is the learnable bias vector. Note that a fully-connected layer can also be

applied to matrix or tensor inputs. In that case, the entries have to be flattened into a vector first. The layer is fully-connected in the sense that every element in the output depends on every element of the input. This can be a favorable property as this layer is very expressive. However, a drawback of this layer is that it requires a large amount of parameters. This is particularly problematic when the inputs are high-dimensional, e.g., high-resolution images. A different type of layer that is particularly suitable for images and addresses the issue of the large number of parameters is the convolutional layer.

2.1.2 *Convolutional Layers*

Instead of treating the inputs as a vector, convolutional layers maintain the spatial arrangement of the entries. For inputs $x \in \mathbb{R}^{c \times p \times q}$ are defined as

$$H_i(x) = w_i * x + b_i, \tag{2.2}$$

where $w_i \in \mathbb{R}^{c \times m \times m}$ is the learnable kernel (also sometimes called filter), $b_i \in \mathbb{R}$ is the learnable bias term and $*$ denotes the convolution operation. There are different ways to implement the convolution operation. In the following we consider the same mode convolution which is defined as

Convolution

$$(w * x)[k, l] = \sum_{j=0}^{c-1} \sum_{s=0}^{m-1} \sum_{t=0}^{m-1} w[j, s, t]\, x[j, k - s + \Delta, l - t + \Delta], \tag{2.3}$$

where we assume that m is odd, $\Delta = (m-1)/2$ and the input is padded with zeros by Δ pixels in the second and third dimension. The output $w * x$ has the same shape as the input, hence the name same. Note, that the inputs have to be padded with zeros appropriately. Intuitively speaking, the kernel w_i is flipped horizontally and vertically and moved step-wise over the image, while at each step the inner product with the current image patch is calculated. Each inner product gives a single entry in the next feature map, i.e., the output layer.

Linearity of the convolution

Note, that the convolution is still a linear operation and thus the convolutional layer is an instance of the fully-connected layer, where weights are reused. However, through their inductive bias[1] convolutions are especially suitable for processing of images. They have fewer parameters than fully-connected layers and require less training data.

[1] The inductive bias is the bias introduced by restricting of the space of functions from which the learned function is taken [14].

The convolution operation is closely related to the cross-correlation and, in fact, for neural network layers, where the kernel is learned, both operations are equivalent (up to vertical and horizontal flips of the kernel). Usually, a convolutional layer uses multiple kernels and stacks the resulting feature maps to form the input for the next layer.

2.1.3 Activation Functions

Fully-connected and convolutional layers are usually followed by an activation function $h : \mathbb{R} \to \mathbb{R}$ that is applied component-wise to each output. The activation function allows the neural network to learn non-linear mappings and also improves expressivity of the network by preventing consecutive layers from collapsing into a single layer. In the following, we give a short overview over the three most common activation functions. All three activation functions are visualized in Figures 2.1, 2.2, and 2.3.

Sigmoid Function. The Sigmoid function is defined as

$$\sigma : \mathbb{R} \to \mathbb{R}, \quad \sigma(x) = \frac{1}{1 + e^{-x}}. \tag{2.4}$$

It is a monotonic, continuous and differentiable function. It has already been used for a long time in classical statistics, e.g., in the context of logistic regression. However, its computation is expensive as it requires a division and the evaluation of the exponential function. Also, its derivative approaches zero for very high and very low inputs that can cause problems during training which relies on derivative-based optimization. The ReLU activation function addresses these problems.

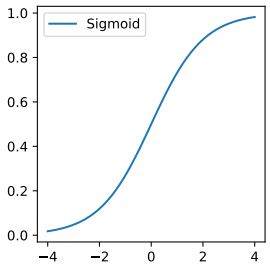


Figure 2.1: The Sigmoid function.

Rectified Linear Unit (ReLU). The ReLU activation function is defined as

$$\mathrm{ReLU} : \mathbb{R} \to \mathbb{R}, \quad \mathrm{ReLU}(x) = \begin{cases} x, & x \geq 0 \\ 0, & x < 0. \end{cases} \tag{2.5}$$

Its computation is easy and only for negative inputs its derivative vanishes. Once the input to the ReLU is negative the ReLU outputs zero and there is no possibility to recover this ReLU by gradient-based optimization (as the derivative is zero). This phenomenon is termed the dying ReLU problem.

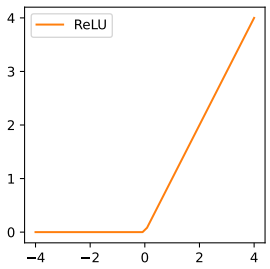


Figure 2.2: The ReLU function.

Leaky ReLU. The leaky ReLU activation function addresses the problem of dying ReLUs by replacing the constant part of the function with a linear part with low slope. It is defined as

$$\mathrm{LeakyReLU}_\alpha : \mathbb{R} \to \mathbb{R}, \quad \mathrm{LeakyReLU}_\alpha(x) = \begin{cases} x, & x \geq 0 \\ \alpha x, & x < 0, \end{cases} \tag{2.6}$$

where $\alpha > 0$ is a small positive number determining the slope for negative inputs. In practice, $\alpha \in (0, 0.2]$ often gives favorable results. Note that α is constant and not a learnable parameter.

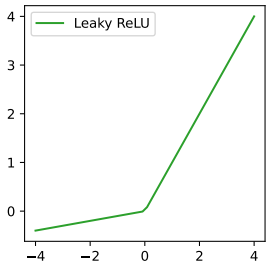


Figure 2.3: The leaky ReLU function ($\alpha = 0.1$).

2.1.4 Pooling Layers

Similar to convolutional layers, pooling layers process the previous feature map patch-wise. The pooling operation aggregates the information of a patch into a single number. There are two commonly used pooling layers: max-pooling and average-pooling. As the name suggests, max-pooling aggregates the entries of the input patch by taking the maximum entry of the patch, while average-pooling calculates the average of all entries of the patch. Pooling layers are usually used in combination with convolutional layers to reduce the spatial size of the feature map and to compress the information extracted by the previous layers.

Max-pooling

Average-pooling

2.1.5 Batch-Normalization Layer

As we have seen before, the behavior of the activation function is highly dependent on the inputs. Therefore, a common strategy is to normalize the activations before they are passed to the activation function. Ioffe and Szegedy [37] introduced batch-normalization which had a great impact on the neural network community as it accelerated training and reduced the need for additional regularization strategies. In detail, their batch-normalization layer normalizes the activations by subtracting the empirical mean of the batch and dividing by the empirical standard deviation of the batch, i.e.,

$$\mathrm{BN}(x) = \gamma \odot \frac{x - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} + \beta, \tag{2.7}$$

where the division is elementwise, μ_B and σ_B^2 are the mean and variance of the activation calculated on the small batch of the data B, γ and β are learnable parameters that have the same shape as the input, $\odot$ denotes elementwise multiplication, and $\epsilon > 0$ is a small positive number that prevents numerical issues. The learnable parameters γ and β are necessary to preserve the expressivity of the neural network.

Learnable parameters

2.2 Loss Functions

For comparing the reconstruction quality of different algorithms we have to define metrics for measuring the reconstruction error or similarity of different images. For simplicity, we define the loss functions for grayscale images. The extension to multiple color channels is straightforward.

2.2.1 Mean Squared Error (MSE)

The most commonly used error metric for two images is the pixelwise mean squared error (MSE). For two vectors $x \in \mathbb{R}^n$ and $\tilde{x} \in \mathbb{R}^n$ the MSE is defined as

$$\mathcal{L}_{\text{MSE}}(x, \tilde{x}) = \frac{1}{n} \sum_{k=0}^{n-1} (x[k] - \tilde{x}[k])^2 . \tag{2.8}$$

For two gray-scale images $x \in \mathbb{R}^{n \times n}$ and $\tilde{x} \in \mathbb{R}^{n \times n}$ the MSE is defined as

$$\mathcal{L}_{\text{MSE}}(x, \tilde{x}) = \frac{1}{n^2} \sum_{k=0}^{n-1} \sum_{l=0}^{n-1} (x[k,l] - \tilde{x}[k,l])^2 . \tag{2.9}$$

and is proportional to the squared Euclidean norm of the pixelwise difference between the two images $\|x - \tilde{x}\|_2^2$. The MSE has several benefits: it is convex and differentiable which makes it especially suitable as a loss function for optimization algorithms. Also, it is usually easy to implement. However, there are also several downsides of using this metric: the MSE punishes errors quadratically which means a strong difference between two pixels of the image results in a very large contribution to the overall error. This is usually not desired. One also has to keep in mind that the MSE is a pixelwise error metric which means that applying some transformation to one of the images (e.g., translating it by just a few pixels) also results in a very large error. In the context of Fourier phase retrieval we will later see that this requires registration of the images, i.e., the images need to be aligned using an appropriate algorithm, before the error is calculated. Also, one needs to compare the original image with the reconstruction that has been rotated by 180° as it also is a valid reconstruction in the context of Fourier phase retrieval. We will further discuss this in Chapter 3.

Problems of MSE

2.2.2 Mean Absolute Error (MAE)

Another error metric for images is the mean absolute error (MAE). For two vectors $x \in \mathbb{R}^n$ and $\tilde{x} \in \mathbb{R}^n$ the MAE is defined as

$$\mathcal{L}_{\text{MAE}}(x, \tilde{x}) = \frac{1}{n} \sum_{k=0}^{n-1} |x[k] - \tilde{x}[k]|. \tag{2.10}$$

For two grayscale images $x \in \mathbb{R}^{n \times n}$ and $\tilde{x} \in \mathbb{R}^{n \times n}$ the MAE is defined as

$$\mathcal{L}_{\text{MAE}}(x, \tilde{x}) = \frac{1}{n^2} \sum_{k=0}^{n-1} \sum_{l=0}^{n-1} |x[k,l] - \tilde{x}[k,l]| \tag{2.11}$$

and is proportional to the ℓ_1-norm of the pixelwise difference between the two images $\|x - \tilde{x}\|_1$. The MAE is still convex but no longer differentiable because the absolute value is not differentiable at zero. As

the MAE punishes deviations using the absolute value it is not as sensitive regarding strong deviations for single pixels as the MSE, which makes it preferable over the MSE. However, the MAE is a pixelwise error measure which means that it suffers from some of the same disadvantages as the MSE.

2.2.3 *Structural Similarity Index Measure (SSIM)*

Different from the previously defined pixelwise metrics the structural similarity index measure (SSIM) [99] calculates the similarity of two input images $x \in \mathbb{R}^{m\times n}$ and $\tilde{x} \in \mathbb{R}^{m\times n}$ by comparing the intensity distributions of the images. The SSIM is defined as

$$\mathrm{SSIM}(x,\tilde{x}) = \frac{(2\mu_x\mu_{\tilde{x}} + C_1)(2\sigma_{x\tilde{x}} + C_2)}{(\mu_x^2 + \mu_{\tilde{x}}^2 + C_1)(\sigma_x^2 + \sigma_{\tilde{x}}^2 + C_2)}, \tag{2.12}$$

where $\mu_x, \mu_{\tilde{x}}$ and $\sigma_x^2, \sigma_{\tilde{x}}^2$ are the means and the variances of the intensities of x and $\tilde{x}$, respectively, $\sigma_{x\tilde{x}}$ is the covariance of the intensities of x and $\tilde{x}$, and C_1 and C_2 are given constants that prevent numerical instabilities.

Rather than calculating the SSIM globally for the whole image, it is usually applied to various sub-images (windows) of the original images. These windows usually have size 8×8 and are moved pixel by pixel over the images. The SSIM values for all windows of the images are then averaged to obtain a single number representing the global similarity of the two input images.

The SSIM is a similarity function with values restricted to $[0, 1]$, where larger values indicate higher similarity, e.g., a value of 1 corresponds to identical inputs, whereas a value of 0 corresponds to completely different input images. Therefore, we have to multiply the SSIM with -1 if we want to use it as loss function (that is minimized), i.e., the corresponding loss function is given as $\mathcal{L}_{\mathrm{SSIM}}(x,\tilde{x}) = -\mathrm{SSIM}(x,\tilde{x})$. Note, that $\mathcal{L}_{\mathrm{SSIM}}$ does not fulfill the triangle inequality which means that it is mathematically speaking not a distance metric (in contrast to the MAE and the MSE).

SSIM as loss function

2.2.4 *Learned Perceptual Image Patch Similarity (LPIPS)*

Loss functions that directly operate on the pixels of an image are not always aligned with human perception. For example, shifting an image for a few pixels results in a high MSE while the human eye might not be able to detect the change at all. Perceptual loss functions try to address this issue. Instead of defining a distance in the pixelspace of the image they measure the distance in the feature space of a given neural network. One of the most common perceptual losses is the learned perceptual image patch similarity (LPIPS) [107] based on a

Perceptual loss functions

VGG-16 network [81]. The VGG-16 net is augmented with additional linear layers such that it outputs a single value (the LPIPS score) for a pair of input images. The neural network is trained on a dataset that was obtained using a two-alternative forced choice (2AFC) method. For each image of the dataset there are also two transformed variants. For each of these triplets (original image and two variants) a human made a decision about which of the two variants is perceived to be more similar to the original image. The network is then trained to mimic this decision.

2.3 *Gradient-based Optimization*

In the previous sections we discussed the building blocks of DNN, but did not explain how the trainable parameters, e.g., the weights, kernels, and biases, are chosen. In this section we want to shed some light onto this matter. Given a dataset $\mathcal{D}$, a loss function $\mathcal{L}$, and a neural network H_θ with trainable parameters θ (e.g., the weight matrices of the fully-connected layers or the filter kernels of convolutional layers) we can now use gradient-based optimization to train the network. There are two notions of gradient-based optimization: (i) gradient ascent that maximizes a function and (ii) gradient descent that can be used to minimize a function. In the context of deep learning, the latter algorithm, i.e., gradient descent, is more relevant as we usually want to minimize the loss function. The insight used by gradient-based optimization is that the gradient of a multivariate function always points toward the direction of steepest ascent. Therefore, gradient ascent takes a small step in the direction of the gradient of a function. In contrast to that, gradient descent takes a small step in the direction of the negative gradient. To train a DNN, one applies gradient descent to the parameters of the neural network to minimize the loss function, i.e.,

Gradient descent

$$g_{k+1} = \frac{1}{|\mathcal{D}|} \sum_{(x,y)\in\mathcal{D}} \nabla_\theta \mathcal{L}(H_\theta(x), y)\big|_{\theta=\theta_k} \tag{2.13}$$

$$\theta_{k+1} = \theta_k - \lambda_{k+1} g_{k+1}, \tag{2.14}$$

where $\lambda_{k+1} > 0$ is the learning rate that determines the size of the step and is usually decayed during optimization. Note, that gradient descent is a greedy algorithm and when applied to non-convex functions (which is usually the case when training neural networks) it is not guaranteed that it always finds a global minimum. Most of the time one has to be satisfied with a local minimum. However, it is believed that because of the many weight space symmetries there exist a large number of local minima that are equally good. Training neural networks usually requires large amounts of data, that is why the sum

Batch gradient descent

calculated in Equation 2.13 is usually only calculated for a small subset of the dataset, i.e.,

$$g_{k+1} = \frac{1}{|B_k|} \sum_{(x,y)\in B_k} \nabla_\theta \mathcal{L}(H_\theta(x), y)\big|_{\theta=\theta_k} \tag{2.15}$$

$$\theta_{k+1} = \theta_k - \lambda_{k+1} g_{k+1}, \tag{2.16}$$

where $\lambda_{k+1} > 0$ is the learning rate and $B_k \subset \mathcal{D}$ is a subset of the dataset. This subset is often referred to as a batch. Since the gradient in Equation 2.15 is no longer the exact gradient, this variant is called stochastic gradient descent (SGD). The size of the batch is usually limited by the memory of the GPU. Although the gradient is no longer exact, it is possible to prove convergence to a local minimum of the loss function under certain conditions, most notably

Convergence

$$\sum_{k=1}^{\infty} \lambda_k = \infty \text{ and } \sum_{k=1}^{\infty} \lambda_k^2 < \infty. \tag{2.17}$$

The SGD method was first discussed by Robbins and Monro [74]. The noise introduced by the inexact gradient calculation is also a useful feature as it is believed that SGD favors flat minima that lead to better generalization on unseen data [44, 33].

There are many modifications that can be applied to Equation 2.15. In the following we give a brief overview over the most relevant ones.

2.3.1 *Momentum*

The speed at which SGD converges is mostly influenced by the curvature of the loss landscape. High curvature leads to a decrease of the speed at which SGD traverses the function. This can be explained by the phenomenon of zig-zagging where iterates cancel out. It is shown in Figure 2.4.

The momentum update tries to counteract this phenomenon by tracking the moving average of previous gradients. The momentum update was introduced by Polyak [72] and reads as

Momentum update

$$\mu_{k+1} = \beta \mu_k + \frac{1}{|B_k|} \sum_{(x,y)\in B_k} \nabla_\theta \mathcal{L}(H_\theta(x), y)\big|_{\theta=\theta_k} \tag{2.18}$$

$$\theta_{k+1} = \theta_k - \lambda_{k+1} \mu_{k+1}, \tag{2.19}$$

where $1 > \beta \geq 0$ is a hyperparameter and $\mu_0 = 0$ is initialized as all-zero. Note, for $\beta = 0$ the momentum is disabled and the update rule corresponds to the vanila SGD update rule. The larger the value of β is the more influence have past updates onto the current update. From a physical viewpoint the variable μ acts as the velocity.

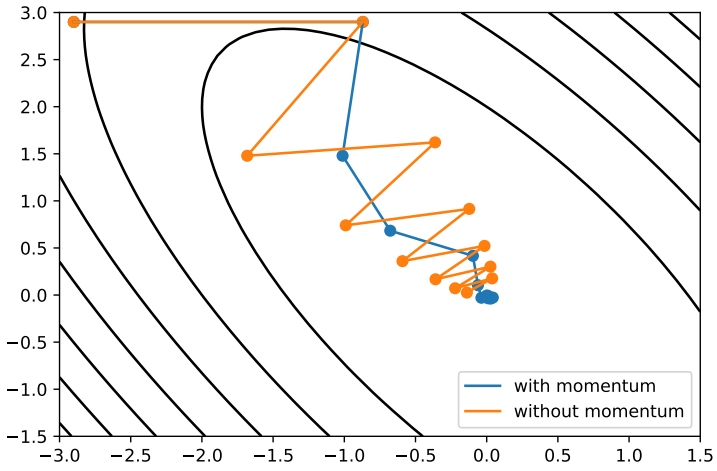


Figure 2.4: The zig-zagging phenomenon of gradient-descent algorithms. The momentum update helps to counteract this by averaging iterates.

2.3.2 *RMSProp*

Instead of tracking the moving average of previous gradients, the RMSProp (Root Mean Square Propagation) update rule calculates the moving average of the squared gradients of previous iterations. These averaged squared gradients are then used to rescale the step in each direction, i.e.,

$$g_{k+1} = \frac{1}{|B_k|} \sum_{(x,y)\in B_k} \nabla_\theta \mathcal{L}(H_\theta(x), y)\big|_{\theta=\theta_k} \tag{2.20}$$

$$v_{k+1} = \beta v_k + (1-\beta) g_{k+1}^2 \tag{2.21}$$

$$\theta_{k+1} = \theta_k - \lambda_{k+1} \frac{g_{k+1}}{\sqrt{v_{k+1}} + \epsilon}, \tag{2.22}$$

where the division and the square roots are meant elementwise and $\epsilon > 0$ is a small constant number preventing numerical issues. The momentum term for the squared gradients is initialized as $v_0 = 0$. The effect of gradient rescaling is that the weights that already have received a lot of large updates in the previous iterations, receive a smaller update at the current iteration. Again, this can counteract the zig-zagging phenomenon. The RMSProp update rule was proposed by Geoffrey Hinton in one of his lectures, but was never formally published. However, it is one of the main ideas of the widely used Adam update rule, which we will discuss next.

2.3.3 *Adam*

Having discussed the momentum update rule and the RMSProp update rule, explaining the famous Adam (Adaptive Moment Estimation) update rule is now straightforward. Adam simply combines the pre-

vious ideas by iterating

$$g_{k+1} = \frac{1}{|B_k|} \sum_{(x,y)\in B_k} \nabla_\theta \mathcal{L}(H_\theta(x), y)\big|_{\theta=\theta_k} \tag{2.23}$$

$$\mu_{k+1} = \beta_1 \mu_k + (1-\beta_1) g_{k+1} \tag{2.24}$$

$$\nu_{k+1} = \beta_2 \nu_k + (1-\beta_2) g_{k+1}^2 \tag{2.25}$$

$$\tilde{\mu}_{k+1} = \mu_{k+1} / (1-\beta_1^{k+1}) \tag{2.26}$$

$$\tilde{\nu}_{k+1} = \nu_{k+1} / (1-\beta_2^{k+1}) \tag{2.27}$$

$$\theta_{k+1} = \theta_k - \lambda_{k+1} \frac{\tilde{\mu}_{k+1}}{\sqrt{\tilde{\nu}_{k+1}} + \epsilon}, \tag{2.28}$$

where λ_{k+1} again denotes the learning rate, $0 < \beta_1 < 1$ and $0 < \beta_2 < 1$ determine the influence of previous gradients and squared gradients, respectively. Default choices, that work well in practice, are $\beta_1 = 0.9$ and $\beta_2 = 0.999$. In addition, Equations 2.26 and 2.27 perform a bias correction for the moving averages. Again, the momentum terms are initialized as $\mu_0 = 0$ and $\nu_0 = 0$. The Adam update rule was introduced by Kingma and Ba [50] and is one of the most cited works in the field of deep learning research.

This is only a small selection of update rules used for deep learning. Up until today not a single best update rule has emerged and which update rule performs best is still highly task-dependent. For a large scale comparison of deep learning update rules in combination with different learning rate decay schedules we refer to the work of Schmidt et al. [79].

2.4 *Autoencoders*

Having introduced neural network layers, loss functions and optimizers, we can now take a look at a complete DNN architecture. One of the simplest architectures is the autoencoder. It solves the task of finding a low-dimensional representation $\mathcal{R} = [z_1, \ldots, z_n]$ of the data $\mathcal{D} = [x_1, \ldots, x_n]$ and can therefore be seen as a non-linear dimensionality reduction technique. It consists of two networks: an encoder network E_ϕ having trainable parameters ϕ and a decoder network D_θ having trainable parameters θ. This is done by feeding the input data to the encoder that outputs a low-dimensional representation. This representation is then fed into the decoder that tries to reconstruct the original images from this representation. The output of the encoder (and therefore also the input to the decoder network) usually has less entries than the original inputs. This bottleneck encourages the encoder to include as much information as possible into the latent representation, such that the decoder has an easier task to reconstruct

the images. Figure 2.5 gives an overview of an autoencoder architecture and shows input images from the MNIST dataset [55], their latent representations, and their reconstructions.

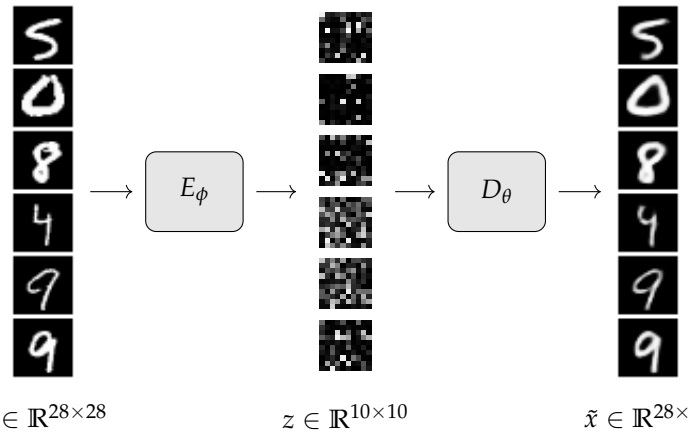


Figure 2.5: An overview of the autoencoder architecture that consists of an encoder network E_ϕ and a decoder network D_θ. The low-dimensional representations z contain enough information such that D_θ can produce reconstructions $\tilde{x}$ that are similar to the original images x.

In order to train the autoencoder, the parameters θ and ϕ are jointly optimized to solve the problem

$$\underset{\phi,\theta}{\operatorname{argmin}} \sum_{x\in\mathcal{D}} \mathcal{L}(x, D_\theta(E_\phi(x))), \tag{2.29}$$

where $\mathcal{L}$ is one of the loss functions presented in Section 2.2. If $\mathcal{L}$ is chosen as the MSE and E_ϕ and D_θ only consist of a single fully-connected layer without an activation function, the optimization problem 2.29 is equivalent to principal component analysis (PCA). Thus, the autoencoder can be seen as a non-linear generalization of PCA.

Note, that we know nothing about the distribution of the latent variable z. Therefore, we cannot generate new samples by plugging random values for z into the decoder network. This issue is addressed in the following section which introduces the VAE – a so called generative model.

2.5 *Generative Models*

Generative models are usually unsupervised learning models, i.e., they only consider a dataset $\mathcal{D} = [x_1, \ldots, x_n]$ consisting of input data points without corresponding labels. The goal of training a generative model is to learn the distribution p_{data} from which the dataset was sampled from. This can either be done explicitly or implicitly: In the first case, the density function is estimated, which allows to evaluate the likelihood of the data. In the second case, one considers oneself satisfied with learning a sampler of the dataset, which can be used to generate new data from the data distribution.

In the following, we give a short overview over the most relevant generative models:

- Explicit models (model the density p_{data} explicitly):
  1. Normalizing flows: RealNVP [18], NADE [92], MADE [70], NICE [17], PixelRNN [94], PixelCNN [93], GLOW [52]
  2. Variational autoencoder (VAEs) [51]
  3. Score-based models: Denoising diffusion probabilistic modeling (DDPM) [83, 32], Score matching with Langevin dynamics (SMLD) [84], Score-based generative modeling through stochastic differential equations [85]
- Implicit models (do not model the density p_{data} explicitly):
  1. Generative adversarial networks (GANs) [25]
  2. Wasserstein GANs [1]
  3. Least-Squares GANs [62]

Since their introduction, GANs and VAEs were the most prominent generative models for a many years. Recently, research in score-based models gained a lot of attention and yields impressive results. Note, that VAEs only learn an approximate likelihood, which is why they should be strictly speaking not be classified as explicit model as it is the case in literature [24].

In the remaining chapters, we will most often employ GANs and VAEs, but will also borrow ideas from the training of Wasserstein GANs.

2.5.1 *Variational Autoencoders (VAEs)*

Autoencoders are a simple neural network architecture consisting of an encoder network E_ϕ with learnable parameters ϕ and a decoder network D_θ with learnable parameters θ. The encoder E_ϕ maps the input images x to a latent representation z, which is then used by the decoder network D_θ to reconstruct the image. To further structure the space of the latent representation z, variational autoencoders (VAE) were introduced by [51].

VAEs are Bayesian models that assume a Gaussian prior for the latent variable $p(z) = \mathcal{N}(0, I)$, where I denotes the identity matrix. The key idea is that the encoder is used to approximate the true posterior density of the latent variable $p(z|x)$. In detail, the encoder E_ϕ no longer predicts a single point in the latent space, but instead predicts means μ_z and variances σ_z of a multivariate Normal distribution. This means, that E_ϕ can now be viewed as a conditional distribution $q_\phi(z|x) = \mathcal{N}(\mu_z, \text{Diag}(\sigma_z))$ describing the latent variable z given an

image x. After training, this conditional distribution $q_\phi(z|x)$ is the approximation for the posterior distribution of z. The decoder D_θ takes the role of modeling the conditional distribution $p_\theta(x|z)$, i.e., the distribution of the reconstructed images given their latent representation.

Training the model requires access to the marginal likelihood (also known as evidence) Marginal likelihood

$$p_\theta(x) = \int p(z)p_\theta(x|z)dz, \tag{2.30}$$

where $p(z)$ is a normal prior for z. Unfortunately, evaluating this integral is intractable, as it requires evaluating the integrand for every value of z. This is why the principle of variational inference, hence the name variational autoencoder, is used to lower-bound the marginal likelihood. This evidence lower bound (ELBO) is then maximized during training.

Evidence lower bound (ELBO). Before we start deriving the ELBO, we have to define the Kullback-Leibler divergence (KL divergence), which can be used to compare two probability distributions. The KL divergence is defined as

$$D_{KL}(p(x)||q(x)) = \int p(x) \log\left(\frac{p(x)}{q(x)}\right) dx, \tag{2.31}$$

where p and q are two continuous probability distributions. The KL divergence is always greater or equal to zero and zero if and only if $p = q$ (almost everywhere). In general, the KL-divergence is not symmetric, i.e., $D_{KL}(p(x)||q(x)) \neq D_{KL}(q(x)||p(x))$.

Following Li et al. [56], the lower bound for the evidence can be derived as:

$$\begin{aligned}
\log p_\theta(x) &= \int \log\left(p_\theta(x)\right) q_\phi(z|x)dz && p_\theta(x) \text{ does not depend on } z \\
&= \int \log\left(\frac{p_\theta(x|z)p(z)}{p_\theta(z|x)}\right) q_\phi(z|x)dz && \text{Bayes' rule and rearranging} \\
&= \int \log\left(\frac{p_\theta(x|z)p(z)}{p_\theta(z|x)}\frac{q_\phi(z|x)}{q_\phi(z|x)}\right) q_\phi(z|x)dz && \text{multiply by one} \\
&= \int \log p_\theta(x|z) q_\phi(z|x)dz - \underbrace{\int \log\left(\frac{q_\phi(z|x)}{p(z)}\right) q_\phi(z|x)dz}_{=D_{\mathrm{KL}}(q_\phi(z|x)||p(z))} \\
&\quad + \underbrace{\int \log\left(\frac{q_\phi(z|x)}{p_\theta(z|x)}\right) q_\phi(z|x)dz}_{=D_{\mathrm{KL}}(q_\phi(z|x)||p_\theta(z|x))\geq 0} \\
&\geq \underbrace{\int \log p_\theta(x|z) q_\phi(z|x)dz}_{=\mathbb{E}_{z\sim q_\phi(z|x)}[\log p_\theta(x|z)]} - \underbrace{\int \log\left(\frac{q_\phi(z|x)}{p(z)}\right) q_\phi(z|x)dz}_{=D_{\mathrm{KL}}(q_\phi(z|x)||p(z))}.
\end{aligned} \tag{2.32}$$

When implementing the ELBO one usually uses Monte-Carlo sampling to estimate the expected value and the KL divergence. The first term

$$\mathbb{E}_{z\sim q_\phi(z|x)}\left[\log p_\theta(x|z)\right] \tag{2.33}$$

is a Gaussian likelihood and maximizing it corresponds to minimizing a MSE reconstruction loss. The second term $-D_{\mathrm{KL}}(q_\phi(z|x)||p(z))$ penalizes deviating from the standard normal prior. For Gaussian distributions, this quantity can be calculated in closed-form. Higgins et al. [31] introduce a hyperparameter $\beta > 1$ to balance the two terms of the ELBO

β-VAE

$$\mathbb{E}_{z\sim q_\phi(z|x)}\left[\log p_\theta(x|z)\right] - \beta D_{\mathrm{KL}}(q_\phi(z|x)||p(z)). \tag{2.34}$$

This promotes disentangling the latent variables and can be used for discovering latent factors in the data. Training a VAE using Equation 2.34 yields the β-VAE.

Figure 2.6 shows the structured latent space of a VAE that was trained on the MNIST dataset [55]. Similar digits are also close in the latent space.

