TECHNISCHE
UNIVERSITÄT
WIEN
Vienna | Austria

DIPLOMARBEIT

Parameter estimation in disordered media

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Technische Physik

eingereicht von

Günther Hackl
Matrikelnummer 01525544

ausgeführt am Institut für Theoretische Physik
der Fakultät für Physik der Technischen Universität Wien

Betreuung:
Dipl.-Ing. Dr.techn. Lukas Michael Rachbauer,
Univ.Prof. Dipl.-Ing. Dr.techn. Stefan Rotter

Wien, May 11, 2025

(Unterschrift Verfasser/in) (Unterschrift Betreuer/in)

Contents

1. Introduction

Neural networks, a subcategory of machine learning, were first conceived in the 1960s [1]. Meanwhile, they have spread to all sciences, where they have been used to investigate solutions to complex tasks, such as predicting protein structures [2] or properties of new materials [3] as well as aiding in medical image analysis [4]. In physics, one of these complex tasks is imaging through a scattering medium, in which neural networks showed promising results [5]. They have even been shown to perform well on diffusers they had not been trained on, suggesting some generality to the scattering process.

One general problem with neural networks is that when they stop improving during their training process, it is hard to say why the improvement stopped [6]. One reason could be that the capabilities of the model are insufficient and further work may lead to better results. Examples for this are the classical benchmarks used by the machine learning community like classifying the MNIST handwritten digits [7] or the ImageNet contest of image classification started initially in 2012 [8], where new model types can be tested. Massive resources are invested nowadays only to gain another 0.5% in accuracy [9]. Another possible explanation is that the data itself has an inherent limit. Such fundamental limits are common in physical systems, one well-known example being the Abbe limit, which defines the maximum resolution of a light microscope based on the wavelength of light used. This means that if a fundamental limit can be found within the data, it could be a unique benchmark that sets the ultimate bound on a network's performance, which is independent of how many resources are invested.

In this thesis, we will investigate a physical system involving a target hidden behind a diffusing slab. A limit for this system is given by the concept of Fisher information, and the corresponding Cramér Rao Lower Bound [10], which can be reached when the maximum information about a parameter is extracted by a measurement device utilizing wavefront shaping [11]. The subsequent derivation of a Fisher information operator, which is Hermitian for this system, suggests that this ultimate limit is an observable property of the system itself. As the Fisher information is a statistical concept not only bound to speckle patterns, this fundamental limit could also find many other use-cases.

For this purpose, we will simulate this system and use the Cramér Rao Lower Bound to answer whether this limit is suitable to be used as a benchmark for the performance of machine learning models.

To get a starting point for the simulations, we analyze experimentally gathered data

[12], so the machine learning model developed here can also be tested on the experimental data, and show an approach how to simulate fully developed speckles, which will be easier to work with than the experimental data itself. We show how to analytically derive an expression to calculate the Fisher information for fully developed speckles without correlations between pixels and give approaches on how correlations can be removed under certain circumstances and how to estimate the influence of noise on the quality of the fisher information calculation. The much more complex task of calculating the Fisher information when correlations are present is beyond the scope of this thesis and will, among other things, be treated in [12]. Finally, we investigate approaches to create a model capable of reaching the limit based on the network used in [5] and explicitly compare its performance against the calculated Cramér Rao Lower Bound.

2. Simulating Speckle Patterns

Before we can train an artificial neural network, we need data that can be used in the training process. In this work, we primarily rely on simulated speckle patterns derived from parameters extracted from the experiment as our data. Compared to the experimental data used in [12], these simulations are easier to control and the data is easier to obtain, but should still provide a valid foundation for developing a network architecture that could later be applied to the more complex experimental datasets. The simulated data can also be analyzed much easier in the Fisher Information step than the experimental data, as the properties of the probability density functions are well understood, and we can eliminate influences, discussed in a later section, that would need to be accounted for otherwise. The goal of this chapter is to analyze experimental data and find an approach to simulate it similarly, but in a simplified way, so we can use it in the following chapters. For these purposes a brief overview of speckle patterns is provided, like the intensity distributions of one point across many similar diffusers and how those change in the presence of ballistic photons, as well as the spatial correlation between two points in a speckle. These distributions are needed to analyze the data from the experimental setup that will be described in section 2.2 and later in section 3 to calculate the Cramér Rao Lower Bound. After obtaining parameters that can describe the experimental data to some extent, we will select a method to that is reasonable for us to generate the dataset necessary throughout this thesis.

2.1. Speckle Patterns Overview

Speckle patterns are an inherent phenomenon of coherent waves and can therefore be observed in a wide range of applications such as synthetic aperture radar (SAR) imagery, medical ultrasound imagery, Optical Coherence Tomography and optical Lithography [13, 14]. In the 1960s, when HeNe-Lasers became commercially available, laser speckles were first observed in optics. They occur whenever a laser is reflected off of a surface that is rough compared to the wavelength of the light. The reason for this is that each facet of the rough surface will reflect the light with a different phase contribution, and these reflected light waves will interfere with one another to create the speckle pattern [14]. For diffusers, as in the experiment treated in this chapter, the different optical paths within the diffuser for each part of the incident wave similarly enables the interference to produce speckle patterns.

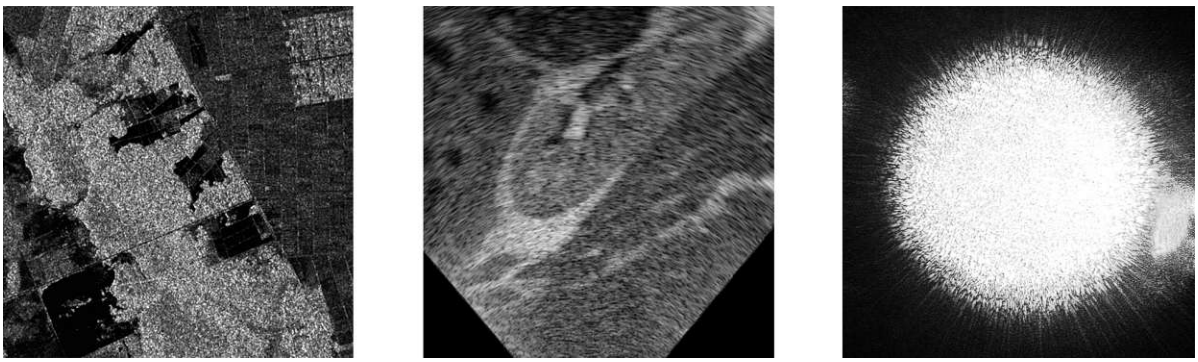

Figure 2.1: Images showing the speckle phenomenon: SAR land cover photo, ultrasound image and laser reflected by paper. Credits: mundialis ecosystem restoration, Fernando C. Monteiro and jurvetson [flickr]

Depending on the use case, speckle can be friend or foe [13]. In the imaging applications above, it is often classified as "speckle noise" and can be especially inhibiting as the Signal-to-Noise ratio for a fully developed speckle is one [14], meaning the intensity of the speckle noise is as large as the signal one would like to observe, which is why many methods to suppress speckle have been devised. In other applications, the statistics of speckle can be exploited to gain insight into the material that produced it and are therefore of high interest in the field of nondestructive testing [13]. For the experiment treated here, where the position of a scatterer behind a diffuser should be estimated, the maximum information that can be used may be coming from both regimes, where the

direct reflection of the scatterer will be obstructed by speckle noise and some information of the scatterer may also be derived from the speckle pattern directly.

2.1.1. Expected First Order Intensity Statistics

A more comprehensive view of speckle statistics can be found in the works of Joseph Goodman [14, 15]. Here we will restrict ourselves to a brief overview of the relevant properties. The first order statistics for speckle concern only one point in space and time, where all the contributions of the scattered light with randomly distributed phases interfere. In our simulation, this would be one pixel of one image. In this context, one often speaks of phasor-sums where a phasor is defined as a combination of an amplitude a_n and a complex phase θ_n. The complex amplitude of phasor-sum of N phasors is then given by

$$\boldsymbol{A} = Ae^{i\theta} = \frac{1}{\sqrt{N}} \sum_{n=1}^{N} a_n e^{i\theta_n} \tag{1}$$

where $1/\sqrt{N}$ normalizes the sum, to conserve finite second statistical moments. To obtain the intensity, which is the measure typically accessible in optical experiments, the amplitude has to be squared. For the case of a linearly polarized laser, we do not have to account for multiple polarizations and simply use

$$I = |\boldsymbol{A}|^2 \tag{2}$$

While we aim to simulate fully developed speckles, in the analysis of the experimental data we will find that it consists of partially developed speckles. In the following, we will discuss the statistics of fully developed speckles and one case that leads to partially developed speckles.

Fully developed speckles can be observed if the three following assumptions are fulfilled [14]:

1. a_n and θ_n are statistically independent from a_m and θ_m if $n \neq m$
2. a_n and θ_n are statistically independent from each other
3. θ_n is uniformly distributed in a range of $(-\pi, \pi)$

where A is the mean amplitude of a large phasor-sum and σ^2 its variance.

From these assumptions, it can be found that the amplitude of the phasor-sum follows Rayleigh statistics:

$$p(A) = \frac{A}{\sigma^2} \exp\left(-\frac{A^2}{2\sigma^2}\right) \tag{3}$$

Using (2) to transform the probability density (derivation can be found in [14]), it can be found that intensity follows an exponential distribution:

$$p(I) = \frac{1}{2\sigma^2} \exp\left(-\frac{I}{2\sigma^2}\right) = \frac{1}{\bar{I}} \exp\left(-\frac{I}{\bar{I}}\right) \tag{4}$$

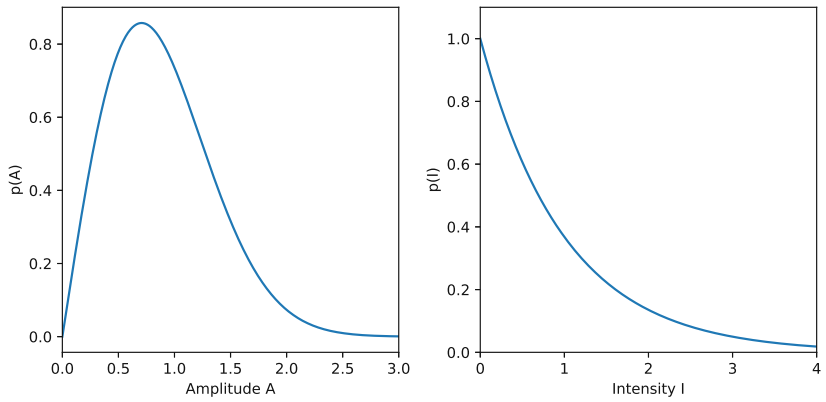

Figure 2.2: The expected statistics for a fully developed speckle pattern show Rayleigh distribution for amplitude and an exponential distribution for intensity.

The first and second statistical moments, mean and variance, of the exponential distribution are given by

$$\bar{I} = \langle I \rangle = 2\sigma^2 = \bar{I} \tag{5}$$

$$\sigma_I^2 = \langle I^2 \rangle - \langle I \rangle^2 = \bar{I}^2 \tag{6}$$

An important metric of speckle patterns is contrast, as it compares the fluctuations in the speckle intensity with the average intensity and is therefore directly related to Signal-to-Noise-Ration (SNR). They are defined as

$$C = \frac{\sigma_I}{\bar{I}} = \frac{1}{SNR} \tag{7}$$

For the fully developed speckle, we can directly see that this expression is equal to 1, showing that the contrast and the noise are at their maximum for fully developed speckles.

Speckles with one constant phasor can be used to model ballistic photons in the speckle, as we will see later. They violate the 3rd of the assumptions that need to be fulfilled for a fully developed speckle, as the phases are no longer uniformly distributed. Without loss of generality, the constant phasor can be chosen to just lie on the real axis, which shifts the possible values for the random phasor sum along this axis and limits the distribution of the phase which can be seen in figure 2.3

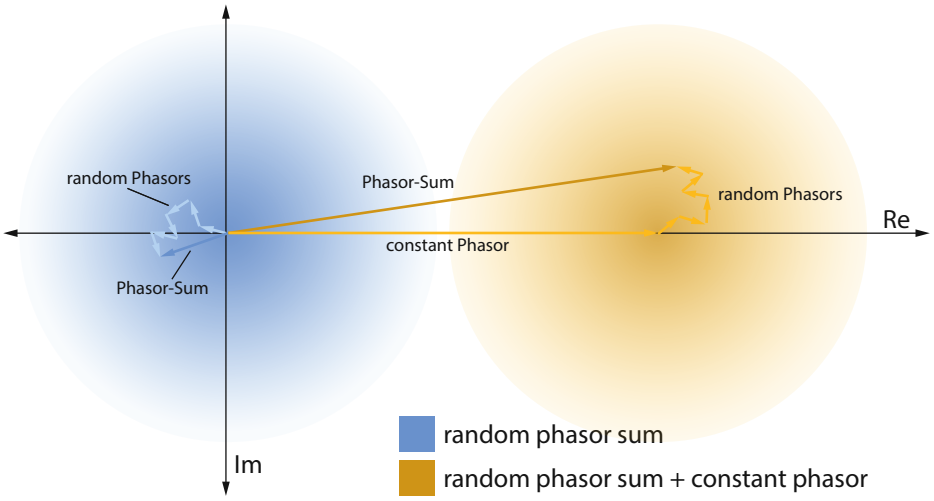

Figure 2.3: The complex amplitudes of the random phasor sum are distributed equally around the origin, showing that the phase θ is equally distributed along $[0, 2\pi]$, while adding a constant phasor restricts the phase.

The derivation of the intensity can be found in [14] and results in a *modified Rician distribution*:

$$p_I(I) = \frac{1}{\bar{I}_n} \exp\left\{-\left(\frac{I}{\bar{I}_n} + r\right)\right\} I_0\left(2\sqrt{\frac{I}{\bar{I}_n} r}\right) \tag{8}$$

where $\bar{I}_n$ is the average intensity of one phasor in the random phasor sum, $r = I_c/\bar{I}_n$

is the beam ratio of the intensity of the constant phasor and the average intensity of one of the random phasors and I_0 is the modified Bessel-function of the first kind. The first and second statistical moments are then given by

$$\bar{I} = \langle I \rangle = (1 + r)\bar{I_n} = I_c + \bar{I_n} \tag{9}$$

$$\sigma_I = \langle I^2 \rangle - \langle I \rangle^2 = \bar{I_n}\sqrt{1 + 2r} \tag{10}$$

which leads to a contrast that is generally smaller than 1 and therefore an SNR typically greater than 1, meaning the speckle noise is less pronounced. It is given by

$$C = \frac{\sigma_I}{\bar{I}} = \frac{\sqrt{1 + 2r}}{1 + r} \tag{11}$$

We can also immediately see that for $I_c \to 0$ the beam ratio r disappears and all the equations simplify to those of the fully developed speckle (as the Bessel function $I_0(0)$ also equals 1) as shown in 2.4.

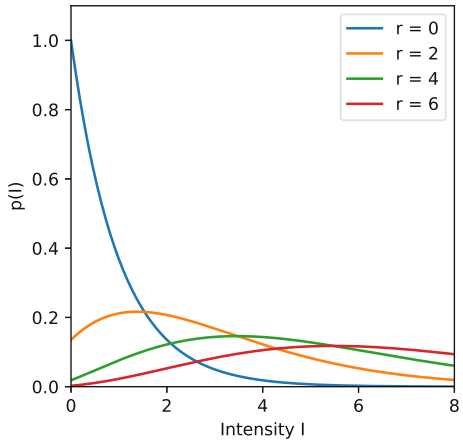

Figure 2.4: The intensity distribution expected for phasor sums with an added constant phasor is the Rician distribution, examples here for different beam ratios r. For a beam ratio, $r = 0$ it shows the expected collapse to the exponential distribution for a phasor sum without a constant phasor.

2.2. Analyzing Experimental Data

In the experimental setup used in [12], the main goal is to measure speckle patterns produced by a scatterer behind a diffuser. The scatterer can be moved along the x-

axis, which will be the parameter we want the artificial neural network to estimate. In this section we will extract some statistical parameters from these datasets to obtain a starting point for the simulations, and we will also discuss some of the differences to the experiment explicitly and some more generally.

