Palacký University Olomouc
Faculty of Science

Department of Optics

Deep neural networks for polarimetric and imaging applications

Bachelor's Thesis

Dominik Vašinka

Palacký University Olomouc
Faculty of Science

Department of Optics

Deep neural networks for polarimetric and imaging applications

Bachelor's Thesis

Author:	Dominik Vašinka
Study programme:	B1701 Physics
Field of study:	General physics and mathematical physics
Form of study:	Full-time
Supervisor:	RNDr. Miroslav Ježek, Ph.D.

Thesis submitted on: .....

Univerzita Palackého
Přírodovědecká fakulta

Katedra optiky

Vícevrstvé neuronové sítě s aplikacemi v polarimetrii a zobrazování

Bakalářská práce

Autor: Dominik Vašinka
Studijní program: B1701 Fyzika
Studijní obor: Obecná fyzika a matematická fyzika
Forma studia: Prezenční
Vedoucí: RNDr. Miroslav Ježek, Ph.D.

Práce odevzdána dne: .....

Abstract

Artificial neural networks are nowadays widely used algorithms, successfully utilized in many challenging applications. Unlike the von Neumann architectures, they do not require specifying the exact rules for processing the input data, which allows for achieving sufficient results even in applications, where all analytical models fail.

In this thesis, the possibility of utilizing these networks for optics related applications is explored. Firstly, we use the deep neural networks for modelling the transformation of the polarization state of light propagated through twisted nematic liquid crystal modules. Then, convolutional neural networks enhancing images affected by the diffraction limit and noise are utilized for counting point-like emitters.

Keywords

machine learning, deep neural networks, convolutional neural networks, polarimetry, twisted nematic liquid crystals, image enhancement, emitter counting.

Acknowledgments

In the first place, I would like to deeply thank my supervisor RNDr. Miroslav Ježek, Ph.D. for his guidance, patience and all the advice given to me. Many thanks go to Mgr. Bc. Martin Bielak for providing me with experimental data and Mgr. Jan Provazník for developing and managing computational grid at the Department of Optics. I would also like to thank all my lectors for sharing their knowledge with me, and my colleagues for a friendly environment. And last but not least, a huge thanks goes to my family for their enormous support.

Dominik Vašinka

Declaration

I hereby declare that I have written this Bachelor's Thesis, and performed all the presented research tasks by myself, while being supervised by RNDr. Miroslav Ježek, Ph.D. I also state that every resource used is properly cited. I agree with the Thesis being used for teaching purposes and being made available at the website of the Department of Optics.

Signed in Olomouc on ....

Dominik Vašinka

Contents

Chapter 1

Introduction

Behind various current technologies, including email spam filtering and autonomous vehicles, are machine learning algorithms. Even though the term machine learning was first used much earlier, its big success started only about a decade ago, which was allowed by discovering new optimization methods and computing technology becoming progressively more efficient, resulting in a decrease of the required computational time.

Machine learning algorithms are different from von Neumann architectures, where the data are provided along with rules for processing them to receive an output. In machine learning, the algorithm uses the data to find rules and patterns. In other words, these algorithms learn from the data, thus the term machine learning. Its methods divide into three primary groups – supervised, unsupervised and reinforcement learning. If we provide the algorithm not only with inputs but also related outputs, we are talking about supervised learning, typically regression and classification. The supervised algorithm learns a mapping between the input and its output, capable of predicting outputs for new data. Unsupervised algorithms provided only with the inputs are supposed to find patterns in the data. For example, they can be used to detect anomalies or to categorize data into groups. And last but not least, reinforcement learning is based on determining the best actions to take in a particular situation to maximize a reward, as finding a way through a maze. They are typically used for autonomous vehicles and in games like chess or Go.

An artificial neural network is an algorithm that learns to perform a nonlinear transformation on its inputs to receive desired outputs. Even though it can be applied to all three branches of machine learning, we will focus on artificial neural networks as part of supervised learning, i.e. with known input and output. They are inspired by the structure and processes in biological brains but do not represent a model of it. The brain consists of an enormous number of neurons connected via synapses which they use to communicate. During the learning process, these synapses alter, some become stronger and other weaker. The same concept is behind the principles of artificial neural networks. First, we provide the network with a dataset representing experience, for example, spam and non-spam emails, each together with its label whether or not it is a spam. Then, the network learns by finding patterns in these data and altering its synapses correspondingly. Once the network has seen enough examples, it is capable of predicting a possibility for a newly received email to be spam.