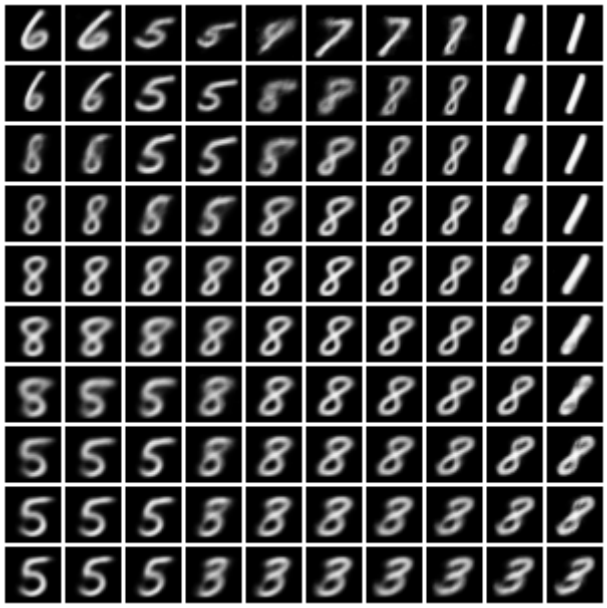


Figure 2.6: Exploring the two-dimensional latent space of a VAE trained on the MNIST dataset: Samples from the decoder that was evaluated on a regular grid over $[0, 1.5] \times [0, 1.5]$.

Reparameterization trick. Training the VAE requires back-propagating through the sampling process of a Gaussian distribution to calculate the gradients with respect to the parameters of the encoder network ϕ. To do so, one uses the so-called reparameterization trick which simply uses a linear transformation of a standard Gaussian random variable, i.e., to sample from $q_\theta(z|x)$ one calculates

$$z = \sigma_z \odot \mathrm{sg}(z_{\mathrm{standard}}) + \mu_z, \tag{2.35}$$

where the mean μ_z and the standard deviation σ_z are predicted by the encoder network, $z_{\text{standard}} \sim \mathcal{N}(0, I)$ is a Gaussian random variable and sg is the stop gradient operation that stops the gradient calculation during backpropation.

2.5.2 *Generative Adversarial Networks (GANs)*

A different class of generative models are the so-called Generative Adversarial Networks (GANs) that were first introduced by Goodfellow et al. [25]. Similar to the VAE, a GAN also consists of two networks: one is the generator G_θ with learnable parameters θ and the other is the discriminator D_ϕ with parameters ϕ. The key idea is that G_θ is trained to transform random samples from a latent distribution p_z to samples similar to those coming from the data set. Figure 2.7 compares samples produced by a GAN that was trained on the MNIST dataset with original samples coming from that dataset. The discriminator D_ϕ takes the role of a critic that is trained to tell real samples from samples that where generated by G_θ apart. This is done by solving a min-max-optimization problem

$$\min_\theta \max_\phi \underbrace{\mathbb{E}_{x\sim p_{\text{data}}}\left[\log(D_\phi(x))\right] + \mathbb{E}_{z\sim p_z}\left[\log(1 - D_\phi(G_\theta(z)))\right]}_{=V(\theta,\phi)}. \quad (2.36)$$

This problem is solved by alternating gradient ascent steps applied to ϕ and gradient descent steps applied to θ. The problem can be viewed as a saddle-point finding problem of the function $V(\theta, \phi)$.

Goodfellow et al. [25] show that if the inner maximization problem is solved exactly then the training algorithm minimizes the Jensen-Shannon-divergence (JS-divergence) between the data distribution and the distribution of the generator

$$D_{\text{JS}}(p_{\text{data}}||p_G) = \frac{1}{2}D_{\text{KL}}(p_{\text{data}}||p_M) + \frac{1}{2}D_{\text{KL}}(p_M||p_G), \quad (2.37)$$

where $p_M = (p_{\text{data}} + p_G)/2$. Furthermore, it is shown that the global minimum of the function $C(\theta) = V(\theta, \phi^*)$ is achieved if and only if $p_G = p_{\text{data}}$, where ϕ^* are the optimal discriminator parameters and p_G denotes the distribution described by G_θ. At that point, the function takes the value $C(\theta) = -\log(4)$.

Due to numerical instabilities, in practice the slightly modified optimization problem

Numerical stability

$$\min_\theta \max_\phi \; \mathbb{E}_{x\sim p_{\text{data}}}\left[\log(D_\phi(x))\right] - \mathbb{E}_{z\sim p_z}\left[\log(D_\phi(G_\theta(z)))\right] \quad (2.38)$$

is solved. Training a GAN can be challenging and is in general considered instable.

Figure 2.7: Samples from a fully-connected GAN trained on the MNIST dataset.

2.5.3 *Conditional Generative Adversarial Networks (CGANs)*

Conditional GANs are an extension of GANs that allow sampling from a conditional distribution $p_{\text{data}}(x|y)$, where y can, for example, be a class label or an image.

The idea is straightforward: both G_θ and D_ϕ get the variable y, which we want to condition on, as an additional input. The min-max objective for the training of a conditional GAN reads as

$$\begin{aligned}\min_\theta \max_\phi \; & \mathbb{E}_{(x,y)\sim p_{\text{data}}} \left[\log(D_\phi(x,y))\right] \\ & + \mathbb{E}_{z\sim p_z, y\sim p_y} \left[\log(1 - D_\phi(G_\theta(z,y),y))\right],\end{aligned} \tag{2.39}$$

where $p_y = \int p_{\text{data}}(x,y)dx$ denotes the distribution of y.

Depending on the first layer of G_θ and D_ϕ and whether y is a vector or an image, there are different ways to realize this: For a fully-connected layer, y is flattened and stacked onto the input. For a convolutional layer the vector y usually turned into a tensor by replicating the values appropriately, e.g., when y is a one-hot-encoded vector, each entry of y forms a new channel of the input. If y is an image, then it is just stacked onto the input. Figure 2.8 shows samples generated by a conditional GAN trained on the MNIST dataset that was conditioned on the class labels of the digits. Thus, G_θ can be used to generate samples of a particular class, e.g., only ones or eights.

Figure 2.8: Samples from a fully-connected conditional GAN trained on the MNIST dataset. The GAN was conditioned on the class label. During sampling from the conditional GAN the class labels $y = 1$ and $y = 8$ were fed into the generator network.

Conditional GANs were already proposed in the original GAN paper [25] and further analyzed by Mirza and Osindero [65]. A notable application of conditional GANs is the pix2pix model [39] that solves a large number of image-to-image translation problems. These image-to-image translation problems are problems where one image needs to be turned into another image. For example, turning satellite images into maps is an image-to-image translation problem. As shown by

Isola et al. [39], it is advisable to augment the training objective with an additional reconstruction loss when solving these kinds of problems. We will further discuss this in Chapter 4.

3
Phase Retrieval

In this chapter, we are going to review some of the mathematical concepts that we need in the later chapters. We also introduce different phase retrieval problems and discuss existing methods to solve them.

3.1 Complex Numbers

We start by repeating some facts about complex numbers. The set of complex numbers $\mathbb{C}$ is an extension of the set of real numbers $\mathbb{R}$. A complex number is defined as a number $z = a + bi$, where $a, b \in \mathbb{R}$ and i is a solution of the equation $i^2 = -1$. The number a is usually referred to as the real part $\mathrm{Re}(z)$ of z, whereas b is the imaginary part $\mathrm{Im}(z)$ of z. For a complex number z, we define its magnitude as

$$|z| = \sqrt{a^2 + b^2} \tag{3.1}$$

and the phase angle as

$$\arg(a + bi) = \begin{cases} \arctan(b/a), & \text{if } a > 0 \\ \arctan(b/a) + \pi, & \text{if } a < 0 \text{ and } b \geq 0 \\ \arctan(b/a) - \pi, & \text{if } a < 0 \text{ and } b < 0 \\ \pi/2, & \text{if } a = 0 \text{ and } b > 0 \\ -\pi/2, & \text{if } a = 0 \text{ and } b < 0 \\ \text{undefined}, & \text{if } a = 0 \text{ and } b = 0. \end{cases} \tag{3.2}$$

The phase angle describes the angle of the complex number in radians, i.e., it takes values in $[-\pi, \pi)$. The complex conjugate z^* of z is defined as $z^* = a - bi$. Figure 3.1 visualizes the complex number $z = 1 + 2i$.

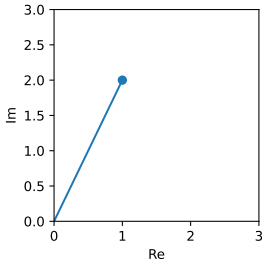


Figure 3.1: The complex number $1 + 2i$.

3.2 Discrete Fourier Transform

The Fourier transform was introduced by Joseph Fourier to study continuous functions in terms of their frequencies. However, in this thesis

we take a look at the discrete variant of the Fourier transform that is applied to a finite sequence of values, e.g., a discrete signal or a digital image. This variant is called the discrete Fourier transform (DFT) and is fundamentally important for modern signal and image processing. In the following, we want to define the Fourier transform and its inverse transform, and give a brief overview of its properties.

3.2.1 One-dimensional Discrete Fourier Transform

The DFT decomposes a vector into a sum of complex sinusoids, where each sinusoid corresponds to a different fixed frequency. This makes it an essential tool for analyzing frequencies that are present in the vector. The one-dimensional DFT maps a real vector $x \in \mathbb{R}^n$ to a complex vector $\hat{x} \in \mathbb{C}^n$ of Fourier coefficients and is defined as

$$\hat{x}[u] = \mathcal{F}(x)[u] = \sum_{k=0}^{n-1} x[k] e^{-2\pi i \left(\frac{ku}{n}\right)}. \tag{3.3}$$

Linearity of the Fourier transform

This definition can be rewritten as

$$\hat{x} = \mathcal{F}(x) = F_n x, \tag{3.4}$$

where

$$F_n = \begin{pmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega & \omega^2 & \cdots & \omega^{n-1} \\ 1 & \omega^2 & \omega^4 & \cdots & \omega^{2(n-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega^{n-1} & \omega^{2(n-1)} & \cdots & \omega^{(n-1)(n-1)} \end{pmatrix} \tag{3.5}$$

is the DFT matrix and $\omega = e^{-2\pi i/n}$ is an n-th primitive root of unity. By writing the DFT in matrix vector notation as we did in Equation 3.4, it is easy to see that the Fourier transform is a linear mapping.

Inverse Fourier transform

For a vector of Fourier coefficients $\hat{x} \in \mathbb{C}^n$ the one-dimensional inverse discrete Fourier transformation (iDFT) is defined as

$$x[k] = \mathcal{F}^{-1}(\hat{x})[k] = \frac{1}{n} \sum_{u=0}^{n-1} \hat{x}[u] e^{2\pi i \left(\frac{ku}{n}\right)}. \tag{3.6}$$

This definition can, again, be rewritten as

$$x = \mathcal{F}^{-1}(\hat{x}) = \frac{1}{n} F_n^* \hat{x}, \tag{3.7}$$

where F_n^* is the DFT matrix defined in Equation 3.5 with complex conjugated entries.

Normalization

There exist different variants to normalize the Fourier coefficients. In Equation 3.3 no normalization is performed. As a consequence we need to multiply by $\frac{1}{n}$ in Equation 3.6. An alternative way to normalizing the Fourier coefficients is to multiply each of the coefficients by

$\frac{1}{\sqrt{n}}$. By using this normalization, the Fourier transform would be an orthonormal linear transformation. In that case, the factor $\frac{1}{n}$ in Equation 3.6 has to be replaced with $\frac{1}{\sqrt{n}}$.

In practice, the fast Fourier transform (FFT) algorithm [15] is used to calculate the DFT of a vector. Applying the FFT reduces the runtime of calculating the Fourier transform of a vector with n entries from $\mathcal{O}(n^2)$, which corresponds to the naïve implementation suggested by Equation 3.3, to $\mathcal{O}(n \log n)$. This is also the reason why the FFT algorithm is among the "top 10 algorithms with the greatest influence on the development and practice of science and engineering in the 20th century" [19].

3.2.2 *Two-dimensional Discrete Fourier Transform*

For a two-dimensional DFT maps a real image $x \in \mathbb{R}^{n \times n}$ to its complex Fourier coefficients $\hat{x} \in \mathbb{C}^{n \times n}$ and is defined as

$$\hat{x}[u, v] = \mathcal{F}(x)[u, v] = \sum_{k=0}^{n-1} \sum_{l=0}^{n-1} x[k, l] e^{-2\pi i \left(\frac{ku+lv}{n}\right)}. \tag{3.8}$$

Again, this definition can be rewritten in matrix-vector notation as

$$\hat{x} = F_n x F_n^T = (F_n \otimes F_n^T) \operatorname{vec}(x) \tag{3.9}$$

where F_n is the symmetric discrete Fourier transformation matrix, $\otimes$ is the Kronecker product and $\operatorname{vec}(x)$ denotes the vector obtained by stacking the columns of x. For better readability, we restrict ourselves to square images, however the definitions can be easily adapted to non-squared images. Using the formula in Equation 3.9, it is easy to see that the two-dimensional discrete Fourier transform is also a linear mapping. Figure 3.2 visualizes the basis-functions of the Fourier transform in the image space.

Figure 3.2: Example 2D basis functions of the Fourier transform.

Once more, the FFT algorithm can be used to calculate the Fourier transform of an image in an efficient way. The Fourier transform of an $n \times n$ image can be calculated in $\mathcal{O}(n^2 \log n)$ using the FFT algorithm. This is drastically faster than the runtime of the naïve implementation, which is $\mathcal{O}(n^3)$.

Later we will also apply the Fourier transform to color images. In this case the two-dimensional Fourier transform is applied channel-wise.

3.2.3 *Phase and Magnitude*

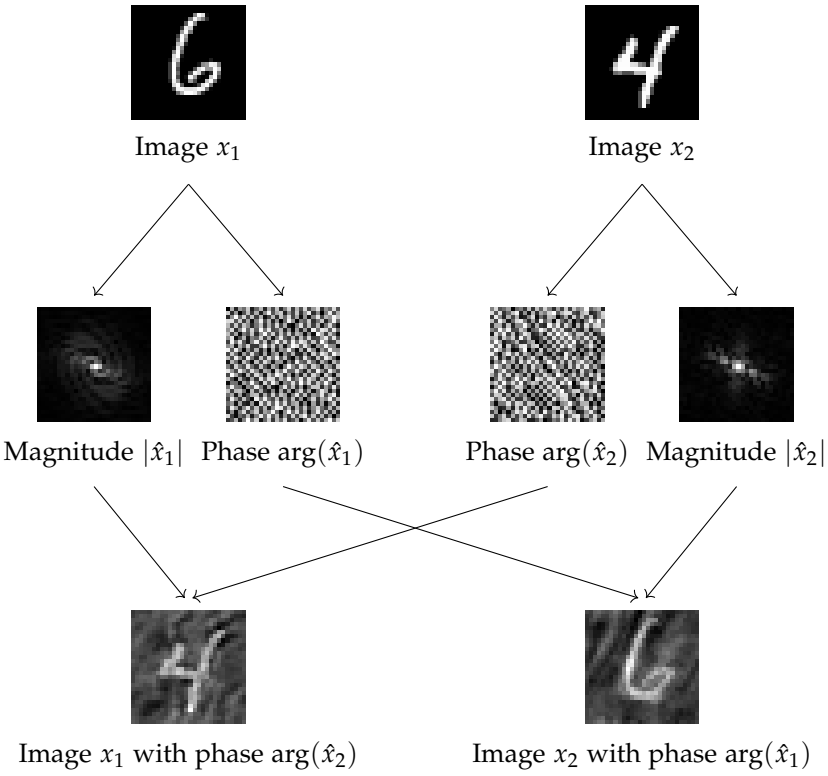


Figure 3.3: Phase swapping experiment: swapping the phase of two images results in an image that is more similar to the image whose phase is used. This demonstrates that the phase contains most of the information about an image.

After having defined the two-dimensional Fourier transform in Equation 3.8, we will now define its phase and magnitude. The magnitude of each Fourier coefficient corresponds to the magnitude of the corresponding sinusoid and the phase of each coefficient describes the phase shift of that sinusoid. The magnitude is obtained by elementwise applying Equation 3.1 to the vector of Fourier coefficients

$$y = |\hat{x}|, \tag{3.10}$$

and the phase of an image is given by element-wise calculating the argument of the Fourier coefficients, i.e.,

$$\varphi = \arg(\hat{x}). \tag{3.11}$$

The phase and the magnitude can be used to obtain the Fourier coefficients

$$\hat{x} = y \odot e^{i\varphi}, \tag{3.12}$$

where the exponentiation is meant elementwise and $\odot$ denotes the Hadamard product.

In general, the phase of an image contains more information about the image than the magnitude. This can be observed by a simple experiment: for two given images x_1 and x_2, one combines the magnitude $y_1 = |\hat{x}_1|$ of image x_1 with the phase $\varphi_2 = \arg(\hat{x}_2)$ to obtain a new image $\mathcal{F}^{-1}(y_1 \odot \exp(i\varphi_2))$ and vice versa.

In Figure 3.3, the results are shown. One can observe that the new images exhibit more characteristics of that image whose phase has been used. Using the wrong magnitude apparently only introduces cloud-like artifacts.

3.2.4 *Invariances of the Magnitude*

The Fourier transform has many interesting properties. In particular, for our setting, two invariances of the magnitudes are of interest: (i) invariance under rotations of 180° (see Figure 3.4), (ii) invariance under circular shifts (see Figure 3.5). Both properties need to be taken into account during the experimental evaluation, as they imply that reconstructions that are shifted or rotated by 180° are equally correct. Thus, the reconstructions need to be registered, i.e., aligned, before any pixel-wise metric like the MSE or the SSIM is calculated.

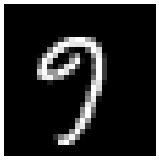


Figure 3.4: Image x_1 rotated.

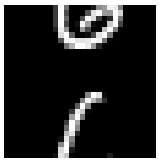


Figure 3.5: Image x_1 shifted.

Invariance Under Rotations of 180°. One can show that the Fourier coefficients $\hat{x}[u, v] = \hat{x}^*[-u, -v]$ holds. Combining this conjugate symmetry with the fact that conjugating a complex number does not change its magnitude and the fact that rotating an image by 180° is equivalent to flipping the image up to down and left to right, this property follows immediately.

Invariance Under Circular Shifts. Let $\text{shift}_{a,b}$ be the operation that circularly shifts an image by a pixels to the right and b pixels to the bottom. The magnitudes of a circularly shifted image $x_{\text{shifted}} = \text{shift}_{a,b}(x)$ are then given as

$$
\begin{aligned}
|\hat{x}_{\text{shifted}}[k,l]| &= \left| \sum_{u=0}^{n-1} \sum_{v=0}^{n-1} x[u-a, v-b] e^{-i2\pi \frac{ku+lv}{n}} \right| && \text{substitute } s = u-a \text{ and } t = v-b \\
&= \left| \sum_{s=-a}^{n-1-a} \sum_{t=-b}^{n-1-b} x[s,t] e^{-2\pi i \frac{k(s+a)+l(t+b)}{n}} \right| && (3.13) \\
&= \underbrace{\left| e^{-i2\pi \frac{ak+bl}{n}} \right|}_{=1} \left| \sum_{s=0}^{n-1} \sum_{t=0}^{n-1} x[s,t] e^{-2\pi i \frac{ks+lt}{n}} \right| \\
&= |\hat{x}[k,l]| \,.
\end{aligned}
$$

3.3 Phase Retrieval Problems

Applications

Phase retrieval is the process of reconstructing images from magnitude measurements, which is a relevant problem in different research areas, e.g., in X-ray crystallography [64], astronomical imaging [21], optics [96], array imaging [7], or microscopy [108].

General Problem

In its most general form, the phase retrieval problem is to reconstruct images $x \in \mathbb{R}^{n \times n}$ from measurements of the form

$$y = |\mathcal{A}(x)|, \tag{3.14}$$

where $\mathcal{A}$ is a known linear operator which depends on the specific application. Note, that the phase retrieval problem is more difficult than linear inverse problems due to the non-linearity (the absolute value) in the forward model.

3.3.1 Fourier Phase Retrieval

A relevant instance of this problem is the Fourier phase retrieval problem, where the operator $\mathcal{A} = \mathcal{F}$ is the two-dimensional discrete Fourier transform. In that case, the measurements are given as

$$y = |\mathcal{F}(x)|. \tag{3.15}$$

Non-oversampled case. In this work, we assume that the Fourier measurements are not oversampled, i.e., we consider the case, where the image x has not been zero-padded. Due to the symmetry of the Fourier transform, this means that the number of measurements is only half the number of pixels. Together with the non-linearity of the measurement process and the fact that most information is contained in the missing phase, this renders the problem particularly hard to solve. This setup corresponds to typical real-world settings, such as X-ray crystallography. We will consider this case in Chapters 4, 5, and 6.

Oversampled case. In contrast, a common assumption in Fourier phase retrieval is that x is zero padded and thus the problem is no longer underdetermined. The zero padding of x leads to an oversampling of the relevant signal and makes the problem much easier to solve, as a large amount of pixels in the image is known to be black. This problem is for example considered in Metzler et al. [63] and Wang et al. [98] and can also be solved using the hybrid input-output (HIO) algorithm [22].

Zero padding

3.3.2 Compressive Gaussian Phase Retrieval

A related phase retrieval problem is the compressive Gaussian phase retrieval problem which is for example discussed in the work of Candes et al. [8] and Shechtman et al. [80]. The compressive Gaussian

phase retrieval problem asks to reconstruct images from measurements that are obtained by multiplying the flattened image with a matrix with random Gaussian entries

$$y = |M \operatorname{vec}(x)|, \tag{3.16}$$

where the measurement matrix is either a real matrix $M \in \mathbb{R}^{m \times n^2}$ or a complex matrix $M \in \mathbb{C}^{m \times n^2}$ and has entries sampled from a (real or complex) Gaussian distribution. While some practitioners consider this problem a toy problem, we include it to compare with existing works that evaluate their method on this problem. Compressive Gaussian phase retrieval is discussed in Chapters 4 and 6.

3.3.3 *Fourier Phase Retrieval With a Reference Image*

Another interesting problem is the Fourier phase retrieval problem with a reference image. Instead of reconstructing the image from plain magnitude measurements, a reference image is added to the original image before the Fourier magnitudes are measured, i.e., we reconstruct the image $x \in \mathbb{R}^{n \times n}$ from the modified measurements

$$y = |\mathcal{F}(x + r)|, \tag{3.17}$$

where $\mathcal{F}$ denotes the discrete two-dimensional Fourier transform and $r \in \mathbb{R}^{n \times n}$ is a known reference that has the same shape as the image. Furthermore, we assume that both the image and the reference have non-negative entries. Figure 3.6 gives an overview of the measurement process. The problem was first mentioned by Kim and Hayes

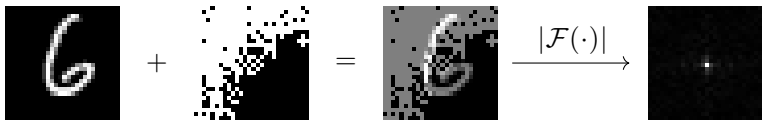


Figure 3.6: An overview of the measurement process with a known reference.

[49, 48] in 1990. Their work considered references consisting of only four adjacent white pixels on a black background. Such a reference could be implemented in a real world measurement setup. Hyder et al. [36] showed that such a reference can also be learned. Whether these learned references are usable in the real world is unclear. We will revisit the problem in Chapter 7.

3.4 *Existing Methods for Fourier Phase Retrieval*

In this section, we want to give an overview of existing methods for Fourier phase retrieval. We focus on the optimization-based algo-

Implementations

rithms that are used as baselines in this thesis, but also briefly discuss existing learning-based methods. Python 3 implementations of the optimization-based algorithms can be found in Appendix A.1.

3.4.1 Error-Reduction (ER) Algorithm

One of the first and most widely used algorithms for phase retrieval from Fourier magnitude measurements is the error reduction (ER) algorithm [22]. Assuming non-negative signals with a limited support described by a set of indices D, the algorithm iteratively enforces constraints in the Fourier domain and the image domain. Enforcing these constraints can be seen as projections onto the sets

$$\mathcal{S}_F = \left\{ x \in \mathbb{R}^{n \times n} \;\middle|\; |\mathcal{F}(x)| = y \right\} \tag{3.18}$$

and

$$\mathcal{S}_O = \left\{ x \in \mathbb{R}^{n \times n}_{\geq 0} \;\middle|\; x[k,l] > 0 \Rightarrow (k,l) \in D \right\}. \tag{3.19}$$

Projections

Note, that $\mathcal{S}_O$ is a convex set, while $\mathcal{S}_F$ is non-convex. The corresponding projections onto the sets $\mathcal{S}_D$ and $\mathcal{S}_O$ are given by

$$\mathcal{P}_F(x) = \mathcal{F}^{-1}\left(y \odot \frac{\mathcal{F}(x)}{|\mathcal{F}(x)|} \right), \tag{3.20}$$

where the division is componentwise and

$$\mathcal{P}_O(x) = \begin{cases} x[k,l], & x[k,l] \geq 0 \text{ and } (k,l) \in D \\ 0, & \text{otherwise.} \end{cases} \tag{3.21}$$

The ER algorithm can thus be seen as a fixed-point iteration

$$x^{(t+1)} = \mathcal{P}_O(\mathcal{P}_F(x^{(t)})). \tag{3.22}$$

The ER algorithm is a modification of the Gerchberg-Saxton (GS) algorithm [23] which solves a slightly different phase retrieval problem that recovers the phase from two different magnitude-only measurements. Gerchberg [23] has proven that the image error per iteration of the ER algorithm does not increase. Fienup [22] later proved that this also holds for the single measurement problem which is discussed in this work. One should note, that in the literature the ER algorithm is sometimes loosely referred to as GS algorithm, however these are, strictly-speaking, two different algorithms.

3.4.2 Fienup's Variants

Building on the ER algorithm Fienup [22] proposed three algorithms which are also based on alternating projections: the input-output (IO)

algorithm, the output-output (OO) algorithm and the hybrid input-output (HIO) algorithm. Similar to the ER algorithm, all three algorithms enforce the Fourier domain constraints in each iteration

$$\bar{x}^{(t+1)} = \mathcal{P}_F(x^{(t)}). \tag{3.23}$$

However, they differ in the second step where the corrected iterate $\bar{x}^{(t+1)}$ (the output) is combined with the previous iterate $x^{(t)}$ (the input). In the following, we show how each of the algorithms defines this second step of the iteration.

The Input-Output (IO) Algorithm. All of the three update rules distinguish between pixels in the output that violate the object domain constraints, i.e, $\bar{x}^{(t)}[k,l] < 0$, and pixels that fulfill the object domain constraints, i.e., $\bar{x}^{(t)}[k,l] \geq 0$. For pixels where the object domain constraint is fulfilled, the (basic) input output (IO) algorithm carries forward the value of the previous iterate $x^{(t)}[k,l]$ (the input), whereas for pixels where the image domain constraint is not fulfilled the new value is obtained by subtracting some amount of the output $\bar{x}^{(t+1)}[k,l]$ from the input. The complete update can be summarized as:

Input

Output

$$x^{(t+1)}[k,l] = \begin{cases} x^{(t)}[k,l], & \text{if } \bar{x}^{(t)}[k,l] \geq 0 \text{ and } (k,l) \in D \\ x^{(t)}[k,l] - \beta \bar{x}^{(t+1)}[k,l], & \text{otherwise,} \end{cases} \tag{3.24}$$

where $\beta > 0$ is a step-size parameter.

The Output-Output (OO) Algorithm. Different from the IO algorithm the output-output (OO) algorithm defines the next iterate $x^{(t+1)}[k,l]$ as the output $\bar{x}^{(t+1)}[k,l]$ for pixels not violating the object domain constraint. For pixels violating this constraint the output-output algorithm takes a step with size $\beta > 0$ in the direction of $-\bar{x}^{(t+1)}[k,l]$ starting from $\bar{x}^{(t+1)}[k,l]$. The update step can be written as:

$$x^{(t+1)}[k,l] = \begin{cases} \bar{x}^{(t+1)}[k,l], & \text{if } \bar{x}^{(t)}[k,l] \geq 0 \text{ and } (k,l) \in D \\ \bar{x}^{(t+1)}[k,l] - \beta \bar{x}^{(t+1)}[k,l], & \text{otherwise.} \end{cases} \tag{3.25}$$

The output-output algorithm can be seen as a generalization of the ER algorithm: for $\beta = 1$ the output-output algorithm produces the same iterates as the ER algorithm. Using a different value for β often improves the performance [22].

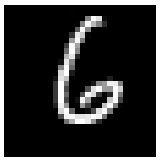


Figure 3.7: HIO reconstruction from oversampled Fourier magnitudes.

The Hybrid Input-Output (HIO) Algorithm. In the following we consider the hybrid input-output (HIO) algorithm. This is the last of Fienup's algorithms, which is the most widely used algorithm and usually produces the best results among these three algorithms. It

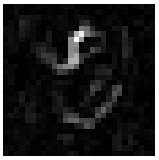


Figure 3.8: HIO reconstruction from non-oversampled Fourier magnitudes.

combines the update rules of the input-output and the output-output algorithm in the following way: for pixels violating the object domain constraints the update rule is the same as for the input-output algorithm. For pixels fulfilling the object domain constraints the hybrid input-output algorithm uses the update rule from the output-output algorithm. Summarized, the update reads as follows:

$$x^{(t+1)}[k,l] = \begin{cases} \bar{x}^{(t+1)}[k,l], & \text{if } \bar{x}^{(t)}[k,l] \geq 0 \text{ and } (k,l) \in D \\ x^{(t)}[k,l] - \beta \bar{x}^{(t+1)}[k,l], & \text{otherwise,} \end{cases} \tag{3.26}$$

where $\beta > 0$ again can be seen as a step-size. Although, this algorithm seems to work quite well in practice, its convergence has not been proven up to today.

Shortcomings of HIO

For reconstructing images from oversampled Fourier magnitudes, the HIO algorithm often converges to a suitable solution (see Figure 3.7). However, this is not the case when the magnitude measurements are not oversampled (see Figure 3.8).

3.4.3 *The Relaxed Averaged Alternating Reflections (RAAR) Algorithm*

The relaxed average alternating reflections (RAAR) algorithm is a more recent projection-based algorithm that was introduced by Luke [59]. It is a relaxation of the averaged alternating reflections (AAR) algorithm that was originally intended for finding the intersection of convex sets. The AAR algorithm was proposed by Bauschke et al. [3]. In our notation, the updates of the RAAR method are

$$x^{(t+1)}[k,l] = \begin{cases} \bar{x}^{(t+1)}[k,l], & \text{if } \bar{x}^{(t+1)}[k,l] \geq 0 \text{ and } (k,l) \in D \\ \beta x^{(t)}[k,l] - (1-2\beta)\bar{x}^{(t+1)}[k,l], & \text{otherwise,} \end{cases} \tag{3.27}$$

where $\beta > 0$ is the relaxation parameter.

3.4.4 *End-to-End Learning (E2E)*

Supervised approach

The most naïve way to apply deep learning to phase retrieval is end-to-end learning (E2E) which trains a deep neural network H_θ with learnable parameters θ that gets the magnitude measurements as input and directly outputs the reconstructions. This is achieved by minimizing the distance between the reconstruction from the network and the ground truth image from the dataset. For example, one could minimize the MSE

$$\mathcal{L}_{\text{rec}}(\theta) = \frac{1}{n} \sum_{i=1}^{n} \| x_i - H_\theta(|\mathcal{A}(x_i)|) \|_2^2 , \tag{3.28}$$

for a given dataset of images $\mathcal{D} = [x_1, \dots, x_n]$. Nishizaki et al. [69] were the first that published an E2E learning approach that uses a convolutional ResNet architecture. Alternatively, one could also use the MAE or the SSIM. We call this approach supervised as it uses images and their corresponding measurements during training. The E2E approach is evaluated in Chapters 4 and 5. During test time an image is reconstructed using a single forward pass through the network H_ϕ. This is computationally very efficient since no further optimization is performed as it is the case in the next approach.