2.2.1. Experimental Setup and Data

Figure 2.5 shows a sketch of the experimental setup, with some optical components for controlling the laser beam being emitted for simplicity:

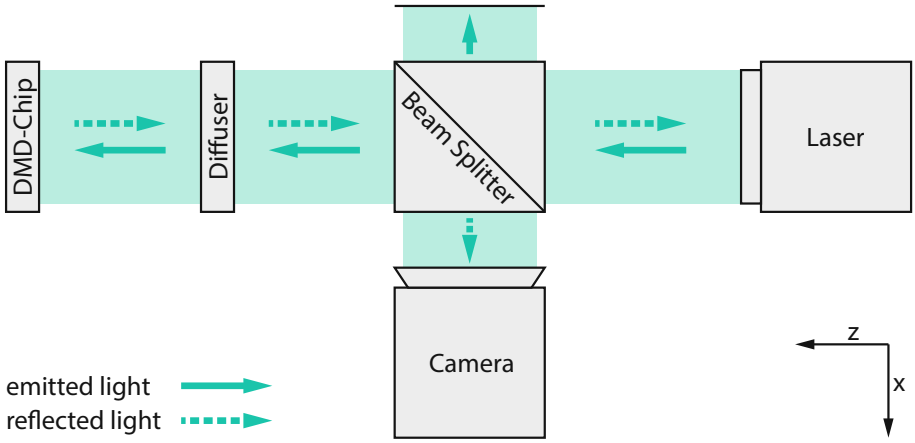

Figure 2.5: Illustration of the experimental setup (with some optical elements controlling the laser light omitted): The light emitted by the laser passes through a diffuser, consisting of TiO_2 particles suspended in glycerol, and is reflected partially on a DMD-chip, which displays a square that can be moved along the x-axis to mimic a scatterer behind the diffuser. The reflected light is then directed into the camera by a beam splitter, after passing through the diffuser

The laser beam passes through the diffuser, which is made of TiO_2 suspended in a glycerol solution that is constantly moved by a pump to guarantee a short correlation time, which allows for gathering speckle patterns fast, while also guaranteeing that two consecutive patterns are not correlated. Then the DMD-Chip is illuminated by the light passing through the diffuser. DMD-chips are arrays of tiny mirrors that can be tilted. In the case of the experiment, they either reflect straight back (activated) or out of the setup (deactivated). After the reflection, the light passes the diffuser a second time and a beam splitter is used to direct the exiting light towards a camera. The concentration of TiO_2 in the diffuser can be varied to effectively change the mean free path. In our

setup, the DMD-Chip displays a square that can be moved along the x-axis, which mimics a scatterer behind the diffuser medium. An example from the data collected by Ilya Starshynov, showing an image and the intensity statistics for the position of this scatterer, where we expect scattered light from the diffuser and unscattered light coming directly from the mirrors on the DMD-Chip, can be seen in figure 2.6.

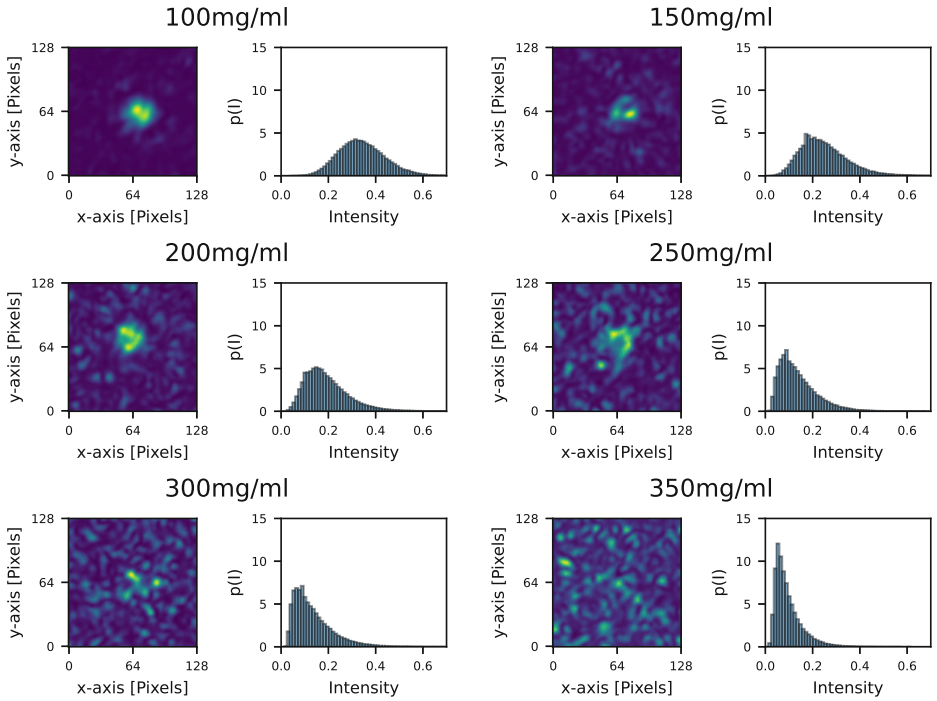

Figure 2.6: Example speckles for different TiO_2 concentration and the intensity statistics for spots in the middle of the frame, where we expect both light scattered by the diffuser and unscattered light coming directly from the DMD-Chip mirrors. For lower concentrations these cannot be modeled with the exponential distribution.

As we can see in 2.6, the intensity statistics of the experiment do not match an exponential distribution as would be expected for a fully developed speckle, which means we are looking at a partially developed speckle.

2.2.2. Ballistic Photons as possible Explanation for Deviations

We can see the scatterer quite clearly in the sample with a low TiO_2 concentration, and the optical thickness is also reported to be in a range where not all photons are expected to scatter in the diffuser. The photons that are not scattered while traveling

through the diffuser are called ballistic photons. In line with our earlier discussion, these ballistic photons can be modeled as an additional constant phasor as for the duration of the measurement (one picture taken) the amplitude and phase of these photons reflected from the scatterer is not influenced by the diffuser and therefore constant in time. Additionally, the presence of the ballistic photons may be important for our success in estimating the scatterer position across many diffusers, as in [16] it is theorized that the generalizeability to unseen statistically similar diffusers of deep neural networks depends solely on ballistic photons however small their contribution, which we will touch on a bit more in the conclusion of the thesis.

From the statistical moments for the *modified Rician distribution* which we can measure in the provided dataset we can solve for the underlying parameters of the distribution for each pixel and obtain

$$\bar{I_n} = \bar{I} \pm \sqrt{\bar{I}^2 - \sigma_I^2} \tag{12}$$

$$r = \frac{\bar{I}}{\bar{I_n}} - 1 \tag{13}$$

where we can discard the solution $\bar{I_n} = \bar{I} + \sqrt{\bar{I}^2 - \sigma_I^2}$ immediately as non-physical, as it would lead to a negative beam ratio, which would in turn mean that the intensity of the constant phasor would have to be negative. The parameters can then be used to show how well the model fits the experimental data and are visualized in figure 2.7.

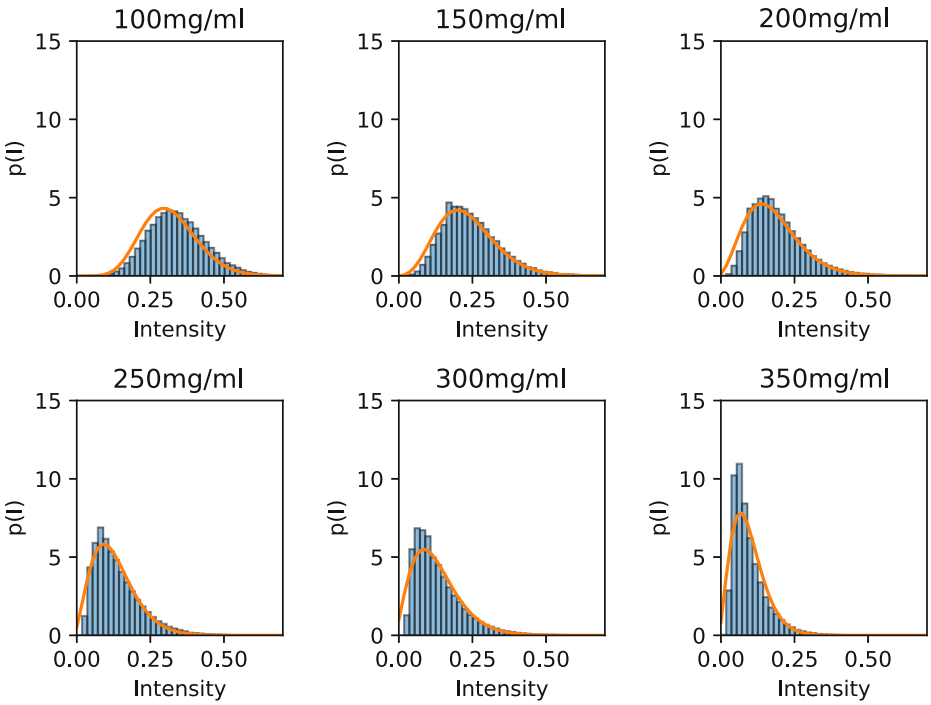

Figure 2.7: Fitted modified Rician statistics for different TiO_2 concentrations show good results for lower concentrations.

As we can see, for lower concentrations of TiO_2 the model fits remarkably well, but for higher concentrations, as the length of the known phasor decreases, we can see that deviations of the histogram from this model increase. Especially for the highest concentration, the intensity distribution can be better fit with an exponential-Gauss mixture. While we expect an exponential distribution in the limit of a disappearing known phasor, the Gaussian part is not captured by only this model. This breakdown can also be observed in the background-pixels of all concentrations as we expect very few ballistic photons, and therefore a vanishingly small known phasor in this region as well.

2.2.3. Spatial parameters

So far we have only inspected the statistics of one pixel, but in the samples we can see that some correlations also exist between neighboring pixels. Goodman [14] shows that this speckle grain size is determined by a pupil or an aperture. A measure for the size of the correlation is the Full-Width-Half-Maximum (FWHM) of the speckle pattern's

auto-correlation function. For the experimental data, we found that the speckle grains, measured from the autocorrelation function in parts of the pattern where they are not distorted by ballistic photons, all show a FWHM of around 8.5 pixels, regardless of the TiO_2 concentrations and therefore stem from the optical setup and not the diffuser.

In the simulations, the correlations between neighboring pixels will be achieved via a band-pass filter, which has only a finite amount of values and will therefore be chosen so that the auto-correlation function of the simulation matches that of the experimental data as closely as possible.

2.2.4. Additional unaccounted factors

The above model is just one correction that can be made to more accurately describe speckles encountered in the real world, and while it may work in some cases, other corrections can not be discarded. It was briefly attempted to take those into account to possibly explain the deviation at low intensities, but while incorporating them at the simulation level would be fairly easy (e.g. adding some Gaussian distributed noise to each pixel), getting a complete probability density to attempt to extract the statistical parameters from the data proved too difficult to be treated in this work. A list of some notable candidates that would introduce a Gaussian part into the statistics is found below, a more comprehensive list can be found in [14, 15]:

Image noise is one aspect that can be expected to be encountered in experimental imaging. Amplifier noise showing a Gaussian distribution would be one possible candidate to explain the deviations. Shot noise following a Poisson distribution, which would approximate a Gaussian distribution for many photons, is a candidate as well [17]. However, it could not be determined how strong the noise would have to be so the observed behavior of one pixel occurs, and whether this would still fit the experimental data.

Unfocused images are treated very thoroughly in [15] and adding this one aspect to fully developed speckles already results in a fairly complicated probability density, although a uniform Gaussian blur with a very small radius over each speckle could also explain some of the statistics seen.

Image distortion could also be a potential issue, as there are quite a few lenses in the experimental setup, lens errors or perspective shifts should be avoided, not only because

they have an effect on the probability density function, but because any distortions in the lenses also break the translation invariance of the scatterer, as the scatterer is distorted differently for every position. This would have to be accounted for in the Fisher information calculation to get the correct Cramér Rao bound.

Composite Speckles occur when the creation of one speckle is driven by another speckle [14]. In the experiment, we expect that the DMD-chip is already illuminated by a speckle-pattern, as it passed through the diffuser before. When the reflected light then passes through the diffuser again, we expect a composite speckle, also called speckled speckle. This could have considerable impact on the statistics, especially for the data where we see less ballistic photons.

2.3. Simulating Speckle Patterns

After obtaining the statistical parameters about the system, we can simulate the experimental data. There are two broad categories of approaches for the simulation. One would be a non-statistical approach of solving actual systems, and the other one more suited for the problem is that of statistical approaches.

2.3.1. Problem with non-statistical Approaches

In theory, one could model one realization of the diffuser exactly in a simulation, by randomly placing TiO_2 scatterers in a glycerol solution with a known refractive index, situated between the walls of the container holding the solution and the configurable mirror behind the container. One realization of just the diffuser is illustrated in 2.8. If we can assume the wall and glycerol solution are both homogeneous, the system with the scatterers can be approximated as piece-wise homogeneous in dielectric permittivity and solved by solving the Helmholtz equation [18]:

$$\left\{\Delta - \frac{\omega^2}{c^2}[1 - \epsilon(\boldsymbol{r})] + \frac{\omega^2}{c^2}\right\} \boldsymbol{E}_{\boldsymbol{\omega}}(\boldsymbol{r}) = \mathbf{0} \tag{14}$$

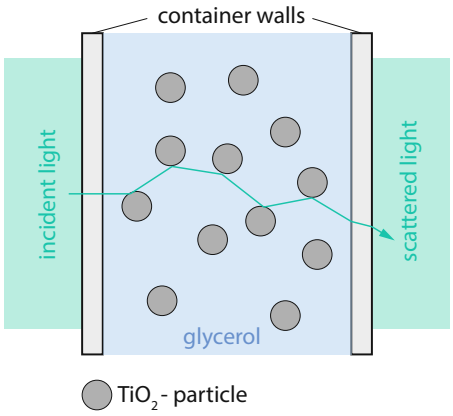

Figure 2.8: Illustration of one possible diffuser realization, where the diffuser consists of TiO_2 particles suspended in glycerol.

To solve this system, numerical methods have to be employed, such as finite elements or a Greens function approach. However, just solving one system would take considerable time. These approaches would be somewhat feasible in two dimensions, but the computation time vastly increases by an added third dimension, making it unreasonable to simulate many realizations of the experiment, as for every new diffuser realization the system would have to be solved again. To obtain accurate statistics and training data, tens to hundreds of thousands of systems would need to be solved, which is impossible in any reasonable time frame. So if the exact description is out of reach, and we are only interested in the statistics anyway, it makes sense to start with a statistical approach in the first place [19].

2.3.2. Simulating fully developed speckles

With the experimental data analyzed and the approach chosen, we can now start simulating data. Starting points for the code for the simulation can be found in [14] and [20], however, the whole process will be described in the following. At first, for simulating fully developed speckles, we will take the mean intensity as our base for generating a phasor sum for each pixel in the speckle pattern we want to create. This leads to a speckle pattern where every speckle is the size of one pixel, and they are not correlated between pixels. In the resulting intensity statistics, we also see what we expect for a fully developed speckle, as seen in figure 2.9

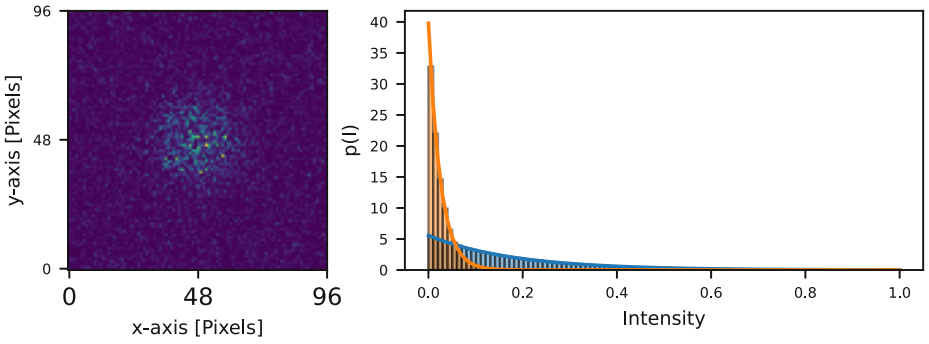

Figure 2.9: A speckle pattern without correlations simulated by sampling from the underlying exponential distributions defined by the desired mean intensity for each pixel (left) and the statistics measured for a pixel in the middle of the frame, where the scatterer is positioned (blue) and a pixel for the background, where no part of the scatterer is positioned (orange), with both showing a good fit to an exponential distribution

To generate a speckle with correlated pixels from the uncorrelated speckle, we use a bandpass to simulate the aperture. In the code, this can be achieved by Fourier transforming the image and mask all values outside the center circle to zero. The radius of the circle has an inverse correlation to the speckle size, meaning the bigger the circle, the smaller the speckles. If the circle is bigger than the image, it naturally gives speckles of the size of one pixel again. The process to achieve the bandpass is sketched in figure 2.10.