Artificial neural networks have successfully spread to many fields, such as cybersecurity, speech recognition, banking, business or routing systems and science is no exception. In medicine, these networks are used for analysing images from X-ray and MRI scans [1, 2], as well as for tumour classification [3] and discovering drugs [4, 5]. As another example, chemists successfully applied them for predicting material properties, for example, of polymers and for analysing spectroscopic data [6]. Many applications can also be found in physics, for example, predicting the energies of atomic nuclei achieving accuracy comparable to state-of-the-art methods but with a lower computational cost [7]. Cosmologists utilized neural networks in estimating the photometric redshift [8], finding gravitational lenses [9], predicting cosmological constraints [10] or classifying of several astronomical objects like supernovas [11, 12]. Neural networks can also be utilized in quantum physics, where they are efficiently used as a classifier of the quantum entanglement without the full information about the states [13], as well as a classifier distinguishing between single and non-single quantum emitters based on their sparse autocorrelation data [14]. Another application can be found in the quantum state tomography [15, 16, 17]. For example, the authors of [15] combine a local-measurement-based QST with artificial neural networks to effectively reconstruct the full quantum state. These are just a few examples as significantly more exist.

Lets now focus on optics, where neural networks found their applications, for example, for classifying objects based on polarimetric radar measurements [18] or for exploring the properties of liquid crystals based on their optical images [19]. However, most of the optics related applications can be found in imaging, where the typical applications consist of image enhancement, image reconstruction, noise reduction and detection, classification and counting of objects. Image reconstruction stands for restoring an original image from a speckle pattern created by propagation of light through scattering media. Neural networks have been used to determine the inverse transmission matrix of multimode fibre from the speckle pattern [20], but also to directly recover and classify the original image [21]. Another neural network reconstructs images of speckle patterns originating from a wide range of diffusers [22]. The image enhancement aims to reduce the limitations of the resolution of optical systems caused by the diffraction limit alongside with optical aberrations. Artificial neural networks have proved to be an effective method for enhancing the resolution of images. The authors of [23] applied a neural network to improve a resolution with a single image and achieved higher output quality compared to state-of-the-art methods. An example of direct application of an image-enhancing network is fluorescence microscopy, where it was used for obtaining super-resolution images under low signal-to-noise ratio [24] as well as significantly faster by requiring up to two orders of magnitude fewer frames than the state-of-the-art methods [25]. Many microscopic objects are so small that diffraction limit does not allow to determine the kind or amount of objects in a picture. These image-enhancing networks can effectively be modified for classifying or counting the objects in images, for example, type of cells [26], or for tracking a single particle as well as multiple particles at the same time under illumination conditions, where classical algorithms tend to fail [27]. A wide range of other examples could be mentioned, as every day, many new applications are discovered, where neural networks overcome the traditional algorithms achieving faster and more accurate results.

The goal of this thesis is to show the benefits of artificial neural networks to optics related applications. In the first chapter, deep neural networks are introduced – their structure, the processes behind them and overfitting preventions. These algorithms are then applied to optic related applications. Namely, the second chapter deals with the transformation of the polarization state of light after passing through twisted nematic liquid crystal devices based on applied voltages. The third chapter focuses on imaging application, where a convolutional neural network is presented as an effective method for image reconstruction, noise reduction and emitter counting. In the end, a discussion of the obtained results and possible future extensions takes place.

Chapter 2

Neural network

2.1 Neural network fundamentals

The working principles of a neural network (NN) emulate operations of the human brain. Basic units, called neurons, are arranged into layers. In a fully, also called densely, connected NN each neuron is connected to all neurons in the following layer. The first layer of NN, called an input layer, represents an entrance for the data and the last one is called an output layer, which results in the final output. It is common to add so-called hidden layers between them, creating a deep neural network (DNN), see Fig. 2.1. A NN without any hidden layers might be limited in its transformations, but adding a single hidden layer with a finite number of neurons allows it to approximate all continuous functions [28]. It is also possible to include one additional neuron to each layer that is not connected to any of the previous neurons. This type of neuron is called bias and always wields a value of 1.