3.4.5 *Deep Generative Priors (DPR)*

A different approach to learning-based phase retrieval are deep generative priors (DPR) which were first used by Hand et al. [30]. They use a generative model G to restrict the search space to reconstructions that come from the distribution that G was trained on. As a generative model one could for example use a GAN or a VAE. Images are reconstructed by minimizing the error between the given magnitudes and the magnitudes of the current output of the generator

$$z^* = \underset{z}{\operatorname{argmin}} \| |\mathcal{A}(G(z))| - y \|_2^2 . \tag{3.29}$$

Hand et al. [30] use gradient descent to solve the optimization problem stated in Equation (3.29). After that, the reconstruction is given by $\hat{x} = G(z^*)$. Drawbacks of this approach is the non-convex optimization at test time which requires usually restarts and many iterations until a suitable solution is reached.

3.4.6 *Unrolling a Gradient Descent Algorithm for Reference Learning*

To learn suitable reference images for the Fourier phase retrieval problem with a reference image, Hyder et al. [36] propose to unroll the following reconstruction algorithm: to reconstruct an image x for a fixed reference r from measurements y that are obtained according to Equation 3.17, the authors minimize

Reconstruction with reference image

$$\underset{x}{\operatorname{argmin}} \| y - \mathcal{F}(x + r) \|_2^2, \tag{3.30}$$

using gradient descent for T steps, where we denote the last iterate of that optimization as $x^{(T)}$. One can now use stochastic gradient descent through the reconstruction algorithm stated above to learn a reference image u from a set of training images. In detail, Hyder et al. [36] calculate the gradient with respect to r of the error

Reference learning

$$\mathcal{L}(x, x^{(T)}) = \| x - x^{(T)} \|_2^2 \tag{3.31}$$

that compares the final reconstruction $x^{(T)}$ (that depends on r) against the ground truth image x coming from a training dataset. In their work, Hyder et al. [36] show that references learned in that way provide high quality reconstructions on similar data and also generalize to different datasets.

We reproduced the results of their work and summarized our findings in a report [76]. Furthermore, in Chapter 7 we show that there is an easier way to obtain references that perform similar.

4
Learning Conditional Generative Models for Phase Retrieval

Corresponding publication. A previous version of this work was published as

Tobias Uelwer, Alexander Oberstraß, and Stefan Harmeling. Phase retrieval using conditional generative adversarial networks. In *2020 25th International Conference on Pattern Recognition (ICPR)*, pages 731–738. IEEE, 2021. DOI: 10.1109/ICPR48806.2021.9412523.

The work presented in this chapter is an extended version of the conference paper and is the result of a collaboration with Sebastian Konietzny. It is currently under review at a journal.

Personal contributions. Tobias Uelwer did the literature review, proposed the model architecture and implemented the initial model together with Alexander Oberstraß. Moreover, Tobias Uelwer designed the experimental evaluation, implemented the HIO, DPR and E2E baseline, designed the ablation experiments and performed parts of the experiments. Alexander Oberstraß and Sebastian Konietzny conducted the hyperparameter tuning and also parts of experiments. Furthermore, Tobias Uelwer created multiple plots and figures and wrote the initial draft of the manuscript which he edited together with Sebastian Konietzny, Alexander Oberstraß and Stefan Harmeling. Stefan Harmeling supervised the project.

Remark. Due to copyright issues the reconstructions of images from the CelebA dataset are not shown in this thesis.

Abstract. Reconstructing images from magnitude measurements is an important and difficult problem arising in many research areas, such as X-ray crystallography, astronomical imaging and more. While optimization-based approaches often struggle with the non-convexity

and non-linearity of the problem, learning-based approaches are able to produce reconstructions of high quality for data similar to a given training dataset. In this work, we analyze a class of methods based on conditional generative adversarial networks (CGAN). We show how the benefits of optimization-based and learning-based methods can be combined to improve reconstruction quality. Furthermore, we show that these combined methods are able to generalize to out-of-distribution data and analyze their robustness to measurement noise. In addition to that, we compare how the methods are impacted by missing measurements. Extensive ablation studies demonstrate that all components of our approach are essential and justify the choice of network architecture.

4.1 Introduction

In this chapter we explore how conditional GANs can be applied to the phase retrieval problem. We treat the phase retrieval problem as an image-to-image translation problem and use different optimization procedures at test time to achieve state-of-the-art results drastically improving upon existing results. However, one drawback of this method is that it is supervised and must be retrained when the measurement process changes. In Chapter 6 we show how an unsupervised method can be used to achieve similar results. Also, the optimization procedure of the PRCGAN at test time is costly. This problem is going to be addressed in Chapter 5.

4.2 Related work

Existing methods to solve the phase retrieval problem can be classified into two categories: optimization-based and learning-based approaches. While optimization-based approaches are especially appealing when the number of measurements is larger than the number of pixels (i.e., the measurements are oversampled), these methods usually fail in the non-oversampled regimes. Learning-based methods use additional information about the distribution of the target images to solve the phase retrieval problem (sometimes also in the non-oversampled case).

There are several other setups that are related to phase retrieval, e.g., ptychography, where we reconstruct an image given a sequence of magnitude measurements of many partially overlapping frames [103]. Another related problem is compressed sensing which asks to reconstruct images from linear measurements. Kim et al. [47] recently also proposed approaching the compressed sensing problem with conditional GANs. Generative priors with sparsity constraints for com-

pressed sensing have been discussed by Killedar et al. [46]. Solving ptychography with conditional generative adversarial networks has been done by Boominathan et al. [4] and Gupta et al. [29].

However, these are different from the problems considered in this work. To best of our knowledge we are the first to apply the conditional GAN framework to solve the Fourier and Gaussian phase retrieval problem.

4.2.1 *Optimization-based Approaches*

One of the first phase retrieval algorithms is the Gerchberg-Saxton (GS) algorithm [23], which starts with a random image and iteratively enforces a magnitude constraint in the Fourier domain and a positivity constraint of pixel intensities in the object domain. Based on the GS algorithm Fienup [22] proposed three extensions: the input-output, the output-output and the hybrid input-output (HIO) algorithm, where the last one is the most popular since it usually produces the best results among the three. Recently, Luke [59] proposed another iterative phase retrieval algorithm which is based on relaxed averaged alternating reflections (RAAR).

4.2.2 *Learning-based Approaches*

Learning-based methods are used for the oversampled Fourier phase retrieval problem and for the non-oversampled Fourier phase retrieval problem.

Oversampled Problem. Deep neural network approaches for the simpler oversampled Fourier phase retrieval problem have, e.g., been discussed by Manekar et al. [61], who propose as passive loss for end-to-end learning which is invariant to symmetries, and by Cha et al. [9], who formulate a novel loss function based on the PhaseCut algorithm [95]. The regularization-by-denoising framework for oversampled Fourier phase retrieval is discussed by Metzler et al. [63], Wang et al. [98] and Wu et al. [101].

Non-oversampled Problem. Deep neural networks for the non-oversampled Fourier phases retrieval problem have first been studied by Nishizaki et al. [69]. Their end-to-end learning approach has later been extended to a deep neural network cascade by Uelwer et al. [87].

Learning-based methods for the Gaussian phase retrieval problem include deep generative priors [30], deep generative priors with sparsity constraints [45] and untrained neural network priors [40].

4.3 Contributions

The contributions of this chapter can be summarized as follows:

1. We describe the PRCGAN, which combines a conditional generative adversarial network (CGAN) and a latent optimization procedure to solve different phase retrieval problems.

2. We study three variants of the PRCGAN: (i) in an end-to-end mode (PRCGAN-D), (ii) in combination with latent optimization (PRCGAN-L), and (iii) in combination with weight optimization of the network (PRCGAN-W).

3. We extensively evaluate all variants of the PRCGAN on the Fourier phase retrieval problem using openly available benchmark datasets such as MNIST, EMNIST, FMNIST, KMNIST, CelebA, and CIFAR-10.

4. We perform thorough ablation studies to examine the impact of each of the components of our method. We also experiment with different loss functions and model architectures.

5. We analyze how our trained models can generalize to out-of-distribution data. To do so we evaluate the performance of the models on datasets that differ from the training set. Furthermore, we create a novel dataset that contains MNIST-like symbols. We also use this dataset to evaluate our models.

6. We investigate the robustness of the different models: we study the impact of Poisson noise and additive Gaussian noise on the measurements and analyze the impact of randomly dropping magnitudes on the reconstruction process.

7. While this work focuses on Fourier phase retrieval, we show in additional experiments that the proposed methods are also applicable to the Gaussian phase retrieval problem.

4.4 End-to-End Learning and DPR

So far, we have discussed two ideas on how deep neural networks can be leveraged to solve phase retrieval problems: in Section 3.4.4 we have seen how end-to-end learning can be used to approach phase retrieval problems. The idea of end-to-end learning is to directly learn a mapping H_ϕ from magnitude measurements to the images. A different idea for a learning-based method is the DPR approach, which we presented in Section 3.4.5. DPR was introduced by Hand et al. [30]

and uses a generative model which can be used to reconstruct images by searching in the latent space of that model.

For our method PRCGAN we combine both ideas and learn a conditional generative model which has access to the magnitude information during training and is subsequently used in an optimization procedure to recover the unknown image.

4.5 PRCGAN: Combining End-to-End Learning and DPR

To increase the quality of the reconstructions we replace the generative model G of DPR with a conditional GAN (CGAN) that is conditioned on the magnitude measurements. In that way, our approach, called PRCGAN (Phase Retrieval based on CGAN) is a hybrid between the E2E and the DPR approach. By doing so, we learn a CGAN that is tailored to phase retrieval reconstruction process. The PRCGAN consists of a discriminator network D_θ with parameters θ and a generator network G_ϕ with parameters ϕ. Both networks are conditioned on the given magnitude measurement. For the latent variable z we choose a multivariate Gaussian with zero mean and unit covariance matrix and denote the latent distribution by q. After training, the generator network G_ϕ takes the role of the reconstruction network.

Generator and discriminator

4.5.1 PRCGAN: Training

The PRCGAN is trained by optimizing a combination of two loss functions: an adversarial component

$$\mathcal{L}_{\text{adv}}(\theta, \phi) = \frac{1}{n}\sum_{i=1}^{n} \log D_\theta(x_i, y_i) + \frac{1}{n}\sum_{i=1}^{n} \log\big(1 - D_\theta(G_\phi(z_i, y_i), y_i)\big), \tag{4.1}$$

where $y_i = |\mathcal{A}(x_i)|$ and $z_i \sim q$ for $i = 1, \dots, n$ and a reconstruction component

Reconstruction loss

$$\mathcal{L}_{\text{rec}}(\phi) = \frac{1}{n}\sum_{i=1}^{n} \big\|x_i - G_\phi(z_i, y_i)\big\|_p^p, \tag{4.2}$$

where p is either 1 or 2. During training the parameters θ and ϕ of D and G are optimized to solve the min-max-problem:

$$\min_\phi \max_\theta \mathcal{L}_{\text{adv}}(\theta, \phi) + \lambda \mathcal{L}_{\text{rec}}(\phi), \tag{4.3}$$

where $\lambda > 0$ is a hyperparameter that balances the two loss components.

4.5.2 PRCGAN: Reconstructions

A trained PRCGAN gives us several options for the reconstruction which span the spectrum between end-to-end learning and

optimization-based image reconstruction. Besides directly using the output of the PRCGAN, we can further enhance the results by optimizing the latent variable or even the weights of the generator itself.

End-to-end approach

PRCGAN-D: Direct Reconstruction. The PRCGAN can be directly used to reconstruct images with a single forward pass. To do so, the given magnitude measurement y and a randomly sampled value z from the latent distribution are fed into the generator G_ϕ. This gives us the reconstruction

$$\hat{x} = G_\phi(z, y). \tag{4.4}$$

While $\hat{x}$ is often a reasonable solution, we can greatly improve the reconstruction quality by employing additional optimization procedures, as we will describe next.

PRCGAN-L: Latent Optimization. Additional to feeding the magnitude y into the generator network, we can tune the latent variable z to ensure that the generated image has the correct magnitude y, or expressed as an optimization problem, we solve

$$z^* = \operatorname*{argmin}_z \left\| y - |\mathcal{A}(G_\phi(z, y))| \right\|_2^2. \tag{4.5}$$

Given an optimal z^*, the reconstruction is the output of the generator network $\hat{x} = G_\phi(z^*, y)$. Note that since the parameters of G_ϕ are fixed, we can solve this optimization in parallel for several magnitudes by passing batches through the generator. We denote this approach as PRCGAN-L.

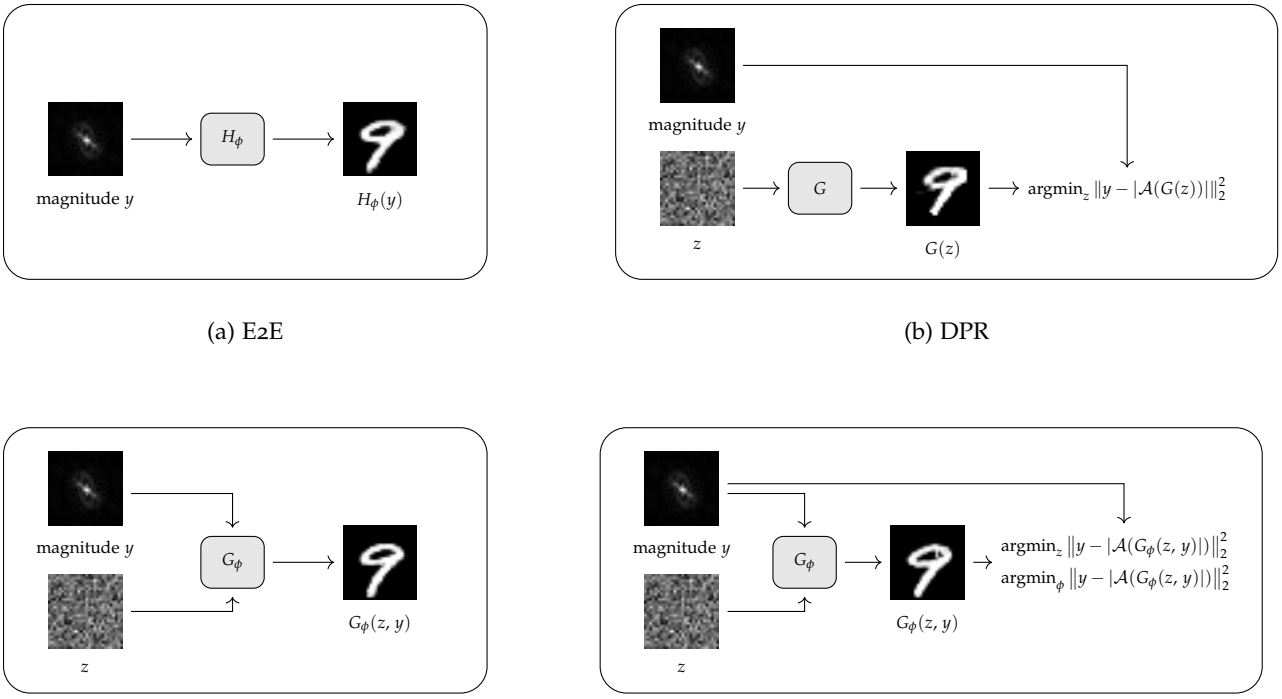
PRCGAN-W: Weight Optimization. Instead of optimizing the latent variable z, we can also search optimal the weights ϕ^* for the generator network G_ϕ to match the given magnitude y for a randomly sampled (and fixed) z, i.e.,

$$\phi^* = \operatorname*{argmin}_\phi \left\| y - |\mathcal{A}(G_\phi(z, y)|) \right\|_2^2. \tag{4.6}$$

Weight optimization in related works

A similar idea was used by Hussein et al. [34] and Ulyanov et al. [91] for linear inverse problems, like compressed sensing and denoising. Plugging the fine-tuned weights ϕ^*, the fixed latent variable z and the given magnitudes y into the generator G_ϕ yields the reconstructed image $\hat{x} = G_{\phi^*}(z, y)$. Different to the latent optimization the weight optimization can only be performed for a single magnitude at a time which makes this approach computationally more expensive. However, we hypothesize that the space of the network weights offers more flexibility to find a good reconstruction. We denote this approach as PRCGAN-W.

Figure 4.1 gives an overview of the different learning based methods considered in this work.



(a) E2E

(b) DPR

(c) PRCGAN-D

(d) PRCGAN-L and PRCGAN-W

Figure 4.1: Phase retrieval: After training the learned-models are used in different ways to reconstruct the image. We propose four variants of the PRCGAN which differ in the reconstruction process: PRCGAN-D, PRCGAN-L, and PRCGAN-W.

4.5.3 *Taxonomy of Learning-based Approaches for Phase Retrieval*

The learning-based methods discussed in the previous sections can be divided into two categories:

Unsupervised Learning: methods like the DPR approach are only trained on images $[x_1, \ldots, x_n]$. Thus we call the DPR approach unsupervised.

Supervised Learning: in contrast to that, other methods are not only trained on the images $[x_1, \ldots, x_n]$ but also have access to the corresponding magnitude measurements $[y_1, \ldots, y_n]$ during training. This allows the model to specialize on that measurement process and usually results in better image reconstruction quality. We call these methods supervised. In this work, E2E and all variants of the PRCGAN are examples thereof.

Note that the term end-to-end learning is in this work exclusively used for methods that reconstruct images in a single forward-pass through the network, having no additional optimization step during reconstruction. Therefore, E2E and PRCGAN-D are considered as end-to-end learning approaches.

4.6 Experimental Setup

4.6.1 Datasets

For our experiments we consider six different datasets. Four of these datasets consist of 28×28 grayscale images, namely, MNIST [55], FMNIST [102], EMNIST [13] and KMNIST [12]. Although, these datasets are considered toy-datasets for classification tasks, solving phase retrieval on these datasets is a non-trivial problem. The other two datasets consists of color-images: the CelebA dataset [58] and the well-known CIFAR-10 dataset [53]. We rescaled both images to 64×64 resolution.

No oversampling

Fourier magnitudes were calculated without oversampling, i.e., the magnitudes had the same dimensionality as the images from the dataset.

4.6.2 Architecture

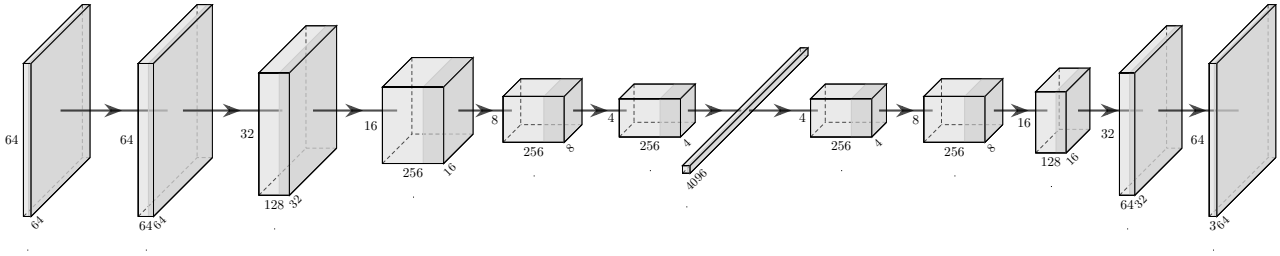


Figure 4.2: Architecture of the generator used to reconstruct the CelebA images from their magnitude.

For the grayscale image datasets (MNIST, FMNIST, KMNIST and EMNIST) we use a multilayer perceptron (MLP) consisting of five layers having each 2048 hidden units. Empirically, we found out that the fully-connected layers are better suited for the global structure of the Fourier phase retrieval problem than convolutional layers. We use batch-normalization [37] and ReLU nonlinearities for the intermediate layers and a sigmoid function for the final layer as the pixel intensities of the images are assumed to be normalized between 0 and 1.

Since the MLP architecture is no longer feasible for the increased pixel count of the color-image datasets CIFAR-10 and CelebA, we use a convolutional neural network (CNN) with two fully-connected intermediate layers for the generator network. Figure 4.2 gives an overview of the used architecture.

For the DPR implementation we follow Hand et al. [30]. That means, a variational autoencoder (VAE) [51] is used for the MNIST-like datasets, where the dimension of the latent variable z is chosen to be 128 and two ReLU activated fully-connected layers with 500 hidden

units each are used. For the CelebA and CIFAR-10 datasets a DCGAN architecture [73] is used with batch-normalization [37] and ReLU activation functions.

4.6.3 *Baselines*

We also compare our PRCGAN approach with classical methods that do not have a learning component. The most commonly used phase retrieval algorithm is Fienup's HIO algorithm [22]. The optimal hyperparameters of HIO (1000 iterations and step-size $\beta = 0.8$) were determined using a validation dataset, and two random restarts (out of three reconstruction we used the one with the lowest magnitude error). Another optimization-based method that we evaluate is the relaxed-averaged-alternating-reflections algorithm (RAAR) proposed by Luke [59]. Here, we also ran 1000 iterations with two random restarts. We set the step-size to $\beta = 0.87$ which was reported to perform best in the original work.

HIO and RAAR

For the learning-based E2E approach, we considered different architectures for the generator and different loss functions. The best performance was obtained with the MSE for the MNIST-like datasets, and with the MAE for the color-image datasets.

E2E

For the DPR approach we tried an non-conditional GAN and a VAE for the generator. For the MNIST-like datasets the VAE performed better, so we only report results for the VAE-based DPR method. For the other datasets, we use a DCGAN [73] as the underlying generative model which produced better results. After training each model, we initialized the latent variable with samples from a standard Gaussian distribution and performed 10,000 iterations of Adam with a step size of 0.1, which we found to perform best. In order to get useful results, we allowed multiple restarts and used the reconstruction with the lowers magnitude error.

Deep generative priors

4.6.4 *Training and Optimization*

We selected all hyperparameters to be optimal on a separate validation dataset. We trained all previously mentioned learning-based models with a batch size of 32 for the MNIST-like datasets and 64 for the color-images, using the Adam optimizer [50]. We trained the PRCGAN for 100 epochs for all datasets except for CIFAR-10, where we increased the number of epochs to 250. We set $\lambda = 100$ for MNIST, EMNIST, and KMNIST and used $\lambda = 1000$ for FMNIST, CelebA, and CIFAR-10.

Hyperparameters

Analogous to the DPR approach, we optimized the latent variable z using 10,000 steps with a learning rate of 0.1. We observed that even without restarting, our approach outperformed DPR, so we decided to eliminate the random restarts to keep the computational effort limited.

In the weight optimization PRCGAN-W, we also used 10,000 steps but with a decreased learning rate of 0.01 for the latent optimizer and 10^{-6} for the weight optimizer.

4.6.5 Evaluation

For grayscale datasets MNIST, FMNIST, EMNIST and KMNIST we compare the mean squared error (MSE), the mean absolute error (MAE) and the structural similarity index measure (SSIM) [99]. For datasets consisting of color-images (CelebA and CIFAR-10) we report the MSE, the LPIPS [107] and the SSIM. For the grayscale images with black background we observe that the reconstructions are sometimes flipped and translated after performing the latent optimization. This is to be expected as flipping and translating the image does not change the magnitudes in the case of Fourier phase retrieval. Therefore, we consider these reconstructions to be equally correct and perform image registration with the target image before calculating the errors. We use cross-correlation [27] to align the reconstruction and the flipped reconstruction with the target image and report the better metric. For the color images from CelebA and CIFAR-10 datasets we did not observe any flips or shifts and therefore omitted the registration step. We use 1024 images in each test set to limit the computational time.

Flipping and shifting

4.7 Experiments

4.7.1 Results for Fourier Phase Retrieval

In the following we discuss the results of the optimization- and learning-based methods for Fourier phase retrieval.

HIO. On the MNIST-like datasets the HIO algorithm reconstructs images with many artifacts. In the other cases HIO is not successful at all, resulting in fragmented, blurry reconstructions. Furthermore, for the color-images of the CIFAR-10 and the CelebA dataset HIO does not produce any useful results.

RAAR. Overall, we observe slightly worse performance when comparing with the results of HIO. For the color-image datasets RAAR also does not succeed at reconstructing the images.

End-to-End. In comparison to the optimization-based methods the E2E approach does not produce fragmented parts. Although some reconstructions are still blurry, E2E performs better in all six datasets than HIO and RAAR.

DPR. On MNIST, the latent optimization approach produces better visual appearance of the digits. To avoid local optima, we did multiple random restarts. However, sometimes DPR still does not work as well as E2E, as can be seen in the first and second image shown in Figure 4.4. While DPR got slightly better MNIST reconstructions than the E2E approach, it is having difficulties on the other datasets.

PRCGAN. The PRCGAN combines end-to-end learning with a subsequent optimization, so we are expecting better results than the E2E and the DPR approach. The most basic PRCGAN-D achieves similar performance than the E2E method, as one can see in Table 4.1 and 4.2. However, due to the adversarial loss component, the PRCGAN alleviates the problem of blurriness and gets much more realistic reconstructions. As one can see in the second and eighth image in Figure 4.4 (columns 2 and 8) the reconstructions show finer texture components like the text and the checkered pattern on the shirts, respectively. Quantitatively, the blurry E2E reconstructions are more favorable than the reconstructions of PRCGAN-D, since the MSE, MAE, and SSIM punish misplaced sharp edges more than blurriness.

The variations PRCGAN-L and PRCGAN-W, that optimize latent variables or weights with respect to the magnitude, produce the best reconstructions regarding both qualitative and quantitative performance on all datasets except for CIFAR-10. It is remarkable that on FMNIST, we were even able to reconstruct the text shown on the second image in Figure 4.4, where all other baseline methods failed. While our presented optimization approaches do not significantly differ in quality, we observe that PRCGAN-L with a few minor exceptions achieves the best reconstructions. One exception are the results of PRCGAN-W, which achieves a lower MAE for MNIST and EMNIST. CIFAR-10 shows the limits of learning-based methods for phase-retrieval: due to the high variation of the dataset, none of the methods recovers good images. Quantitatively, E2E is slightly better than the other approaches. E2E produced blurry images, while our PRCGAN-D is trying to create images with edges.

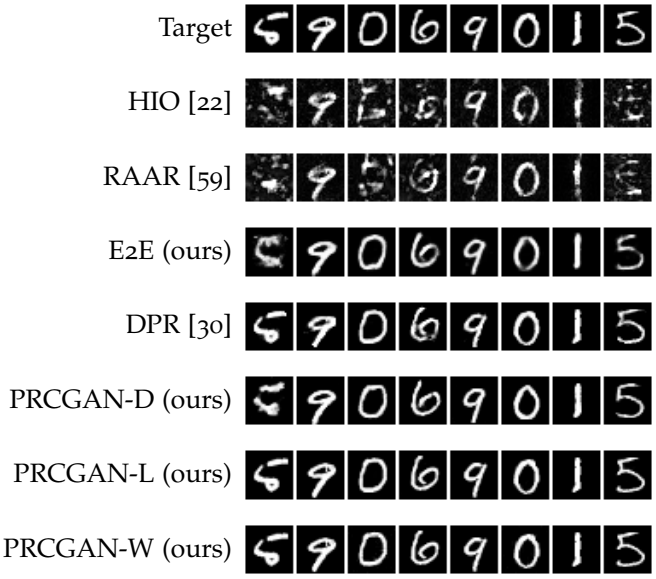


Figure 4.3: Registered reconstructions from the Fourier magnitudes of samples from the MNIST test dataset for each model.

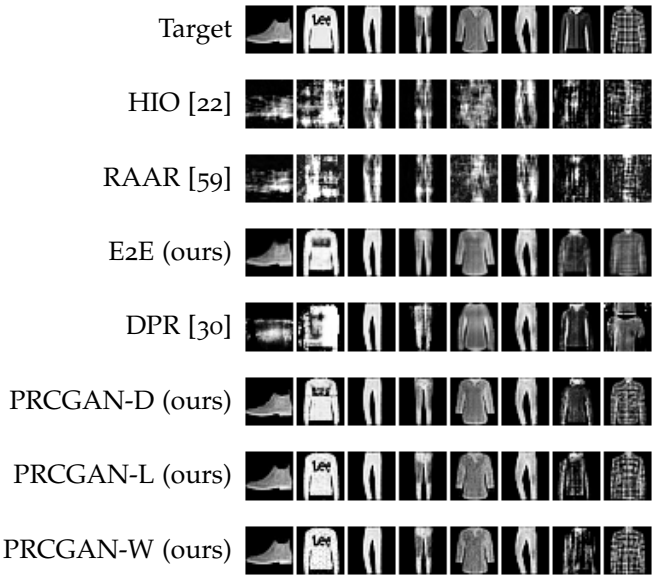


Figure 4.4: Registered reconstructions from the Fourier magnitudes of samples from the FMNIST test dataset for each model.

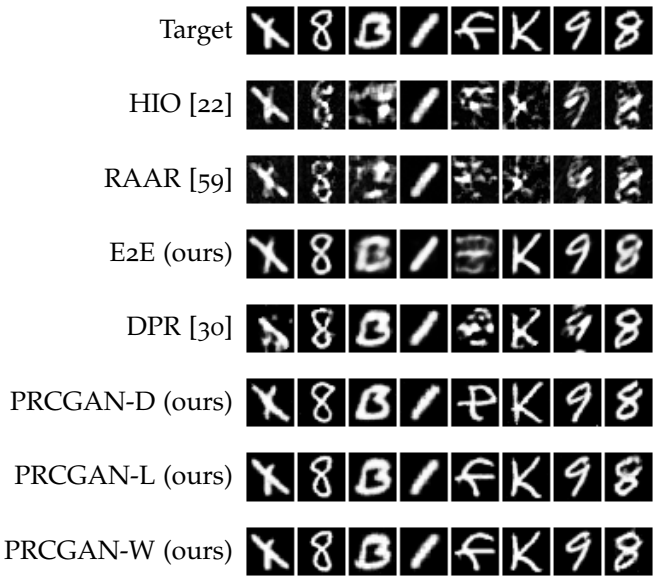


Figure 4.5: Registered reconstructions from the Fourier magnitudes of samples from the EMNIST test dataset for each model.

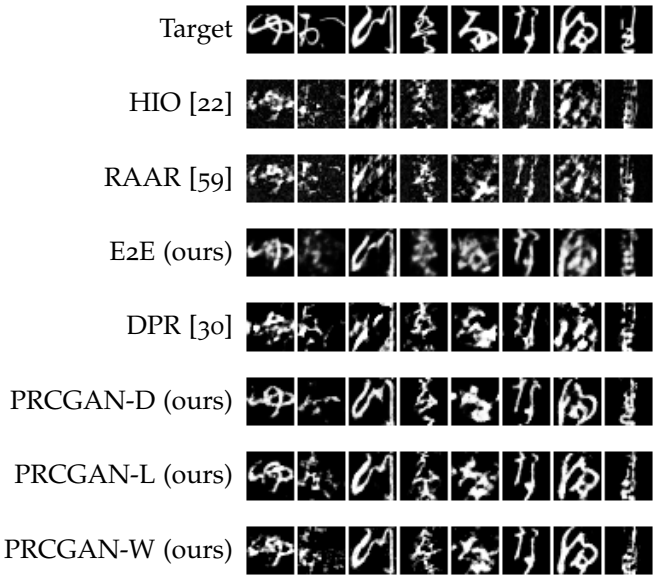


Figure 4.6: Registered reconstructions from the Fourier magnitudes of samples from the KMNIST test dataset for each model.