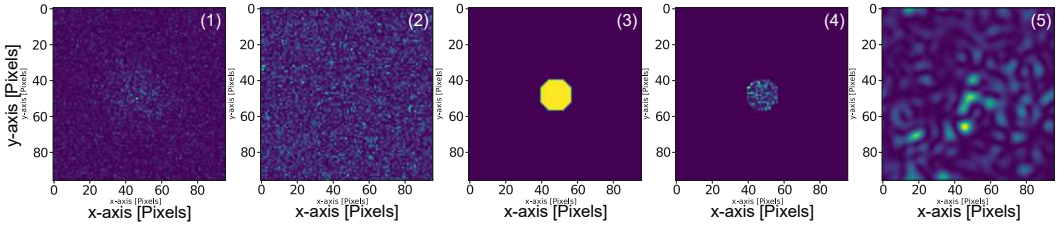

Figure 2.10: Process of going from uncorrelated speckle (1) to Fourier transform (2) and via mask (3) to masked Fourier (4) to the correlated speckle (5). Note: the images from Fourier space are showing the squared complex values.

This removes higher frequencies that lead to speckles that are only pixel sized and increasing the speckle size by smearing the complex phasor sums out over more pixels and in consequence creating correlations between the pixels. For the case of fully developed speckles, this process also leaves the underlying statistics for every pixel untouched. The

result can be seen in figure 2.11, but we can still make out a visible difference to the experimental data ssn in figure 2.6.

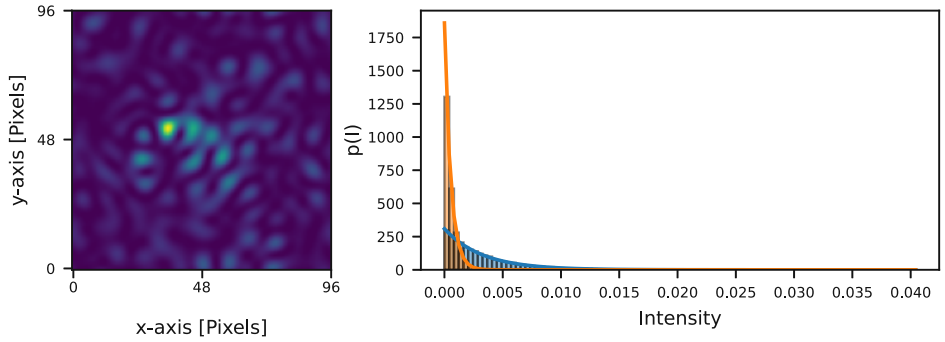

Figure 2.11: Adding correlations to fully developed speckles via a bandpass filter also leads to exponential distributions for a pixel in the middle of the frame, where the scatterer is positioned (blue) and a pixel for the background, where no part of the scatterer is positioned (orange)

A code snippet for generating one of these speckle patterns can be found in the appendix, using NumPy [21] for generating random phasor sums and general matrix manipulation and SciPy [22] for the Fourier transform. The code for generating hundreds of thousands of speckle patterns has to be structured a bit differently to make full use of the packages and stay within the given resource limits and a reasonable time frame.

2.3.3. Problems with simulating experimental data

As we saw in the analysis, the experimental data can be better fitted when including ballistic photons, which are modeled with one constant phasor. From the experimental data, we can try to calculate the length of this constant phasor and the average length of the other phasors for each pixel. Then we can simulate an uncorrelated speckle pattern just as we did before with the one small adjustment that one element of the phasor sum is set to the constant value before adding everything. Generating many speckle patterns, we can see that the expected statistics arise in the sample in figure 2.12.

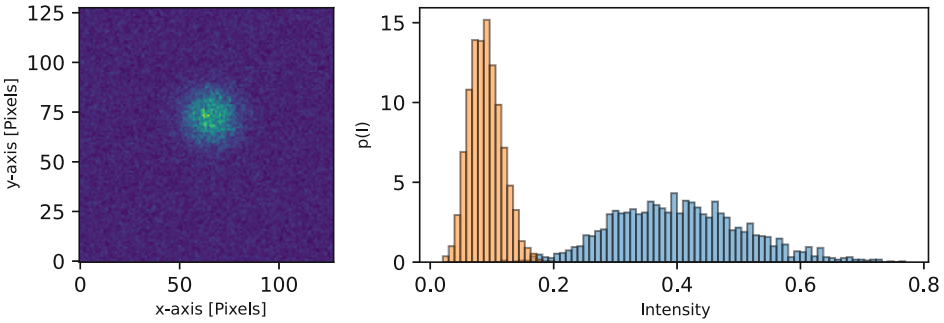

Figure 2.12: A simulated speckle pattern with ballistic photons incorporated, but without correlations (left). The statistics measured for a pixel in the middle of the frame, where the scatterer is positioned (blue) and a pixel for the background (orange) show distributions that can be fitted with a Rician distribution as shown earlier.

The main problem arises when trying to introduce correlations again. For the phasor sums with the constant phasor it can be observed that the statistics per pixel change when applying the bandpass. This means the intensities for the constant phasor and the random phasors can not be easily extracted from the experimental data in the ways we tried. The differences in the statistics resulting from applying the bandpass can be seen in the plot in figure 2.13

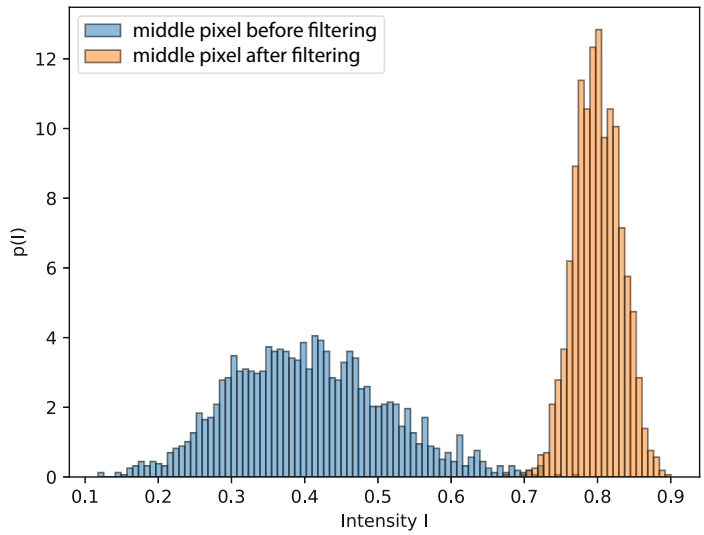

Figure 2.13: The statistics of one pixel for the speckle patterns with ballistic photons change significantly when applying the band-pass filter to introduce correlations between neighboring pixels (orange) compared to the unfiltered statistics (blue), meaning the parameters extracted from the experimental data can not be used as a starting point for simulations.

Additionally, for the samples with higher optical thickness, we have seen in figure 2.7 that the model with just ballistic photons fits not as well, as we can see quite a difference between the expected Rician distribution and the observed distribution. For these higher optical thicknesses, we expect fewer ballistic photons, and the distribution to trend to an exponential distribution, which would be non-zero at zero intensity. As we can see in figure 2.14 the distribution trends to zero for zero intensity. This points to some unaccounted factors, we discussed in section 2.2.4, like image noise with Gaussian form or a slightly out of focus image, where the convolution between an exponential distribution and a Gaussian distribution could explain the intensity trending toward zero.

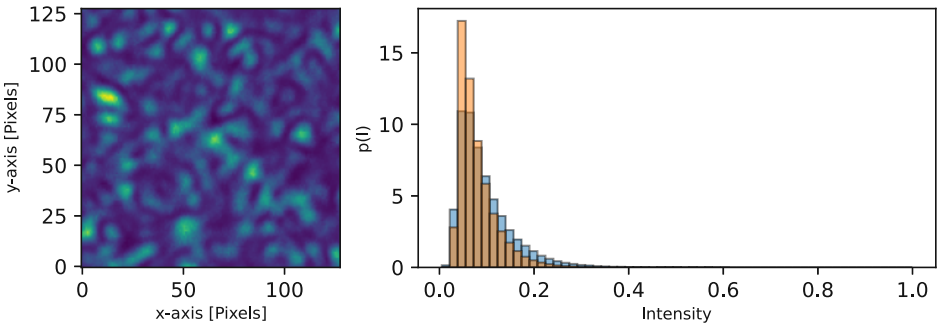

Figure 2.14: A Speckle with high TiO_2 concentration should show an exponential distribution, but the statistics can be better fitted with a Gaussian-exponential mixture for both a point where the scatterer is positioned (blue) and a point from the background (orange)

Further investigation into the exact probability density would be needed to extract all the parameters used for simulation. This would not change much for the complexity of the simulation as it would most likely just result in additional computationally non-intensive steps added on top of the current code, but obtaining the parameters for these steps to match the experiment is impossible without the probability density. If the experimental data could be simulated well, an interesting follow-up topic would be to train the model on just simulated data and test the performance on experimental data. As obtaining training data can be difficult and time-consuming, this could be used to verify a model on reliable data before setting up the experiment and obtaining the data. However, this is out of scope for this thesis, but can be a direction for future research.

3. Calculating Fisher Information of Speckles

Fisher Information is used in estimation theory to derive the limit of the precision that an estimator can reach with the Cramér Rao Lower Bound [23]. In this chapter, we will show how we can accurately calculate the Fisher information and subsequent Cramér Rao Lower Bound from the dataset simulated in chapter 2. For this, we introduce the general concept of Fisher information and how it can be adapted to account for biases that might help an estimatior achieve higher precision. Then we will derive a way to obtain the Fisher information from our dataset of speckle patterns without correlations for the estimation of an arbitrary parameter θ and demonstrate it explicitly with simulated data, where the parameter θ is a displacement of the scatterer on the x-axis. As the calculation heavily depends on the change of the intensity distributions, which have to be derived from a finite sample, the quality and usefulness of the Fisher information will suffer if the sample is too small. We will present a method to estimate the number of speckles that is needed to obtain these distributions to the desired accuracy. In between, we are also highlighting an interesting, more general approach for speckles with correlation that Max Weimar used in [12] to also obtain the correct Fisher information for the experimental data, although it is not needed for our well understood statistics. In the end, we have an approach to accurately calculate the Fisher information for the simulated fully developed speckles, which can be used to calculate the Cramér Rao Lower Bound for this data, which the estimations of the artificial neural network can be compared to.

3.1. Fisher information in general

The Fisher Information is typically encountered when one wants to estimate the parameters of a probability density function (PDF) from a dataset using the maximum likelihood method. For this one assumes a model for the PDF with some parameters θ, e.g. a Gaussian distribution where the mean μ should be estimated

$$f(x|\theta) = f(x|\mu) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left\{-\frac{(x-\mu)^2}{2\sigma^2}\right\} \tag{15}$$

From the model the likelihood function can be constructed which gives a distribution for how well a specific parameter θ can explain the observed data $x_1, ..., x_n$

$$L(\theta) = P(x_1, x_2, ..., x_n|\theta) = \prod_{i=1}^{n} f(x_i|\theta) \tag{16}$$

It is often easier to work with the log-likelihood function, as one can work with a sum of PDFs instead of the product. The strictly monotonic property of the logarithm and the fact that PDFs cannot be negative means that this change has no consequence on further discussion other than being easier to handle.

$$\ln L(\theta) = \sum_{i=1}^{n} \ln f(x_i|\theta) \tag{17}$$

Next, we obtain the score function, which gives the sensitivity of the likelihood function with respect to the parameter θ. If there is a wide peak in the likelihood function, we expect that the parameter is harder to estimate than if there is a very narrow peak. This is achieved by differentiating with regard to the parameter:

$$U(\theta) = \frac{\partial}{\partial\theta} \ln L(\theta) \tag{18}$$

The Fisher information is the expected value of the score function [24], and can be expressed as follows:

$$F(\theta) = \mathbb{E}\left[\left(\frac{\partial}{\partial\theta} \ln L(\theta)\right)^2\right] \tag{19}$$

Note that the Fisher information in literature is often denoted with I, but will be denoted with F in this work to separate it clearly from the Intensity. An alternative also often used is the second derivative of the log likelihood function, that leads to the same value if the PDF satisfies certain regularity conditions [23].

The most important result involving the Fisher information is that it determines a lower bound on the variance of the estimator, which was named Cramér Rao Lower Bound (CRLB):

$$Var(\hat{\theta}) \geq \frac{1}{F(\theta)} \tag{20}$$

3.1.1. Bias correction

The Fisher information we derived so far is only valid for unbiased estimators. In real life applications, biased estimators often lead to more desirable results for the accuracy, as the introduction of some bias can lead to smaller variances in the estimation. The machine learning model trained in chapter 5 also shows bias, necessitating a correction of the CRLB, the process of which is explained in this section. Given the bias

$$b(\theta) = \mathbb{E}(\hat{\theta}) - \theta \tag{21}$$

it can be derived [25] that the CRLB will then be

$$Var(\hat{\theta}) \geq \frac{(1 + \frac{\partial b}{\partial \theta})^2}{F(\theta)} \tag{22}$$

This also shows explicitly that for a biased estimator it can be possible to achieve a lower variance than for an unbiased one as the derivative of the bias can be negative. However, this may not be the case along the whole parameter space.

The biased result is not as commonly discussed as the regular CRLB, as it is often hard to obtain the derivative of the bias, but for machine learning models especially a large set of training data is usually available, where both $\hat{\theta}$ and the ground truth θ are known to calculate the bias and then obtain the derivative. For this thesis, the bias is easily accessible as all the training data for the model is simulated.