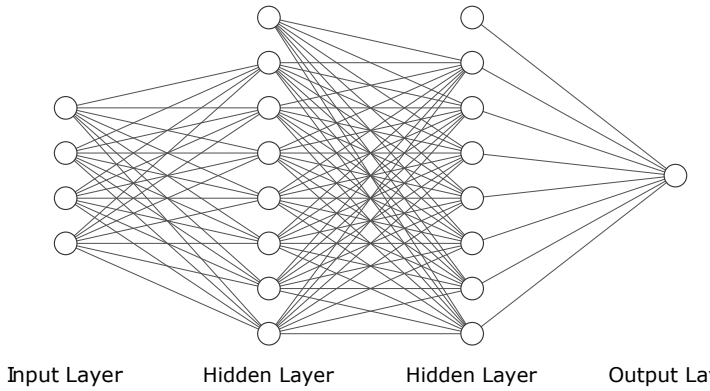

Figure 2.1: An example of a densely connected neural network with three neurons plus a bias in an input layer, two hidden layers with seven neurons plus a bias each, and a single neuron in an output layer. The input neurons could represent age, mileage and horsepower of a car, and the output neuron could be its price. Such a trained network would then predict the price of a car based on its parameters.

For a future description of a DNN structure, I will use a notation M-N(O)-P, where M stands for the number of neurons in the input layer, O is the number of hidden layers each with N neurons and P is the number of neurons in the output layer. All these numbers are bias excluded. The structure shown in Fig. 2.1 is then described as 3-7(2)-1.

Each connection has a certain value assigned to it called weight. These are at initialization random but altered as the NN sees learning examples. Via these weights, the network performs a transformation on the input data. The value v_j of the neuron j is calculated from an impulse z_j received by this neuron. The impulse is a sum of all neuron values v_i in the previous layer multiplied by their corresponding weight w_{ij},

$$z_j = \sum_{i=1}^{N} w_{ij} v_i. \tag{2.1}$$

This weighted sum is then used as an input to a so-called activation function f whose output is the value v_j of the considered neuron,

$$v_j = f(z_j). \tag{2.2}$$

An activation function can be a simple linear function, but the network would be able to perform only linear transformations on its dataset. Usually, the applications require a more complex transformation than just linear. Even though many non-linear functions are commonly used, the most popular is ReLU (rectified linear unit)

$$f(z) = \max(0, z), \tag{2.3}$$

due to its ability to learn very quickly even in a structure with many layers and connections [29]. Using this function is inspired by operations in brains once again. Biological neurons only activate themselves when the received nerve impulse is stronger than a certain threshold. Similarly, the values of artificial neurons with ReLU are zero if the received signal (2.1) is lower or equal to zero.

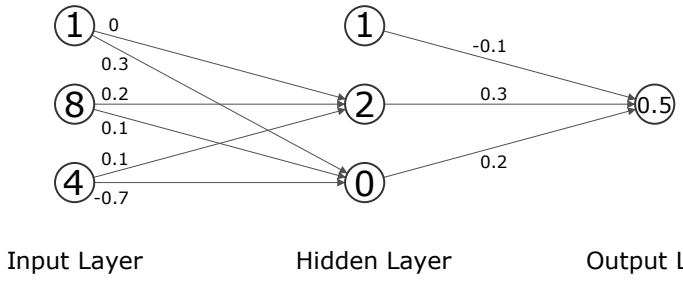

Figure 2.2: A simple 2-2(1)-1 structure of DNN using ReLU as an activation function. Each neuron has its value displayed inside – biases always wield the value 1, while other neuron values can vary – and the smaller numbers near the corresponding connection represent weights. Values of neurons in the hidden and output layer are computed using (2.1) and (2.2) where f stands for ReLU function (2.3).

Once the network has performed a transformation on all the input data, it compares the difference between received and target output in a so-called loss function to evaluate the error. The backpropagation algorithm [30] then computes a gradient of the loss function with respect to the weights and adjusts them in the opposite direction leading to a minimum. Usually, complicated functions have also so-called local minima with lower values than its neighbourhood, but higher than the global minimum. A gradient descend algorithm might get stuck in one of them, but typically, a DNN might consist of hundreds as well as hundred thousands of neurons arranged into many layers, making the vector space very high-dimensional. It was shown that local minimum does not represent a problem in high-dimensional spaces, as they are very close to the global minimum in their error value [31, 32].

Showing the whole dataset to the network once would by itself allow only one adjustment to the weights. Instead, we let the network learn from the dataset multiple times in cycles, called epochs, consisting of performing the transformation on these data and adjusting weights based on the gradient. Another useful method is to divide the dataset into smaller parts called batches. Using small batches leads to better results in the generalization quality, as it converges to a flat minimum, whereas the numerical evidence suggests that large batches tend to converge to a sharp one [33]. The network learns from these batches one at the time and adjusts the weights based on their gradients. The method of computing gradient for parts of the dataset is called mini-batch stochastic gradient descent (SGD) and is typically used along with multiple epochs [34].