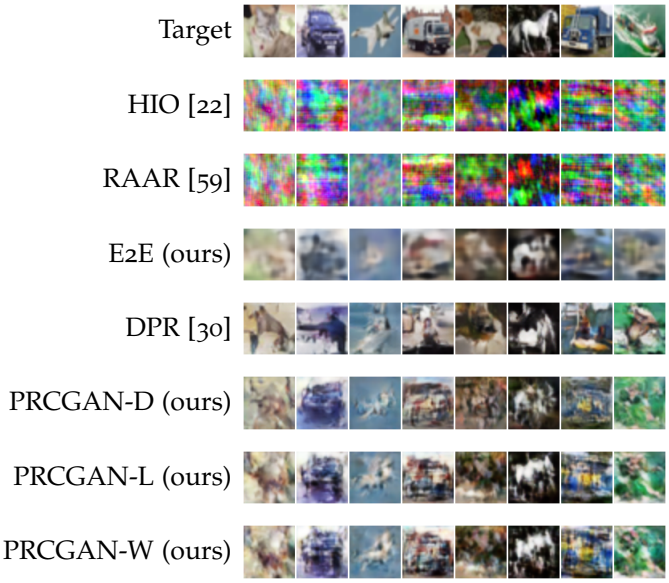


Figure 4.7: Registered reconstructions from the Fourier magnitudes of samples from the CIFAR-10 test dataset for each model.

	MNIST			FMNIST		
Method	MSE (↓)	MAE (↓)	SSIM (↑)	MSE (↓)	MAE (↓)	SSIM (↑)
HIO [22]	0.0440	0.1016	0.5274	0.0649	0.1608	0.4019
RAAR [59]	0.0489	0.1150	0.4879	0.0668	0.1673	0.3491
E2E (ours)	0.0164	0.0429	0.8191	0.0129	0.0564	0.7400
DPR [30]	0.0139	0.0302	0.8685	0.0288	0.0855	0.6093
PRCGAN-D (ours)	0.0185	0.0415	0.8196	0.0149	0.0569	0.7414
PRCGAN-L (ours)	**0.0009**	0.0045	**0.9890**	**0.0084**	**0.0403**	**0.8376**
PRCGAN-W (ours)	0.0010	**0.0034**	0.9867	0.0090	0.0429	0.8266
	EMNIST			KMNIST		
Method	MSE (↓)	MAE (↓)	SSIM (↑)	MSE (↓)	MAE (↓)	SSIM (↑)
HIO [22]	0.0645	0.1364	0.4942	0.0920	0.1721	0.3592
RAAR [59]	0.0668	0.1425	0.4767	0.0931	0.1774	0.3773
E2E (ours)	0.0223	0.0656	0.7598	0.0538	0.1193	0.5510
DPR [30]	0.0326	0.0685	0.7480	0.0884	0.1446	0.4504
PRCGAN-D (ours)	0.0318	0.0692	0.7546	0.0694	0.1192	0.5535
PRCGAN-L (ours)	0.0066	0.0255	0.9416	**0.0393**	**0.0775**	**0.7384**
PRCGAN-W (ours)	**0.0063**	**0.0218**	**0.9436**	0.0402	0.0787	0.7259

Table 4.1: Quantitative evaluation for MNIST, FMNIST, EMNIST and KMNIST for the registered reconstructions from the Fourier magnitudes. MSE, MAE: lower is better. SSIM: higher is better. Best values are printed **bold** and second-best are underlined.

	CelebA			CIFAR-10		
Method	MSE (↓)	SSIM (↑)	LPIPS (↓)	MSE (↓)	SSIM (↑)	LPIPS (↓)
HIO [22]	0.1005	0.0510	0.8228	0.0814	0.0881	0.7827
RAAR [59]	0.1011	0.0537	0.8183	0.0808	0.0927	0.7779
E2E (ours)	0.0123	0.6367	0.2683	**0.0390**	**0.2735**	0.5852
DPR [30]	0.0388	0.4185	0.3529	0.0707	0.1713	0.5819
PRCGAN-D (ours)	0.0155	0.5653	0.2655	0.0402	0.2297	0.5403
PRCGAN-L (ours)	**0.0093**	**0.6846**	**0.2182**	0.0489	0.2219	**0.5401**
PRCGAN-W (ours)	0.0115	0.6405	0.2835	0.0492	0.2173	0.5487

Table 4.2: Quantitative evaluation for CelebA and CIFAR-10 for the reconstructions from the Fourier magnitudes. Note that we do not register the reconstructions for these datasets. Best values are printed **bold**.

4.7.2 *Computational Runtime*

Reconstructing a single image of shape $3\times64\times64$ using RAAR or HIO takes approximately 3.70 seconds on an AMD EPYC 7742 CPU (including two restarts). The end-to-end approaches E2E and PRCGAN-D, which reconstruct the images in a single forward pass, are the fastest methods considered in this work. They take 0.16 and 0.02 seconds, respectively. The latent optimization of the DPR approach takes 137.24 seconds, as it used two random restarts. In contrast to that PRCGAN-L takes 85.76 seconds but does not use any restarts. The weight optimization approach (PRCGAN-W) runs for 138.89 seconds to reconstruct a single image. In contrast to DPR and PRCGAN-L, which can process a dataset of images batchwise, PRCGAN-W needs to process each image separately. Thus it scales linearly with the number of images and not with the number of batches as it is the case for DPR and PRCGAN-L. The runtimes of the learning-based methods are measured on an NVIDIA A100 GPU. In conclusion PRCGAN-L gives excellent reconstructions in most cases and has reasonable runtime.

4.7.3 *Dissecting the PRCGAN: Ablation Experiments*

In this section, we explain what the PRCGAN has learned by presenting the results of the following ablation experiments:

Which impact does the choice of the reconstruction loss have? We are interested which impact the choice of the reconstruction loss function $\mathcal{L}_{\text{rec}}$ has on the performance of the PRCGAN-D and PRCGAN-L model. In addition to the MAE and the MSE we also consider LPIPS, which is a perceptual loss function. Results are shown in Table 4.3. Overall, the performance of the PRCGAN-D is improved by using LPIPS as reconstruction loss, however when considering PRCGAN-L the effect decreases. This justifies our initial choice of the reconstruction loss.

Method	$\mathcal{L}_{\text{rec}}$	CelebA			CIFAR-10		
		MSE (↓)	SSIM (↑)	LPIPS (↓)	MSE(↓)	SSIM (↑)	LPIPS(↓)
PRCGAN-D	MAE	0.0155	0.5653	0.2655	0.0402	0.2297	**0.5403**
PRCGAN-D	MSE	0.0149	0.5535	0.3028	0.0422	**0.2432**	0.5878
PRCGAN-D	LPIPS	**0.0143**	**0.5856**	**0.2418**	**0.0385**	0.2390	0.5409
PRCGAN-L	MAE	**0.0093**	**0.6846**	0.2182	0.0489	0.2219	0.5401
PRCGAN-L	MSE	0.0125	0.6110	0.2811	0.0534	0.2133	0.5692
PRCGAN-L	LPIPS	0.0096	0.6804	**0.2126**	**0.0479**	**0.2279**	**0.5359**

Table 4.3: Comparison of reconstruction performance for different choices of reconstruction loss functions $\mathcal{L}_{\text{rec}}$. Best values are printed **bold**.

Is the magnitude passed to the CGAN being used for the reconstruction? In the basic PRCGAN-D approach, the magnitudes are processed by the CGAN to recover the image. However, for the other PRCGAN variants we are also employing the magnitudes in the subsequent optimization. To evaluate the influence of these two roles of the magnitude, we run experiments where we used wrong magnitudes as inputs to the model on purpose while afterwards solving the optimization problem stated in Equation 4.5 with the correct magnitudes. Table 4.4 shows that the performance completely drops, which shows that for PRCGAN-L, the magnitude input to the generator is essential.

Method	MNIST		
	MSE (↓)	MAE (↓)	SSIM (↑)
PRCGAN-L	**0.0009**	**0.0045**	**0.9890**
PRCGAN-L (conditioned on labels)	0.0160	0.0361	0.8442
PRCGAN-L (wrong magnitudes)	0.0244	0.0455	0.7801
PRCGAN-L (without adversarial loss)	0.0289	0.0523	0.7363

Table 4.4: Quantitative evaluation of the different ablation experiments. Best values are printed **bold**.

Is it sufficient to condition on the label? Instead of feeding magnitudes to the CGAN, one could argue that just the labels should be sufficient for successful reconstruction. To answer this question we train a CGAN conditioned on the labels and attempt to reconstruct the image given the correct label information. While the label might not be available in practice, this experiment helps us to understand what information is relevant. As in the previous study, the results of only using label information are inferior to using the magnitude (see Table 4.4).

Can we drop the adversarial component in the loss? Next, we train a PRCGAN by only minimizing the reconstruction loss $\mathcal{L}_{\text{rec}}$. Note that this approach is not identical to E2E since have a latent noise variable and

	CelebA					
	with fully-connected layer			without fully-connected layer		
	MSE (↓)	SSIM (↑)	LPIPS (↓)	MSE (↓)	SSIM (↑)	LPIPS (↓)
PRCGAN-D	**0.0155**	**0.2655**	**0.5653**	0.0260	0.4902	0.2903
PRCGAN-L	**0.0093**	**0.2182**	**0.6846**	0.0153	0.5899	0.2697
PRCGAN-W	**0.0115**	**0.2835**	**0.6405**	0.0156	0.5834	0.3021

Table 4.5: Ablation study for the intermediate fully-connected layer used in the neural network architecture for the CelebA and CIFAR-10 dataset. Best values are printed **bold**.

a subsequent optimization of it. Again, the performance worsens (see Table 4.4).

Can we drop the fully-connected intermediate layer? For the color images, the PRCGAN consists of several convolutional layers, two intermediate fully-connected layers and several transposed convolutional layers (see Figure 4.2). The motivation for the fully connected layers is that it helps to model the global structure of the phase retrieval problem. To measure the influence of it, we train the same model without the intermediate fully-connected layers. The results are shown in Table 4.5 and confirm our design choice to include a fully connected intermediate layer. Also, the model without it was more difficult to train due to numerical instabilities.

4.7.4 Robustness Against Distributional Shifts

In real-world applications we often do not have examples from the true image distribution. Instead we might be able to train a model on a dataset that is only similar to some degree. To simulate this situation, we consider models trained on the MNIST digits and compare to what extend we can reconstruct letters from the EMNIST dataset. Since the MNIST dataset is a subset of the EMNIST dataset, we ensure that only images showing letters are used for evaluation. Table 4.6 shows that all three variants of the PRCGAN are the best approaches for this setup. For several exemplary reconstructions refer to Figure 4.9. To further confirm these generalization abilities, we expanded our experiments on the four grayscale datasets. We evaluate E2E, DPR, PRCGAN-D, PRCGAN-L, and PRCGAN-W trained on MNIST, FMNIST, EMNIST and KMNIST images, respectively, with each of the other datasets. Figure 4.8 summarizes the reconstruction performance of the learned methods on out-of-distribution data. PRCGAN-L and PRCGAN-W perform best in eight out of twelve cases (only considering those where training and testing datasets differ).

Carrying on the idea of reconstructing arbitrary shapes we have created a small dataset consisting of 32 MNIST-like symbols and bench-

	EMNIST (only letters)		
Method (trained on MNIST)	MSE (↓)	MAE (↓)	SSIM (↑)
E2E (ours)	<u>0.0535</u>	0.1026	0.5486
DPR [30]	0.0427	0.0814	0.6670
PRCGAN-D (ours)	0.0567	0.1025	0.5710
PRCGAN-L (ours)	**0.0269**	**0.0604**	**0.7751**
PRCGAN-W (ours)	**0.0269**	<u>0.0636</u>	<u>0.7492</u>

Table 4.6: Training on MNIST digits leads to reasonable results on letters of EMNIST. Quantitative evaluation for the registered reconstructions from the Fourier magnitudes. MSE, MAE: lower is better. SSIM: higher is better. Best values are printed **bold** and second-best are <u>underlined</u>.

mark the different methods on this dataset. Here, we consider models trained on MNIST and EMNIST, and evaluate both using Fourier measurements. Refer to as Table 4.7 and Figure 4.10. Note that although the results are generally better for the models trained on EMNIST, the PRCGAN variants always perform better than E2E and DPR.

	trained on MNIST		
Method	MSE (↓)	MAE (↓)	SSIM (↑)
E2E (ours)	0.0916	0.1583	0.3974
DPR [30]	0.0712	0.1192	0.6221
PRCGAN-D (ours)	0.1095	0.1610	0.3770
PRCGAN-L (ours)	<u>0.0498</u>	<u>0.0942</u>	<u>0.7337</u>
PRCGAN-W (ours)	**0.0484**	**0.0930**	**0.7356**

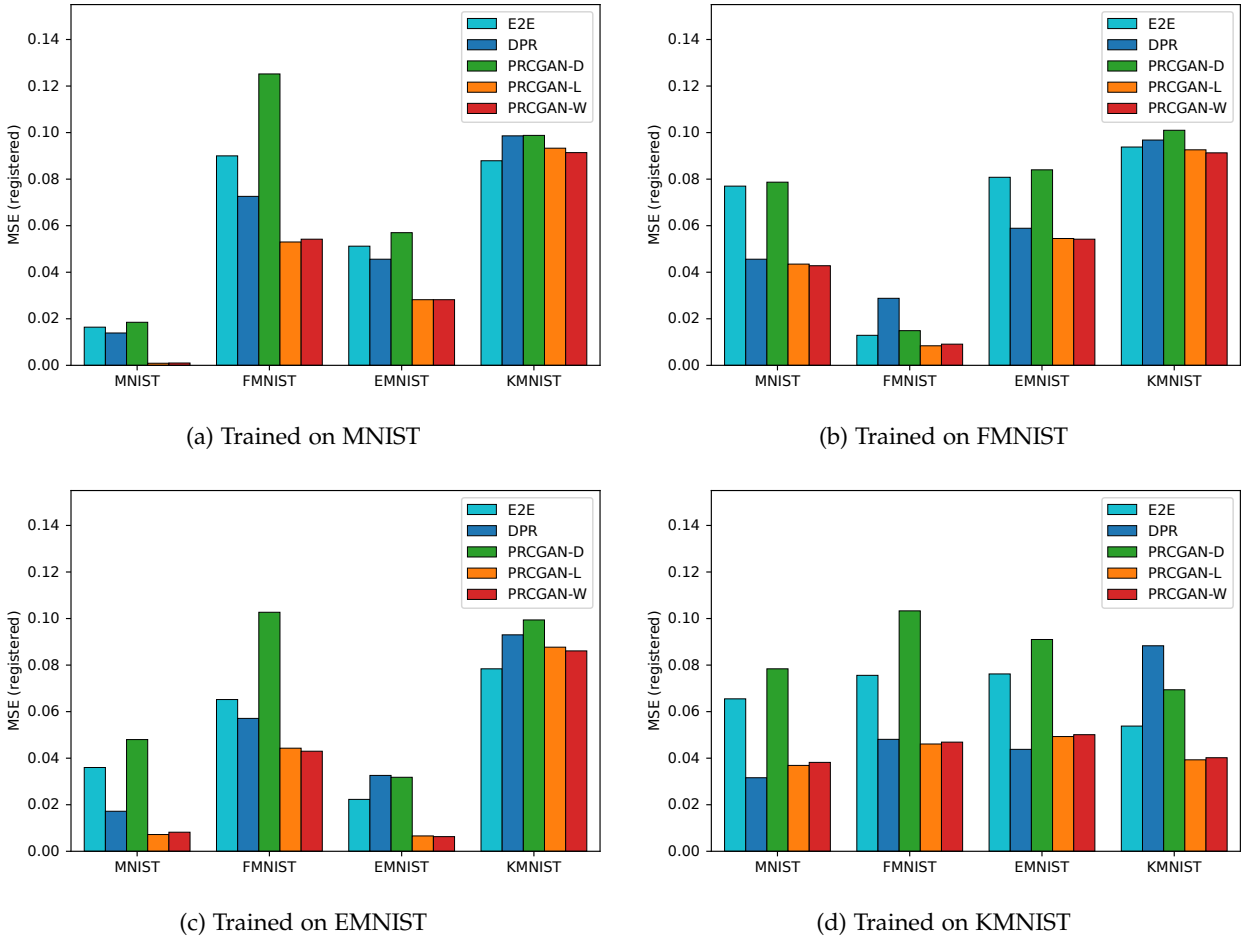
Table 4.7: Evaluation of 32 MNIST-like symbols for the registered reconstructions from the Fourier magnitudes, with the specified methods trained on MNIST. MSE, MAE: lower is better. SSIM: higher is better. Best values are printed **bold** and second-best are <u>underlined</u>.

	trained on EMNIST		
Method	MSE(↓)	MAE (↓)	SSIM (↑)
E2E (ours)	0.0653	0.1413	0.4812
DPR [30]	0.0499	0.0940	0.7280
PRCGAN-D (ours)	0.0967	0.1522	0.4422
PRCGAN-L (ours)	<u>0.0452</u>	<u>0.0902</u>	<u>0.7282</u>
PRCGAN-W (ours)	**0.0347**	**0.0751**	**0.7910**

Table 4.8: Evaluation of 32 MNIST-like symbols for the registered reconstructions from the Fourier magnitudes, with the specified methods trained on EMNIST. MSE, MAE: lower is better. SSIM: higher is better. Best values are printed **bold** and second-best are <u>underlined</u>.

4.7.5 Robustness Against Noise

All learning-based phase retrieval approaches discussed in this work were trained on synthetic noiseless measurements. However, in practice Fourier magnitudes measurements often exhibit different kinds of noise which may influence the reconstruction process. In this section, we train the different models on noise-free images and then study the



(a) Trained on MNIST

(b) Trained on FMNIST

(c) Trained on EMNIST

(d) Trained on KMNIST

Figure 4.8: Generalization to out-of-distribution data: MSE of registered reconstructions from noisy Fourier magnitudes from the grayscale datasets.

their robustness to noise during the reconstruction process. More precisely, we consider magnitude measurements corrupted by two types of noise. First, we take a look at Poisson noise, which is the type of noise usually present in many phase retrieval applications [105]. The noisy magnitude measurements are then given as

$$\tilde{y} = \alpha \sqrt{s}, \text{ with } s \sim \text{Poisson}\left(\frac{y^2}{\alpha^2}\right), \tag{4.7}$$

where the parameter α controls the amount of noise (larger values for α correspond to stronger noise). A similar measurement process has been considered by Metzler et al. [63]. Second, we consider additive white Gaussian noise, i.e., the measurements are given as

$$\tilde{y} = y + \alpha s, \text{ with } s \sim \mathcal{N}(0,1), \tag{4.8}$$

where, again, α controls the amount of noise.

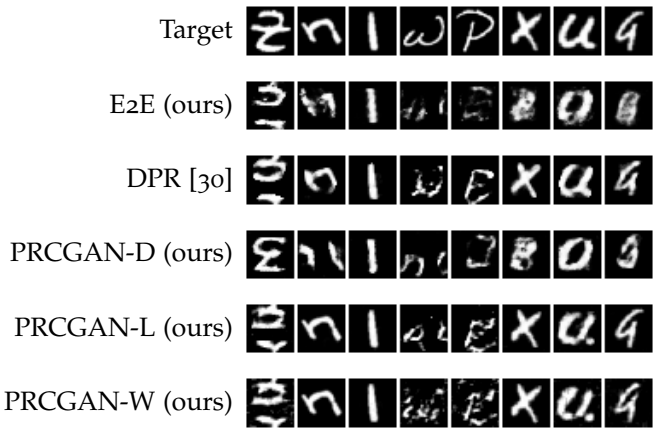


Figure 4.9: Generalization to out-of-distribution data: registered reconstructions of the letters from EMINST. All models were trained on MNIST.

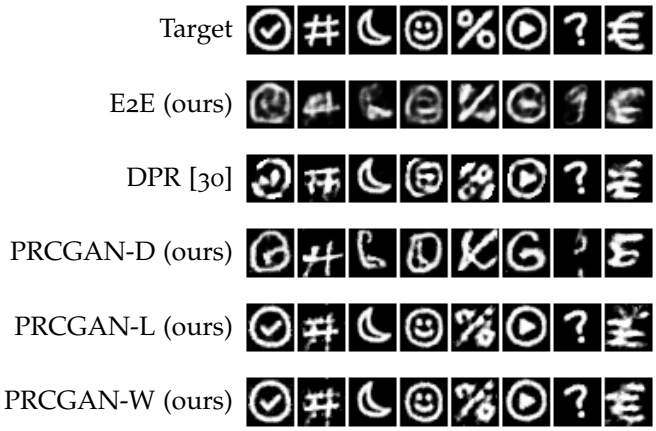
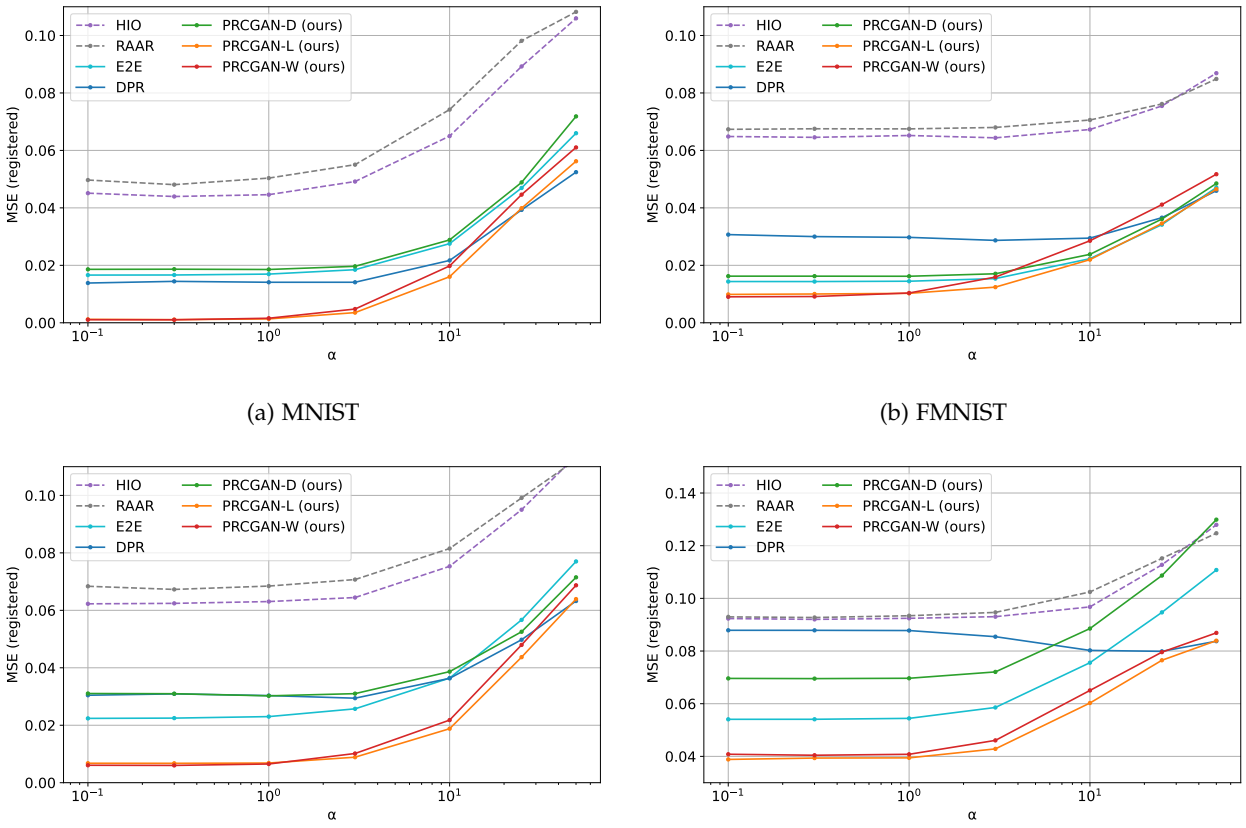


Figure 4.10: Generalization to out-of-distribution data: registered reconstructions of images from our MNIST-like symbols dataset. All models were trained on EMNIST.



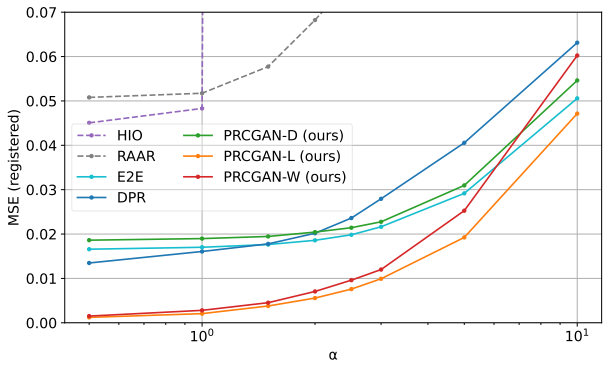
(a) MNIST

(b) FMNIST

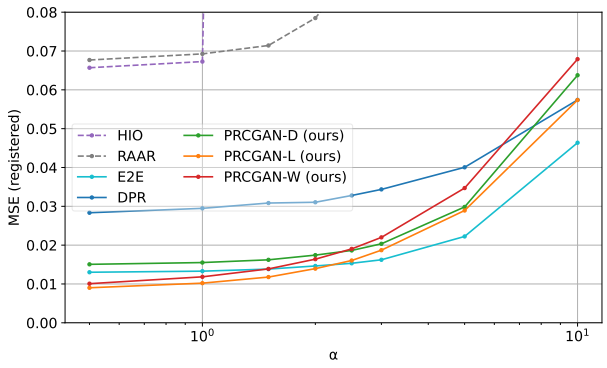
(c) EMNIST

(d) KMNIST

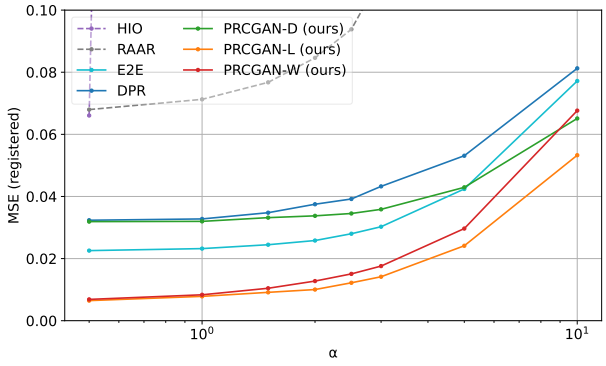
Figure 4.11: Noise robustness: MSE of registered reconstructions from Fourier magnitudes perturbated with Poisson shot noise. Larger α implies stronger noise.



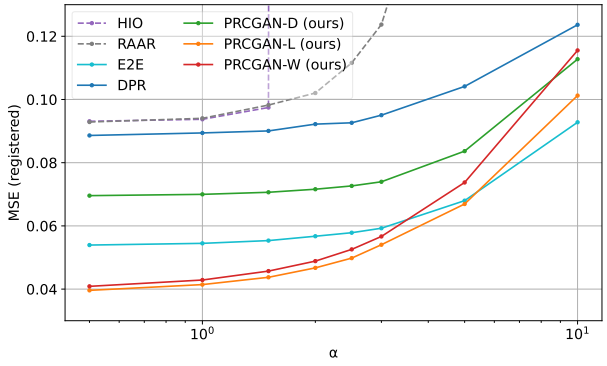
(a) MNIST



(b) FMNIST



(c) EMNIST



(d) KMNIST

Figure 4.12: Noise robustness: MSE of registered reconstructions from Fourier magnitudes perturbed with additive white Gaussian noise. Larger α implies stronger noise.

Figure 4.11 compares the performance of all methods regarding the robustness against noise in the measurements used for the reconstruction. More precisely, we plot the MSE of 1024 reconstructions against different noise levels, i.e., values of α, on the MNIST (top-left panel), FMNIST dataset (top-right panel), EMNIST (bottom-left panel) and KMNIST (bottom-right panel). On MNIST, the proposed methods are robust to noise up to $\alpha = 3$. On FMNIST, however, PRCGAN-W is less robust compared to PRCGAN-L. While for small amounts of noise our approaches gives the best results, for the larger amounts of noise, DPR is equally good or even slightly better. This might be due to the fact, that DPR uses random restarts, which might have helped coping with local optima in the large noise regime. Also, PRCGAN-L and PRCGAN-W perform best on EMNIST and KMNIST.

4.7.6 *Robustness Against Randomly Missing Measurements*

Furthermore, we consider the task of reconstructing images from partial Fourier measurements and analyze how the different methods are impacted by missing measurements. Some of the entries are randomly set to zero by multiplying the measurements with a binary mask b (that exhibits the same symmetries as the Fourier transform). In detail, the measurements are given as

Random binary masks

$$y = b \odot |\mathcal{F}x|, \tag{4.9}$$

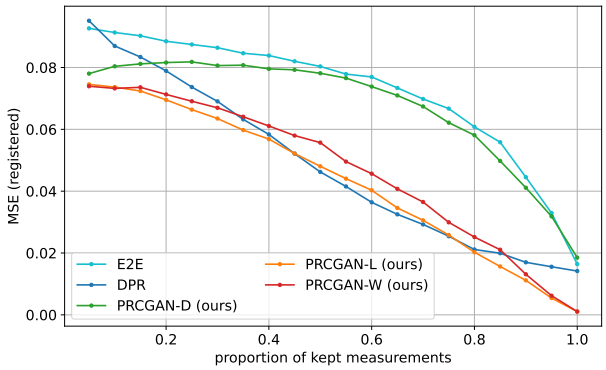
where $\odot$ denotes elementwise multiplication.

Both unsupervised and supervised methods use the magnitudes for the reconstruction error, where the terms of the missing entries are omitted, i.e., images were reconstructed by minimizing the loss $\|y - b \odot |\mathcal{F}G(z)|\|_2^2$ for DPR and $\left\|y - b \odot |\mathcal{F}G_\phi(z, y)|\right\|_2^2$ for PRCGAN-L. We did not retrain the models.

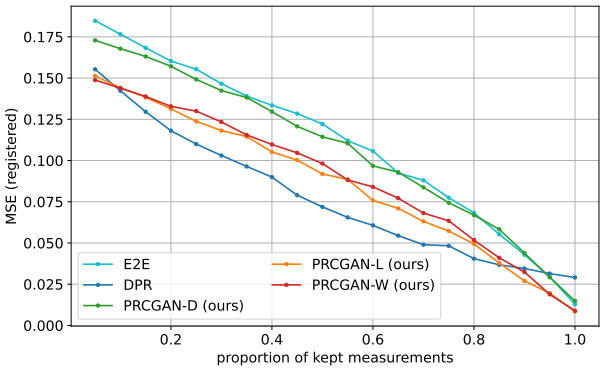
Figure 4.13 shows the MSE on the MNIST and the FMNIST dataset. Interestingly, methods that get the measurements as input perform worse than the DPR approach. The supervised methods are further affected by missing magnitudes, since they also use them as input for the network. Further training on masked Fourier magnitudes could help mitigating this effect.