3.2. Fisher Information for fully developed speckles without spatial correlation

In the same way as the Fisher information was derived to determine the maximum information states [10] in a different experimental setup including a reference beam, we will derive the Fisher information here in this section for our own specific setup. The derivation specifically depends on the assumption that the pixels in the speckle pattern are independent, meaning the speckle grains are smaller than one pixel, and we therefore don't have to account for correlations between them. The Fisher information F is then given as just the sum of the Fisher information for each independent pixel F_i [23]:

$$F = \sum_{i=1}^{N} F_i \tag{23}$$

which means we only have to derive the Fisher information for one pixel, where we know that the distribution for for a fully developed speckle looks like

$$p_i(x, \theta) = \frac{1}{I_i} \exp\left\{-\frac{x}{I_i}\right\} \tag{24}$$

We start from the definition of the Fisher information, with the expectation value written as an integral

$$F_i = \int_{-\infty}^{\infty} dx\, p_i(x, \theta) \left[\frac{\partial \ln p_i(x, \theta)}{\partial \theta}\right]^2 \tag{25}$$

As the exponential distribution $p_i(x, \theta)$ is determined uniquely by the parameter I_i and so every change in θ only results in a change of I_i we can use the chain rule

$$\frac{\partial \ln p_i(x, \theta)}{\partial \theta} = \frac{\partial \ln p_i(x, \theta)}{\partial I_i} \frac{\partial I_i}{\partial \theta} \tag{26}$$

which gives

$$F_i = \left(\frac{\partial I_I}{\partial \theta}\right)^2 \int_{-\infty}^{\infty} dx\, p_i(x, \theta) \left[\frac{\partial \ln p_i(x, \theta)}{\partial I_i}\right]^2 \tag{27}$$

Then we carry out the differentiation of the logarithm and get

$$F_i = \left(\frac{\partial I_i}{\partial \theta}\right)^2 \int_{-\infty}^{\infty} dx \frac{1}{p_i(x, \theta)} \left[\frac{\partial p_i(x, \theta)}{\partial I_i}\right]^2 \tag{28}$$

Differentiating the exponential distribution generally yields

$$\frac{\partial p_i(x, \theta)}{\partial I_i} = p_i(x, \theta) \frac{x - I_i}{I_i^2} \tag{29}$$

In our derivation for the Fisher information, this gives

$$F_i = \left(\frac{\partial I_i}{\partial \theta}\right)^2 \int_{-\infty}^{\infty} dx\, p_i(x, \theta) \frac{x^2 - 2xI_i + I_i^2}{I_i^4} \tag{30}$$

Going back from the integral form of the expectation value, we can use the property for the moments of the exponential distribution $E(x^k) = I_i^k k!$:

$$F_i = \left(\frac{\partial I_i}{\partial \theta}\right)^2 \frac{\mathbb{E}(x^2) - \mathbb{E}(x)2I_i + \mathbb{E}(x^0)I_i^2}{I_i^4} = \left(\frac{\partial I_i}{\partial \theta}\right)^2 \frac{1}{I_i^2} \tag{31}$$

This will give us a useful approach to calculate the Fisher information in practice. We can get I_i for the exponential distribution by calculating the mean of a large enough sample and can then calculate the derivative through finite differences from samples where the parameter θ has been varied by $\pm\Delta\theta$

$$\frac{\partial I_i}{\partial \theta} = \frac{I_i(\theta + \Delta\theta) - I_i(\theta - \Delta\theta)}{2\Delta\theta} \tag{32}$$

The correctness of this derivative depends therefore greatly on obtaining the intensity accurately, which in the case of the exponential distribution is the mean of a sample. Due to the inherent statistics of speckle patterns, we will have to choose a sufficient sample size, as the means for each pixel would seem noisy, which will be treated in section 3.4.

3.3. Fisher Information for more general speckle patterns

To calculate the Fisher information for the more general speckle patterns we discussed in the previous chapter, correlations between pixels have to be dealt with. The correlations pose a problem, as this means two pixels partly share a unit of information. Imagine two pixels that always display the same value. This means they have a correlation coefficient of 1. You could in every situation add an arbitrary number of these pixels, but this would not increase information. This means summing over all pixels without taking correlations into account would lead to serious over counting of information if the correlation coefficient is positive. In the case of speckle patterns, we only have to deal with positive correlations, as speckle grains larger than a pixel lead to neighboring pixels being positively correlated.

For Gaussian joint distributions with correlations, there already exists a well studied model where there already exists a closed formula for the Fisher information accounting for correlations. For exponential distributions this is not the case and while some approaches exist to model correlations, for the number of variables (i.e. pixels) one would need to derive the model using Copulas [26], and from there derive a calculation of the Fisher information if possible. This would still only be one step towards understanding the experimental data we received, as this shows different and more complex behavior

from an exponential distribution. So even if this model was derived, it is unsure how expensive it is to compute, as the marginal distribution for every pixel has to be fitted to the assumed distribution. In case of the exponential this is just finding the mean, but e.g. for an exponential Gauss distribution the fit procedure is a much more involved optimization problem. Therefore, we decided to inspect one easy approach that is often used to get rid of the correlations, and also highlight its problems. Then we will highlight one approach developed by Max Weimar, where he can provide an estimation for the Fisher information for arbitrary distributions.

3.3.1. Resampling to remove correlations

Resampling is the first thing that comes to mind when correlations pose a problem [27]. If we assume that a speckle grain carries one unit of information and covers multiple pixels, the information would be overcounted when counting it by pixels, as every pixel represents the same information. Sampling to a lower resolution where a speckle grain only covers one pixel should give us still all the information available while preventing overcounting due to correlations. Resampling the resolution of Images is achieved by processing a set of pixels, most commonly a square grid, with a function to retrieve one value from the set. Common functions that are used are:

- getting the center value of the grid
- getting the average value of the grid
- getting the maximum value of the grid

Although there are many options to resample, the safest choice for us is to always get a specific position in the grid that should be sampled and not change this position between images to leave the underlying probability distribution intact, so we can apply the Fisher information formula for uncorrelated exponential distributions without any modification. The next question is how big the grid for resampling should be. For this, we simulated speckles with small speckle grains, but still noticeably larger than one pixel as shown in figure 3.1.

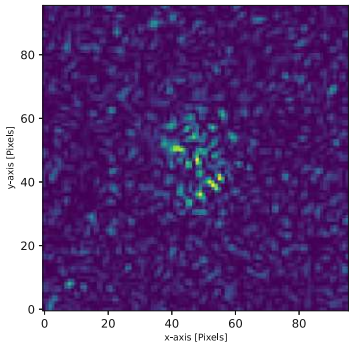

Figure 3.1: An example speckle pattern simulated with very small speckle grains.

With these speckles we can calculate the autocorrelation function and from there measure the peak of said function to get a measure of the speckle grain size. One popular measure is again the FWHM, which is obtained by measuring the peak width at fifty percent height. This makes sense as for example with Gaussian peaks the peak width at the very bottom would in theory be infinite and therefore impossible to meassure. For our simulated speckles, we averaged the autocorrelation function of many simulated speckles and measure the peak at different heights to obtain the following table:

% of height	width
50	1.99 pixels
25	2.98 pixels
1	3.96 pixels

Table 1: The peak of the autocorrelation function measured at different heights. The typical FWHM at 50% with 2 pixels can give an indication of how far the correlations reach.

Now that we have a range for how big the resampling grid should be, we can try each one and observe the effect it has on the correlation coefficient between two neighboring pixels in the final image shown in figure 3.2:

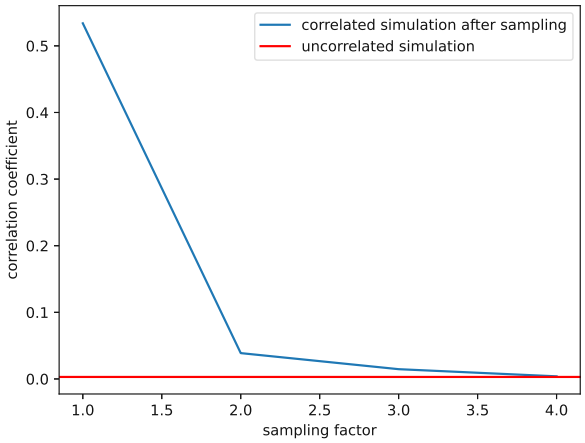

Figure 3.2: The correlation coefficient between two neighboring pixels decreases with the sampling factor. A significant decrease is achieved already by using the FWHM (sampling factor = 2.0) but to approach uncorrelated pixels a sampling factor in the size of the whole autocorrelation function peak is necessary (sampling factor = 4.0).

The graph reveals that only taking the FWHM is not enough to get rid of the correlations, while when taking the width of the whole peak one can transition the correlated case completely back to the uncorrelated case. A measurement of the correlation coefficient in simulated uncorrelated speckles reveals the same very small fluctuations we can see here in the case of resampling with four pixels.

There are however problems with this approach. The neighboring pixels are not a hundred percent correlated, which makes sense as the speckle grains can lie anywhere, but the grid gives a fixed position. Therefore, information is necessarily lost by this process. For our experimental data the speckle grains are much bigger than in the simulated example, leading to a huge information loss. A second problem we have with the experimental data is that we expect ballistic photons. On a focused picture we would expect every pixel to receive some amount of ballistic photons and therefore each pixel would deliver, at least partially, unique information regardless of the speckle grain size. All this information is equally erased in the resampling process. These problems make a different approach that can potentially retain more information desirable, which will be highlighted in the following section.

3.3.2. Fisher Information from independent component analysis

The key to finding a reasonable estimate for the Fisher information for the experimental data presented in [12] is a result from Max Weimar. Although beyond the scope of the thesis, we will briefly highlight the approach. More information can be found in [12] and its supplemental materials.

For his approach, independent component analysis (ICA) is used to transform the data to a base where correlations are minimized. As not every joint distribution has a base without correlations there can still be some mutual information between the marginal distributions, but by checking that this mutual information is small we can be sure that the problem of over-counting information does not have a significant effect. As the over-counting leads to a larger Fisher information and therefore a smaller Cramér Rao Lower Bound, this still signifies a limit that should not be surpassed by any estimator. This means the over-counting is minimized, leading to a good compromise between the actual lowest bound that should be reachable and a bound so low that it bears no useful application.

To get the Fisher information, the derivative is directly numerically computed from the independent components with finite differences, making the process applicable for arbitrary distributions. One important point is to choose the step size for the derivative so that the calculation is stable, which can be achieved by plotting the Fisher information over the step size and finding the first plateau in this function.

3.4. Influence of noise

If we would now calculate the Fisher information from the speckle patterns, we would notice that the number of speckles we use to calculate has an impact on the result. The reason for this is that the derivative is extremely sensitive to noise and for the calculation of the intensities the speckle noise in particular is a hindrance as can be seen in figure 3.3.

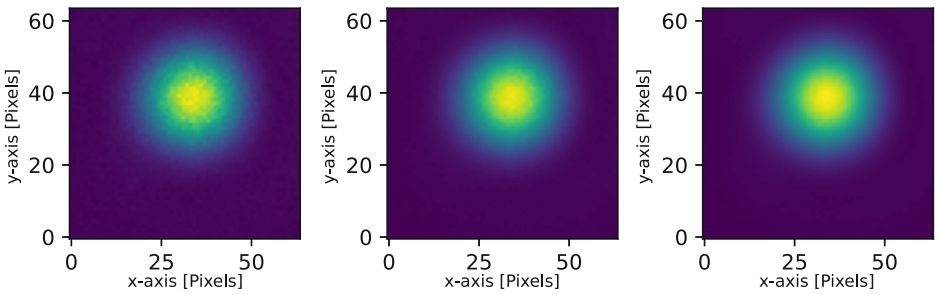

Figure 3.3: Calculating the mean of speckle patterns shows visible noise when using too few samples (left) while the noise disappears with increasing sample sizes (right).

As the Fisher information depends on the squared derivative of the PDF, the error resulting from noise will only increase the Fisher information. This means if the noise is too large, the calculated Cramér Rao bound will be much lower than the actual one, losing its usefulness as a tool for benchmarking estimators. In the paper on Maximum Information States [10] this noise problem was also encountered, although there the goal was to get more signal out of larger $\Delta\theta$. To get an indication of the impact of noise two different measurements without the parameter θ changing have to be taken. Calculating the derivative via finite differences, and subsequent Fisher information, with those measurements and the $\Delta\theta$ used for the real calculation would lead to a fisher information of zero in the absence of noise. This means all the Fisher information resulting from this calculation are purely coming from noise in the measurements. Many plane waves were measured in [10], each carrying part of the Fisher information, allowing for one histogram to be plotted for the noise measurement and one for a measurement where θ actually changed. These two histograms then show the difference in signal and noise. As we have many pixels that are each carrying part of the Fisher information, we can plot similar histograms. While taking two different measurements might be time-consuming in the experiment, for this work it is as easy as starting two simulation runs with the same parameters. With the same number of speckles as for the mean intensities above and the parameter θ being the displacement on the x-axis, these histograms are shown in figure 3.4.

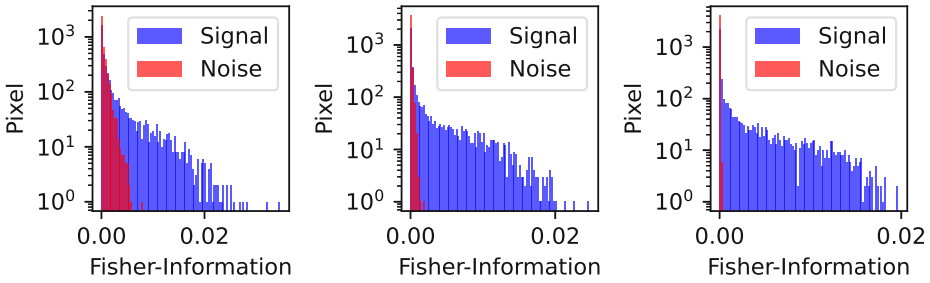

Figure 3.4: The histograms show how many pixels carry how much Fisher information. For the signal (blue) the calculation uses two samples where the parameter θ changed. The noise is obtained by taking two samples, between which the parameter did not change, meaning the only contributions to the derivative used for the Fisher information come from noise. As the size of the samples is increased (left to right), we can see that the noise has less of an effect.

While these histograms give a good indication of the noise, we can also do a large simulation run and slice the resulting dataset into smaller ones, taking care that no overlapping data is used in the same calculation, and plot $1 - \frac{Noise}{Signal}$ to see how much of the calculated Fisher information is resulting from the actual signal:

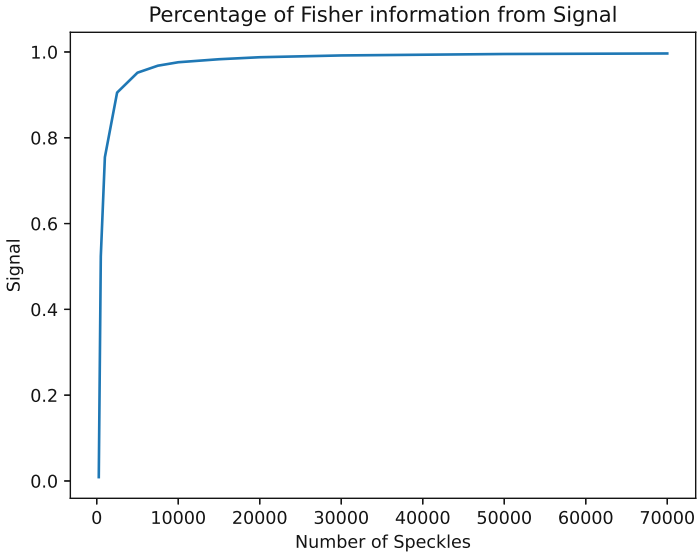

Figure 3.5: Percentage of Fisher Information that stems from Signal with regard to number of speckles used for calculations.

This then allows for a point at which we are satisfied with the noise reduction to be chosen. For experimental data in [12] this was necessarily used in the beginning

as collecting a large amount of data proved difficult because of settling TiO_2, making long experiment runtimes impossible. The calculation of the Fisher information at this time was done without accounting for correlations, but as the over counting affects both signal and noise equally, and we were only interested in the ratio of the two this still proved a valid approach to get an estimate on how many speckle patterns are needed per position.

3.5. Calculations for simulated data

For the calculations of the Fisher information, with θ being the displacement of the scatterer on the x-axis in our experimental setup, we will proceed with uncorrelated data, as we have shown that, with some downsides, correlated data can be transformed into uncorrelated data. The better approach of using ICA is not a result of this thesis and therefore not described here. In this section, we will find the Fisher information for simulated datasets, using the intensity distributions from the experimental samples with varying optical thicknesses as starting point for the simulations. To get the correct Fisher information, we first need to examine how many speckles are needed for each optical thickness so that the noise induced error is small. In this thesis we set the threshold to 5%, checked a list of speckle counts and took the first that surpassed this threshold as the number of samples we need for calculating the Fisher information. When everything is calculated, we get the following results:

TiO_2 concentration	number of speckles needed	$1 - \frac{N}{S}$	Fisher information
100mg/ml	2500	0.972	30.934
150mg/ml	5000	0.970	13.894
200mg/ml	5000	0.951	8.898
250mg/ml	15000	0.962	3.746
300mg/ml	50000	0.961	1.076
350mg/ml	-	-	-

Table 2: Table showing the number of speckles needed to be sampled per scatterer position for a specific optical thickness to get the contribution of noise to the Fisher information below 5% along with the calculated Fisher infomation.

We can then also plot the dependence of the Fisher information on the optical thickness in figure 3.6.