2.2 Datasets

One of the most important things for training networks is dataset. Not all the available data examples will be used for training as there have to be enough data examples for testing the quality of the network. Only approximately 60 % of the data examples will be a part of the training set. The remaining data are usually equally split between validation and test set, whose purpose will be explained in what follows.

The training set should contain the biggest portion of all the available data as the backpropagation algorithm uses these examples to adjust the weights. Naturally, the error of the training set will gradually decrease, but that does not necessarily mean that the network is learning the correct mapping. It is possible for the network to memorize these examples and predict the required output. It would fail, however, in predicting output for data examples it has never seen before. This phenomenon is called overfitting, described in details in Sec. 2.3, and using a second set of data called validation helps to distinguish its occurrence.

The validation set allows comparing the training error with a reference one on the data not used for training. For example, when choosing between different types of network architectures, validation errors of these trained networks can be compared to determine which one provides a better generalization mapping. Another example is evaluating validation error continuously during training, typically after each epoch, to see how the network can generalize on new data. An example of training and validation error development after each epoch is shown in Fig. 2.3.

The test set contains the remaining available data that the NN used neither for training nor during validation. Once the network finishes the entire training, i.e. no additional weight adjusting will take place, its ability to predict correct outputs is verified on the test set. This serves as the final check whether the network can make effective predictions on completely new examples. As the validation set helps with choosing the architecture of the NN and evaluations during training, the network might also overfit the validation set to some extent. Therefore, only the test error is a valid scale for the measurement of the network generalization capability.

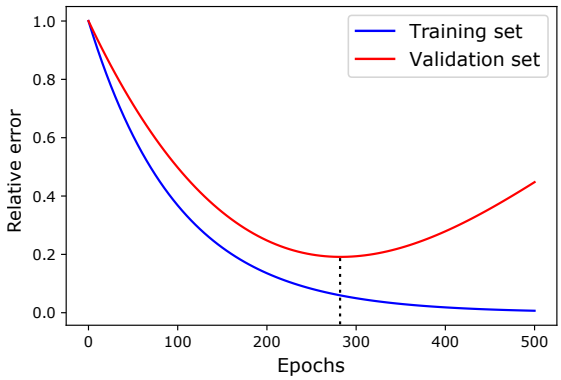

Figure 2.3: An example of a relative error development after each epoch. The blue curve representing a train set error keeps getting lower as the network learns better mapping for these examples. On the other hand, the red validation curve stars rising after a certain number of epochs due to overfitting. After that, the network only learns how to remember the training set. For this example, the ideal number of epochs is approximately 280, as to this point, no severe overfitting is present.

2.3 Overfitting

Overfitting and underfitting are problems that might occur while representing a dataset with a function. To sufficiently represent the data, we have to fit them with a function that not only has a low error on this data but can keep it low if new examples were provided. If new data points were added to the dataset and the error increased dramatically, we would talk about overfitting. In Fig. 2.4 (b), a high order polynomial function is a typical case of overfitting, as it can interpolate the data, but can not generalize well. On the other hand, if the error is always very high, no matter what are the parameters of the function, see Fig. 2.4 (a), then it might be a case of underfitting, and a more complex function is required for the data. It could also mean that a wrong type of function is used, for example, sine instead of an exponential function.

In conjunction with DNN, underfitting is not often discussed as a problem requiring complicated methods to prevent it. If a network suffers from underfitting, adding more weights by modifying the architecture can usually solve it. On the other hand, overfitting represents a serious issue. Let us continue with Fig. 2.3 from Sec. 2.2 where overfitting occurs after a certain number epochs.

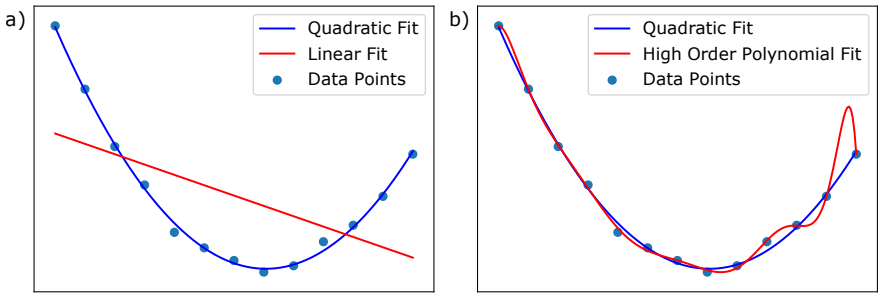

Figure 2.4: An example of a) underfitting and b) overfitting. The dataset represented by blue dots follows a quadratic function, making it a sufficient fit. a) A linear function has a high error rate which does not depend on its parameters. In other words, a linear fit is a typical example of underfitting as its error is very high no matter what the parameters are. b) A high order polynomial function can represent the dataset very well, as its error rate is practically zero. But if new examples appeared, the predictions of this function would fail, especially in the right part of the picture.