4.7.7 *Results for Compressive Phase Retrieval*

Besides the Fourier phase retrieval problem, a common studied problem is the compressive Gaussian phase retrieval problem. In this setup a measurement matrix $A \in \mathbb{R}^{m \times n}$ is sampled from a Gaussian distribution $\mathcal{N}(0, 1/m)$ for varying numbers of measurements m. By increasing the number of measurements m we can smoothly adjust the



(a) MNIST

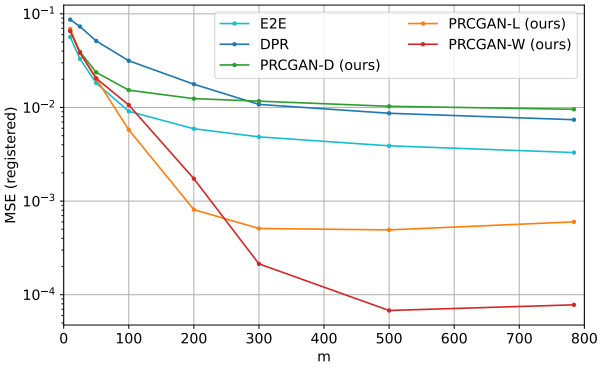


(b) FMNIST

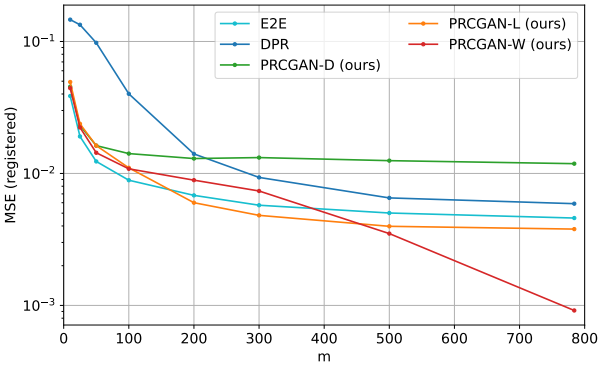
Figure 4.13: Reconstructions from partial Fourier magnitudes: MSE of registered reconstructions from partial Fourier magnitudes.

difficulty of the problem. For a comprehensive evaluation we consider eight different values for m. We start with a small number of 10 measurements and increase it up to the dimension of the flattened image ($m = 784$, number of pixel in MNIST). For each m we sample a measurement matrix A and we keep it fixed to train the different approaches. Analogous to the Fourier phase retrieval, we take the same model architectures as described in Section 4.6.2. In Figure 4.14 we plot the MSE of 1024 reconstructions against different values of m. For the PRCGAN variants and the E2E approach, we retrain the model for each value of m. Note that the training of the underlying VAE for DPR is independent of the chosen measurement matrix A, so we can optimize the latent space of the same VAE that we also use in the Fourier phase retrieval experiments. However, for MNIST we observe worse results using the latent dimension of 128, so in this case we keep the latent dimension of 20 from the original work [30]. As expected,

Figure 4.14: Reconstruction from Gaussian measurements: comparison of the PRCGAN, the E2E and generative prior approaches for different number of measurements m.



(a) MNIST



(b) FMNIST

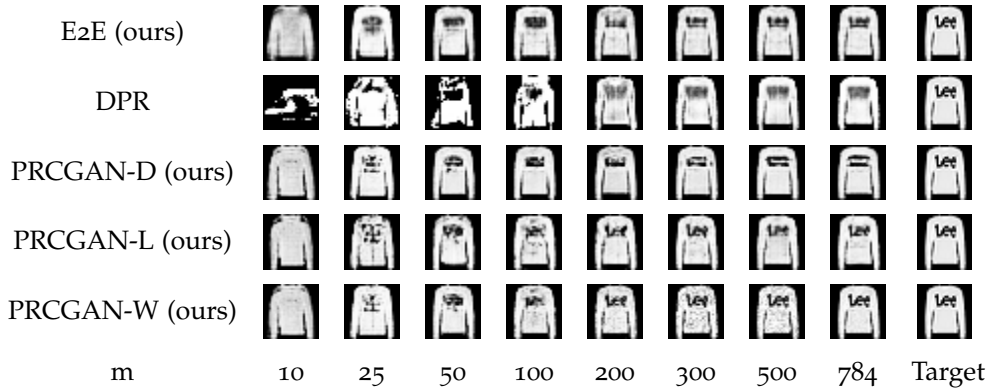


Figure 4.15: Reconstructions from Gaussian measurements of images from FMNIST for a varying number of measurements m.

reducing the number of measurements results in higher errors for all methods as one can see from the plots in Figure 4.14. Note that DPR is has the worst performance and is strongly influenced by the number of measurements on both datasets. Lastly, we show an example for the reconstruction performance of the proposed methods on FMNIST in Figure 4.15. DPR fails to reconstruct the letters at all and E2E only achieves useful results for the maximum number of measurements. In contrast to that, PRCGAN-L and PRCGAN-W successful reconstruct the images even for a small number of measurements.

4.8 Conclusion

In this work, we propose the PRCGAN for solving Gaussian and Fourier phase retrieval. Our method can be seen as an end-to-end learning approach augmented with an additional optimization procedure combining the best of both worlds. On the one hand, the learning component allows our method to reconstruct images for particularly difficult instances of the phase retrieval problem, and on the other hand, the subsequent latent or weight optimization produce high quality reconstructions. Our ablation study shows that each of the components of our model is necessary to achieve this. Furthermore, we show that our method is robust to Poisson noise and additive Gaussian noise and (at least to some extend) generalize better to out-of-distribution data than other methods.

5
Phase Retrieval with Cascaded Neural Networks

Corresponding publication. The contents of this chapter have been published as:

Tobias Uelwer, Tobias Hoffmann, and Stefan Harmeling. Non-iterative phase retrieval with cascaded neural networks. In *Artificial Neural Networks and Machine Learning–ICANN 2021: 30th International Conference on Artificial Neural Networks, Bratislava, Slovakia, September 14–17, 2021, Proceedings, Part II 30*, pages 295–306. Springer, 2021. DOI: 10.1007/978-3-030-86340-1_24.

Personal contributions. Tobias Uelwer did the literature review, created the example figure, proposed the model architecture and designed the experimental evaluation. The implementation of the model and the tuning of the hyperparameters were performed by Tobias Hoffmann. Tobias Uelwer performed the experiments using the implementation of the model provided by Tobias Hoffmann. Furthermore, Tobias Uelwer created the plots and figures and wrote the initial draft of the manuscript. He edited the manuscript together with Stefan Harmeling. The project was supervised by Stefan Harmeling.

Abstract. Fourier phase retrieval is the problem of recovering an image given only the magnitude of its Fourier transformation. Approaches that are based on optimization, like the well-established Gerchberg-Saxton or the hybrid input output algorithm, struggle at reconstructing images from magnitudes that are not oversampled. This motivates the application of learned methods, which allow reconstruction from non-oversampled magnitude measurements after a learning phase. In this chapter, we want to push the limits of these learned methods by means of a deep neural network cascade that reconstructs the image successively on different resolutions from its

non-oversampled Fourier magnitude. We evaluate our method on four different datasets (MNIST, EMNIST, FMNIST, and KMNIST) and demonstrate that it yields improved performance over other non-iterative methods and optimization-based methods.

5.1 Introduction

In this chapter a purely learning-based neural network cascade for the Fourier phase retrieval problem is discussed. This means that the model does not involve any kind of optimization procedure besides the training, and thus does not require any tuning at test time. While the method discussed in this chapter achieves better results than previous non-iterative methods, its performance still lags behind methods involving an optimization procedure at test time.

End-to-end learning

To tackle the non-oversampled phase retrieval problem we formulate phase retrieval as a learning problem. Concretely, end-to-end learning for phase retrieval directly recovers the image from the magnitude only using a mapping that has been learned to solve the problem in a particular problem domain. The mapping is parameterized by a neural network G that is trained to invert the measurement process, i.e.,

$$\tilde{x} \approx G(y). \tag{5.1}$$

Since the measurement process is known, training pairs can be generated on-the-fly from sample images of a given dataset. The weights of G can then be learned using stochastic gradient descent by minimizing a loss function. The benefit of end-to-end methods is the fast computation of the reconstruction because only a single forward-pass through the neural network is used to calculate the reconstruction.

5.2 Contributions

This chapter addresses the challenge of improving the performance of non-iterative phase retrieval methods based on neural networks. We show that a multi-scale approach based on cascading neural networks is able to improve previous non-iterative phase retrieval methods.

5.3 Related Work

Cascades of neural networks have been proposed previously by Schlemper et al. [77] but in the context of compressed sensing which is a related but different problem than phase retrieval. Phase retrieval has applications in many areas of research, e.g., in X-ray crystallography [64], astronomical imaging [21] or microscopy [108]. We distinguish between three classes of methods for phase retrieval:

1. Iterative methods without a learned component: Gerchberg [23] proposed a simple algorithm that is based on alternating reflections. The idea behind this algorithm is to iteratively enforce the constraints in the Fourier space and the image space. Later Fienup modified the Gerchberg-Saxton algorithm in different ways which led to the input-output, the output-output and the hybrid-input-output (HIO) algorithm [22], where the HIO algorithm is most commonly used for phase retrieval. Luke [59] analyzed the relaxed averaged alternating reflection (RAAR) algorithm. In general, these iterative methods without a learning component work well when the signal is oversampled.

2. Iterative methods with a learned component: For non-oversampled phase retrieval Işıl et al. [38] extend the HIO algorithm by a neural network that removes artifacts. Metzler et al. [63] and Wu et al. [101] use the regularization-by-denoising framework [75] to solve oversampled phase retrieval problems. Another class of learned methods rely on the optimization of a latent variable of a learned generative model [30, 88] and produce high quality results. However, these methods require a training phase and an optimization phase during application and are therefore very costly.

3. Non-iterative methods with a learned component: Non-iterative phase retrieval with a deep convolutional neural network that is trained end-to-end is proposed by Nishizaki et al. [69]. Recently, Manekar et al. [60] use symmetry breaking to solve the oversampled phase retrieval problem with neural networks. The benefit of non-iterative learned methods is the highly efficient reconstruction of images using only a single forward-pass through the model while also producing good results in the non-oversampled case.

5.4 Proposed Method

In this chapter, we propose to use a cascaded neural network architecture for Fourier phase retrieval. In the following, we refer to it as cascaded phase retrieval (CPR) network. The CPR network consists of multiple sub-networks $G^{(1)}, \dots, G^{(q)}$ which are updated successively to reconstruct the different down-sampled instances of the original image, where $G^{(1)}, \dots, G^{(q)}$ are fed with the intermediate reconstruction produced by the previous network. In that way, each of these sub-networks can iteratively refine the reconstruction. In addition to that, each of the sub-networks is provided with the measurement y as an input. The first few sub-networks are trained to reconstruct a down-sampled version of the image, where we denote the resolutions by $n_p \times n_p$ for $p = 1, \dots, q$. The last sub-networks predict the image

at full-resolution $n_q \times n_q$. The nearest-neighbor interpolation scheme is used for down-sampling the training images. Figure 5.1 shows an overview of the CPR network architecture.

5.4.1 Loss Functions

A common choice for reconstruction tasks is the mean squared error (MSE) which is defined for an image x and a corresponding reconstruction $\hat{x}$ as

$$\mathcal{L}^{(p)}_{\text{MSE}}(x, \tilde{x}) = \frac{1}{n_p^2} \sum_{k=1}^{n_p} \sum_{l=1}^{n_p} \left(x[k,l] - \tilde{x}[k,l]\right)^2 . \tag{5.2}$$

Although, it seems to work well in practice and provides good gradients for training, the reconstructions tend to be blurry. This phenomenon has been discussed by Pathak et al. [71]. Hence, we also implement the mean absolute error (MAE), i.e.,

$$\mathcal{L}^{(p)}_{\text{MAE}}(x, \tilde{x}) = \frac{1}{n_p^2} \sum_{k=1}^{n_p} \sum_{l=1}^{n_p} |x[k,l] - \tilde{x}[k,l]| \tag{5.3}$$

for measuring the reconstruction error.

5.4.2 Training

During training, each sub-network $G^{(p)}$ is trained using an individual loss $\mathcal{L}^{(p)}$. Each sub-network is updated one after another, where the loss $\mathcal{L}^{(p)}$ influences only $G^{(p)}$ and does not impact the parameters of the previous sub-networks.

Algorithm 1: Training algorithm for CPR network

Input: Dataset X, downsampling functions $g_1, \ldots, g_q$, networks $G^{(1)}, \ldots, G^{(q)}$, loss functions $\mathcal{L}^{(1)}, \ldots, \mathcal{L}^{(q)}$

for $e = 1, \ldots, N$ **do**
  for batch $(x_1, \ldots, x_b)$ in X **do**
    Calculate magnitudes $y = (y_1, \ldots, y_b)$ with $y_k = |\mathcal{F}(x_k)|$, for $k = 1, \ldots, b$
    for $p = 1, \ldots, q$ **do**
      Calculate $\tilde{X}^{(p)} = (\tilde{x}_1, \tilde{x}_2, \ldots, \tilde{x}_b)$, where $\tilde{x}_k = g_p(x_k)$ for $k = 1, \ldots, b$
      if $p == 1$ **then**
        $\hat{X}^{(p)} = G_p(y)$
      else
        $\hat{X}^{(p)} = G_p(y, \hat{X}^{(p-1)})$
      Update network parameters using $\nabla \mathcal{L}^{(p)} \left(\hat{X}^{(p)}, \tilde{X}^{(p)}\right)$
    end
  end
end

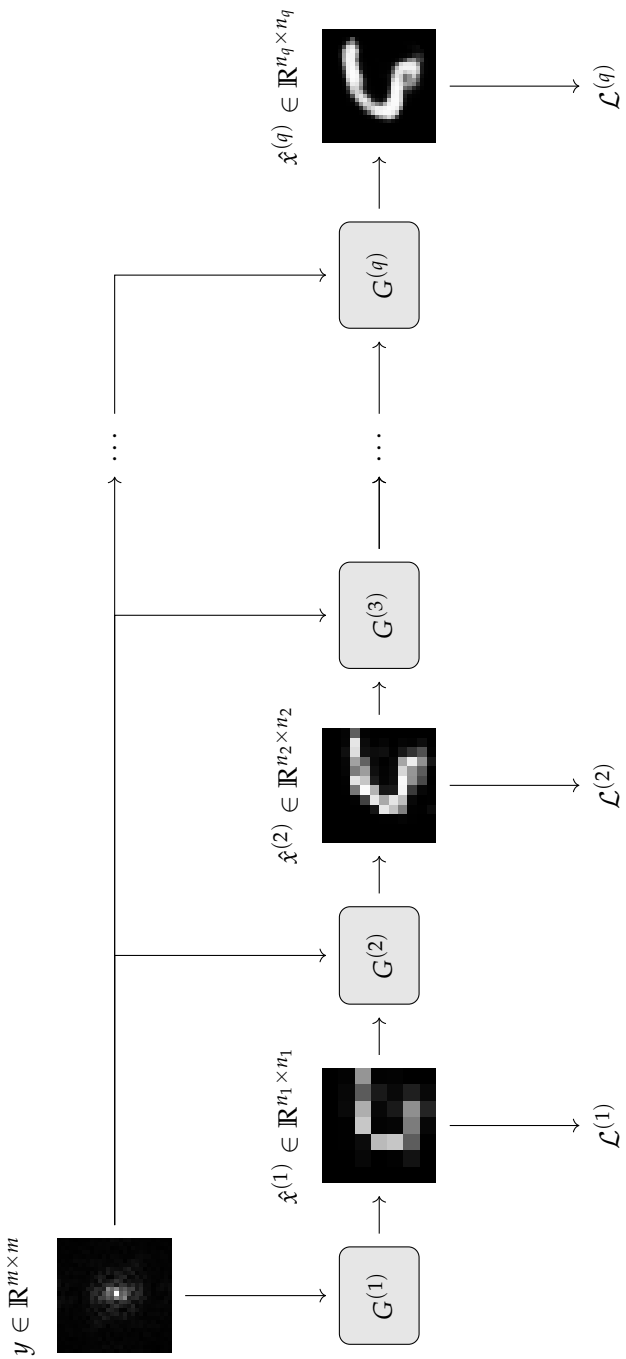


Figure 5.1: An overview of the network architecture of the CPR approach. The magnitude image is fed to each of the networks. The sub-networks are updated stage-wise, i.e., we use $\mathcal{L}_1$ to update $G^{(1)}$, then the output of $G^{(1)}$ is passed as additional input to $G^{(2)}$ and so on. The first few networks focus on reconstructing a sub-sampled instance of the image, whereas the last sub-network predict the image at full-resolution.

Alternatively, the CPR network could be trained in an end-to-end fashion, however, since the intermediate reconstructions have different resolutions, we would need to carefully choose weights to balance the influence of each loss function $\mathcal{L}^{(1)}, \ldots, \mathcal{L}^{(q)}$. The training procedure is shown in more detail in Algorithm 1.

5.5 Experimental Evaluation

In this section, we empirically evaluate the performance of our model. In order to do this, we report the results of the fully-convolutional residual network (ResNet) employed by Nishizaki et al. [69], the multi-layer-perceptron (MLP) used in our previous work [88] and the PRC-GAN [88]. In addition to these learned networks we include the results of the well-established HIO algorithm [22] and the RAAR algorithm [59] as a baseline.

5.5.1 Datasets

For the experimental evaluation we use the MNIST [55], the EMNIST [13], the FMNIST [102] and the KMNIST [12] datasets. All datasets consist of 28×28 grayscale images. MNIST contains images of digits, EMNIST contains images of letters and digits, FMNIST contains images of clothing and KMNIST contains images of cursive Japanese characters. Although these datasets are considered to be toy datasets when it comes to classification tasks, they provide quite challenging data for two-dimensional Fourier phase retrieval. For the EMNIST dataset we use the balanced version of the dataset.

5.5.2 Experimental Setup

We compare our CPR approach with the MLP and the ResNet that are trained to minimize $\mathcal{L}_{\text{MSE}}$ for the MNIST, the EMNIST and the KMNIST dataset. The $\mathcal{L}_{\text{MAE}}$ is used for the FMNIST dataset. Furthermore, we report the results of an MLP trained with an adversarial loss in combination with $\mathcal{L}_{\text{MAE}}$ (PRCGAN) as proposed in our previous work [88]. For our proposed CPR network we consider a cascade of five MLPs with three hidden layers where we increased the scales of the (intermediate) reconstructions according to Table 5.1. The number of hidden units for each sub-network is also shown in Table 5.1. Furthermore, we compare the results with a CPR network that produces intermediate reconstructions at full scale. We refer to this variant as CPR-FS. All sub-networks are trained using dropout [86], batch-normalization [37] and ReLU activation functions. For the last layer we use a Sigmoid function to ensure that the predicted pixels are in $[0, 1]$. To optimize the weights we used Adam [50] with learning rate

Architecture

10^{-4}. We train all versions of the CPR network for 100 epochs with the $\mathcal{L}_{\text{MSE}}$, except for the FMNIST dataset where we use $\mathcal{L}_{\text{MAE}}$ for the final layer. These choices gave the best results on the validation dataset.

We ran the HIO algorithm and the RAAR algorithm for 1000 steps each and allowed three random restarts, where we selected the reconstruction $\hat{x}$ with the lowest magnitude error. For HIO we set $\beta = 0.8$ and for RAAR we set $\beta = 0.87$.

		$G^{(1)}$	$G^{(2)}$	$G^{(3)}$	$G^{(4)}$	$G^{(5)}$
Scale	CPR	7×7	12×12	17×17	22×22	28×28
	CPR-FS	28×28	28×28	28×28	28×28	28×28
Layer size	CPR	1136	1336	1536	1736	1936
	CPR-FS	1936	1936	1936	1936	1936

Table 5.1: Scales used for the (intermediate) reconstructions and number of hidden units used for each network of the cascade.

5.5.3 *Metrics*

For a quantitative evaluation we compare the MSE and the MAE as defined in Equation 5.2 and Equation 5.3.

Shifts and rotations

Because translating signals by a constant shift or rotating them by 180° does not change their Fourier magnitude, we considered these reconstructions equally correct. Thus, we register the predictions (and their rotated variants) using cross-correlation as described by Brown [6] before calculating the evaluation metrics.

5.5.4 *Results*

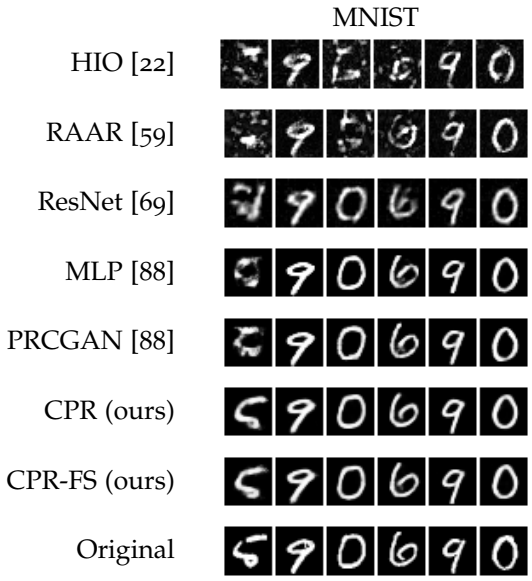
Figure 5.2 compares six reconstructions by the different methods on the MNIST and the FMNIST test dataset. We observe that the HIO algorithm and the RAAR algorithm fail to recover the image in most of the cases. From all learned methods, the Resnet produced the worst reconstructions. The estimated images are very blurry and in some cases the reconstruction exhibit deformations (e.g., the last two images from the FMNIST dataset that are shown in Figure 5.2). The PRCGAN produces reconstructions that are sharp and overall the visual quality is similar to the reconstructions of the MLP. Most of the learned methods struggle to recover the first image of the MNIST dataset (depicting the "5") . We suppose that this sample is very different from the samples that were used to train the networks. Only, the CPR and the CPR-FS network are capable of recovering this image.

Table 5.2 shows the MSE and the MAE of the reconstructions. Over-

Table 5.2: Quantitative comparison of the reconstructions produced by the different methods. We report MSE and MAE between the reconstructions and the original images of the test dataset. The best result is printed **bold** and the second-best are underlined.

	MNIST		EMNIST	
	MSE (↓)	MAE (↓)	MSE (↓)	MAE (↓)
HIO [22]	0.0441	0.1016	0.0653	0.1379
RAAR [59]	0.0489	0.1150	0.0686	0.1456
ResNet [69]	0.0269	0.0794	0.0418	0.1170
MLP [88]	0.0183	0.0411	0.0229	0.0657
PRCGAN [88]	0.0168	0.0399	0.0239	0.0601
CPR (ours)	**0.0123**	**0.0370**	0.0153	0.0525
CPR-FS (ours)	0.0126	0.0373	**0.0144**	**0.0501**
	FMNIST		KMNIST	
	MSE (↓)	MAE (↓)	MSE (↓)	MAE (↓)
HIO [22]	0.0646	0.1604	0.0835	0.1533
RAAR [59]	0.0669	0.1673	0.0856	0.1559
ResNet [69]	0.0233	0.0820	0.0715	0.1711
MLP [88]	0.0128	0.0526	0.0496	0.1168
PRCGAN [88]	0.0151	0.0572	0.0651	0.1166
CPR (ours)	0.0115	0.0503	0.0447	0.1068
CPR-FS (ours)	**0.0113**	**0.0497**	**0.0433**	**0.1034**

Figure 5.2: Reconstructions from the Fourier magnitudes of samples from the MNIST test dataset.



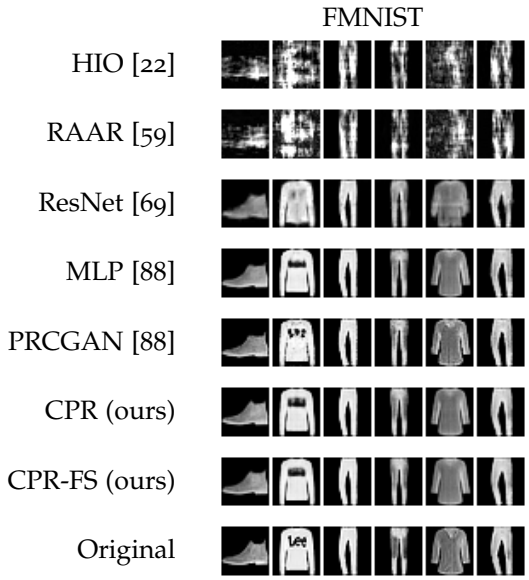


Figure 5.3: Reconstructions from the Fourier magnitudes of samples from the FMNIST test dataset.

all, the learned methods outperform RAAR and HIO by a large margin. For MNIST, EMNIST and KMNIST we see that the CPR network greatly improves the reconstruction quality compared to the other learned methods. We hypothesize that our proposed CPR network yields better results when the signals of interest have a small support (e.g., MNIST, EMNIST, KMNIST). However, for signals with a large support (e.g., Fashion MNIST) we only observe a small improvement compared to the other learned methods.

5.5.5 Intermediate Prediction at Full-Scale

We briefly study the effect of predicting down-sampled versions of the image. Therefore, we evaluate the CPR-FS network which produces full-scale intermediate reconstructions. Table 5.2 also shows that the CPR-FS network performs similarly in terms of the overall reconstruction quality. For the EMNIST, the FMNIST and the KMNIST dataset the full-scale variant is slightly better. However, due to the larger input, the sub-networks need to have more parameters and thus training is more expensive.

5.5.6 Ablation Study

In this section, we demonstrate that increasing the number of sub-networks has a beneficial effect on the overall reconstruction quality. To do so, we train five network cascades exemplarily on the EMNIST dataset where we increase the number of sub-networks from one to five. We report the MSE on the test dataset after 50 epochs. Fig-

ure 5.4 shows that the MSE for the EMNIST dataset decreases with an increasing number of sub-networks used for the CPR-FS approach. Furthermore the gain in terms of MSE saturates after $q = 5$, such that additional sub-networks do not bring any further improvements. We expect the same relative behavior on the other datasets when increasing q.

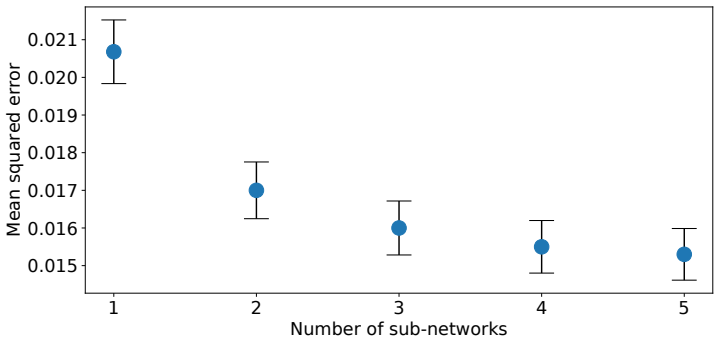


Figure 5.4: Test MSE on the EMNIST test dataset for different number of sub-networks. Error bars indicate the 95% confidence interval.

5.6 *Conclusion and Future Work*

In this chapter, we use a cascade of neural networks for non-oversampled Fourier phase retrieval. Our approach successively reconstructs images from their Fourier magnitudes and outperforms other existing non-iterative networks noticeably in terms of the reconstruction quality. However, non-iterative methods do not yet reach the reconstruction quality of iterative methods with a learning component which require high computational cost at test time.

Future work could also evaluate different strategies for training the neural network cascade. For example, greedy sub-network-wise training could be implemented and compared with our training procedure. Moreover, the CPR network architecture can easily be adapted to solve inverse problems other than Fourier phase retrieval.

6
Optimizing Intermediate Representations of Generative Models for Phase Retrieval

Corresponding publication. The contents of this chapter have been published as:

Tobias Uelwer, Sebastian Konietzny, and Stefan Harmeling. Optimizing intermediate representations of generative models for phase retrieval. *Transactions on Machine Learning Research*, 2022. ISSN 2835-8856. The first two authors contributed equally.

This work has also been presented at the *NeurIPS 2022 AI4Science Workshop*.

Personal contributions. Tobias Uelwer conceived the idea of applying intermediate layer optimization of generators to the phase retrieval problem. He did the literature review, chose the used generative models together with Sebastian Konietzny and designed the experimental evaluation and ablation experiments. Sebastian Konietzny proposed the initialization schemes, implemented the model and performed hyperparameter tuning and experiments. Tobias Uelwer wrote the initial draft of the manuscript which was later edited together with Stefan Harmeling. The project was supervised by Stefan Harmeling.

Remarks. Due to copyright issues the reconstructions of images from the CelebA dataset are not shown in this thesis. Please, refer to the published paper. Furthermore, we report MSE values instead of PSNR values (which were used in the published version) to improve comparability with the other methods in this thesis.

Abstract. Phase retrieval is the problem of reconstructing images from magnitude-only measurements. In many real-world applications the

problem is underdetermined. When training data is available, generative models allow optimization in a lower-dimensional latent space, hereby constraining the solution set to those images that can be synthesized by the generative model. However, not all possible solutions are within the range of the generator. Instead, they are represented with some error. To reduce this representation error in the context of phase retrieval, we first leverage a novel variation of intermediate layer optimization (ILO) to extend the range of the generator while still producing images consistent with the training data. Second, we introduce new initialization schemes that further improve the quality of the reconstruction. With extensive experiments on the Fourier phase retrieval problem and thorough ablation studies, we can show the benefits of our modified ILO and the new initialization schemes. Additionally, we analyze the performance of our approach on the Gaussian phase retrieval problem.

6.1 Introduction

In this chapter, we revisit the DPR method [30] that is based on optimizing the latent space of a trained generative model. It is known, that optimizing intermediate layers in generative models can lead to excellent results in various linear inverse problems like inpainting, super-resolution, denoising and compressed sensing, as shown by Daras et al. [16]. We demonstrate that this approach, termed intermediate layer optimization (ILO), can be extended to solve the more difficult non-linear inverse problem of phase retrieval. In particular, we introduce an additional additional initialization schemes and a refinement step that further improves reconstruction performance. Additionally, we propose different learned and unlearned initialization schemes. The learned initialization scheme trains an encoder to predict a point in the latent space that corresponds to the reconstruction. In contrast to the conditional GAN approach presented in Chapter 4 this encoder can be trained for a fixed generator network.

6.2 Related Work

In the following, we give a short overview of the various approaches distinguishing between classical optimization-based and learning-based methods.

6.2.1 Methods without Learning for Phase Retrieval

One of the first (Fourier) phase retrieval algorithms is Fienup's error-reduction (ER) algorithm [22] that is based on alternating projections.

In his work, Fienup [22] also introduced the hybrid-input-output (HIO) algorithm which improved the ER algorithm. The Gaussian phase retrieval problem was approached by Candes et al. [8] who used methods based on Wirtinger derivatives to minimize a least-squares loss. Wang et al. [97] considered a similar idea but used a different loss function and so-called truncated generalized gradient iterations. Holographic phase retrieval is a related problem, which aims to reconstruct images from magnitude measurements where a known reference is assumed to be added onto the image. This problem was approached by Lawrence et al. [54] using untrained neural network priors. Untrained neural networks have also been used by Chen et al. [10] for solving different phase retrieval problems. Phase retrieval using the RED framework [75] was done by Metzler et al. [63], Wang et al. [98] and Chen et al. [11]. However, these works focus on the reconstruction of images from coded diffraction pattern measurements or $4\times$ oversampled Fourier magnitudes.

6.2.2 *Learning-based Methods for Phase Retrieval*

Recently, learning-based methods for solving phase retrieval problems gained a lot of momentum. They can be classified into two groups: supervised methods and unsupervised methods.

Supervised methods. These methods directly learn a mapping that reconstructs images from the measurements. Learning neural networks for solving Fourier phase retrieval was done by Nishizaki et al. [69]. This approach was later extended to a neural network cascade by Uelwer et al. [87], who also used conditional generative adversarial networks to solve various phase retrieval problems [88]. While giving impressive results, all of these supervised approaches have a drawback for the Gaussian phase retrieval setup, since the measurement operator must be known at training time. Supervised learning of reference images for holographic phase retrieval was done by Hyder et al. [36] and Uelwer et al. [89] gave further insights.