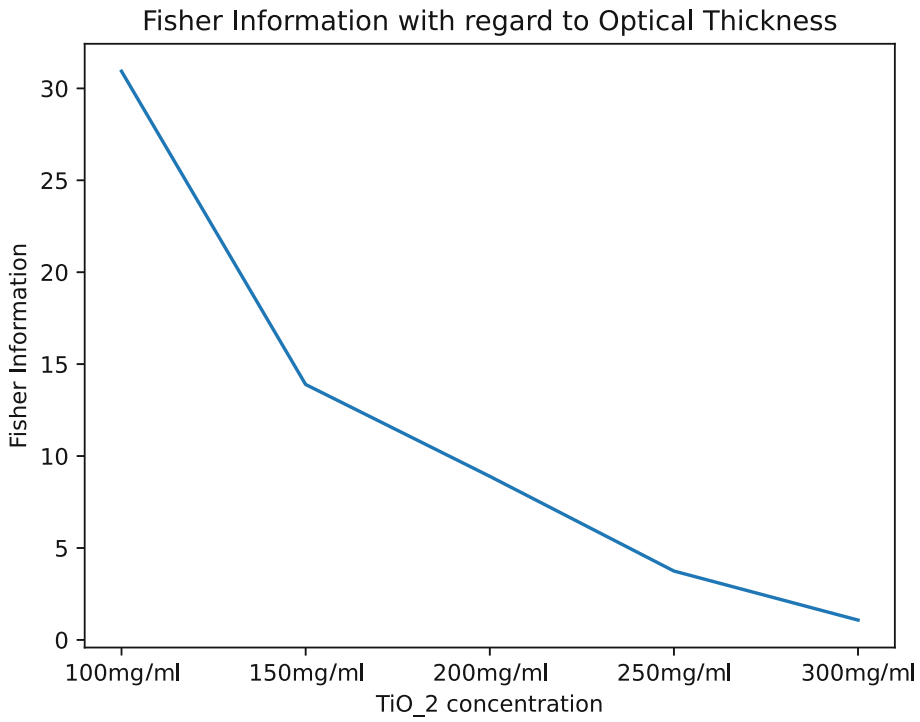

Figure 3.6: Fisher information decreases with increasing TiO2 concentration

4. Developing the Machine Learning Model

As the main focus of this thesis is to develop a model for estimating a parameter from speckle patterns in simulated data that can also be used in [12] to verify the Cramér Rao bound as limit for estimators, we will tackle this task in this chapter. For this, we will develop some basic knowledge about machine learning to understand the revolutionary UNet model that was used by Lei Tian to show that deep learning can reconstruct images from speckle patterns [5] and on top of that develop a model that can be used to estimate a parameter in our dataset. The focus is therefore only on the concepts necessary to understand this model. For a more complete technical, proof-heavy introduction into these and further concepts, we refer to the diploma thesis of Lukas Michael Rachbauer [28]. For a more general introduction to machine learning, including other concepts beside neural networks, we refer to [29] or a textbook like [30]. After establishing the foundation, we will evaluate two network architectures, rooted in the concepts of UNet, but on a different dataset than discussed so far, as the experimental data was not yet

available at the time the approaches were tested. The chapter then concludes with the choice of a network architecture candidate we want to use for chapter 5.

4.1. Fully Connected Networks

Fully connected neural networks, also called dense neural networks, were the first type of deep learning algorithm that emerged, under the name of multi-layer perceptron [1]. As the name suggests, it consists of multiple layers of perceptrons, which evolved to the neurons we use today, with the main difference being that neurons have a wider variety of activation functions. The output of a neuron is determined by the weighted output of neurons in a previous layer and an inherent bias to the neuron. All of this is combined in an activation function:

$$a = \sigma(\boldsymbol{x} \cdot \boldsymbol{w} + b) \tag{33}$$

where a is the activation or output of the neuron, $\boldsymbol{x}$ is the vector of inputs, $\boldsymbol{w}$ are the weights for the inputs and b is a bias inherent to the neuron, making it more or less likely to activate and σ denotes the activation function.

For a fully connected neural network, every neuron in one layer has weighted connections to all the neurons in the previous layer. These weights for the connections and the biases of the neurons are the parameters of the network, which can be derived from so-called hyperparameters, like the number of layers and the number of neurons per layer, which define the network.

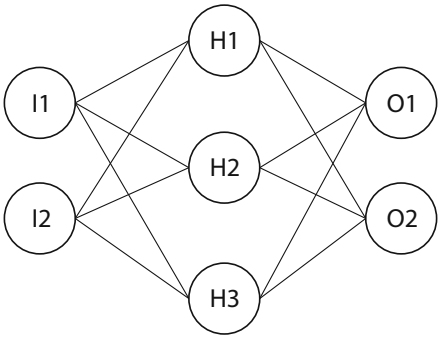

Figure 4.1: Illustration of a fully connected network with two input neurons, three neurons in a hidden layer and 2 neurons in the output layer.

For neural networks to be able to learn non-linear relationships between inputs and outputs, the activation function σ has to be non-linear. If one were to use a linear function for all neurons, all the linear transformations within the network could be collapsed into one matrix multiplication, which cannot map non-linearity and defeats the purpose of a layered model. The two most common activation functions used in literature are

- Sigmoid:

$$\sigma = \frac{1}{1 + e^{-x}} \tag{34}$$

- Rectified linear unit (ReLU):

$$\sigma = \left\{ \begin{array}{ll} x, & x \geq 0 \\ 0, & otherwise \end{array} \right\} \tag{35}$$

While the sigmoid function, due to being smooth, lends itself very well to explaining the gradient descent used to train the network, it is computationally expensive compared to ReLU. Due to being a saturating function, with $0 \leq \sigma \leq 1$, it can also lead to vanishing gradients if the inputs for the activation are very big, inhibiting the training process. Thus, the ReLU function and some of its variations are more commonly used nowadays.

4.1.1. Training a Network

In theory, it would be possible to tune the weight and bias parameters by hand, but it quickly becomes unfeasible as the number of neurons grows. The power of neural networks can only be realized if these parameters can be adjusted by an optimization algorithm, which is usually some form of gradient descent. This algorithm works by defining a cost function $C(x, \hat{x})$ that defines a penalty for the network depending on how far the expected outputs x deviate from the predicted outputs $\hat{x}$ of the network. The cost function has to be chosen with the specific task in mind and although there are standard functions for specific tasks, many variations exist. Some of these standards are

- For regression (estimating a continuous variable, e.g. house prices from a set of

input features) Mean Squared Error (MSE) is used:

$$C(x, \hat{x}) = \frac{1}{N} \sum (x - \hat{x})^2 \tag{36}$$

where N is the number of data points.

- For single label classification (assigning only one category depending on input, e.g. classifying an animals' species from an image) one uses categorical cross-entropy:

$$C(x, \hat{x}) = -\sum_i x_i ln(\hat{x}_i) \tag{37}$$

where i is the number of categories.

- For multi label classification (assigning multiple categories depending on input, e.g. estimating which demographic groups a person belongs to depending on browsing behavior) one uses the sum of binary cross-entropies:

$$C(x, \hat{x}) = -\sum_i x_i ln(\hat{x}_i) + (1 - x_i) ln(1 - \hat{x}_i) \tag{38}$$

where i is the number of categories.

To train the network, the partial derivatives $\frac{\partial C(x,\hat{x})}{\partial \theta_i}$ of the cost function with respect to the weight and bias parameters are taken, and each parameter can be adjusted with

$$\theta_i(t+1) = \theta_i(t) - \eta \frac{\partial C(x, \hat{x})}{\partial \theta_i} \tag{39}$$

where η defines the step-size of the adjustment. This step size can be either set manually as a learning rate hyperparameter or in more advanced versions be adjusted on a momentum basis, where the last few step adjustments are used to calculate momentum. If they all point in the same direction the step size is increased for faster learning, or if they are alternating the step size is decreased to get further into the minimum. One popular momentum based learning rate optimizer is the Adam optimizer. [31]

On the computational side, gradient descent is very efficient for neural networks. As the derivatives of the activation functions are known analytically, one can obtain analytical expressions for every partial derivative via the chain rule. In the first step,

the weights at the output layer can be resolved using the cost. Then the obtained partial derivatives can be used to resolve the weights of the layer right before the output layer, and so on. This behavior led to the name Backpropagation for the application of gradient descent to neural networks because the parameter updates propagate from the back of the network to the front. [32]

4.1.2. Training-, Validation- and Test-Dataset

When naively using all the available data for training, one can run into one of the most common problems in machine learning: The model works well on the training data, but does not perform nearly as well on other similar datasets. This is called Overfitting and arises because during the training process the network picked up patterns that are unique to the training set to improve the cost function at the cost of generalizeability. To detect this early, a good practice is to take a small part of the training data and use it as validation data. The validation step is typically done after every training step, to see how the model performs on unseen data. If the performance on the validation data stops improving, the training process can be stopped, as every step afterward just improves the model on recognizing patterns unique to the training data, maybe even worsening performance on the validation data. Usually, the person training the network will choose the configuration which performs best on the validation data after trying many different combinations of hyperparameters. However, this is itself a training process, and it is good practice to set aside another set of the training data to test the final chosen model with data both unseen in the network training process and unseen during the hyperparameter tweaking process.

4.1.3. Adding randomness to the training

The gradient descent learning process can be implemented by utilizing the whole training dataset to calculate one step for adjusting the parameters, and repeating this until a minimum is reached. Assuming that the parameter initialization is always the same, this method is repeatable, meaning that it leads to the same minimum in the cost function every time. While repeatability usually may be desirable for experiments, some addition of randomness can have many benefits in the training process. Saddle points on the original cost function are then less of a concern for stopping training progress, and the process gains the ability to break free from local minima and potentially discover a

better, more general minimum.

Batching is one such approach to add randomness and has the additional benefit of speeding up the training. For this technique, one training step is often split up in batches only containing a small amount of the whole training data and updating the parameters after every batch. When the network has gone through all the training data by using all batches, this is called an epoch. After each epoch, the training data is shuffled and separated into new batches. Though the progress on the original cost function graph may seem more erratic, as technically the cost function and therefore the gradient looks different for every batch, it will generally lead to better minima faster [29].

Dropout layers are also adding randomness to the network, with the purpose of helping generalizeability. The idea for dropout is that during a training step, a certain percentage of neurons in a layer are set to zero. When the training is finished, the activations have to be adjusted with this percentage to account for more active neurons.

In the original paper [33], the creators of this approach hypothesize that this algorithm improves performance because it prevents co-adaptation of neurons. Co-adaptation means that errors of badly performing neurons are fixed up by other neurons, leading to complex chains of neurons depending on each other, which in turn may have reduced performance on more general data. By randomly setting neurons to zero and making them unreliable for others to fix errors, one neuron can not lean on these and complex chains are broken up. Another conceptual explanation they provide for the performance increase through dropout is that by switching off a number of neurons, each training step one new smaller network is trained. By switching on all neurons after training and adjusting the activations accordingly, one could see this as averaging over all these networks, making the bigger network an ensemble of these smaller similar networks that have been trained at the same time.

4.2. Convolutional Networks

In fully connected networks, every neuron takes all the inputs of the previous layer into account. While this works great for tabular datasets, where every input column could have an effect on the output, they can be very inefficient for problems that show some kind of locality or translational invariance, like an image. An example which would be problematic for dense networks is to detect whether a picture contains a cat. If the training data would not show a cat in the upper right sector in any of the training images,

a fully connected network would not be able to detect a cat there. This means that the training data would not only need to contain many cats, to train the network how a group of pixels resembling a cat looks like, but also have these cats at every possible position within the picture. A more efficient approach would be to only learn what a cat looks like and test if such a group of pixels can be found in the picture. This is where convolutional neural networks come in, which we will be explained in this section.

4.2.1. Filters in Image Processing

In image processing, the idea of filters, also called kernels, are a well established concept [34]. These kernels are used for a variety of things like blurring or sharpening an image or edge detection and are expressed through small matrices, which are used in a discrete convolution with the image to obtain an output image, where every pixel in the output results from taking a group of neighboring pixels with the same size of the filter, multiplying every element of this pixel-matrix with the corresponding element in the filter matrix and in the end summing all the results to get one output pixel. A visual representation can be seen in figure 4.2.

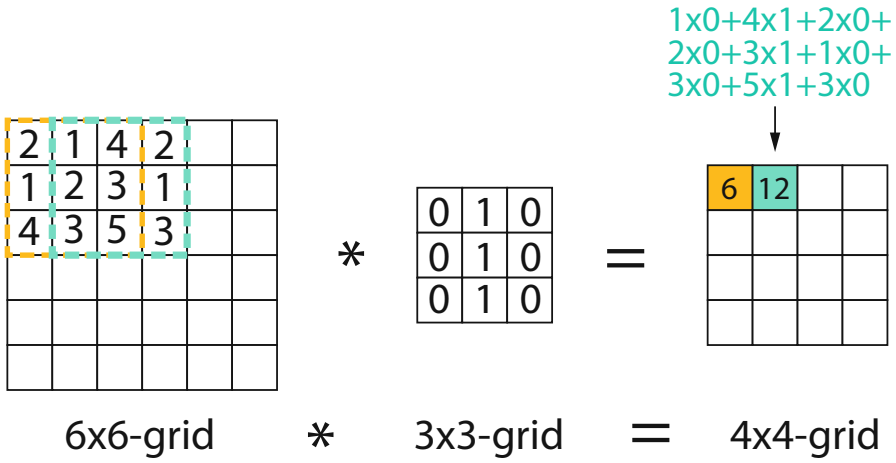

Figure 4.2: Discrete convolution of a 6x6 matrix with a 3x3 matrix, the second matrix is laid over the first in the top left, multiplicated element-wise and summed up to produce the first element of the output. Then the 3x3 matrix is moved one step to calculate the next output element.

The filters in image processing are specifically crafted for their purpose. An example for a filter for detecting a horizontal edge would look like:

$$A = \frac{1}{6}\begin{pmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{pmatrix} \tag{40}$$

On an image where the top half is black and the bottom half is white, a discrete convolution with this filter would result in an output with almost exclusively zeros and a horizontal line of ones where the edge between top and bottom half lies, as the filter only produces a non-zero output when it is convolved with a 3x3-grid where the top row and bottom row are different.

If the image size needs to be preserved it would need to be padded, as the filter would otherwise have undefined numbers when calculating the very edges. This is usually done either with pure ones, pure zeroes or by mirroring the picture at the edges. Otherwise, one would get a smaller picture, e.g. 2 pixels smaller in both directions when using a 3x3 filter.

4.2.2. Filters in Neural Networks

In convolutional neural networks, the idea is for these filters to be trained on data. Each matrix element of the filter is a weight that is adjusted through the backpropagation algorithm. With convolution deriving the algorithm is more elaborate, but an analytical form for the partial derivatives exists, making it relatively inexpensive for a computer to calculate. Often times in convolutional networks the inputs are not 2D, but 3D. For example, if an input picture is in color, one would typically split this up into the three color channels for red, green and blue. Another example would be that a convolutional layer with 32 filters would produce an output with 32 channels, which in turn can serve as 3D input for the next convolutional layer. For processing, there are then two approaches:

- Normal convolution with C three-dimensional filters with size $N \times N \times I$, where N is the desired filter size, I is the number of input channels and C is the desired number of output channels.
- Depth-wise separable convolution with C two-dimensional filter with size, $N \times N$ and a second convolution of the resulting channels with I filters with size 1×1.

Both approaches are valid. The convolution with multiple 3D-filters has more parameters, which gives the network more flexibility to learn, whereas the separable convolution

is faster and needs less memory at the cost of said flexibility.

4.2.3. Pooling Layers

With only padded convolutional layers the image-size would always stay the same, but for some use-cases distilling the gathered information to a smaller image is necessary. We have seen a similar objective with resampling, and in machine learning this is usually called pooling. The pooling layers separate the image into non-overlapping groups, typically 2×2 and performing an operation that results in a single pixel. In Max-Pooling layers the maximum of the pixels is propagated, In Average-Pooling layers the average is calculated. A visual representation can be seen in figure 4.3.