The network had seen the training examples so many times that it was able to remember them without generalizing. The training error keeps getting lower, but the validation error starts rising. One of the commonly used methods to prevent this from happening is called dropout regularization [35]. It assigns each neuron from an input and hidden layers a value representing a probability p of dropping this neuron out of the network for one learning iteration, i.e. one back-propagation cycle. Such a neuron with all its weights is then skipped and does not receive, nor emit any signals. Even during the backpropagation algorithm, it acts like it is not a part of the network, and its weights are ignored and stay identical. After each iteration, the same probability denotes whether the neuron will be left out again. For the evaluation purpose, the weights of neuron with probability p are multiplied by $(1 - p)$ value, assuring that the mean output value during training is equal to the one during testing. In [35], its ability to lower the test error by reducing the overfitting is explored and compared to other regularization methods for vision and speech recognition, as well as other applications.

An especially interesting case of overfitting appears in the context of the DNN complexity. From the classical understanding of overfitting, increasing the complexity of the function will decrease the test error rate until a so-called sweet spot is reached, see the typical regime in Fig. 2.5. Going beyond this point leads to an increase in the test error rate as the function overfits the data instead of generalizing. But if the DNN complexity increases even beyond the point of interpolation, the test error rate starts decreasing again, as shown in the over-parameterized region in Fig. 2.5 [36, 37]. The network might even reach a point, where its test error is lower than the one in the sweet spot. Such behaviour does not have an analogy in classical curve fitting and, to my knowledge, only appears in the context of DNNs. Typically, the time required for a DNN to reach minimal error increases with its complexity. However, over-parameterized networks also report a decrease in the learning time without any other modifications.

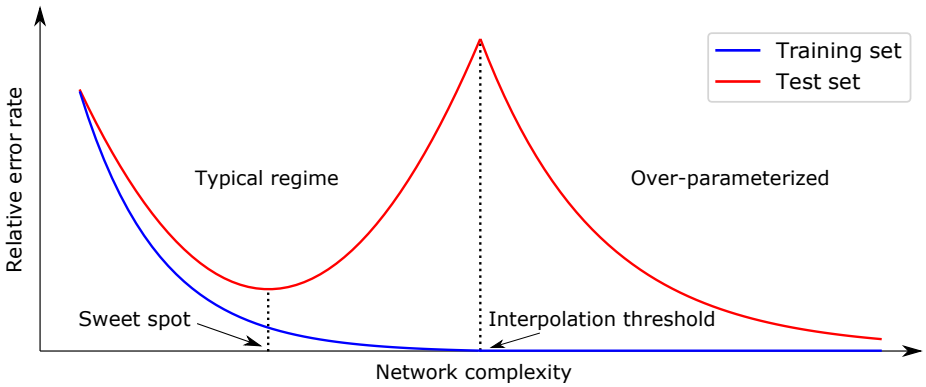

Figure 2.5: The double descent curve consisting of typical overfitting regime in the left part and over-parameterized network in the right part of the figure. The sweet spot represents the balanced under and overfitting, and the interpolation threshold is the point of zero training error. The classical understanding of overfitting only expects the existence of the typical regime, but in the context of DNN, increasing the complexity of the network beyond the interpolation threshold leads to a decrease in test error [36].

2.4 Hyperparameters and their optimization

Besides weights as parameters adjusted during training, there are also so-called hyperparameters, specified at the beginning. These stay the same during training but affect the final results. The goal is to optimize these hyperparameters and find the best possible combination evaluated on the validation set. The network then trains with this combination and its performance on the test set is the final evaluation of its quality. Among typical examples are the number of neurons, hidden layers, epochs and data examples in batches, and dropout probability. In this section, further important hyperparameters and methods of their optimization are covered.