Unsupervised methods. Unsupervised methods are agnostic with respect to the measurement operator, because they are trained on a dataset of images without the corresponding measurements. Generative models for Gaussian phase retrieval have been analyzed by Hand et al. [30] and Liu et al. [57]. Killedar and Seelamantula [45] apply a sparse prior on the latent space of the generative model to solve Gaussian phase retrieval. Alternating updates for Gaussian phase retrieval with generative models was proposed by Hyder et al. [35]. Manekar et al. [61] used a passive loss formulation to tackle the non-

oversampled Fourier phase retrieval problem.

In the context of these related works, the method that we propose in this chapter can be seen as an unsupervised learning-based approach that can optionally be extended with a supervised component, as we detail in Section 6.3.2.

6.2.3 *Optimizing Representations of Generators for Inverse Problems*

Generative models have been used to solve linear inverse problems, e.g., compressed sensing [5], image inpainting [106]. To decrease the representation error and thereby boost the expressiveness of these generative models, the idea of optimizing intermediate representations was successfully applied in the context of compressed sensing [82], image inpainting and super-resolution [16]. The former paper called this approach generator surgery, the latter intermediate layer optimization (ILO).

6.2.4 *Contributions*

The contributions made in this chapter are the following:

1. We show that the idea of optimizing intermediate representations in generative models can be applied to solve phase retrieval problems. However, given the difficulty of these non-linear inverse problems an additional subsequent optimization step is required to obtain good results.

2. We also show that the need for multiple runs with random restarts for generator-based approaches to solve inverse problems can be reduced by using different initialization schemes.

Further intuition and motivation will be given in Section 6.3.1. In extensive experiments, we consider the underdetermined Fourier and Gaussian phase retrieval (with real and complex measurement matrices) and show that our method provides state-of-the-art results. By analyzing various combinations of methods and generator networks, we show the influence of the choice of the generator. In further ablation studies, we show the importance of each component of our method.

6.3 *Phase Retrieval with Generative Models*

The basic idea of applying a trained generative model G to the phase retrieval problem is to plug $G(z)$ into the least-squares data fitting term and to optimize over the latent variable z, i.e.,

$$\min_z \big\| |\mathcal{A}(G(z))| - y \big\|_2^2. \tag{6.1}$$

This method has been used for Gaussian phase retrieval (with real-valued A) by Hand et al. [30]. While this approach yields good results, the reconstruction quality is limited by the range of the generator network G. This issue is, for example, also discussed in the work of Asim et al. [2]. In the following, we explain how the range of G can be extended by optimizing intermediate representations of G instead of only solving Equation 6.1 with respect to the latent variable z.

Notation. For the exposition of the method we use the following notation: The generator network $G = G_k \circ \cdots \circ G_1$ can be written as the concatenation of k layers $G_1, \ldots, G_k$. The subnetwork consisting of layers i through j is denoted by $G_i^j = G_j \circ \cdots \circ G_i$. Note, that $G = G_1^k = G_{i+1}^k \circ G_1^i$ for $1 \leq i \leq k$. The output of layer G_i is written as z_i, i.e., $z_i = G_i(z_{i-1})$. The input of G is z_0.

Furthermore, we define the ℓ_1-ball with radius $r > 0$ around z as, ℓ_1-ball

$$B_r(z) = \{x \mid \|x - z\|_1 \leq r\}. \tag{6.2}$$

6.3.1 Phase Retrieval with Intermediate Layer Optimization (PRILO)

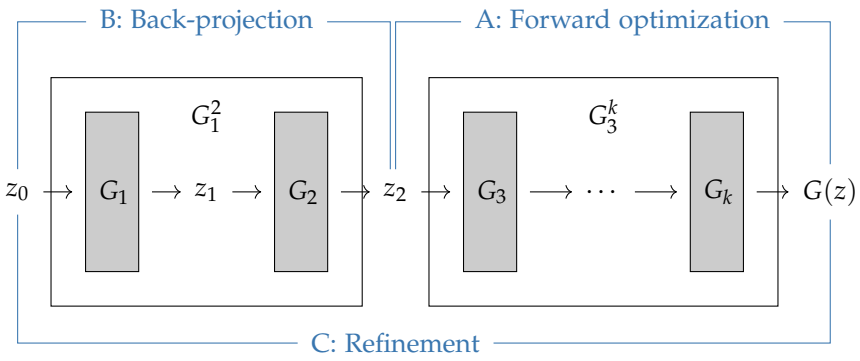


Figure 6.1: Structure of the generator network and, in blue, the parts relevant for steps A, B and C.

The key idea of ILO is to vary the intermediate representations learned by a sub-network of the generator G while ensuring an overall consistency of the solution. The latter is particularly challenging for non-linear problems like phase retrieval. As we will show in the experiments, additional steps and special initialization schemes are essential to obtain useful results. All of these steps will be explained next.

The generator network captures the prior knowledge and restricts the solutions of the (underdetermined) phase retrieval problem. In this work, we use two important ideas:

Range extension: Hidden representations z_i for $i > 0$ are allowed to vary outside the range of G_1^i. This will reduce the representation error of the generator.

Image consistency: Solutions must be realistic, non-degenerate and similar to the the training dataset. This is achieved by back-projection ensuring that z_i is not too far away from the image of some z_0.

Note that one has to strive for a trade-off between range extension and image consistency, where range extension facilitates the optimization, while image consistency regularizes the solution.

Subdividing the generator network

Concretely, we repeatedly split the generator G into sub-networks G_1^i and G_{i+1}^k and iteratively optimize the intermediate representations z_i resulting from G_1^i (instead of simply optimizing the initial latent variable z_0). In this way, we go outside the range of the sub-network G_1^i to reduce the mean-squared magnitude error. To make the resulting image more consistent we back-project z_i closer to the range of G_1^i. We will show that z_0 from the back-projection is not only a good candidate for the overall optimization, but also serves as a good initialization for further optimization leading to a significant refinement of the overall solution.

To get started, we initially optimize the input variable z_0 (minimize Equation 6.1). This provides starting points for all intermediate representations z_i for $i > 0$. The overall procedure then iteratively applies the following three steps to different intermediate layers:

A. Forward optimization: vary the intermediate representation z_i to minimize the magnitude error while staying in the ball with radius r_i around the current value of z_i, i.e.,

$$z_i^* = \underset{z \in B_{r_i}(z_i)}{\operatorname{argmin}} \left\| |\mathcal{A}(G_{i+1}^k(z))| - y \right\|_2^2. \tag{6.3}$$

Note, that this step possibly leaves the range of G_1^i (range extension). By introducing ball-constraints on the optimized variable we avoid overfitting the measurements.

B. Back-projection: find an intermediate representation $G_1^i(z_0)$ that is close to the optimal z_i^* from step A, i.e.,

$$\bar{z}_0 = \underset{z \in B_{s_i}(\mathbf{o})}{\operatorname{argmin}} \left\| G_1^i(z) - z_i^* \right\|_2^2, \tag{6.4}$$

where $\mathbf{o} \in \mathbb{R}^l$ denotes the vector of all zeros. By doing so we ensure that there exists a latent z_0 that yields the reconstruction. Thereby, we regularize the solution from step A to obtain image consistency.

While these two steps have been shown to improve the overall performance of linear inverse problems, we found that for phase retrieval the following refinement step significantly improves the reconstructions.

C. Refinement: Our intuition here is that the back-projected latent $\bar{z}_0$ serves as a good initialization for further optimization. Starting from the $\bar{z}_0$ found in step B, we again optimize the measurement error

$$z_0^* = \underset{z \in B_{r_0}(\bar{z}_0)}{\operatorname{argmin}} \left\| |\mathcal{A}(G(z))| - y \right\|_2^2. \tag{6.5}$$

Note, that as we only optimize the latent variable of G, we do not have to apply the back-projection again for this refinement step.

These steps are applied repeatedly to the different intermediate representations. Multiple strategies for selecting the layers to optimize are possible. In practice it turned out to be sufficient to treat the layers once from left to right. Thus, we recommend to start with the layers closer to the input and then progress further with the layers closer to the output. Starting with the later layers can cause problems as they offer too much flexibility and no regularizing effect. The final reconstruction is obtained as $x^* = G(z_0^*)$. Steps A, B and C are visualized in Figure 6.1.

The three optimization problems stated in Equations 6.3-6.5 are solved using projected gradient descent. This means that after each gradient descent step the iterate is projected back onto the appropriate ℓ_1-ball. In our implementation this projection is implemented using the method described by Duchi et al. [20].

6.3.2 *Initialization Schemes*

Due to the difficulty of the phase retrieval problem, many methods are sensitive to the initialization. Candes et al. [8] proposed a spectral initialization technique that is often used for compressive Gaussian phase retrieval. Since it requires one to solve a large eigenvalue problem, this method can be quite costly. A common approach to improve reconstruction performance is to use random restarts and to select the solution with the lowest magnitude error after the optimization [30]. However, this requires running the method multiple times. Instead, we propose two fast initialization schemes that use the generative model and the available magnitude information before the optimization.

Random restarts

Magnitude-Informed Initialization (MII). Deep generative models can generate a lot of images at relatively low cost: instead of optimizing Equation 6.1 with random restarts, we sample a set of starting points $Z = \{z_0^{(0)}, \ldots, z_0^{(p)}\}$ and select the one with the lowest magnitude mean-squared error (MSE),

$$z_0^{\text{init}} = \underset{z \in Z}{\operatorname{argmin}} \left\| |\mathcal{A}(G(z))| - y \right\|_2^2, \tag{6.6}$$

before the optimization. The empirical reason for this choice is that the MSE between the unknown target image and the initial reconstructions $G(z)$ for $z \in Z$ strongly correlates with the MSE of their magnitudes (correlation $\rho = 0.91$). This initialization scheme is applicable to other reconstruction algorithms that search in the latent space of a generative model as well.

Learned Initialization (LI). The generator is trained on a dataset of images that are characteristic for the problem. We use the generator to train an encoder network E_θ that maps magnitude measurements y to latent representations z_0. Once the encoder is trained, we can use it to predict an initialization for the optimization discussed in Section 6.3.1.

A naive way to train the encoder network is to create input/output pairs by starting with random latent vectors z_0 and combining them with the magnitudes $|\mathcal{A}(G(z_0))|$ of the corresponding image. The disadvantage is that the original training images are only implicitly used in this approach (because those were used to train the generator). To leverage the generator and also the training images, we estimate the weights θ of the encoder E_θ, such that encoded magnitudes $E_\theta(y)$ generate an image $G(E_\theta(y))$ that is close to the original image x. This idea originates from GAN inversion [109].

Loss function

More precisely, we minimize a combination of three loss functions to train the encoder

$$\begin{aligned}\mathcal{L}_{\text{MSE}}(G(E_\theta(y)), x) + \lambda_{\text{perc}}\mathcal{L}_{\text{perc}}(G(E_\theta(y), x)) \\ +\lambda_{\text{adv}}\mathcal{L}_{\text{adv}}(D_\phi(G(E_\theta(y)), D_\phi(x))),\end{aligned} \tag{6.7}$$

where $\mathcal{L}_{\text{MSE}}$ is the image MSE, $\mathcal{L}_{\text{perc}}$ the LPIPS loss [107], and $\mathcal{L}_{\text{adv}}$ is a Wasserstein adversarial loss [1] with gradient penalty [28] (using the discriminator network D_ϕ). In our experiments, we set $\lambda_{\text{perc}} = 5 \cdot 10^{-5}$ and $\lambda_{\text{adv}} = 0.1$. Note, that the generator network is fixed and we only optimize the encoder weights θ and the discriminator weights ϕ to solve a learning objective.

Random perturbation

Additionally, we also found it helpful to apply a small normally-distributed perturbation with mean 0 and standard deviation $\sigma = 0.05$ to the predicted latent representation. We do so because using a fixed initialization leads to a deterministic optimization trajectory when running the optimization multiple times.

Gradient noise

For better exploration of the optimization landscape we also apply gradient noise during optimization. In combination with the projection onto the feasible set the update reads as

$$z^{(k+1)} = P\left(z^{(k)} - \alpha\left(\nabla_z f\left(z^{(k)}\right) + u^{(k)}\right)\right), \tag{6.8}$$

where f is the objective function corresponding to the current step, $u^{(k)} \sim \mathcal{N}(0, \sigma_k^2 I)$ with $\sigma_k^2 = \frac{\eta}{(1+k)^\gamma}$ and P is the projection onto $B_r(x^{(0)})$.

In our experiments we set $\eta = 0.02$ and $\gamma = 0.55$. This noise decay schedule was proposed by Welling and Teh [100] and is also discussed by Neelakantan et al. [67] in the context of neural network training.

One drawback of the learned initialization is that it requires one to retrain the encoder network when the measurement matrix changes. In the following, we only evaluate this approach for the Fourier phase retrieval problem.

6.4 Experimental Evaluation

Datasets

We evaluate our method on the following datasets: MNIST [55], EMNIST [13], FMNIST [102], and CelebA [58]. The first three datasets consist of 28×28 grayscale images, whereas the latter dataset is a collection of $200,000$ color images that we cropped and rescaled to a resolution of 64×64.

Architectures

Similar to Hand et al. [30], we use a fully-connected variational autoencoder (VAE) for the MNIST-like datasets and a DCGAN [73] for the CelebA dataset. Both architectures were also used by Bora et al. [5] for compressed sensing. Details about the VAE training are described in Appendix B.1. Going beyond existing works, we are interested in improving the performance on the CelebA dataset even further, thus we considered deeper, more expressive generators, like the Progressive GAN [42] and the StyleGAN [43]. Since the optimization of the initial latent space for deeper generators is more difficult, we expect significantly better results by adaptively optimizing the successive layers of these models. Our implementation is based on open-source projects[1] [2] [3]. In total our computations took two weeks on two NVIDIA A100 GPU. Detailed hyperparameter settings can be found in Appendix B.2.

[1] https://github.com/rosinality/progressive-gan-pytorch

[2] https://github.com/rosinality/style-based-gan-pytorch

[3] https://github.com/giannisdaras/ilo

6.4.1 Phase Retrieval with Fourier Measurements

For the problem of Fourier phase retrieval, we compare our method first with the ER algorithm [22] and the HIO algorithm [22] (both having no learning component) and then with the following supervised learning methods: an end-to-end (E2E) learned multi-layer-perceptron [88], a residual network [69], a cascaded multilayer-perceptron [87], and a conditional GAN approach [88]. Additionally, we apply DPR which was originally only tested on Gaussian phase retrieval [30].

Table 6.1 and Table 6.2 show the mean squared error (MSE) and the mean structural similarity index measure (SSIM) [99]. Each reported number was calculated on the reconstructions of 1024 test samples. We allow four random restarts and select the generated sample resulting in the lowest measurement error. As one can see, more expressive

models result in better reconstructions. Furthermore, we note that our proposed initialization schemes, i.e., the LI and MII, lead to improved performance. Reconstructions using the DCGAN and the Progressive GAN can be found in the appendix of the published paper.

Surprisingly, our unsupervised PRILO-MII approach often quantitatively outperforms the supervised competitors (MNIST and EMNIST). Only for FMNIST we get slightly worse results which we attribute to the performance of the underlying generator of the VAE. Notably, we outperform DPR that uses the same generator network, which shows that the modifications explained in this chapter are beneficial. Summarizing, among classical and unsupervised methods our new approach performs best, often even better than the supervised methods.

The CelebA dataset is more challenging and the influence of the choice of the generator networks is substantial. We consider the DCGAN [73], the Progressive GAN [42] and the StyleGAN [43] as base models for our approach (second column in Table 6.2). Furthermore, the new initialization schemes have a strong impact. For fairness, we also combined the existing DPR approach [30] with the various generator models and the initialization schemes. Thus we are able to study the influence of the different components. Among the generators, StyleGAN performs best. MII improves the results, while LI achieves even better reconstructions. For all previously mentioned combinations, our new method PRILO is better than DPR.

To further support our claims, we performed additional experiments on the high-resolution FFHQ dataset [43] using the StyleGAN architecture. Although, the reconstruction capabilities of all considered methods are still limited, our proposed changes improve the reconstructions. Please, refer to the published paper to view the reconstruction.

6.4.2 *Phase Retrieval with Gaussian Measurements*

Beyond Fourier phase retrieval, compressed Gaussian phase retrieval is a similarly challenging problem, where the measurement matrix A has real- or complex-valued normally-distributed entries. For the real-valued case, we sample from a zero-mean Gaussian with variance $1/m$,

Sampling a real-valued measurement matrix

$$A = (a_{kl})_{\substack{k=1,\dots,m \\ l=1,\dots,n}} \sim \mathcal{N}\left(0, \frac{1}{m}\right), \tag{6.9}$$

where m is the number of measurements. The entries of the complex-valued measurement matrix are obtained by sampling real and imaginary parts from a zero-mean Gaussian with variance $1/(2m)$,

Sampling a complex-valued measurement matrix

$$A = (a_{kl} + ib_{kl})_{\substack{k=1,\dots,m \\ l=1,\dots,n}} \text{ with } a_{kl}, b_{kl} \sim \mathcal{N}\left(0, \frac{1}{2m}\right). \tag{6.10}$$

Table 6.1: Fourier phase retrieval: Our proposed method (being unsupervised) performs best among classical, unsupervised and supervised approaches for MNIST and EMNIST, and second best for FMNIST (in terms of MSE). Best values are printed **bold** and second-best are underlined.

		MNIST		Fashion-MNIST		EMNIST	
	Learning	MSE (↓)	SSIM (↑)	MSE (↓)	SSIM (↑)	MSE (↓)	SSIM (↑)
ER [22]	–	0.0487	0.5504	0.0684	0.3971	0.0676	0.5056
HIO [22]	–	0.0440	0.5274	0.0649	0.4019	0.0645	0.4942
DPR [30]	unsup.	0.0094	0.9005	0.0191	0.6846	0.0171	0.8568
PRILO (ours)	unsup.	0.0014	0.9843	0.0173	0.7133	0.0082	0.9367
PRILO-MII (ours)	unsup.	**0.0004**	**0.9949**	0.0135	0.7560	**0.0033**	**0.9719**
ResNet [69]	sup.	0.0239	0.7292	0.0221	0.6070	0.0413	0.5616
E2E [88]	sup.	0.0164	0.8191	0.0129	0.7400	0.0223	0.7598
CPR [87]	sup.	0.0127	0.8529	0.0113	0.7768	0.0147	0.8537
PRCGAN-L [88]	sup.	0.0009	0.9890	**0.0084**	**0.8376**	0.0066	0.9416

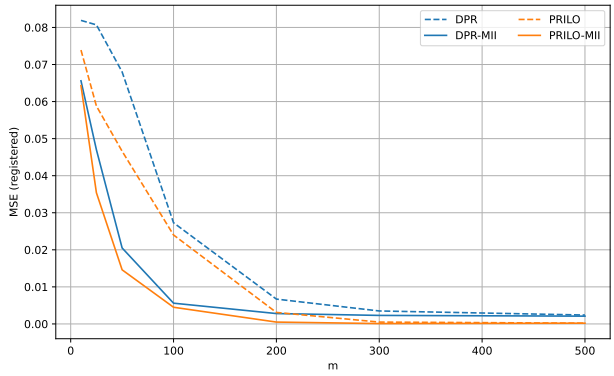
	Base model	Learning	CelebA MSE (↓)	CelebA SSIM (↑)
ER [22]	–	–	0.1015	0.0637
HIO [22]	–	–	0.1005	0.0510
DPR [30]	DCGAN	unsup.	0.0321	0.4457
DPR-MII	DCGAN	unsup.	0.0260	0.4898
PRILO	DCGAN	unsup.	0.0259	0.5159
PRILO-MII	DCGAN	unsup.	0.0222	0.5264
DPR	Progressive GAN	unsup.	0.0285	0.5276
DPR-MII	Progressive GAN	unsup.	0.0217	0.5738
PRILO	Progressive GAN	unsup.	0.0244	0.5665
PRILO-MII	Progressive GAN	unsup.	0.0203	0.5910
DPR	StyleGAN	unsup.	0.0352	0.4859
DPR-MII	StyleGAN	unsup.	0.0209	0.5972
PRILO	StyleGAN	unsup.	0.0307	0.5143
PRILO-MII	StyleGAN	unsup.	0.0139	0.6358
E2E [88]	–	sup.	0.0123	0.6367
PRCGAN-L [88]	–	sup.	**0.0093**	0.6846
DPR-LI	StyleGAN	sup.	0.0099	<u>0.7021</u>
PRILO-LI	StyleGAN	sup.	<u>0.0094</u>	**0.7247**

Table 6.2: Fourier phase retrieval on CelebA: Our proposed PRILO combined with LI and StyleGAN achieves the best results in terms of MSE and SSIM, also against enhanced version of DPR, which uses LI and StyleGAN. Best values are printed **bold** and second-best are <u>underlined</u>.

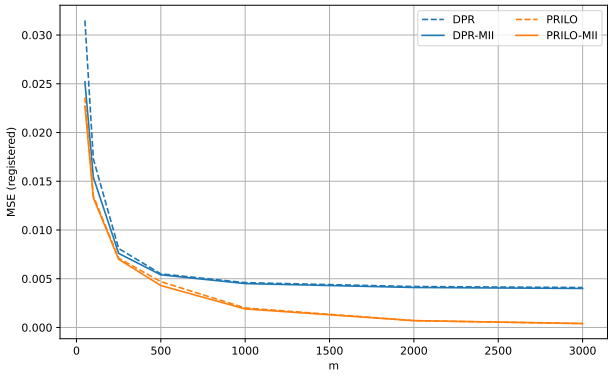
We directly compare our results with the results of the DPR method for real- and complex-valued measurement matrices (the real-valued case was already covered in [30]). Figure 6.2 shows the MSE and the SSIM values for different numbers of measurements m. Our proposed method PRILO-MII (orange) improves DPR's results (blue) by a margin. Note that for compressive Gaussian phase retrieval only unsupervised methods are tested, since each measurement matrix requires retraining the model.

6.4.3 Ablation Studies

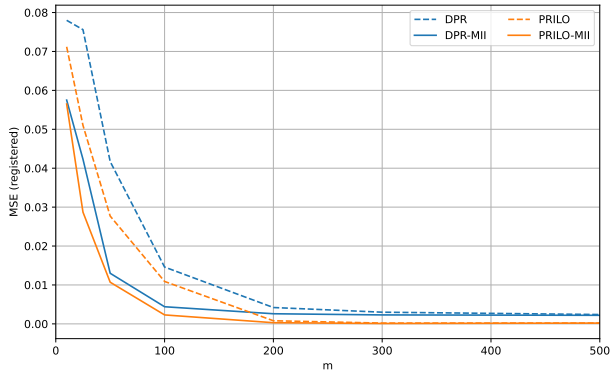
In our experiments, we observed that the performance can be improved by re-running the optimization procedure with a different initialization and selecting the result with the lowest magnitude error which is a common practice in image reconstruction. However, this approach is quite costly. Our initialization schemes described in Section 6.3.2 can be used to achieve a similar effect without the need of re-running the whole optimization procedure multiple times. In Figure 6.3, we compare the single randomly initialized latent variable, 5000 initialization using the MII approach and slightly perturbed learned initialization (LI). Again, we consider a test set of 1024 sam-



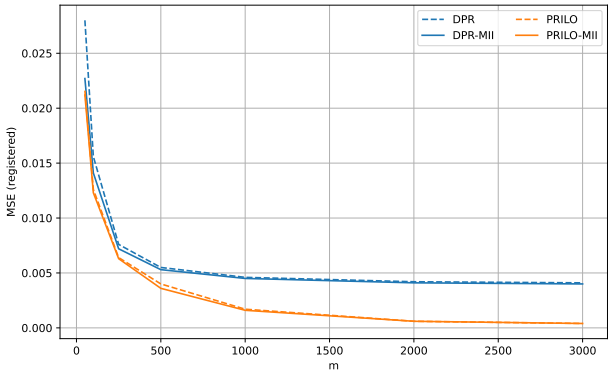
(a) MNIST: real-valued measurement matrices



(b) CelebA: real-valued measurement matrices



(c) MNIST: complex-valued measurement matrices



(d) CelebA: complex-valued measurement matrices

Figure 6.2: Real and complex Gaussian phase retrieval on MNIST and CelebA: PRILO-MII greatly improves DPR's reconstruction results. Note, that we also combine DPR with our MII. For MNIST we used the VAE and for CelebA the StyleGAN as base model.

ples. Although, we are still using restarts in combination with our initialization, we observe that already for a single run our initialization outperforms the results of 5 runs.

Table 6.3: Ablation experiments using StyleGAN on CelebA for Fourier measurements.

	MSE (↓)	SSIM (↑)
PRILO-LI	0.0094	0.7247
only initialization	0.0246	0.4777
only step A	0.0102	0.7108
only step A and B	0.0108	0.7063
no ball constraint	0.0115	0.6974
PRILO-MII	0.0139	0.6358
only initialization	0.0380	0.3565
only step A	0.0220	0.5861
only step A and B	0.0236	0.5719
no ball constraint	0.0260	0.5540

In order to assess the impact of each step on the reconstruction performance, we perform an ablation study. We compare our complete PRILO-MII and PRILO-LI models with different modified versions of the model: for one, we consider omitting every optimization, i.e., we only compare with the initialization. Next, we omit the back-projection step (B) and the refinement step (C). We also compare with the variant of our model which we only omit step C. Finally, we analyze the impact of the ball-constraints. Table 6.3 shows that each of the components is important to reach the results. Notably, only using step A and B (which corresponds to naïvely applying the approach by Daras et al. [16] to phase retrieval) gives worse results than only step A. However, in combination with our proposed step C the approach gives better results.

6.5 Conclusion

Generative models play an important role in solving inverse problem. In the context of phase retrieval, we observe that optimizing intermediate representations is essential to obtain excellent reconstructions. Our method PRILO, used in combination with our new initialization schemes, produces better reconstruction results for Gaussian and Fourier phase retrieval than existing (supervised and unsupervised) methods. Notably, in some cases our unsupervised variant PRILO-MII even outperforms existing supervised methods. We also show that the initialization schemes we introduced can easily be adapted to different methods for inverse problems that are based on generative models, e.g., DPR. Our ablation study justifies that each of our used compo-

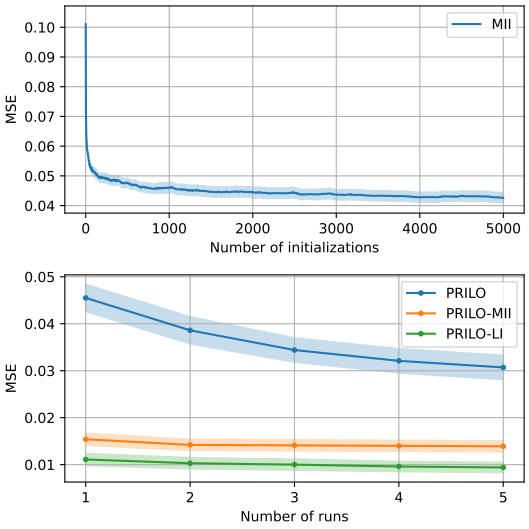


Figure 6.3: Effect of the number of initializations and number of restarts onto the reconstruction performance. Shaded areas indicate the 95%-confidence interval.

nents is essential to achieve the reported results.

Limitations

During our experiments, we observed that some hyperparameter tuning is necessary to achieve good results: the selected intermediate layer highly influences the results and also the radii for the constraints play an important role. Furthermore, our approach is in some cases not able to reconstruct the finer details and is still (to some extend) limited by the generative model (FMNIST results reported in Table 6.1). Furthermore, generator-based methods for non-oversampled Fourier phase retrieval still struggle to reconstruct high-resolution images as demonstrated in appendix of the published paper.

Broader Impact Statement

This work discusses a method for reconstructing images from their magnitude measurements. Whether this method can have negative societal impacts is difficult to predict and depends on the specific application, e.g., in X-ray crystallography or microscopy. For example, in some applications the reconstructed images could contain sensitive information, which could lead to data security issues if the method is not applied properly.

7
A Closer Look at Reference Learning for Phase Retrieval

Corresponding publication. The contents of this chapter have been published as:

Tobias Uelwer, Nick Rucks, and Stefan Harmeling. A closer look at reference learning for fourier phase retrieval. In *NeurIPS 2021 Workshop on Deep Learning and Inverse Problems*, 2021.

Personal contributions. Tobias Uelwer did the literature review, implemented the unrolled error-reduction algorithm, proposed the new baseline reference and designed and conducted the experiment evaluation. Furthermore, he wrote large parts of the manuscript and created the plots and figures. He edited the manuscript together with Stefan Harmeling and Nick Rucks. Stefan Harmeling supervised the project.

Abstract. Reconstructing images from their Fourier magnitude measurements is a problem that often arises in different research areas. This process is also referred to as phase retrieval. In this work, we consider a modified version of the phase retrieval problem, which allows for a reference image to be added onto the image before the Fourier magnitudes are measured. We analyze an unrolled error-reduction (ER) algorithm that can be used to learn a good reference image from a dataset. Furthermore, we take a closer look at the learned reference images and propose a simple and efficient heuristic to construct reference images that, in some cases, yields reconstructions of comparable quality as approaches that learn references.

7.1 Introduction

The problem discussed in this chapter differs from the more general phase retrieval problem considered in the previous chapters. In the fol-

lowing we want to shed some light onto reference learning for Fourier phase retrieval. On the on hand side, we propose a modified Error-reduction algorithm that can be used to reconstruct images and learn references in such a setting. On the other hand, we argue that learning reference images is not always necessary and one can instead use a more simple reference image that yields comparable results to learned references.

7.2 Related Work

The idea of unrolling algorithms has been considered before for other phase retrieval problems: The performance of an unrolled ER algorithm for phase retrieval has been analyzed by Schlieder et al. [78]. Naimipour et al. [66] unrolled the Wirtinger flow algorithm for compressive phase retrieval. Unlike our work, these papers augment existing phase retrieval algorithms with learnable parameters which is different from the reference based phase retrieval we consider in this work.

Phase retrieval with a reference

The Fourier phase retrieval problem with a reference image was first analyzed by Kim and Hayes [49, 48]. Recently, Hyder et al. [36] showed that such a reference can be learned from a rather small dataset using an unrolled gradient descent algorithm. While it is interesting that this is possible, a couple of questions arise which we discuss in this work.

7.3 Contributions

The contributions of this chapter can be summarized as follows:

1. We modify the well-established error-reduction (ER) algorithm [22] so that it utilizes a reference. Furthermore, we show that this modified ER algorithm can be unrolled to learn a reference image (similar to the approach of Hyder et al. [36]).

2. We answer the question under which conditions learning a reference image for Fourier phase retrieval is really necessary. For this, we propose a simple baseline reference image that is easily constructed and compare its performance with the learned references in the oversampled and the non-oversampled case.

7.4 Unrolling the ER Algorithm to Learn a Reference

The phase retrieval algorithm discussed by Hyder et al. [36] is a gradient descent method that requires extensive parameter tuning of its step size. For this reason, we unroll the ER algorithm, which typically converges within a few iterations. The ER algorithm iteratively replaces

the magnitude of the current estimate with the measured magnitude and enforces a positivity-constraint on the image after the reference has been subtracted. Algorithm 2 shows our modified version of ER that utilizes a reference image. We learn the reference by calculating the mean squared error (MSE) between outputs of the ER algorithm after n iterations and the corresponding original images. Then we perform backpropagation to calculate the gradient with respect to the reference u and update it using a gradient descent step. These steps are repeated for multiple batches until convergence is achieved.