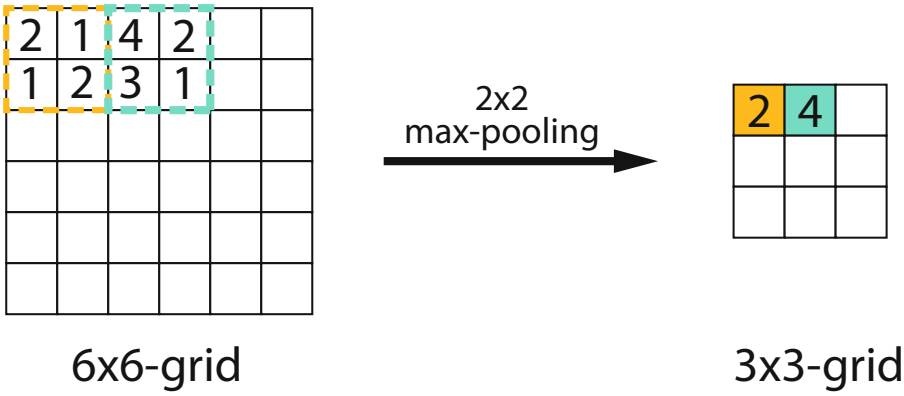

Figure 4.3: A 2x2 maxpooling takes the maximum element of every non-overlapping 2x2 section of the matrix to produce the output

When putting a max-pooling layer between two convolutional layers with 3×3 filters, the locality limitation is eased for the second layer, as its 3×3 grid now effectively covers a 6×6 pixel area from the original image. In this sense it is observed in convolutional nets that the first layers have fairly primitive filters like the edge-detection and through max-pooling subsequent layers combine these simple filters to form bigger and more complex shapes.

4.2.4. Combining Convolutional and Dense Layers

For detecting whether a cat is in a picture, using only convolutional and pooling layers is not enough, as the output would again be some image with arbitrarily many channels.

This would make generating a training set difficult, as we would need to define how this output image would have to look and define a cost function to make the backpropagation algorithm work. For a problem like this the convolutional layers are in practice only there to pre-process and enhance the information and then use a few fully connected layers to reduce this information to a singular output estimation, that represents the confidence of the network, if there is a cat in the picture. This architecture is illustrated in figure 4.4. Defining the desired answer is then easy, by labeling the pictures with true/false and using binary cross-entropy as a cost function. Another less used option would be a max-pooling layer at the very end, that spans the whole image and outputs only one integer, and again use binary cross entropy as cost function. This last layer is also called global pooling layer, but usually performs worse than a few connected layers.

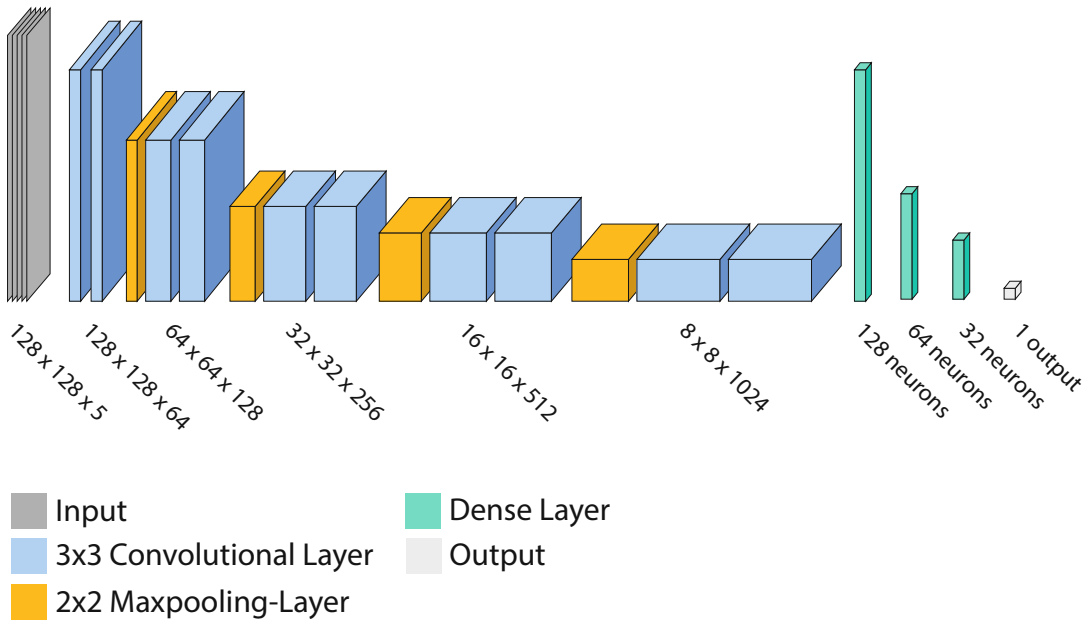

Figure 4.4: Illustration of convolutional neural network with a few convolutional layers and a few dense layers resulting in one single output

There are of course networks where we want the output to be an image again. An example would be the UNet which will be discussed in a later section.

4.3. Residual Neural Networks and DenseBlocks

As deeper layers in a convolutional network mean more complex filters and filter-combinations, these deeper networks historically performed better on very complex tasks like the ImageNet dataset, a dataset with images from 20,000 categories to classify. When adding more layers, training becomes harder, because of the vanishing/exploding gradient problem. When using the sigmoid activation function, large activation input values lead to the function saturating, which results in a derivative close to zero preventing learning. For activation functions which permit large partial derivatives, the opposite becomes the problem, where training adjustments may become too large for effective training and prevent the model from converging. While this problem can be mitigated with specific weight initialization and normalization layers, very deep networks still would not perform to expectations. What allowed for even deeper networks was the idea of residual learning via skip-connections.

With skip-connections, a layer not only has access to the outputs from the directly preceding layer, but also other layers that came earlier in the network. While this provides the layer with the possibility to combine less complex filters from earlier layers with more complex filters from the directly preceding layer, it also provides a shorter path to calculate parts of the partial derivative for these earlier layers which mitigates the vanishing gradients.

Denseblocks were then introduced in the DenseNet as a logical conclusion to this concept [35]. A denseblock is a block of convolutional layers with the same output size, where every layer has access to the outputs of all previous layers within the block, as well as the input of the block itself.

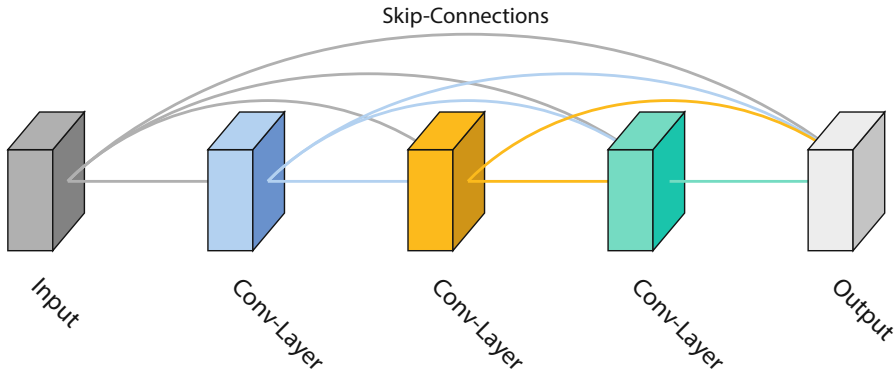

Figure 4.5: Illustration of one dense block, where the input for each convolutional layer consists of the output of all the layers in the block before it

4.4. UNet

The UNet architecture was introduced in [36] for biological image segmentation. These image segmentation tasks usually get an image as input and have to output a classification for each pixel. Fully convolutional networks were used for image segmentation before [37], but the difference is that these networks first compressed the information with convolutional and max-pooling layers and then used a last convolutional layer to get the expected output size from the condensed information. UNet takes a different approach by gradually upsampling and creating a symmetric path to get back to the desired resolution. On the way up, skip-connections are used to incorporate the outputs of the layers on the way down. This allows for residual learning, as discussed in a previous section. An example of how a UNet might look can be seen in the following figure:

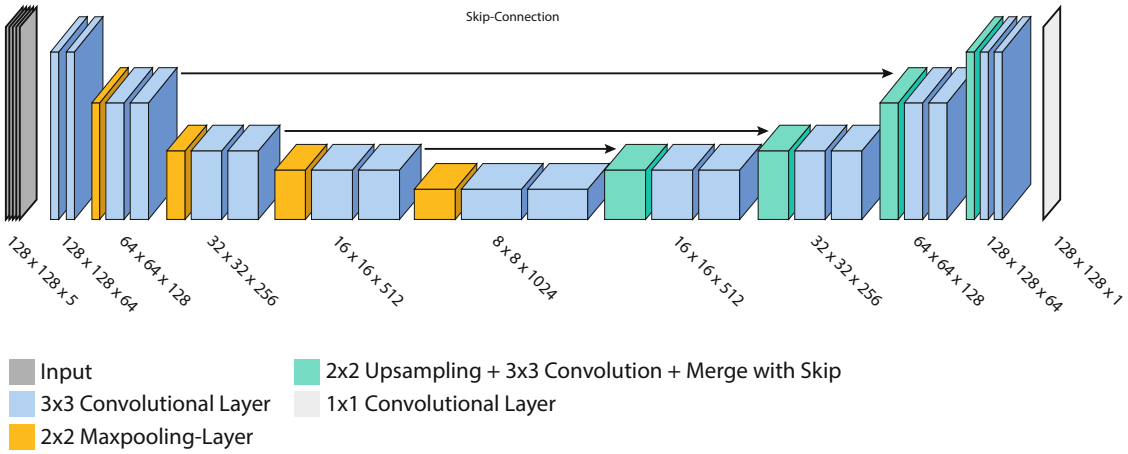

Figure 4.6: Illustration of the U-Net architecture

The team around Lei Tian also used this architecture in [5] to reconstruct an image behind a scatterer from the speckle pattern that was recorded. One modification they added was to use the previously described dense-blocks instead of just a few convolutional layers at each level.

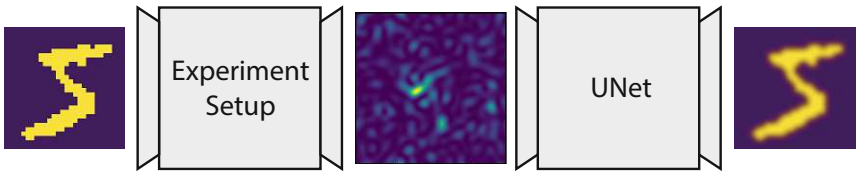

Figure 4.7: Illustration of how the Lei Tian paper could show that neural networks can retrieve the picture on the DMD from the speckle pattern

In [5] it was shown how to reconstruct images behind the same diffuser it was trained on, and they further showed that if enough diffusers are in the training set, the network can also pick up more general patterns and reconstruct patterns from similar unseen diffusers. As we expect a similar but different diffuser for each image from both our experiment and simulation, this network may be a good starting point for any investigations.

4.5. Combining models or End-to-End

Combining models can yield good results for more complex tasks. Imagine you want to label the types of cats in a picture. There are already available models that can detect objects (including cats) and draw a bounding box. One could use this bounding box of detected cats and feed it into a classifier trained on the different types of cats to get the exact type. This way the only work necessary is training the classifier and not retrain the much bigger object detection model. For our project there was a similar suggestion in the beginning to use the already known and available model from [5] to reconstruct the image of the scatterer without speckle noise and on top of that build a peak detection, similar to the detection filters we discussed earlier in the chapter, to determine the position of said scatterer. Another possible approach would be an end-to-end training where we would apply the first half of the UNet and use the condensed information to feed into dense layers which would result in a single output neuron corresponding to the x-axis-position of the scatterer, essentially creating one of the classic convolutional networks.

On the project timeline we decided to investigate this first in a slightly different problem domain, while waiting for the experimental data, so we would have a starting point for the model once the data arrived. Our experimental collaborator Ilya Starshynov provided a dataset he collected previously, where he replicated the experiment in [5], producing speckle patterns with numbers from the MNIST handwritten digit dataset [38] behind a diffuser. To get closer to our actual problem domain, we will not just reconstruct the image like in [5], but also to classify which digit was behind the diffuser. Although somewhat distant from the original problem domain, we can still gather valuable insights and gain some experience with the UNet model and the machine learning frameworks Keras [39] and Tensorflow [40]. For this, the combined model approach would be to use the UNet for reconstruction and hand it to a digit classifier using logistic regression to get 10 outputs, each giving the probability of the input being the class the output corresponds to. For the end-to-end approach, we will recreate the first half of the UNet and then add some dense layers, also with 10 outputs for the probability of the corresponding digit classes.

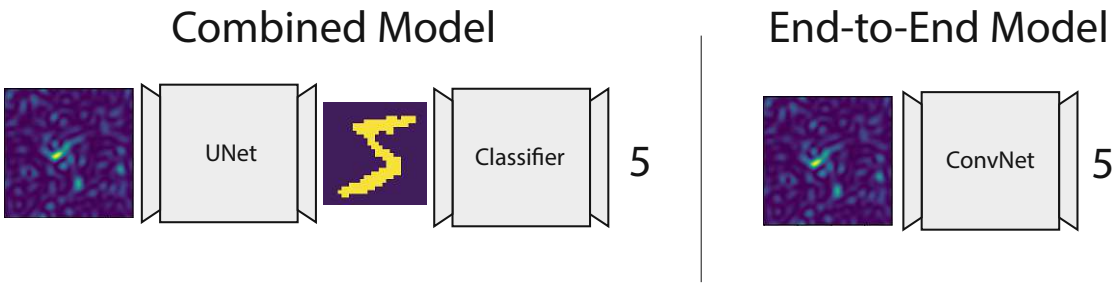

Figure 4.8: Sketch of both the combined model approach and the end-to-end model approach, where the combined model has an intermediate step of reconstructing the image before classifying, while the end-to-end model directly classifies from the speckle pattern

For the combined model, the UNet ended up having 21 million trainable parameters and took 2 hours for one epoch on our system specified in Appendix A. After training through 8 epochs, the training was stopped as the validation loss stagnated. On the test dataset, it reached an accuracy of 92.2%. While the only possible pixel values in the MNIST dataset are 0 and 1, the model allows for values in between, which shows up in a transition from 0 to 1 in the output images as seen in 4.7

The classifier was able to reach an accuracy of 94.4% when trained on the MNIST dataset directly, however when using the output data of the UNet the combined model could then only reach 82.1%. While it probably would improve accuracy if the classifier was trained directly on the output of the UNet, this approach seems unrealistic for the actual problem of peak detection we want to solve, as the peak detection is usually not a machine learning model trained on data but an algorithm.

For the end-to-end model, the trainable parameters were overall only 7.3 million. This resulted in only half the time used per epoch. After 11 epochs, the validation loss again stagnated and we stopped the training. The overall accuracy reached was 94.2%, vastly outperforming the combined model. With fewer parameters, it needs less memory to train and has the benefit of shorter epoch times. For these reasons, we decided to move forward with the end-to-end model once the experimental data is gathered, to solve the task of estimating the x-axis position of the scatterer.

5. Testing the ultimate limit on simulated uncorrelated data

Over the last few chapters, we built the knowledge and tools necessary to answer the question whether the Fisher information can be a good benchmark for the performance of an estimator in the context of wave scattering in disordered media. In this chapter, we will calculate the Fisher information for a simulated sample, train the chosen model architecture on this sample and compare the model performance to the CRLB to explicitly answer this question.

The data used in this chapter consists of simulated uncorrelated fully developed speckles with the data, based on the sample where the diffuser was calibrated to 200mg TiO_2 per ml. The parameter θ will again be the displacement on the x-axis, where 7 scatterer positions in a region $[-3, 3]$ are simulated. In chapter 3 we calculated the Fisher information F for this sample with enough images to achieve an arbitrary noise contribution set below 5%. For this chapter, we will use 70,000 samples for the calculation, as our hardware is easily capable of this, and we get a more exact value for the Fisher information, to compare to the machine learning model. With this sample, we get $F = 7.664$ as the value for the Fisher information and from the signal-to-noise ratio plot in figure 5.1 we can see that this value is accurate, as the histogram from the noise measurement shows that all pixels have a fisher information have a value very close to zero, while some pixels of the signal measurement contribute significantely more.