Alphabetically, the next hyperparameter is an activation function implementing non-linearity to the network's transformation, see Sec. 2.1. Besides its typical representative, ReLU function, many others can be used, for example, a logistic function (sigmoid), arctangent, softmax, hyperbolic tangent or, to some extent, even linear function. Although ReLU is usually safe to pick due to its results and ability to learn very quickly, a different one might sometimes achieve better results. Each layer can have a different activation function, and the right choice is especially important in the output layer. For example, if you want your results to be bipolar values, ReLU is not an option, as its outputs are only non-negative values, but arctangent, multiplied by a constant to extend its range, might work just fine there. Or in the case of multi-class classification, as is deciding whether the picture is a dog, cat or mouse, softmax is the correct choice. In conclusion, for hidden layers, a ReLU function is usually an excellent pick, but you have to choose the activation function for the output layer very carefully.

Another hyperparameter also mentioned in the same section is a loss function, evaluating the error between target and obtained output. Typical neural network applications can be divided into two groups – regression and classification. In regression applications, the goal is to make the obtained result values as close to the target ones as possible and therefore loss functions like mean squared error (MSE), mean squared logarithmic error or mean absolute error (MAE) tend to work well. On the other hand, binary cross-entropy and categorical cross-entropy are common examples of the classification loss function. The aim is choosing the correct category, and these functions provide probabilities of the input examples belonging to each category.

The last two hyperparameters discussed here are optimizer and learning rate, specifying what type of SGD the network uses for training and how big changes to weights does each iteration respectively. I discuss these two together as they are very closely connected. The plain SGD is described in Sec. 2.1, but other types with smaller or bigger modifications exist. Typically, they add a form of adaptiveness to the learning rate, which alters its value during learning. For example, it might be a decay that slowly decreases the rate after each iteration or momentum that accelerates it. A bigger value of learning rate corresponds to more significant weight adjustments during learning and vice versa. Therefore it is beneficial to have a relatively large learning rate at the start when the weights are random but lower with more epochs. Typical examples of adjusted SGD optimizer are RMSprop, Adagrad or very popular Adam, which combines the benefits of both previous examples [38].

These are the most significant hyperparameters, even though many others could have been mentioned. Finding their optimal combination is crucial to achieving the best results. But simply trying different combinations by hand is not very efficient. The first idea would be to evaluate the error for all possible combination and choose the one with the lowest error. This method is called grid search and is very computationally expensive as there usually is an enormous number of combinations and each requires training a different network to evaluate the error. An alternative to using all combinations is randomly picking a fraction of those and choose the best result just from this limited sample. The computational time is reduced, but nothing guarantees to find the optimum result. For this purpose, more advanced methods were developed, minimizing the number of evaluations.

At first, we considered implementing a so-called genetic algorithm [39] inspired by Darwin's natural selection theory. Described without unnecessary details, the first generation of examples, hyperparameter combinations, is chosen randomly and then evaluated. The best results are selected for breeding new individuals that replace the weak ones in the next generation. Together with a mutation, i.e. a small chance to slightly alter the examples, this method achieves better results than random-sampling search with the same number of evaluations. A considered alternative method is a black-box optimization software called Nomad [40]. It was developed for optimizing functions, whose evaluations are highly computationally expensive. Implementing Mesh Adaptive Direct Search algorithm [41], Nomad allows finding the best solution with a relatively small number of evaluations. The hyperparameters can be real values, integers or even text-like objects, for example, a type of SGD method. For this reason, we used this method during the NN optimization process.

2.5 Convolutional neural network

In this section, a particular type of neural network called convolutional [42] is introduced, typically used in imaging applications. Its significant advantage is the invariance of object location in the image, due to the shared-weights architecture. In other words, it does not matter which direction is a man looking or in which section of the picture is he situated. The convolutional neural network (CNN) can locate this object and extract its relevant information anywhere in the picture. This feature makes them an ideal candidate for an image or even video reconstruction and classification as well as several other applications.

To describe a layer of a CNN, I will use the notation AxBxC, where B and C are numbers of neurons in a channel, and A is the number of channels. An RGB coloured picture with 1600 by 1200 pixels is then represented as 3x1600x1200, whereas a black and white version of this image would be 1x1600x1200. A CNN consists of two types of layers, namely convolutional and pooling, performing a different operation to determine the values of their neurons. A convolutional layer applies a filter, unique for each channel, represented by an X by Y matrix of values. These values are the only parameters adjusted during the learning process, just as were the weights in a fully-connected DNN. A pooling layer purpose is to reduce the dimensionality of the channels, as well as making the network invariant to small translations. Its parameters are fixed and represent a size of the area of a channel reduced to a single neuron. A max-pooling layer keeps only the highest value of these neurons, whereas an average-pooling layer keeps their average. These layers can be arranged in any order and even mixed with dense layers, as shown in Fig. 2.6 to create a classifier.