Unrolling

Algorithm 2: ER algorithm for phase retrieval with reference

Input: Fourier magnitude $y \in \mathbb{R}^{d \times d}$, reference image $r \in \mathbb{R}^{d \times d}$, initialization $x_0 \in \mathbb{R}^{d \times d}$, number of iterations n

Output: Reconstruction $x_n \in \mathbb{R}^{d \times d}$

for $k = 0, \ldots, n-1$ **do**
  $p_{k+1} \leftarrow \mathcal{F}(x_k + r)/|\mathcal{F}(x_k + r)|$ `// Estimate phase`
  $\bar{x}_{k+1} \leftarrow \mathcal{F}^{-1}(p_{k+1} \odot y) - r$ `// Fourier constraints`
  $x_{k+1} \leftarrow \max(0, \bar{x}_{k+1})$ `// Image constraints`
end
return x_n

7.5 *Constructing References Mimicking the Learned Ones*

Figure 7.1: Comparison of different learned references. For each of the two learned methods, the shown references are trained on the following datasest: MNIST, EMNIST, FMNIST, SVHN, CIFAR-10.

Learning the reference image is computationally expensive. The most obvious baseline reference is a random image. For this, we sample the pixel values from a uniform distribution $\mathcal{U}(0,1)$. Looking at various learned reference images produced by the algorithm of Hyder et al. [36] and our unrolled ER algorithm (Figure 7.1), we see that most pixels are either zero or one. This suggests to also consider random binarized reference images.

However, looking more closely at those learned reference images (Figure 7.1), we see that all references exhibit flat areas. Furthermore, the learned references do not show any symmetries. These observations suggest trying a simple reference image as a baseline mimicking

Figure 7.2: Our simple reference.

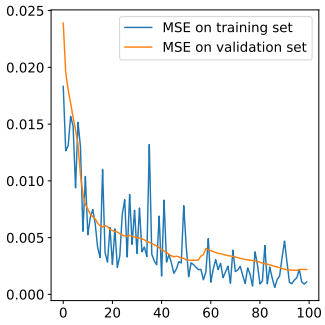


Figure 7.3: Training and validation MSE between reconstructions and original images of the FMNIST dataset.

Hyperparameters

those features. To construct such a reference, we start with a black image that has a white square in the bottom right corner. After blurring this image with a Gaussian filter, we normalize and add weighted Poisson noise to every pixel. Finally, we binarize the image appropriately. A resulting simple reference is shown on the right in Figure 7.2.

In the next section, we compare the performance of these different references for reconstructing images from their oversampled and non-oversampled magnitude measurements.

7.6 Comparing Baseline and Learned References

Experimental Setup. To study whether references are necessary at all, we also reconstruct images from measurements that were taken without adding a reference onto the image. In that case, the trivial ambiguities (translation and flip) cannot be resolved by the reconstruction algorithm. Therefore, we register the reconstructions appropriately before calculating the error. For fair comparison, we register the reconstructions of the other methods before calculating the MSE as well.

We evaluate the reconstructions of images from the MNIST [55], the EMNIST [13], the FMNIST [102], the CIFAR-10 [53], and the SVHN [68],datasets. We convert the images from the CIFAR-10 and the SVHN dataset to grayscale. We consider non-oversampled magnitude measurements as well as measurements that are oversampled by a factor of two in each dimension. To solve the Fourier phase retrieval problem, we use a similar algorithm as Hyder et al. [36]. We run the algorithm for 500 steps and set the step size to $\alpha = 1.95$, which was chosen on a validation dataset. For each of the datasets, we use 1000 images for testing and we train the references on 100 training images using Adam [50] with a learning rate of 0.01. We use batches of size 10. During training, we unroll the ER algorithm for 15 steps and use 500 steps to reconstruct the images at test time.

Similar to the intensities of the images, we restrict the entries of the reference image to lie between 0 and 1 as using larger values makes the problem trivial to solve, i.e., even a random reference with values between 0 and 100 yields perfect reconstructions.

Results. Table 7.1 shows that our unrolled ER algorithm is able to learn references of similar quality as the method proposed by Hyder et al. [36]. We see that the reconstruction errors of our simple reference are quite close to the errors of the learned methods in the oversampled case. Figure 7.3 shows the training and the validation loss for a reference trained using our unrolled ER algorithm on the FMNIST dataset. We can observe that only a few stochastic gradient descent steps are necessary to drastically decrease training and validation error. Also,

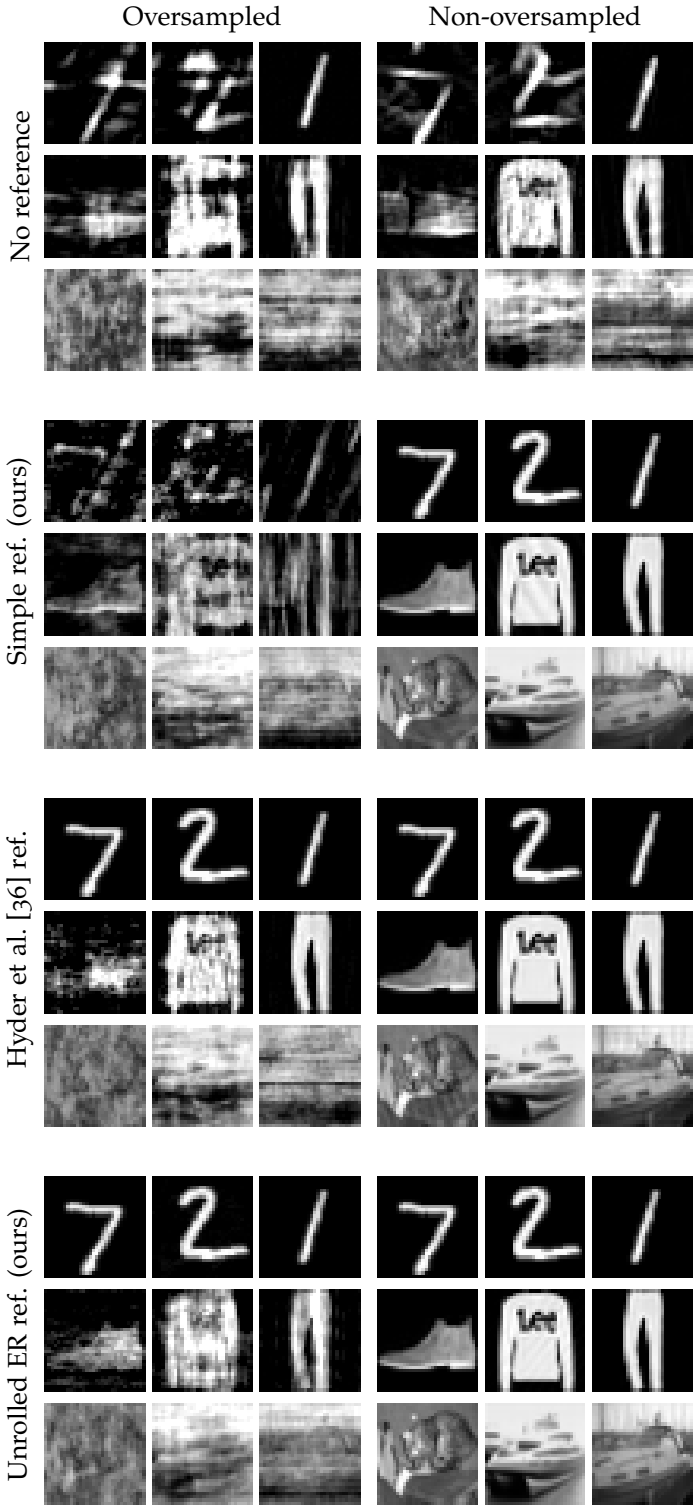


Figure 7.4: Reconstructions of the MNIST, the FMNIST and the CIFAR-10 dataset from non-oversampled and oversampled Fourier magnitudes using different references.

	Method	MNIST	EMNIST	FMNIST	SVHN	CIFAR-10
Non-oversampled	No reference	0.035615	0.063414	0.042417	0.013985	0.035134
	Random reference	0.052724	0.079784	0.046670	0.012393	0.030222
	Random reference (binary)	0.055324	0.081130	0.049436	0.012447	0.029299
	Simple reference (ours)	0.060027	0.089347	0.067549	0.011270	0.024128
	Hyder et al. [36] reference	0.002607	**0.014687**	**0.013649**	0.010131	0.024141
	Unrolled ER reference (ours)	**0.002181**	0.015427	0.019863	**0.008775**	**0.020020**
Oversampled	No reference	0.020566	0.032907	0.021068	0.007516	0.020518
	Random reference	0.005350	0.025308	0.011484	0.003670	0.009848
	Random reference (binary)	0.001170	0.010994	0.006842	0.002462	0.007310
	Simple reference (ours)	0.000761	0.003681	0.000848	0.000187	0.000495
	Hyder et al. [36] reference	0.000132	**0.000023**	0.000073	0.000126	0.001415
	Unrolled ER reference (ours)	**0.000071**	0.000257	**0.000055**	**0.000055**	**0.000125**

Table 7.1: Comparison of the mean squared reconstruction error (lower is better) using different references for Fourier phase retrieval from non-oversampled and oversampled magnitudes. Best values are printed **bold** and second-best are underlined.

our ER algorithm requires only a small number of iterations in training to learn a good reference. Figure 7.4 visualizes some reconstructions. In the non-oversampled case, the learned references produce better reconstructions than our simple reference. Surprisingly, the random references that we use as baseline perform worse on the MNIST-like datasets than using no baseline in the non-oversampled case. In the oversampled case, the learned references, as well as our simple reference, produce almost perfect reconstructions. Therefore, we conclude that learning references is not necessary when the Fourier magnitudes are oversampled.

7.7 *Conclusion*

In this chapter, we discuss how the ER algorithm can be unrolled in order to learn a reference image for phase retrieval. We show that our unrolled ER algorithm performs comparably to an existing method and requires only a few unrolling steps while having no hyperparameters that require extensive tuning. Furthermore, we provide a simple reference which puts the benefit of a learned reference into perspective, especially when oversampled magnitude measurements are available.

7.8 *Broader Impact*

Fourier phase retrieval is a fundamental and relevant imaging problem in many areas of science, e.g., in X-ray crystallography or microscopy. Ethical or societal consequences depend on the exact application of the phase retrieval algorithm. Our insights set the performance gain

of learned reference images for phase retrieval into perspective. We propose a simpler and more efficient approach for reference construction which yields similar results and requires less computation and thus less energy.

8
Conclusion

Summary

In this thesis, we have shown how deep learning can be applied to phase retrieval problems. We have taken a look at unsupervised methods that train a generative model on similar images without having knowledge about the corresponding measurements. These methods usually rely on optimizing the latent variable of the generative model to reconstruct the image. Going beyond this latent optimization, we have shown that optimization of intermediate representations of the generative model can further improve the reconstruction quality. In contrast to that, supervised methods use images paired with their corresponding measurements during training. One instance of supervised learning is end-to-end learning, which learns a mapping from magnitude measurements to the images. We combine both ideas and show how this can further improve the reconstruction performance of (conditional) generative adversarial networks.

These learning-based methods produce outstanding reconstructions that surpass the performance of existing methods when training data is available. In addition, we analyzed the robustness of learning-based models to different forms of noise in the measurement process and studied the generalization abilities of these models to different data distributions. We also have taken a look on reference based phase retrieval and discussed whether learning an additive reference is necessary.

Table 8.1 gives an overview of the optimization-based and learning-based methods discussed in this thesis.

Future work

Future work could focus on improving the computational runtime of learning-based methods, especially those involving costly optimization procedures.

Real world data

In this thesis, we evaluated the methods on benchmark image datasets and simulated measurements, which is a shortcoming of our experiments. However, this has been addressed by the work of Ye et al. [104] who applied our PRCGAN (among other methods) to measurements that were obtained in an optical experiment: a lens-based system consisting of a Helium-neon laser, a phase-only

Method	Publication	Learning	End-to-end	Iterative	Reference Image
ER	Fienup [22]	no	no	yes	no
IO	Fienup [22]	no	no	yes	no
OO	Fienup [22]	no	no	yes	no
HIO	Fienup [22]	no	no	yes	no
RAAR	Luke [59]	no	no	yes	no
ResNet	Nishizaki et al. [69]	supervised	yes	no	no
E2E	Uelwer et al. [88]	supervised	yes	no	no
DPR	Hand et al. [30]	unsupervised	no	yes	no
PRCGAN-D	Uelwer et al. [88]	supervised	yes	no	no
PRCGAN-L	Uelwer et al. [88]	supervised	no	yes	no
PRCGAN-W	Uelwer et al. [88]	supervised	no	yes	no
CPR	Uelwer et al. [87]	supervised	yes	no	no
CPR-FS	Uelwer et al. [87]	supervised	yes	no	no
PRILO	Uelwer et al. [90]	unsupervised	no	yes	no
PRILO-MII	Uelwer et al. [90]	unsupervised	no	yes	no
PRILO-LI	Uelwer et al. [90]	supervised	no	yes	no
Unrolled GD	Hyder et al. [36]	supervised	no	yes	yes
Unrolled ER	Uelwer et al. [89]	supervised	no	yes	yes

Table 8.1: Overview over the methods for Fourier phase retrieval considered in this thesis. All methods can also be modified to solve compressive Gaussian phase retrieval.

spatial light modulator and a CMOS camera was used to capture the magnitude-only measurements. This shows that our method is also applicable to realistic settings.

Furthermore, it would be interesting to further evaluate learning-based methods in other realistic measurement setups. Whether deep learning can be used to solve problems like X-ray crystallography remains an open question, but recent breakthroughs in other areas of research like AlphaFold [41] gives cause for optimism.

References

[1] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein generative adversarial networks. In *International conference on machine learning*, pages 214–223. PMLR, 2017.

[2] Muhammad Asim, Max Daniels, Oscar Leong, Ali Ahmed, and Paul Hand. Invertible generative models for inverse problems: mitigating representation error and dataset bias. In *International Conference on Machine Learning*, pages 399–409. PMLR, 2020.

[3] Heinz H Bauschke, Patrick L Combettes, and D Russell Luke. Finding best approximation pairs relative to two closed convex sets in hilbert spaces. *Journal of Approximation theory*, 127(2):178–192, 2004.

[4] Lokesh Boominathan, Mayug Maniparambil, Honey Gupta, Rahul Baburajan, and Kaushik Mitra. Phase retrieval for fourier ptychography under varying amount of measurements. *arXiv preprint arXiv:1805.03593*, 2018.

[5] Ashish Bora, Ajil Jalal, Eric Price, and Alexandros G Dimakis. Compressed sensing using generative models. In *International Conference on Machine Learning*, pages 537–546. PMLR, 2017.

[6] Lisa Gottesfeld Brown. A survey of image registration techniques. *ACM computing surveys (CSUR)*, 24(4):325–376, 1992.

[7] Oliver Bunk, Ana Diaz, Franz Pfeiffer, Christian David, Bernd Schmitt, Dillip K Satapathy, and J Friso Van Der Veen. Diffractive imaging for periodic samples: retrieving one-dimensional concentration profiles across microfluidic channels. *Acta Crystallographica Section A: Foundations of Crystallography*, 63(4):306–314, 2007.

[8] Emmanuel J Candes, Xiaodong Li, and Mahdi Soltanolkotabi. Phase retrieval via wirtinger flow: Theory and algorithms. *IEEE Transactions on Information Theory*, 61(4):1985–2007, 2015.

[9] Eunju Cha, Chanseok Lee, Mooseok Jang, and Jong Chul Ye. Deepphasecut: Deep relaxation in phase for unsupervised fourier phase retrieval. *arXiv preprint arXiv:2011.10475*, 2020.

[10] Mingqin Chen, Peikang Lin, Yuhui Quan, Tongyao Pang, and Hui Ji. Unsupervised phase retrieval using deep approximate mmse estimation. *IEEE Transactions on Signal Processing*, 70:2239–2252, 2022.

[11] Zhuojie Chen, Yan Huang, Yu Hu, and Zhifeng Chen. Phase recovery with deep complex-domain priors. *IEEE Signal Processing Letters*, 29:887–891, 2022.

[12] Tarin Clanuwat, Mikel Bober-Irizar, Asanobu Kitamoto, Alex Lamb, Kazuaki Yamamoto, and David Ha. Deep learning for classical japanese literature. *arXiv preprint arXiv:1812.01718*, 2018.

[13] Gregory Cohen, Saeed Afshar, Jonathan Tapson, and Andre Van Schaik. Emnist: Extending mnist to handwritten letters. In *2017 international joint conference on neural networks (IJCNN)*, pages 2921–2926. IEEE, 2017.

[14] Nadav Cohen and Amnon Shashua. Inductive bias of deep convolutional networks through pooling geometry. In *International Conference on Learning Representations*, 2017.

[15] James W Cooley and John W Tukey. An algorithm for the machine calculation of complex fourier series. *Mathematics of computation*, 19(90):297–301, 1965.

[16] Giannis Daras, Joseph Dean, Ajil Jalal, and Alex Dimakis. Intermediate layer optimization for inverse problems using deep generative models. In *International Conference on Machine Learning*, pages 2421–2432. PMLR, 2021.

[17] Laurent Dinh, David Krueger, and Yoshua Bengio. Nice: Non-linear independent components estimation. *arXiv preprint arXiv:1410.8516*, 2014.

[18] Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. Density estimation using real nvp. *arXiv preprint arXiv:1605.08803*, 2016.

[19] J. Dongarra and F. Sullivan. Guest editors introduction to the top 10 algorithms. *Computing in Science & Engineering*, 2(1):22–23, 2000. DOI: 10.1109/MCISE.2000.814652.

[20] John Duchi, Shai Shalev-Shwartz, Yoram Singer, and Tushar Chandra. Efficient projections onto the l 1-ball for learning in

high dimensions. In *Proceedings of the 25th international conference on Machine learning*, pages 272–279, 2008.

[21] C Fienup and J Dainty. Phase retrieval and image reconstruction for astronomy. *Image recovery: theory and application*, 231:275, 1987.

[22] James R Fienup. Phase retrieval algorithms: a comparison. *Applied optics*, 21(15):2758–2769, 1982.

[23] Ralph W Gerchberg. A practical algorithm for the determination of phase from image and diffraction plane pictures. *Optik*, 35: 237–246, 1972.

[24] Ian Goodfellow. Nips 2016 tutorial: Generative adversarial networks. *arXiv preprint arXiv:1701.00160*, 2016.

[25] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in Neural Information Processing Systems*, volume 27. Curran Associates, Inc., 2014.

[26] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.

[27] Manuel Guizar-Sicairos, Samuel T Thurman, and James R Fienup. Efficient subpixel image registration algorithms. *Optics letters*, 33(2):156–158, 2008.

[28] Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron C Courville. Improved training of wasserstein GANs. *Advances in neural information processing systems*, 30, 2017.

[29] Shashank Gupta, Sumohana S Channappayya, et al. Perceptually driven conditional GAN for fourier ptychography. In *2019 53rd Asilomar Conference on Signals, Systems, and Computers*, pages 1267–1271. IEEE, 2019.

[30] Paul Hand, Oscar Leong, and Vlad Voroninski. Phase retrieval under a generative prior. In *Advances in Neural Information Processing Systems*, pages 9136–9146, 2018.

[31] Irina Higgins, Loic Matthey, Arka Pal, Christopher Burgess, Xavier Glorot, Matthew Botvinick, Shakir Mohamed, and Alexander Lerchner. beta-VAE: Learning basic visual concepts with a constrained variational framework. In *International Conference on Learning Representations*, 2017.

[32] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *Advances in Neural Information Processing Systems*, 33:6840–6851, 2020.

[33] Sepp Hochreiter and Jürgen Schmidhuber. Flat minima. *Neural computation*, 9(1):1–42, 1997.

[34] Shady Abu Hussein, Tom Tirer, and Raja Giryes. Image-adaptive gan based reconstruction. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 3121–3129, 2020.

[35] Rakib Hyder, Viraj Shah, Chinmay Hegde, and M Salman Asif. Alternating phase projected gradient descent with generative priors for solving compressive phase retrieval. In *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 7705–7709. IEEE, 2019.

[36] Rakib Hyder, Zikui Cai, and M Salman Asif. Solving phase retrieval with a learned reference. In *European Conference on Computer Vision*, pages 425–441. Springer, 2020.

[37] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. PMLR, 2015.

[38] Çağatay Işıl, Figen S Oktem, and Aykut Koç. Deep iterative reconstruction for phase retrieval. *Applied optics*, 58(20):5422–5431, 2019.

[39] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A Efros. Image-to-image translation with conditional adversarial networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1125–1134, 2017.

[40] Gauri Jagatap and Chinmay Hegde. Phase retrieval using untrained neural network priors. In *NeurIPS 2019 Workshop on Deep Learning and Inverse Problems*, 2019.

[41] John Jumper, Richard Evans, Alexander Pritzel, Tim Green, Michael Figurnov, Olaf Ronneberger, Kathryn Tunyasuvunakool, Russ Bates, Augustin Žídek, Anna Potapenko, et al. Highly accurate protein structure prediction with alphafold. *Nature*, 596(7873):583–589, 2021.

[42] Tero Karras, Timo Aila, Samuli Laine, and Jaakko Lehtinen. Progressive growing of GANs for improved quality, stability, and variation. *arXiv preprint arXiv:1710.10196*, 2017.

[43] Tero Karras, Samuli Laine, and Timo Aila. A style-based generator architecture for generative adversarial networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4401–4410, 2019.

[44] Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. On large-batch training for deep learning: Generalization gap and sharp minima. *arXiv preprint arXiv:1609.04836*, 2016.

[45] Vinayak Killedar and Chandra Sekhar Seelamantula. Compressive phase retrieval based on sparse latent generative priors. In *2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 1596–1600. IEEE, 2022.

[46] Vinayak Killedar, Praveen Kumar Pokala, and Chandra Sekhar Seelamantula. Learning generative prior with latent space sparsity constraints. *arXiv preprint arXiv:2105.11956*, 2021.

[47] Kyung-Su Kim, Jung Hyun Lee, and Eunho Yang. Compressed sensing via measurement-conditional generative models. *IEEE Access*, 2021.

[48] Wooshik Kim and Monson H Hayes. Iterative phase retrieval using two fourier transform intensities. In *International Conference on Acoustics, Speech, and Signal Processing*, pages 1563–1566. IEEE, 1990.

[49] Wooshik Kim and Monson H Hayes. Phase retrieval using two fourier-transform intensities. *JOSA A*, 7(3):441–449, 1990.

[50] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[51] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[52] Durk P Kingma and Prafulla Dhariwal. Glow: Generative flow with invertible 1x1 convolutions. *Advances in neural information processing systems*, 31, 2018.

[53] Alex Krizhevsky. Learning multiple layers of features from tiny images. Citeseer, 2009.

[54] Hannah Lawrence, David A Barmherzig, Henry Li, Michael Eickenberg, and Marylou Gabrié. Phase retrieval with holography and untrained priors: Tackling the challenges of low-photon nanoscale imaging. *arXiv preprint arXiv:2012.07386*, 2020.

[55] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[56] Fei-Fei Li, Jiajun Wu, and Ruohan Gao. CS231n: Deep learning for computer vision. University Lecture, 2022.

[57] Zhaoqiang Liu, Subhroshekhar Ghosh, and Jonathan Scarlett. Towards sample-optimal compressive phase retrieval with sparse and generative priors. *Advances in Neural Information Processing Systems*, 34, 2021.

[58] Ziwei Liu, Ping Luo, Xiaogang Wang, and Xiaoou Tang. Deep learning face attributes in the wild. In *Proceedings of International Conference on Computer Vision (ICCV)*, December 2015.

[59] D Russell Luke. Relaxed averaged alternating reflections for diffraction imaging. *Inverse problems*, 21(1):37, 2004.

[60] Raunak Manekar, Kshitij Tayal, Vipin Kumar, and Ju Sun. End to end learning for phase retrieval. In *ICML workshop on ML Interpretability for Scientific Discovery*, 2020.

[61] Raunak Manekar, Zhong Zhuang, Kshitij Tayal, Vipin Kumar, and Ju Sun. Deep learning initialized phase retrieval. In *NeurIPS 2020 Workshop on Deep Learning and Inverse Problems*, 2020.

[62] Xudong Mao, Qing Li, Haoran Xie, Raymond YK Lau, Zhen Wang, and Stephen Paul Smolley. Least squares generative adversarial networks. In *Proceedings of the IEEE international conference on computer vision*, pages 2794–2802, 2017.

[63] Christopher Metzler, Phillip Schniter, Ashok Veeraraghavan, et al. prdeep: robust phase retrieval with a flexible deep network. In *International Conference on Machine Learning*, pages 3501–3510. PMLR, 2018.

[64] Rick P Millane. Phase retrieval in crystallography and optics. *JOSA A*, 7(3):394–411, 1990.

[65] Mehdi Mirza and Simon Osindero. Conditional generative adversarial nets. *arXiv preprint arXiv:1411.1784*, 2014.

[66] Naveed Naimipour, Shahin Khobahi, and Mojtaba Soltanalian. Unfolded algorithms for deep phase retrieval. *arXiv preprint arXiv:2012.11102*, 2020.

[67] Arvind Neelakantan, Luke Vilnis, Quoc V Le, Ilya Sutskever, Lukasz Kaiser, Karol Kurach, and James Martens. Adding gradient noise improves learning for very deep networks. *arXiv preprint arXiv:1511.06807*, 2015.

[68] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y Ng. Reading digits in natural images with unsupervised feature learning. *NIPS Workshop on Deep Learning and Unsupervised Feature Learning*, 2011.

[69] Yohei Nishizaki, Ryoichi Horisaki, Katsuhisa Kitaguchi, Mamoru Saito, and Jun Tanida. Analysis of non-iterative phase retrieval based on machine learning. *Optical Review*, 27(1):136–141, 2020.

[70] George Papamakarios, Theo Pavlakou, and Iain Murray. Masked autoregressive flow for density estimation. *Advances in neural information processing systems*, 30, 2017.

[71] Deepak Pathak, Philipp Krahenbuhl, Jeff Donahue, Trevor Darrell, and Alexei A Efros. Context encoders: Feature learning by inpainting. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2536–2544, 2016.

[72] Boris T Polyak. Some methods of speeding up the convergence of iteration methods. *Ussr computational mathematics and mathematical physics*, 4(5):1–17, 1964.

[73] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*, 2015.

[74] Herbert Robbins and Sutton Monro. A stochastic approximation method. *The annals of mathematical statistics*, pages 400–407, 1951.

[75] Yaniv Romano, Michael Elad, and Peyman Milanfar. The little engine that could: Regularization by denoising (RED). *SIAM Journal on Imaging Sciences*, 10(4):1804–1844, 2017.

[76] Nick Rucks, Tobias Uelwer, and Stefan Harmeling. [Re] solving phase retrieval with a learned reference. In *ML Reproducibility Challenge 2021 (Fall Edition)*, 2021.

[77] Jo Schlemper, Jose Caballero, Joseph V Hajnal, Anthony N Price, and Daniel Rueckert. A deep cascade of convolutional neural networks for dynamic mr image reconstruction. *IEEE transactions on Medical Imaging*, 37(2):491–503, 2017.

[78] Lennart Schlieder, Heiner Kremer, Valentin Volchkov, Kai Melde, Peer Fischer, and Bernhard Schölkopf. Learned residual gerchberg-saxton network for computer generated holography. 2020.

[79] Robin M Schmidt, Frank Schneider, and Philipp Hennig. Descending through a crowded valley-benchmarking deep learning optimizers. In *International Conference on Machine Learning*, pages 9367–9376. PMLR, 2021.

[80] Yoav Shechtman, Yonina C Eldar, Oren Cohen, Henry Nicholas Chapman, Jianwei Miao, and Mordechai Segev. Phase retrieval with application to optical imaging: a contemporary overview. *IEEE signal processing magazine*, 32(3):87–109, 2015.

[81] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[82] Niklas Smedemark-Margulies, Jung Yeon Park, Max Daniels, Rose Yu, Jan-Willem van de Meent, and Paul Hand. Generator surgery for compressed sensing. *arXiv preprint arXiv:2102.11163*, 2021.

[83] Jascha Sohl-Dickstein, Eric Weiss, Niru Maheswaranathan, and Surya Ganguli. Deep unsupervised learning using nonequilibrium thermodynamics. In *International Conference on Machine Learning*, pages 2256–2265. PMLR, 2015.

[84] Yang Song and Stefano Ermon. Generative modeling by estimating gradients of the data distribution. *Advances in Neural Information Processing Systems*, 32, 2019.

[85] Yang Song, Jascha Sohl-Dickstein, Diederik P Kingma, Abhishek Kumar, Stefano Ermon, and Ben Poole. Score-based generative modeling through stochastic differential equations. *arXiv preprint arXiv:2011.13456*, 2020.

[86] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

[87] Tobias Uelwer, Tobias Hoffmann, and Stefan Harmeling. Non-iterative phase retrieval with cascaded neural networks. In *Artificial Neural Networks and Machine Learning–ICANN 2021: 30th International Conference on Artificial Neural Networks, Bratislava, Slovakia, September 14–17, 2021, Proceedings, Part II 30*, pages 295–306. Springer, 2021. DOI: 10.1007/978-3-030-86340-1_24.

[88] Tobias Uelwer, Alexander Oberstraß, and Stefan Harmeling. Phase retrieval using conditional generative adversarial networks. In *2020 25th International Conference on Pat-*

tern Recognition (ICPR), pages 731–738. IEEE, 2021. DOI: 10.1109/ICPR48806.2021.9412523.

[89] Tobias Uelwer, Nick Rucks, and Stefan Harmeling. A closer look at reference learning for fourier phase retrieval. In *NeurIPS 2021 Workshop on Deep Learning and Inverse Problems*, 2021.

[90] Tobias Uelwer, Sebastian Konietzny, and Stefan Harmeling. Optimizing intermediate representations of generative models for phase retrieval. *Transactions on Machine Learning Research*, 2022. ISSN 2835-8856. The first two authors contributed equally.

[91] Dmitry Ulyanov, Andrea Vedaldi, and Victor Lempitsky. Deep image prior. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 9446–9454, 2018.

[92] Benigno Uria, Marc-Alexandre Côté, Karol Gregor, Iain Murray, and Hugo Larochelle. Neural autoregressive distribution estimation. *The Journal of Machine Learning Research*, 17(1):7184–7220, 2016.

[93] Aaron Van den Oord, Nal Kalchbrenner, Lasse Espeholt, Oriol Vinyals, Alex Graves, et al. Conditional image generation with pixelcnn decoders. *Advances in neural information processing systems*, 29, 2016.

[94] Aäron Van Den Oord, Nal Kalchbrenner, and Koray Kavukcuoglu. Pixel recurrent neural networks. In *International conference on machine learning*, pages 1747–1756. PMLR, 2016.

[95] Irene Waldspurger, Alexandre d'Aspremont, and Stéphane Mallat. Phase recovery, maxcut and complex semidefinite programming. *Mathematical Programming*, 149(1):47–81, 2015.

[96] Adriaan Walther. The question of phase retrieval in optics. *Optica Acta: International Journal of Optics*, 10(1):41–49, 1963.

[97] Gang Wang, Georgios B Giannakis, and Yonina C Eldar. Solving systems of random quadratic equations via truncated amplitude flow. *IEEE Transactions on Information Theory*, 64(2):773–794, 2017.

[98] Yaotian Wang, Xiaohang Sun, and Jason Fleischer. When deep denoising meets iterative phase retrieval. In *International Conference on Machine Learning*, pages 10007–10017. PMLR, 2020.