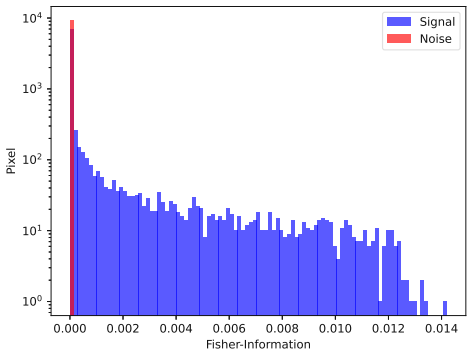

Figure 5.1: The noise calculation for the Fisher information (red), where both sides of the finite difference are computed with simulated samples at the same parameter θ, show all the pixels have a Fisher information close to zero. The signal calculation (blue), where the parameter θ changes, shows pixels contributing far larger values.

From this value, we can calculate the smallest possible standard deviation an unbiased estimator can achieve with

$$\sigma^2 = \frac{1}{F} \tag{41}$$

$$\sigma = \frac{1}{\sqrt{F}} \tag{42}$$

which gives the values $\sigma^2 = 0.130$ and $\sigma = 0.361$ for this sample. On the following graph it is visualized how we expect the predictions to be distributed, where we can see at least some overlap as seen in figure 5.2 rather than completely separate peaks. Non-overlapping peaks can lead to the network recognizing each peak as a distinct category, having a high bias towards one value representing the category in the predictions and therefore not delivering predicitons on a more continuous scale, like we would prefer for our analysis.

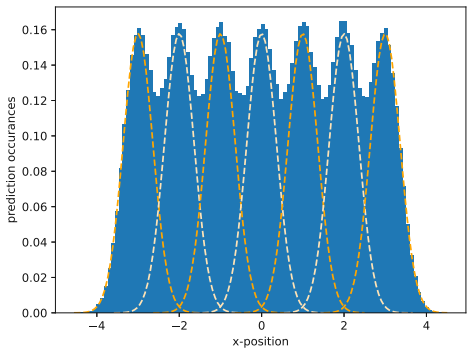

Figure 5.2: The theoretical optimal achievable distribution of predictions (blue) is constructed by creating a Gauss distribution for every position of the scatterer (white and orange), with the variance taken from the Fisher information.

5.1. Model Results

The general model architecture consists of some dense blocks to condense the information and some dense layers. The explicit code to generate the model can be found in Appendix B.3. It shows a few hyperparameters, such as number of dense layers and number of neurons per dense layer. During experimenting, many iterations of the model were tested and the hyperparameters for the model, which led to the results in this chapter, are documented in B.4.

The model was trained with 20,000 speckles per position, a validation set of size 500 was used for the validation loss and 4,500 speckles per position were kept in a test dataset.

Due to the weights of the model being randomized every time a training run is started, the same hyperparameters might lead to slightly different results. However, they should all trend towards a similar mean squared error, which makes sense as the mean squared error is related to the variance of the estimator with

$$MSE(\hat{\theta}) = Var(\hat{\theta}) + Bias(\hat{\theta}, \theta)^2 \tag{43}$$

but could potentially also contain bias, which will be investigated in the following. When plotting the predictions of one of these models all together in figure 5.3 we can see a similar distribution as is figure 5.2, with seven distinct peaks at the positions of the scatterer. However, there is a deviation from the expected distribution at the edges that suggests that the model is biased.

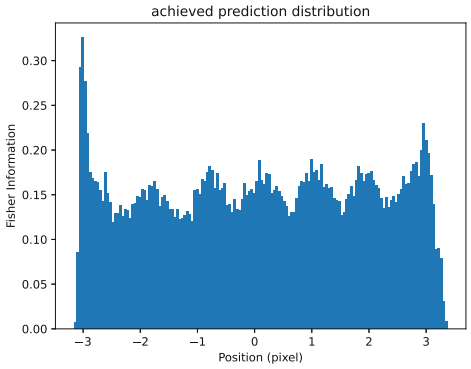

Figure 5.3: The prediction distribution achieved by the model shows distinct peaks at the possible positions for the scatterer. It also shows bias at the fringes, as no data beyond those exist, explaining some difference to figure 5.2.

The occurrence of bias in the trained models makes sense as the general model has the ability to learn that θ only lies in $[-3, 3]$ and there would be no reason to predict a value outside this region. This necessarily introduces bias, which has to be treated as shown in 3.1.1 to obtain the correct CRLB, but this form of bias is certainly a benefit for model performance on this specific dataset as it stems from a logical conclusion that is based on the training data.

5.2. Comparison between Cramér Rao Bound and Model Performance

To compare the model performance to the CRLB, we need to calculate the bias for the model predictions. This is done with the test dataset, where each speckle is fed to the model and the prediction is recorded alongside the desired ground truth. After this, the bias can easily be calculated for every position of the scatterer to obtain the bias function. In this section, we will discuss two models, where due to the randomized weights at the start of the training, one ended up with lower bias for the middle position and one ended up with higher bias.

After computing the predictions for the test set and the bias we find for the lower bias model, that the mean squared error is 0.1332, which is very close to the optimal variance. The total comparison looks like this:

position	unbiased CRLB	biased CRLB	prediction standard deviation
-2.0		0.329	0.363
-1.0		0.359	0.373
0.0	0.361	0.357	0.358
1.0		0.359	0.372
2.0		0.325	0.369

Table 3: The model with lower bias for the middle position achieved a standard deviation very close to the optimum for the middle position. The unbiased CRLB is equal for all positions, as it only depends on the TiO_2-concentration, while the bias can vary due to the machine learning model. Position -3.0 and 3.0 are not included, as no derivative for the bias of these positions can be calculated due to them being endpoints.

In the table, we see that the bias for the middle position is very low, as the CRLB needs almost no correction, and the model is able to achieve a variance close to the biased CRLB. Towards the edges, we see that the bias increases, as the biased CRLB decreases, but the model variance stays in a similar range as the middle peak. The distribution of the middle position and the biased CRLB is plotted in figure 5.4, where we can see visualized that the predictions are centered very well around the ground truth of $\mu = 0.0$ and the standard deviation of the predictions is very close to both unbiased and biased CRLB.

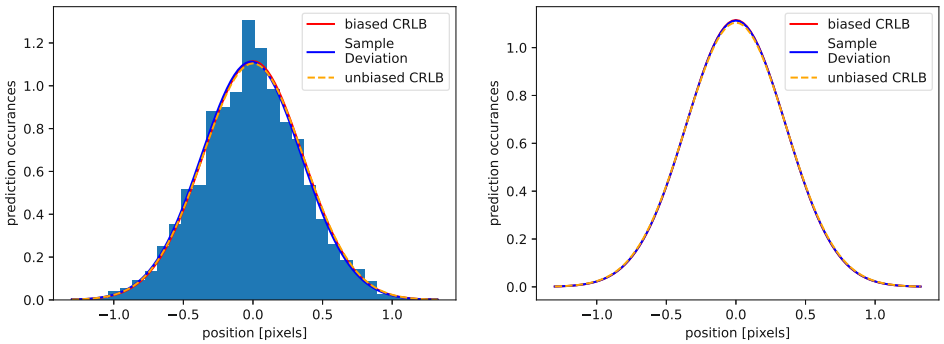

Figure 5.4: Isolating the test set predictions from a model iteration which shows low bias to a sample where the ground truth position is $x = 0.0$, we can see that the standard deviation for this sample is almost equal to both the unbiased Cramér Rao and the biased Cramér Rao bound, which can be seen by their normal distributions overlapping (right). The histogram of predictions is also centered very well around the ground truth of $\mu = 0.0$ (left). The model was trained with 20000 simulted speckles and the test set consisted of 4500 speckles per position

For the model with higher bias, we see a mean squared error of 0.147, but focusing on the middle position this bias seems to help the variance surpass the unbiased CRLB as seen in the following table:

position	unbiased CRLB	biased CRLB	prediction standard deviation
-2.0		0.322	0.344
-1.0		0.341	0.358
0.0	0.361	0.339	0.346
1.0		0.334	0.346
2.0		0.321	0.348

Table 4: The higher bias model surpassed the unbiased CRLB by quite some margin, but is not able to surpass the bias-corrected CRLB. The unbiased CRLB is equal for all positions, as it only depends on the TiO_2-concentration, while the bias can vary due to the machine learning model. Position -3.0 and 3.0 are not included, as no derivative for the bias of these positions can be calculated due to them being endpoints.

The distribution of the middle position and the biased CRLB is plotted in figure 5.5 and shows that the standard deviation of the predictions surpasses the unbiased CRLB and approaches the biased CRLB, but we can also see that the bias for the middle position leads to the distribution not being centered around the ground truth at $\mu = 0.0$.

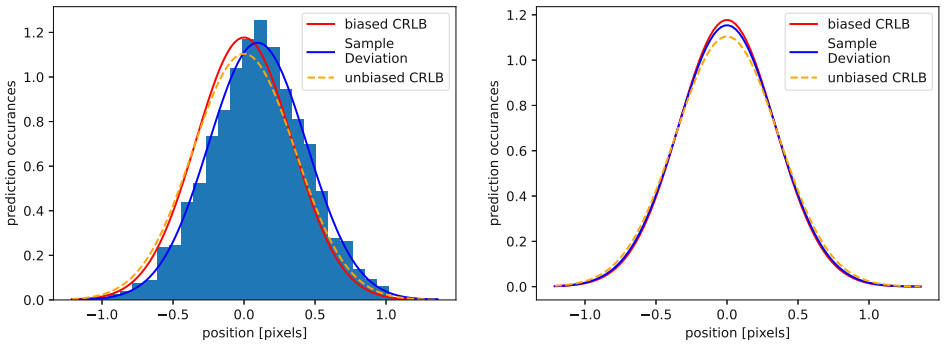

Figure 5.5: Isolating the test set predictions from a model iteration which shows high bias to a sample where the ground truth position is $x = 0.0$, the standard deviation of the sample of predictions surpasses the biased Cramér Rao bound and approaches the unbiased Cramér Rao (right). The sample of predictions is, however, not centered on the ground truth of $\mu = 0.0$ (left). Note: The distributions on the right were all plotted around the mean $\mu = 0$ to compare them, while the histogram of the predictions shows that some bias exists as it is not centered. The model was trained with 20000 simulted speckles and the test set consisted of 4500 speckles per position

We trained eight more models to the point where the validation error was no longer decreasing and compared if the Cramér Rao Lower Bound was violated for any position of the scatterer. As the bias is different for every position in every model, the bias correction of the Cramér Rao Lower Bound has to be calculated for each combination individually. To compare all standard deviations σ of the predictions, we normalized to a relative distance with

$$d = \frac{\sigma - \sigma_{CRLB}}{\sigma_{CRLB}} \tag{44}$$

The results can be seen in figure 5.6, once with bias-corrected Cramér Rao Lower Bound and once without the bias correction. For the bias-corrected case, we can see that nothing could get beyond the Cramér Rao Lower Bound. For the majority of cases, the models got very close to the limit, with half the cases showing a relative distance between 0.025 and 0.075. The closest combination got to a relative distance of 0.00003, but does not surpass the limit. The bias correction leads in general to a lower Cramér Rao Lower bound, which can be seen in the fact that without the bias correction the models are likely to go beyond the limit, again highlighting the importance of the correction, when close to the limit.

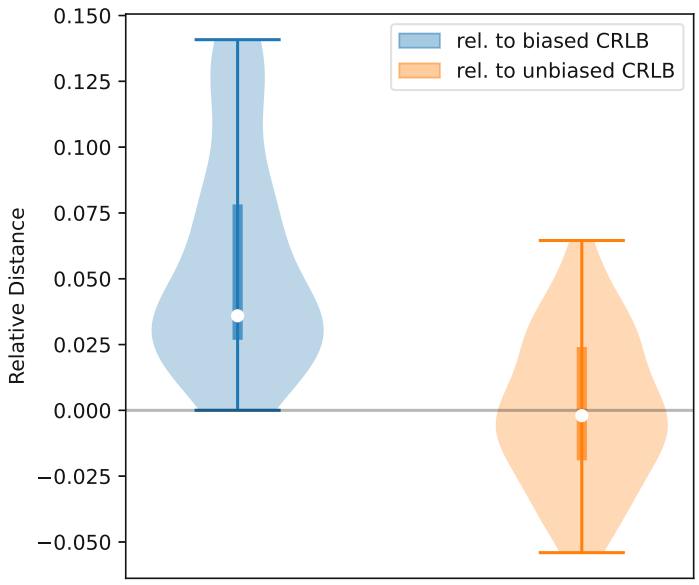

Figure 5.6: Showing the relative distance to the CRLB for 10 randomly initialized models with 5 possible positions of the scatterer each. The relative distance to the CRLB for each model-position-combination was calculated with $d = \frac{\sigma - \sigma_{CRLB}}{\sigma_{CRLB}}$ using the respective CRLB. For the bias-corrected case the majority was close, but not surpassing the limit, while going beyond was possible without the bias correction. White dots show the median values of the relative distance, vertical bars indicate the corresponding first and third quartiles, the whiskers show the minimum and maximum, and the coloured areas show the associated histograms. The horizontal line at zero represents the respective Cramér-Rao Lower Bound.

In conclusion, we can say that a machine learning model can not surpass the biased CRLB, which makes it a limit for the precision of a machine learning model. The bias needed to surpass the unbiased CRLB may affect the accuracy of the model negatively. Some bias arise due to a natural limitation of the parameter space in the data, like seen at the edges in our parameterspace and may be welcome to improve model performance. Other bias, however, may stem from overfitting on the specific training set. So while the biased CRLB is a good benchmark for the variance the model can achieve, it should also be accompanied by an analysis of the biases, to make sure that the model is predicting the parameter as expected. In the end, this relates again to one of the fundamental problems of supervised learning, the bias-variance-tradeoff, where a lower variance can be achieved through the introduction of bias, but the bias may introduce a fundamental error that is reflected in every prediction. Another factor that may impact the usefullness of this method as a benchmark is the number of samples needed to calculate the Fisher information. Although we learned in chapter 3 that for the Fisher information of the

sample treated in this chapter to be below 5% error we needed 5000 samples, to get the error very close to zero a sample size of 75000 was needed, because of diminishing returns. Meanwhile, as seen figure 5.6, the machine learning model was able to come close to the Cramér Rao Lower Bound while using only 20000 samples per position for training. In some situations it may not be feasible to create that many more samples, just to verify if the model could get better, especially when the model might already perform close to the Cramér Rao Lower Bound, and a high accuracy for the Fisher information is needed. Another limiting factor is the accessibility of the derivative of the probability density function with regard to the parameter, that is needed to compute the fisher information. In these simulations these could easily be generated and computed with a finite differences approach, but in experimantal settings one might not be able to accurately change a parameter that should be estimated. So finally we can say, that this approach can be used to benchmark a machine learning model, although the practical usefulness depends on the caveats mentioned above. With the generality of the Cramér Rao Lower Bound beyond just this physical system treated here, and the approach using independent component analysis, developed by Max Weimar, many use cases may potentially benefit from being able to find a limit to the achievable precision.

6. Summary and Outlook

In the first chapter we analyzed the experimental data and showed the influence of ballistic photons on the probability distribution function and saw that there are still more factors at play to get to the true distribution function that models the experimental data. We used the insights we gained as a starting point for our simulations of fully developed speckles, both with and without correlations.

In the second chapter we discussed the Fisher information and derived a way to calculate it for the case of fully developed speckles without correlations and attempted to remove the correlations by resampling and showed that this process would remove too much information for the experimental data. We also discussed the impact of noise on the Fisher information and showed that for our case the noise can be reduced by simulating a sufficient amount of speckle patterns for every θ value and how many speckle patterns are necessary for a desired signal-to-noise ratio.

In the third chapter, we built up all the basics needed to understand the machine

learning model used in [5] and how to extend it for our use-case. An important part here was the evaluation of whether to really build on top of the UNet model or only use parts of it that easily allows for an End-to-End training approach. The End-to-End approach was found to be faster, less resource intensive, more accurate and easier to work with than the combined model, which saved time over the whole project.