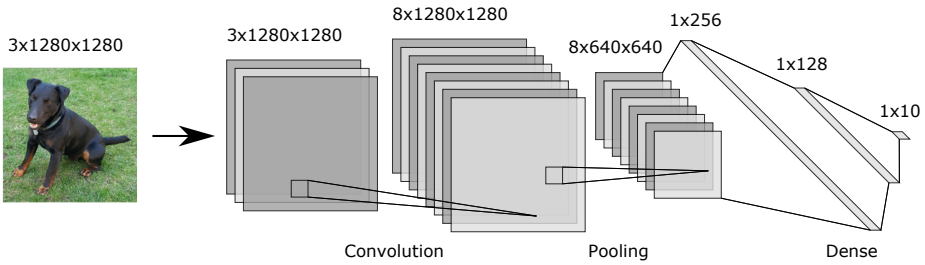

Figure 2.6: An example of a classification convolutional neural network predicting a type of animal in an image. At first, the picture is separated to 3 channels corresponding to a red, green and blue intensity of each pixel. Then, a convolutional layer applies different types of filters, with sizes 160x160, expanding the number of channels to 8. After that, a pooling layer performing a 2x2 reduction decreases the dimensions of the picture. The final part of this network consists of three dense layers whose outputs are probabilities for ten predefined categories, i.e. types of animals.

A specific kind of convolutional neural network is an autoencoder, that learns to output a copy of its input. This task could be trivial, but autoencoder consists of a so-called bottleneck, see Fig. 2.7, which is a part with a reduced amount of neurons compared to the input. The bottleneck forces the network to extract only the relevant information and reconstruct the image from this fraction of data. An autoencoder is a part of unsupervised learning as only input

data are presented, and the network searches for patterns in the data without the necessity of having the training examples labelled. A typical application for an autoencoder is a reduction of noise, as a well-trained autoencoder does not let the noise pass through the bottleneck. Another application is image classification using the first part of the network and replacing the second half by a dense network. The convolutional part extracts the relevant information about an image and fully-connected part uses them to assign the correct category.

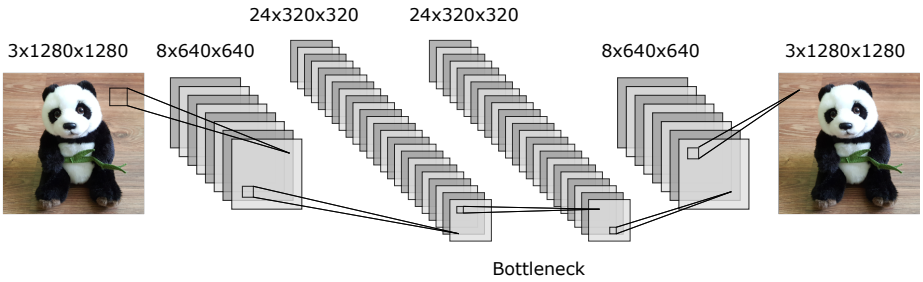

Figure 2.7: An example of an autoencoder with 3x1280x1280 input layer and 24x320x320 neurons in the bottleneck. This network downsamples the number of neurons to one half and then upsamples back to the original amount. Each layer consists of convolution and pooling, as the dimensions are reduced and the number of channels increased. The second half could be replaced with dense layers to transform this autoencoder into a classifier.

Chapter 3

Liquid crystal polarimetry

A polarization state of light plays a significant role in many optical systems and application, and therefore its precise preparation, modulation and detection are vital. In this chapter, the polarization state and Stokes parameters as a way to describe it are explained. Then, polarization modulators consisting of nematic crystal modules are introduced, and a deep neural network is applied to model a transformation of polarization state after passing through these modules. At the end of this chapter, the results obtained using DNN and using conventional methods are compared.

3.1 Polarization state of light

Polarization state is defined for a transverse wave as a development of electric and magnetic field vectors in time and space. Typically, we focus on the vector of an electric intensity $\vec{E}$ in a plane perpendicular to the direction of the wave. If the endpoint of this vector moves on a well-defined trajectory, we speak about a polarized wave, whereas disordered movement corresponds to an unpolarized one. A mixture of these waves is partially polarized, and we use a degree of polarization p to characterize the fraction of a polarized wave. There are six basis states of fully polarized light based on trajectories of vector $\vec{E}$: horizontal H, vertical V, diagonal D and anti-diagonal A, all with lines as trajectories in corresponding directions, and right-hand circular R and left-hand circular L following a circle in clockwise and anti-clockwise directions respectively.