[99] Zhou Wang, Alan C Bovik, Hamid R Sheikh, and Eero P Simoncelli. Image quality assessment: from error visibility to structural similarity. *IEEE transactions on image processing*, 13(4):600–612, 2004.

[100] Max Welling and Yee W Teh. Bayesian learning via stochastic gradient langevin dynamics. In *Proceedings of the 28th international conference on machine learning (ICML-11)*, pages 681–688. Citeseer, 2011.

[101] Zihui Wu, Yu Sun, Jiaming Liu, and Ulugbek Kamilov. Online regularization by denoising with applications to phase retrieval. In *Proceedings of the IEEE/CVF International Conference on Computer Vision Workshops*, 2019.

[102] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*, 2017.

[103] Rui Xu, Mahdi Soltanolkotabi, Justin P Haldar, Walter Unglaub, Joshua Zusman, Anthony FJ Levi, and Richard M Leahy. Accelerated wirtinger flow: A fast algorithm for ptychography. *arXiv preprint arXiv:1806.05546*, 2018.

[104] Qiuliang Ye, Li-Wen Wang, and Daniel PK Lun. SiSPRNet: End-to-end learning for single-shot phase retrieval. *Optics Express*, 30 (18):31937–31958, Aug 2022.

[105] Li-Hao Yeh, Jonathan Dong, Jingshan Zhong, Lei Tian, Michael Chen, Gongguo Tang, Mahdi Soltanolkotabi, and Laura Waller. Experimental robustness of fourier ptychography phase retrieval algorithms. *Optics express*, 23(26):33214–33240, 2015.

[106] Raymond A Yeh, Chen Chen, Teck Yian Lim, Alexander G Schwing, Mark Hasegawa-Johnson, and Minh N Do. Semantic image inpainting with deep generative models. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 5485–5493, 2017.

[107] Richard Zhang, Phillip Isola, Alexei A Efros, Eli Shechtman, and Oliver Wang. The unreasonable effectiveness of deep features as a perceptual metric. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 586–595, 2018.

[108] Guoan Zheng, Roarke Horstmeyer, and Changhuei Yang. Wide-field, high-resolution fourier ptychographic microscopy. *Nature photonics*, 7(9):739, 2013.

[109] Jiapeng Zhu, Yujun Shen, Deli Zhao, and Bolei Zhou. In-domain GAN inversion for real image editing. In *European conference on computer vision*, pages 592–608. Springer, 2020.

Appendices

A Code

A.1 Fienup's Phase Retrieval Algorithms

```python
import numpy as np


def fienup_phase_retrieval(
    mag, mask=None, beta=0.8, steps=200, mode="hybrid", verbose=True
):
    """
    Implementation of Fienup's phase-retrieval methods. This function
    implements the input-output, the output-output and the hybrid method.

    Note: Mode 'output-output' and beta=1 results in
    the Gerchberg-Saxton algorithm.

    Parameters:
        mag: Measured magnitudes of Fourier transform
        mask: Binary array indicating where the image should be
              if padding is known
        beta: Positive step size
        steps: Number of iterations
        mode: Which algorithm to use
              (can be 'input-output', 'output-output' or 'hybrid')
        verbose: If True, progress is shown

    Returns:
        x: Reconstructed image

    Author: Tobias Uelwer
    Date: 30.12.2018

    References:
    [1] E. Osherovich, Numerical methods for phase retrieval, 2012,
        https://arxiv.org/abs/1203.4756
    [2] J. R. Fienup, Phase retrieval algorithms: a comparison, 1982,
        https://www.osapublishing.org/ao/abstract.cfm?uri=ao-21-15-2758
    [3] https://github.com/cwg45/Image-Reconstruction
    """

    assert beta > 0, "step size must be a positive number"
    assert steps > 0, "steps must be a positive number"
    assert (
        mode == "input-output" or mode == "output-output" or mode == "hybrid"
    ), "mode must be 'input-output', 'output-output' or 'hybrid'"

    if mask is None:
        mask = np.ones(mag.shape)

    assert mag.shape == mask.shape, "mask and mag must have same shape"

    # sample random phase and initialize image x
    y_hat = mag * np.exp(1j * 2 * np.pi * np.random.rand(*mag.shape))
    x = np.zeros(mag.shape)
```

```
    # previous iterate
    x_p = None

    # main loop
    for i in range(1, steps + 1):
        # show progress
        if i % 100 == 0 and verbose:
            print("step", i, "of", steps)

        # inverse fourier transform
        y = np.real(np.fft.ifft2(y_hat))

        # previous iterate
        if x_p is None:
            x_p = y
        else:
            x_p = x

        # updates for elements that satisfy object domain constraints
        if mode == "output-output" or mode == "hybrid":
            x = y

        # find elements that violate object domain constraints
        # or are not masked
        indices = np.logical_or(
            np.logical_and(y < 0, mask), np.logical_not(mask)
        )

        # updates for elements that violate object domain constraints
        if mode == "hybrid" or mode == "input-output":
            x[indices] = x_p[indices] - beta * y[indices]
        elif mode == "output-output":
            x[indices] = y[indices] - beta * y[indices]

        # fourier transform
        x_hat = np.fft.fft2(x)

        # satisfy fourier domain constraints
        # (replace magnitude with input magnitude)
        y_hat = mag * np.exp(1j * np.angle(x_hat))
    return x
```

A.2 RAAR Algorithm

```
import numpy as np


def P_M(x, mag):
    """
    Implements Equation (2).

    Parameters:
        x: current iterate
        mag: measured magnitude
    """
    F_x = np.fft.fft2(x)
    v0_hat = mag * (F_x / (np.abs(F_x) + 1e-9))
    return np.real(np.fft.ifft2(v0_hat))


def P_Spos(x, mask=None):
    """
    Implements Equation (3).

    Parameters:
        x: current iterate
        mask: support mask, mask=None corresponds full support
    """
    if mask is None:
```

```
        return np.maximum(0, x)
    else:
        return np.maximum(0, x) * mask


def R_M(x, mag):
    """
    Implements reflector for M.

    Parameters:
        x: current iterate
        mag: measured magnitude
    """
    return 2 * P_M(x, mag) - x


def R_Spos(x, mask=None):
    """
    Implements reflector for Spos.

    Parameters:
        x: current iterate
    """
    return 2 * P_Spos(x, mask) - x


def T_star(x, mag, mask=None):
    return 0.5 * (R_Spos(R_M(x, mag), mask) + x)


def V(x, mag, beta, mask=None):
    return beta * T_star(x, mag, mask) + (1 - beta) * P_M(x, mag)


def RAAR(mag, beta=0.8, steps=200, verbose=True, mask=None):
    """
    Implementation of the relaxed averaged alternating reflections
    (RAAR) algorithm.

    Parameters:
        mag: Measured magnitudes of Fourier transform
        mask: Binary array indicating where the image should be
              if padding is known
        beta: Positive step size
        steps: Number of iterations
        verbose: If True, progress is shown

    Returns:
        x: Reconstructed image

    Author: Tobias Uelwer
    Date: 04.10.2022

    References:
    [1] Luke, D. Russell. "Relaxed averaged alternating reflections
    for diffraction imaging." Inverse problems 21.1 (2004): 37.
    """

    assert beta > 0, "step size must be a positive number"
    assert steps > 0, "steps must be a positive number"

    x = np.random.rand(*mag.shape)

    for i in range(1, steps + 1):

        if i % 10 == 0 and verbose:
            print("step", i, "of", steps, "(beta:" + str(beta) + ")")

        x = V(x, mag, beta, mask)
        # beta +=0.001

    x = np.real(x)

    return x
```

A.3 Neural Network Architectures

```python
import torch
import torch.nn as nn
import numpy as np


class MLP(nn.Module):
    def __init__(self, imsize=(1, 28, 28), outsize=(1, 28, 28), h=2048):
        super(MLP, self).__init__()
        self.imsize = imsize
        self.outsize = outsize
        self.layers = nn.Sequential(
            nn.Linear(imsize[0] * imsize[1] * imsize[2], h),
            nn.ReLU(inplace=False),
            nn.Linear(h, h),
            nn.BatchNorm1d(h),
            nn.ReLU(inplace=False),
            nn.Linear(h, h),
            nn.BatchNorm1d(h),
            nn.ReLU(inplace=False),
            nn.Linear(h, h),
            nn.BatchNorm1d(h),
            nn.ReLU(inplace=False),
            nn.Linear(h, outsize[0] * outsize[1] * outsize[2]),
            nn.Sigmoid(),
        )

    def forward(self, x):
        N = x.shape[0]
        out = self.layers(x.view(N, -1))
        return out.view(N, *self.outsize)


class FCIL(nn.Module):
    def __init__(self, h):
        super(FCIL, self).__init__()
        self.h = h
        self.layers = nn.Sequential(
            nn.Linear(h, h),
            nn.BatchNorm1d(h),
            nn.ReLU(),
            nn.Linear(h, h),
            nn.BatchNorm1d(h),
            nn.ReLU(),
        )

    def forward(self, x):
        shape = x.shape
        x = x.view(shape[0], self.h)
        return self.layers(x).view(*shape)


class ConvolutionalNetwork(nn.Module):
    def __init__(self, imsize=(3, 64, 64), outsize=(3, 64, 64), s=32):
        super(ConvolutionalNetwork, self).__init__()

        self.imsize = imsize
        self.outsize = outsize
        self.s = s

        padding, fcil_size = self.calculate_padding_and_fcil_size()
```

```
        self.layers = nn.Sequential(
            nn.Conv2d(imsize[0], imsize[0], kernel_size=1, padding=padding),
            nn.BatchNorm2d(imsize[0]),
            nn.ReLU(),
            nn.Conv2d(imsize[0], 1 * s, kernel_size=5, padding=2),
            nn.BatchNorm2d(1 * s),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2, stride=2),
            nn.Conv2d(1 * s, 2 * s, kernel_size=3, padding=1),
            nn.BatchNorm2d(2 * s),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2, stride=2),
            nn.Conv2d(2 * s, 4 * s, kernel_size=3, padding=1),
            nn.BatchNorm2d(4 * s),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2, stride=2),
            nn.Conv2d(4 * s, 4 * s, kernel_size=3, padding=1),
            nn.BatchNorm2d(4 * s),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2, stride=2),
            FCIL(h=fcil_size),
            nn.UpsamplingNearest2d(scale_factor=2),
            nn.ConvTranspose2d(4 * s, 4 * s, kernel_size=3, padding=1),
            nn.BatchNorm2d(4 * s),
            nn.ReLU(),
            nn.UpsamplingNearest2d(scale_factor=2),
            nn.ConvTranspose2d(4 * s, 2 * s, kernel_size=3, padding=1),
            nn.BatchNorm2d(2 * s),
            nn.ReLU(),
            nn.UpsamplingNearest2d(scale_factor=2),
            nn.ConvTranspose2d(2 * s, 1 * s, kernel_size=3, padding=1),
            nn.BatchNorm2d(1 * s),
            nn.ReLU(),
            nn.UpsamplingNearest2d(scale_factor=2),
            nn.ConvTranspose2d(
                1 * s, self.outsize[0], kernel_size=5, padding=2
            ),
            nn.Sigmoid(),
        )

    def calculate_padding_and_fcil_size(self):
        imsize = self.imsize
        p0 = 2 ** (int(np.ceil(np.log2(imsize[-2])))) - imsize[-2]
        p1 = 2 ** (int(np.ceil(np.log2(imsize[-1])))) - imsize[-1]
        padding = (p0, p1)
        fcil_size = (
            4 * ((imsize[1] + p0) // 16) * ((imsize[2] + p1) // 16) * self.s
        )
        return padding, fcil_size

    def forward(self, x):
        N = x.shape[0]
        out = self.layers(x)[..., : self.outsize[-2], : self.outsize[-1]]
        return out


class ConvolutionalDiscriminator(nn.Module):
    def __init__(self, imsize=(3, 64, 64), s=64):
        super(ConvolutionalDiscriminator, self).__init__()
        self.layers = nn.Sequential(
            nn.Conv2d(imsize[0], s, 4, 2, 1, bias=False),
            nn.LeakyReLU(0.2, inplace=False),
```

```
            nn.Conv2d(
                s, 2 * s, kernel_size=4, stride=2, padding=1, bias=False
            ),
            nn.BatchNorm2d(2 * s),
            nn.LeakyReLU(0.2, inplace=False),
            nn.Conv2d(
                2 * s, 4 * s, kernel_size=4, stride=2, padding=1, bias=False
            ),
            nn.BatchNorm2d(4 * s),
            nn.LeakyReLU(0.2, inplace=False),
            nn.Conv2d(
                4 * s, 8 * s, kernel_size=4, stride=2, padding=1, bias=False
            ),
            nn.BatchNorm2d(8 * s),
            nn.LeakyReLU(0.2, inplace=False),
            nn.Conv2d(
                8 * s, 1, kernel_size=4, stride=1, padding=0, bias=False
            ),
            nn.Sigmoid(),
        )

    def forward(self, x):
        output = self.layers(x)
        return output.view(-1)
```

A.4 Training and Testing Code for PRCGAN

```
import copy
import matplotlib.pyplot as plt
import numpy as np
import torch
import torch.nn as nn
import torch.optim as optim
import lpips as lp
import tqdm
import models


def poisson_noise(magn, alpha=1, device=None):
    """
    Add poisson noise to magnitudes.

    Parameters:
        magn: Batch of magnitudes.
        alpha: Level of noise.
        device: Device on which noise is stored (PyTorch device)."""
    intens = magn**2
    alpha_2 = alpha**2
    lmd = intens / alpha_2
    if device is not None:
        lmd[(lmd == 0)] = 1e-15
        intens_noise = (
            alpha_2 * torch.distributions.poisson.Poisson(lmd).sample()
        )
        magn_noise = torch.sqrt(intens_noise)
    else:
        intens_noise = alpha_2 * np.random.poisson(lmd, magn.shape)
        magn_noise = np.sqrt(intens_noise)
    return magn_noise
```

```
33
34
35  def calc_mags(x):
36      """
37      Calculated Fourier magnitudes for a batch of images x.
38
39      Paramters:
40          x: batch of images.
41      """
42      ft = torch.fft.fft2(x)
43      mags = torch.abs(ft)
44      return mags
45
46
47  def calc_meas(A, x):
48      """
49      Calculated Gaussian magnitudes for a batch of images x.
50
51      Paramters:
52          A: measurement matrix.
53          x: batch of images.
54      """
55      m = A.shape[1]
56      n = x.shape[-1] * x.shape[-2]
57      assert m <= n
58      N = x.shape[0]
59      C = x.shape[1]
60      compressed = (x.view(N, C, -1) @ A).view(N, C, m)
61      compressed = torch.abs(compressed)
62
63      compr_pad = nn.functional.pad(
64          compressed, (0, n - m), mode="constant", value=0
65      ).reshape(x.shape)
66
67      return compressed, compr_pad
68
69
70  def train_e2e(
71      model,
72      loss_fn,
73      optimizer,
74      device,
75      trainloader,
76      epochs,
77      val=None,
78      lpips_lmd=None,
79      A=None,
80      plot=False,
81      file=None,
82  ):
83      model = model.to(device=device)
84      history = []
85
86      loss_fn_vgg = lp.LPIPS(net="vgg").to(device)
87
88      for e in range(epochs):
89          pbar = tqdm.tqdm(trainloader)
90          for x in pbar:
91
92              model.train()
93
94              # move x to device and reduce to magnitude information in fourier space
95              x = x.to(device=device)
```

```
            if A is not None:
                _, x_in = calc_meas(A, x)
            else:
                x_in = calc_mags(x)

            # compute loss as pixel wise distance to original input
            x_pred = model(x_in)
            if lpips_lmd is not None:
                loss = torch.mean(
                    torch.mean(torch.abs(x - x_pred), dim=(1, 2, 3))
                    + lpips_lmd * loss_fn_vgg(x, x_pred)
                )
            else:
                loss = loss_fn(x_pred, x)
            history.append(loss.item())

            optimizer.zero_grad()
            loss.backward()
            optimizer.step()

            pbar.set_description(
                "epoch: {:3d} loss: {:3.4}".format(e + 1, loss.item())
            )

        if plot:
            plt.plot(history)
            plt.show()
        if file is not None:
            torch.save(model.state_dict(), file + "_{:03d}.sd".format(e + 1))
        if val is not None:
            x = next(iter(val)).to(device)
            x_rec = model(calc_mags(x))
            print(
                "SSIM:",
                ssim(x_rec.detach().cpu().numpy(), x.detach().cpu().numpy())[
                    0
                ],
            )

    return history


def train_cgan(
    gen,
    disc,
    g_opt,
    d_opt,
    device,
    trainloader,
    start_epoch=0,
    end_epoch=100,
    val=None,
    loss_fn="mae",
    lmd=1,
    lpips_lmd=None,
    plot=False,
    file=None,
):
    """
    Training algorithm for PRCGAN.

    Parameters:
        gen: Generator network (PyTorch model).
```

```
        disc: Discriminator network (PyTorch model).
        g_opt: Optimizer for generator network (PyTorch optimizer).
        d_opt: Optimizer for discriminator network (PyTorch optimizer).
        device: Device used for training (PyTorch device).
        trainloader: Dataloader for training set.
        start_epoch: Epoch at which to start.
        end_epoch: Epoch at which to end.
        val: Whether after model is evaluated after each epoch.
        loss_fn: Reconstruction loss. Can be "mae" or "mse".
        lmd: Weight for reconstruction loss.
        lpips_lmd: Weight for LPIPS loss. If None, LPIPS is not used.
        plot: Indicates whether intemedate reconstructions and loss history
              are shown.
        file: Path to checkpoints folder.
    """
    disc = disc.to(device=device)
    gen = gen.to(device=device)

    history_g = []
    history_d = []
    history_mse = []

    loss_fn_vgg = lp.LPIPS(net="vgg").to(device)

    if start_epoch > 0:
        assert file is not None
        disc.load_state_dict(
            torch.load(file + "_netD_{:03d}.sd".format(start_epoch))
        )
        gen.load_state_dict(
            torch.load(file + "_netG_{:03d}.sd".format(start_epoch))
        )

    re = False  # Helper variable to restart if loss is NaN.
    for e in range(start_epoch, end_epoch):
        pbar = tqdm.tqdm(trainloader)
        for x in pbar:

            x = x.to(device=device)
            mags = calc_mags(x)

            z = torch.randn(*x.shape).to(device)
            zm = torch.cat([z, mags], dim=1)

            gen.train()
            gen_output = gen(zm)

            xm = torch.cat([x, mags], dim=1)
            disc.train()
            disc_output_real = disc(xm)
            disc_output_generated = disc(torch.cat([gen_output, mags], dim=1))

            if loss_fn == "mse":
                rec_loss = torch.mean((x - gen_output) ** 2)
            elif loss_fn == "mae":
                rec_loss = torch.mean(torch.abs(x - gen_output))

            if lpips_lmd is not None:
                rec_loss = rec_loss + lpips_lmd * torch.mean(
                    loss_fn_vgg(x, gen_output)
                )

            g_loss = (
```

```
                -torch.mean(torch.log(disc_output_generated)) + lmd * rec_loss
            )

            history_g.append(g_loss.item())
            g_opt.zero_grad()
            g_loss.backward()
            g_opt.step()

            disc_output_real = disc(xm)
            disc_output_generated = disc(
                torch.cat([gen_output.detach(), mags], dim=1)
            )
            d_loss = -torch.mean(
                (
                    torch.log(disc_output_real)
                    + torch.log(1 - disc_output_generated)
                )
                / 2.0
            )
            history_d.append(d_loss.item())
            d_opt.zero_grad()
            d_loss.backward(retain_graph=True)

            d_opt.step()

            mse_loss = torch.mean((x - gen_output) ** 2)
            history_mse.append(mse_loss.detach().cpu().item())

            pbar.set_description(
                "epoch: {:3d} loss: {:3.4}".format(e + 1, rec_loss.item())
            )

            if torch.isnan(rec_loss):
                re = True
                break
        if re:
            break

        if plot:
            # Plotting code
            plt.plot(history_g[-10000:], label="Gen loss")
            plt.plot(history_d[-10000:], label="Disc loss")
            plt.plot(history_mse[-10000:], label="MSE")
            plt.legend()
            plt.show()

            test_outputs, test_images = test_generator(
                gen, device, dataloader["val"]
            )
            plot_grid(
                np.vstack(
                    [
                        test_images[:16],
                        test_outputs[:16],
                        test_images[16:32],
                        test_outputs[16:32],
                    ]
                ),
                figsize=(8, 8),
            )

            mse_val = np.mean((test_outputs - test_images) ** 2)
            print("Mean squared dist:", mse_val)
```

```
        if file is not None:
            # Save checkpoints
            torch.save(
                gen.state_dict(), file + "_netG_{:03d}.sd".format(e + 1)
            )
            torch.save(
                disc.state_dict(), file + "_netD_{:03d}.sd".format(e + 1)
            )
        if val is not None:
            # Validate model
            x = next(iter(val)).to(device)
            z = torch.randn(*x.shape).to(device)
            gen.eval()
            zm = torch.cat([z, calc_mags(x)], dim=1)
            x_rec = gen(zm)
            print(
                "SSIM:",
                ssim(x_rec.detach().cpu().numpy(), x.detach().cpu().numpy())[
                    0
                ],
            )
    if re:
        # Reset training
        gen = models.ConvNet(imsize=(6, 64, 64), outsize=(3, 64, 64), s=64)
        disc = models.ConvDiscriminator(imsize=(6, 64, 64), s=64)
        train_cgan(
            gen,
            disc,
            g_opt,
            d_opt,
            device,
            trainloader,
            start_epoch=e,
            end_epoch=end_epoch,
            val=val,
            lmd=lmd,
            lpips_lmd=lpips_lmd,
            plot=plot,
            file=file,
        )


def test_cgan(model, device, testloader, alpha=None, A=None):
    """
    Testing algorithm for PRCGAN.

    Parameters:
        model: Trained PyTorch model
        device: Indicates on which PyTorch device the model
        is executed
        testloader: Dataloader to obtain test data.
        alpha: Level of noise. If None, no noise is added.
        A: Measurement matrix for Gaussian phase retrieval. If None,
        Fourier measurements are considered.
    """
    model = model.to(device=device)
    model.eval()

    test_images = []
    outputs = []

    for i, data in enumerate(testloader):
```

```
        if i == 16:
            break
        z = torch.randn(*data.shape).to(device)
        data = data.to(device=device)

        if alpha is not None:
            x_in = poisson_noise(calc_mags(data), alpha, device)
        elif A is not None:
            _, x_in = calc_meas(A, data)
        else:
            x_in = calc_mags(data)

        net_input = torch.cat([z, x_in], dim=1)
        output = model(net_input)

        test_images.append(data.cpu().numpy())
        outputs.append(output.cpu().detach().numpy())

    return np.concatenate(outputs), np.concatenate(test_images)


def opt_prc(
    sample_in,
    model,
    device,
    mode="prc-l",
    lr1=1e-2,
    lr2=1e-6,
    max_steps=10000,
    A=None,
):
    """
    Optimization algorithms for PRCGAN-L and PRCGAN-W.

    Parameters:
        sample_in:
        model: Trained PyTorch model of generator.
        device: Indicates on which PyTorch device the model
        is executed
        mode: Can be "latent" or "weight" for latent and weight
              optimization, respectively.
        lr1: Learning rate for latent optimizer.
        lr2: Learning rate for weight optimizer.
        max_steps: Maximum number of iterations.
        A: Measurement matrix for Gaussian phase retrieval. If None,
        Fourier measurements are considered.
    """
    model.to(device)
    model.eval()

    z = torch.randn((sample_in.shape), requires_grad=True, device=device)

    state_dict = copy.deepcopy(model.state_dict())
    z_opt = optim.Adam([z], lr=lr1)
    gen_opt = optim.Adam(list(model.parameters()), lr=lr2)

    losses = []
    last_out = model(torch.cat([z, sample_in], dim=1))

    for i in tqdm.tqdm(range(max_steps)):
        if A is not None:
            _, x_in = calc_meas(A, last_out)
        else:
```

```
            x_in = calc_mags(last_out)

        mse_all = torch.mean((x_in - sample_in) ** 2, dim=(1, 2, 3))
        loss = torch.sum(mse_all)

        if mode == "latent":
            z_opt.zero_grad()
        elif mode == "weight":
            gen_opt.zero_grad()
        else:
            z_opt.zero_grad()
            gen_opt.zero_grad()
        loss.backward()
        if mode == "latent":
            z_opt.step()
        elif mode == "weight":
            gen_opt.step()
        else:
            z_opt.step()
            gen_opt.step()

        last_out = model(torch.cat([z, sample_in], dim=1))

        losses.append(loss.item())

    model.load_state_dict(state_dict)

    return last_out.detach(), losses, mse_all
```

A.5 Unrolled ER Algorithm

```
import torch
import torchvision
import matplotlib.pyplot as plt
import numpy as np
from data import *
from util import *
from imageio import imsave
from skimage.registration import phase_cross_correlation

# Load data
x_train, x_test = load_CIFAR10()
x_train = x_train[0:100]
x_val = x_test[-100:]
x_test = x_test[0:1000]

s = (64, 64)  # Size of magnitude
si = (32, 32)  # Size of image

def ER_reference(y, u, imsize=(32, 32), s=(64, 64), steps=100):
    """
    Implements the error-reduction algorithms with a reference image.

    Parameters:
        y: magnitude measurement.
        u: known reference image.
        imsize: size of image.
        s: size of magnitude.
        steps: number of iterations.
```

```
    """

    # Initialize reconstruction with zeros.
    x = torch.zeros(len(y), 1, *imsize)

    for i in range(steps):
        # Fourier domain constraints
        intermed = torch.fft.ifft2(phase(x + u, s=s) * y)[:, :, :32, :32].real

        # Image domain constraints
        x = intermed - u
        x[x < 0] = 0  # Enforce Positivity
    return x

if __name__ == "__main__":
    train_generator = torch.utils.data.DataLoader(
        x_train, batch_size=10, shuffle=True
    )
    u = torch.rand(32, 32, requires_grad=True)
    opt = torch.optim.Adam([u], lr=0.01)
    losses = []

    # Training loop
    for epoch in range(10):
        print(epoch)
        for i, batch in enumerate(train_generator):
            # Calculate measurements of training batch
            y = magnitude(batch + u, s=s, norm="backward")

            # Reconstruct batch using current reference
            x_rec = ER_reference(y, u, imsize=(32, 32), s=s, steps=15)

            # Calculate loss and back-propagate through ER algorithm
            loss = torch.mean(torch.square(x_rec - batch))
            losses.append(loss.item())
            loss.backward()
            opt.step()

            # Clip reference to [0,1]
            u.data[u < 0] = 0.0
            u.data[u > 1] = 1.0

    # Calulcate measurements
    y = magnitude(x_test + u, s=s, norm="backward")

    # Reconstruct test images
    rec = ER_reference(y, u, imsize=(32, 32), s=s, steps=500).detach()

    def cross_correlation(moving, fixed):
        """
        Registers two images using cross-correlation.

        Parameters:
            moving: moving image.
            fixed: fixed image.
        """
        if moving.shape[-1] == 3:
            moving_gray = rgb2gray(moving)
            fixed_gray = rgb2gray(fixed)
        elif moving.shape[-1] == 1:
            moving_gray = moving[..., 0]
            fixed_gray = fixed[..., 0]
        else:
```

```
            print("Image channel Error!")

        shift, error, diffphase = phase_cross_correlation(
            moving_gray, fixed_gray
        )
        out = np.roll(moving, -np.array(shift).astype(np.int), axis=(0, 1))
        return out, error

    def register_cross_correlation(predicted_images, true_images, torch=True):
        """
        Registers two batches of images using cross-correlation.

        Parameters:
            moving: moving image.
            fixed: fixed image.
            torch: indicates whether input batches are torch.tensors.
        """
        pred_reg = np.empty(
            predicted_images.shape, dtype=predicted_images.dtype
        )

        for i in range(len(true_images)):
            if torch:
                true_image = true_images[i].transpose(1, 2, 0)
                predicted_image = predicted_images[i].transpose(1, 2, 0)
            else:
                true_image = true_images[i]
                predicted_image = predicted_images[i]

            shift_predict, shift_error = cross_correlation(
                predicted_image, true_image
            )
            rotshift_predict, rotshift_error = cross_correlation(
                np.rot90(predicted_image, k=2, axes=(0, 1)), true_image
            )

            if torch:
                pred_reg[i] = (
                    shift_predict.transpose(2, 0, 1)
                    if shift_error <= rotshift_error
                    else rotshift_predict.transpose(2, 0, 1)
                )
            else:
                pred_reg[i] = (
                    shift_predict
                    if shift_error <= rotshift_error
                    else rotshift_predict
                )

        return pred_reg

    # Calculate registered MSE for test images.
    x_rec_reg = register_cross_correlation(
        np.array(rec),
        x_test.numpy(),
    )
    mse_vals = [
        mse(torch.tensor(x_rec_reg[i, None]), x_test[i, None]).item()
        for i in range(len(x_rec_reg))
    ]
    print("MSE: %f +/- %f" % (np.mean(mse_vals), np.std(mse_vals)))
```

B Supplementary Materials

In the following we provide some supplementary materials for Chapter 6.

B.1 VAE Architecture

For the MNIST, FMNIST and EMNIST dataset we use a variational autoencoder (VAE) as a generator. The encoder and the decoder each consist of three layers with ReLU activation function, where each layer had 500 hidden units. The latent space was chosen 100-dimensional. It was trained using a Bernoulli-likelihood and Adam with learning rate 10^{-3} for 100 epochs.

B.2 Hyperparameters

Table 2: Hyperparameters for PRILO and PRILO-MII on the MNIST and EMNIST datasets.

Repetitions		Variable	Steps	ℓ_1-radius
1	Initial optimization	z_0	150	100
1	Forward optimization	z_1	150	50
	Back-projection	z_0	-	-
	Refinement	z_0	-	-
9	Forward optimization	z_2	300	100
	Back-projection	z_0	-	-
	Refinement	z_0	-	-

Table 3: Hyperparameters for PRILO and PRILO-MII on the Fashion-MNIST dataset.

Repetitions		Variable	Steps	ℓ_1-radius
1	Initial optimization	z_0	100	100
1	Forward optimization	z_1	100	10
	Back-projection	z_0	-	-
	Refinement	z_0	-	-
4	Forward optimization	z_2	200	40
	Back-projection	z_0	-	-
	Refinement	z_0	-	-

Repetitions		Variable	Steps	ℓ_1-radius (z_i)	ℓ_1-radius (z_0)
1	Initial optimization	z_0	100	1000	1000
1	Forward optimization	z_1	100	200	100
	Back-projection	z_0	100	1000	1000
	Refinement	z_0	100	1000	1000
1	Forward optimization	z_2	100	200	100
	Back-projection	z_0	100	1000	1000
	Refinement	z_0	100	1000	1000

Table 4: Hyperparameters for PRILO (based on StyleGAN) on the CelebA dataset. Note that z_0 is passed into each layer. That is why we have to apply ℓ_1-regularization also for z_0 when optimizing the latter layers.

Repetitions		Variable	Steps	ℓ_1-radius (z_i)	ℓ_1-radius (z_0)
1	Initial optimization	z_0	2000	1000	1000
3	Forward optimization	z_1	400	100	100
	Back-projection	z_0	100	1000	1000
	Refinement	z_0	2000	1000	1000

Table 5: Hyperparameters for PRILO-MII and PRILO-LI (based on StyleGAN) on the CelebA dataset. Note that z_0 is passed into each layer. That is why we have to apply ℓ_1-regularization also for z_0 when optimizing the latter layers.