In the last chapter, we then used all the tools and insights gathered to explicitly show that the Fisher information can indeed be a good benchmark for how low of a variance a machine learning model can achieve, but should also be accompanied by an analysis of the biases of the model. We have explicitly shown this for uncorrelated fully developed speckle patterns, which were simulated, and found it to be useful with some caveats. Future work that should also show this for data with correlations will be treated in [12] where due to the more general approach for calculating the Fisher information devised by Max Weimar the theory can be directly tested on experimental data.

For other future work, thought should be put into the most prominent weak point that this theory brings. We saw that the calculation for the Fisher information is very sensitive to errors in the derivative with respect to θ. In the simulation or a laboratory setting where θ is a highly controllable parameter, obtaining this derivative might not be that difficult, but real world applications of this theory will need a good way to estimate these derivatives accurately without having control over θ. Another point to further investigate would also be to determine the amount of information that is carried by ballistic photons vs the correlations within the speckle pattern. If the claims of [16] are true and the machine learning models used in [5] depend mostly on ballistic photons, then the speckle noise in the speckle pattern would only be noise, not carrying any information about the scatterer. Indeed, in the beginning of the project, we started out without accounting for correlations between pixels and the pixel-wise sum of the Fisher information yielded a Cramér Rao Bound that would almost be reached by the machine learning model, as we would expect. When we later noticed and tried to correct for correlations by dividing by speckle grain size, the model would reach beyond the corrected CRLB which should be impossible. This suggests that a majority of the information is carried by ballistic photons, which would give unique information at every pixel, instead of the speckle where only every speckle grain, across many pixels would deliver unique information. One approach to investigate this further could be to compare the calculation with the independent component analysis with a calculation where we intentionally over-count, as the difference could be an indication how much information

is really carried by the speckle correlations.

7. Acknowledgments

First and foremost, I would like to thank my supervising professor Stefan Rotter for giving me the chance to be part of a project group that produced a scientific paper, giving me lots of insights into academia. Naturally, I also want to extend these thanks to the project group explicitly: Dorian Bouchet, Ilya Starshynov and Daniele Faccio, for all the support I got, and Lukas Rachbauer and Max Weimar especially for all the discussions and feedback regarding this thesis. Furthermore, I want to thank all the friends and family who motivated me along the way, be it with advice, support or writing sessions in the library. Who knows how much longer it would have taken without you.

Appendix

A. Hardware System

The hardware system used in this thesis had the following specifications:

- AMD Ryzen 7 3700X
- Nvidia GeForce GTX 1070 8GB DDR5
- 64GB DDR4 Memory

B. Code Examples

Here are some simplified examples for the code needed to achieve the results in this thesis. You need the python packages tensorflow in version 2.16.0 (follow their instructions on installing it with gpu support if you want to train a neural network), numpy in version 1.26.4 nand matplotlib in version 3.10.0.

B.1. Simulating Speckle Pattern

Simulating fully developed speckle patterns with or without correlations from an intensity distribution previously saved as numpy array to mean_speckle.npy can be done with the following code:

```
import numpy as np
from matplotlib import pyplot as plt

def draw_cicle(image_size, circle_diamiter):

    circle = np.zeros(image_size,dtype=np.bool_)
    center = np.array(circle.shape)/2.0

    for y in range(image_size[0]):
        for x in range(image_size[1]):
            circle[y,x] = (y - center[0])**2 + (x - center[1])**2 <
   circle_diamiter **2
```

```
    return(circle)

num_phasors = 100
filter_size = 10 #pixel (radius of circle)

mean_speckle = np.load("mean_speckle.npy") #load the intensity
    distribution from a saved numpy array

phi = np.random.uniform(-np.pi,np.pi,(num_phasors, mean_speckle.shape
    [1], mean_speckle.shape[0]))
phasorfield = np.sum(np.exp(1j * phi) * np.sqrt(mean_speckle)/np.sqrt(
    num_phasors), axis = 0)

fourier = np.fft.fftshift(np.fft.fft2(phasorfield))

mask = draw_cicle(mean_speckle.shape, filter_size)

masked = fourier*mask

filtered = np.square(np.absolute(np.fft.ifft2(np.fft.ifftshift(masked))
    ))

fig, axs = plt.subplots(1,1)
axs.imshow(filtered)

plt.show()
```

B.2. Fisher Information Signal and Noise

The following code can be used to calculate the Fisher information for fully developed speckles without correlation and the proportion that the noise makes up compared to the signal. Four files with speckle patterns have to be supplied as numpy arrays with the dimensions [specklenumber, y, x].

```
import numpy as np
from matplotlib import pyplot as plt

Fake_Fisher_Information = []
Fisher_Information = []

```

```
delta = 1 #pixel

# how many speckles are sampled for the signal to noise plots below
position_durations_plot = [1000, 5000, 25000]

fig, ax = plt.subplots(1,3, figsize=(6,2))

for index, sample in enumerate(position_durations_plot):

    #load one position sample for x+dx and x-dx and two different samples for x
    mean_intensity = [
        np.mean(np.load(intensity0.npy)[0,:sample],axis = 0),   #load sample for position 0
        np.mean(np.load(intensity1_0.npy)[0,:sample],axis = 0), #load first sample for position 1
        np.mean(np.load(intensity1_1.npy)[0,:sample],axis = 0), #load second sample for position 1
        np.mean(np.load(intensity2.npy)[0,:sample],axis = 0),   #load sample for position 2
    ]
    mean_intensity = np.array(mean_intensity)

    f_deriv = (mean_intensity[1]-mean_intensity[2])/(2.0*delta)
    deriv = (mean_intensity[3]-mean_intensity[0])/(2.0*delta)

    I_avg = 0.5*(mean_intensity[1]+mean_intensity[2])

    f_F = np.square(f_deriv)/(np.square(I_avg))
    F = np.square(deriv)/(np.square(mean_intensity[1]))

    Fake_Fisher_Information.append(np.sum(f_F))
    Fisher_Information.append(np.sum(F))

    bins = np.linspace(0,max(F.max(),f_F.max()),100)

    ax[index].hist(F.flatten(), bins, alpha = 0.65, label = 'Signal', color = "b")
    ax[index].hist(f_F.flatten(), bins, alpha = 0.65, label = 'Noise', color = "r")
```

```
40
41      ax[index].set_yscale("log")
42      ax[index].legend()
43      ax[index].set_xlabel("Fisher-Information")
44      ax[index].set_ylabel("Pixel")
45
46  fig.tight_layout()
47  plt.show()
```

B.3. Tensorflow Model

The following code is used to generate the tensorflow model, based on a few hyperparameters. To better understand the model.summary() and tf.keras.utils.plot_model(model, "filename.png") might help.

```
1  import tensorflow as tf
2  from tensorflow.keras.layers import Input, Dropout, Conv2D,
       SeparableConv2D, Activation, MaxPooling2D, Concatenate,
       BatchNormalization, Flatten, Dense
3  from tensorflow.keras.regularizers import L2
4  from tensorflow.keras.initializers import HeUniform
5  from tensorflow.keras.models import Model
6  from tensorflow.keras.optimizers import Adam
7
8  def get_model(db_depth, db_filter_shape, db_growth, db_bypass,
       dropout_rate = 0.5,speckle_size = (32, 64)):
9      weight_decay = 1E-4
10     inputs = Input((speckle_size[0],speckle_size[1], 1))
11     x = inputs
12
13     for i in range(len(db_depth)):
14         input_list = [x]
15         for j in range(db_depth[i]):
16             x = BatchNormalization(axis=3,
17                                    gamma_regularizer=L2(weight_decay),
18                                    beta_regularizer=L2(weight_decay))(x)
19             x = Activation('relu')(x)
20             x = Conv2D(db_growth[i], db_filter_shape[i],
21                        kernel_initializer=HeUniform(),
22                        padding="same",
```

```
                        kernel_regularizer=L2(weight_decay))(x)
            if dropout_rate:
                x = Dropout(dropout_rate)(x)
            input_list.append(x)
            x = Concatenate(axis=3)(input_list)
        bypass = SeparableConv2D(db_bypass[i], 1, activation='relu',
    padding='same')(x)
        transition = BatchNormalization(axis=3,
                            gamma_regularizer=L2(weight_decay),
                            beta_regularizer=L2(weight_decay))(x)
        transition = Activation('relu')(transition)
        transition = Conv2D(db_depth[i]*db_growth[i], db_filter_shape[i
    ],
                        kernel_initializer=HeUniform(),
                        padding="same",
                        kernel_regularizer=L2(weight_decay))(transition)
        x = Concatenate(axis=3)([transition, bypass])
        x = MaxPooling2D(pool_size=(2, 2))(x)

    x = SeparableConv2D(128, 1, activation='relu', padding='same')(x)
    flat = Flatten()(x)

    dense1 = Dense(128, activation='relu', kernel_initializer=HeUniform
    ())(flat)
    dense2 = Dense(64, activation='relu', kernel_initializer=HeUniform
    ())(dense1)
    out = Dense(1, activation='linear')(dense2)

    opt = Adam(learning_rate = 1e-3, amsgrad=True)

    model = Model(inputs=inputs, outputs=out)
    model.compile(optimizer=opt, loss='mse')
    return model

db_depth = (3,3,4)
db_filter_shape = ((2,2),(2,2),(2,2))
db_growth = (64,96,128)
db_bypass = (128,128,192)
dropout_rate = 0.5

```

```
59 model = get_model(db_depth, db_filter_shape, db_growth, db_bypass,
       dropout_rate, pattern_size)
60
61 tf.keras.utils.plot_model(model, to_file="model.png")
62
63 model.summary()
```

B.4. Final model hyperparameters

parameter	value
db_depth	(3,3,4)
db_filter_shape	((2,2), (2,2), (2,2))
db_growth	(64,96,128)
db_bypass	(128,128,192)
dropout_rate	0.5

Table 5: Model parameters used in the final convolutional neural network

References

[1] J. Schmidhuber, "Deep learning in neural networks: An overview," *Neural Networks*, vol. 61, pp. 85–117, jan 2015.

[2] J. Jumper, R. Evans, A. Pritzel, T. Green, M. Figurnov, O. Ronneberger, K. Tunyasuvunakool, R. Bates, A. Žídek, A. Potapenko, *et al.*, "Highly accurate protein structure prediction with alphafold," *nature*, vol. 596, no. 7873, pp. 583–589, 2021.

[3] T. Xie and J. C. Grossman, "Crystal graph convolutional neural networks for an accurate and interpretable prediction of material properties," *Physical review letters*, vol. 120, no. 14, p. 145301, 2018.

[4] G. Litjens, T. Kooi, B. E. Bejnordi, A. A. A. Setio, F. Ciompi, M. Ghafoorian, J. A. Van Der Laak, B. Van Ginneken, and C. I. Sánchez, "A survey on deep learning in medical image analysis," *Medical image analysis*, vol. 42, pp. 60–88, 2017.

[5] Y. Li, Y. Xue, and L. Tian, "Deep speckle correlation: a deep learning approach toward scalable imaging through scattering media," *Optica*, vol. 5, p. 1181, sep 2018.

[6] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[7] P. Y. Simard, D. Steinkraus, J. C. Platt, *et al.*, "Best practices for convolutional neural networks applied to visual document analysis.," in *Icdar*, vol. 3, Edinburgh, 2003.

[8] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Advances in neural information processing systems*, vol. 25, 2012.

[9] J. Yu, Z. Wang, V. Vasudevan, L. Yeung, M. Seyedhosseini, and Y. Wu, "Coca: Contrastive captioners are image-text foundation models," 2022.

[10] D. Bouchet, S. Rotter, and A. P. Mosk, "Maximum information states for coherent scattering measurements," *Nature Physics*, vol. 17, p. 564–568, Jan. 2021.

[11] A. P. Mosk, A. Lagendijk, G. Lerosey, and M. Fink, "Controlling waves in space and time for imaging and focusing in complex media," *Nature photonics*, vol. 6, no. 5, pp. 283–292, 2012.

[12] I. Starshynov, M. Weimar, L. M. Rachbauer, G. Hackl, D. Faccio, S. Rotter, and D. Bouchet, "Model-free estimation of the cramér-rao bound for deep-learning imaging in complex media," *in review*, 2024.

[13] A. I. of Physics, "Speckle: Friend or foe?," *AIP conference proceedings*, vol. 1537, no. 1, p. 5, 2013.

[14] J. Goodman, *Speckle Phenomena in Optics: Theory and Applications*. Roberts and Company Publishers, 2007.

[15] J. C. Dainty, *Laser speckle and related phenomena*, vol. 9. Springer science & business Media, 2013.

[16] X. Zhan, J. Gao, Y. Gan, C. Song, D. Zhang, S. Zhuang, S. Han, P. Lai, and H. Liu, "Roles of scattered and ballistic photons in imaging through scattering media: a deep learning-based study," *arXiv preprint arXiv:2207.10263*, 2022.

[17] R. Verma and J. Ali, "A comparative study of various types of image noise and efficient noise removal techniques," *International Journal of advanced research in computer science and software engineering*, vol. 3, no. 10, 2013.

[18] S. Rotter and S. Gigan, "Light fields in complex media: Mesoscopic scattering meets wave control," *Reviews of Modern Physics*, vol. 89, no. 1, p. 015005, 2017.

[19] R. Carminati, "Waves in complex media," *Lecture notes*, 2018.

[20] J. W. Goodman, *Simulating Speckle with Python*. SPIE Press, 2024.

[21] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. F. del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E. Oliphant, "Array programming with NumPy," *Nature*, vol. 585, pp. 357–362, Sept. 2020.

[22] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. J. Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. J. Carey, İ. Polat, Y. Feng, E. W. Moore, J. VanderPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa, P. van Mulbregt, and SciPy 1.0 Contributors, "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python," *Nature Methods*, vol. 17, pp. 261–272, 2020.

[23] S. M. Kay, *Fundamentals of Statistical Signal Processing: Estimation Theory*. Prentice Hall, 1993.

[24] R. A. Fisher, "On the mathematical foundations of theoretical statistics," *Philosophical transactions of the Royal Society of London. Series A, containing papers of a mathematical or physical character*, vol. 222, no. 594-604, pp. 309–368, 1922.

[25] V. Namrata, "Detection and estimation theory," *Lecture notes*, 2018.

[26] R. B. Nelsen, *An introduction to copulas*. Springer, 2006.

[27] Starshynov, Ilya and Rachbauer, Lukas Michael and Hackl, Günther and Faccio, Daniele and Bouchet, Dorian. private discussions, 2021.

[28] L. M. Rachbauer, *Inverse Scattering in One-dimensional Random Media Using Deep Learning*. Institute for Theoretical Physics, TU Vienna, 2019.

[29] P. Mehta, M. Bukov, C.-H. Wang, A. G. Day, C. Richardson, C. K. Fisher, and D. J. Schwab, "A high-bias, low-variance introduction to machine learning for physicists," *Physics Reports*, vol. 810, pp. 1–124, may 2019.

[30] O. Dürr, B. Sick, and E. Murina, *Probabilistic deep learning: With python, keras and tensorflow probability*. Manning Publications, 2020.

[31] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2014.

[32] R. P. Lippmann, "An introduction to computing with neural nets," *ACM SIGARCH Computer Architecture News*, vol. 16, no. 1, pp. 7–25, 1988.

[33] G. Hinton, "Improving neural networks by preventing co-adaptation of feature detectors," *arXiv preprint arXiv:1207.0580*, 2012.

[34] R. Chandel and G. Gupta, "Image filtering algorithms and techniques: A review," *International Journal of Advanced Research in Computer Science and Software Engineering*, vol. 3, no. 10, 2013.

[35] G. Huang, Z. Liu, L. van der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," 2016.

[36] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," 2015.

[37] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 3431–3440, 2015.

[38] L. Deng, "The mnist database of handwritten digit images for machine learning research," *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 141–142, 2012.

[39] F. Chollet *et al.*, "Keras." https://keras.io, 2015.

[40] T. Developers, "Tensorflow." https://tensorflow.org, May 2022.