Using Stokes parameters to describe a polarization of light is advantageous as it can cover both fully and partially polarized states. The first parameter S_0 represents the intensity I of the light beam, whereas parameters S_1, S_2 and S_3 express the differences between intensities of two related basis states

$$\begin{aligned} S_0 &= I, \\ S_1 &= I_H - I_V, \\ S_2 &= I_D - I_A, \\ S_3 &= I_R - I_L. \end{aligned} \tag{3.1}$$

The parameter S_0 does not affect a type of polarization and can be normalized, $S_0 = 1$. Remaining parameters S_1, S_2 and S_3 after the normalization can

possess only values from interval $\langle -1, 1 \rangle$, and $\sqrt{S_1^2 + S_2^2 + S_3^2} = p$. Examples of Stokes parameters for basis polarization states are shown in Tab. 3.1. These three parameters can be used as a Cartesian coordinate system displaying fully polarized states on a surface of a sphere with a centre in the origin and a radius of 1, and partially polarized states inside this sphere, as shown in Fig. 3.1.

Polarization state	H	V	D	A	R	L
Stokes parameters	$\begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}$	$\begin{pmatrix} -1 \\ 0 \\ 0 \end{pmatrix}$	$\begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}$	$\begin{pmatrix} 0 \\ -1 \\ 0 \end{pmatrix}$	$\begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}$	$\begin{pmatrix} 0 \\ 0 \\ -1 \end{pmatrix}$

Table 3.1: A table with the six basis polarization states of light and their corresponding Stokes parameters S_1, S_2 and S_3.

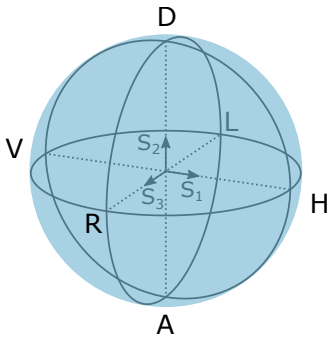

Figure 3.1: A Poincaré sphere in a Cartesian coordinate system of Stokes parameters with basis states in corresponding positions. Fully polarized states of light are positioned on the surface of this sphere, partially polarized are inside it, and unpolarized light is situated in the origin of a coordinate system.

Using the Stokes parameters, we can define a density matrix, which is another way of describing a polarization state, as

$$\rho = \frac{1}{2} \begin{pmatrix} 1 + S_1 & S_2 - iS_3 \\ S_2 + iS_3 & 1 - S_1 \end{pmatrix}, \tag{3.2}$$

where i stands for the imaginary unit. To determine the closeness or similarity of two polarization states, we typically use the fidelity F, defined as

$$F(\sigma, \rho) = \left[Tr \left(\sqrt{\sqrt{\rho} \sigma \sqrt{\rho}} \right) \right]^2, \tag{3.3}$$

where σ and ρ are the density matrices of two polarization states. A fidelity can possess a value from interval $\langle 0, 1 \rangle$, where $F(\rho, \rho) = 1$. Multiplying the fidelity by a factor of 100 allows expressing its value as a percentage.

3.2 Twisted nematic liquid crystals modules

A preparation of polarization states of light is typically done using half-wave and quarter-wave plates. This approach requires rotating these plates, which takes a certain amount of time and might cause vibrations to the optical system. An alternative is to replace these plates with modules made of twisted nematic liquid crystals (TN LC), typically used in displays, controlled by applied voltages. Replacing the plates with these modules would both speed up the process and get rid of vibrations caused by motorized rotation stages. A downside is the lack of accurate analytical description of polarization transformation of TN LC. Numerical modelling can help, to some extent, with the prediction of polarization state from the control voltages but is inefficient in the inverse task. As the precision of polarization state preparation and measurement is the key to many applications, we will present an alternative approach of TN LC modelling using a deep neural network. But first, in this section, the vital information about these modules are covered.

A TN LC module consists of liquid crystal molecules enclosed between two perpendicular alignment layers, which causes them to arrange in a twist. On the outer side of these layers are electrodes, which create a control electric field, see Fig. 3.2. Once applied, this electric field forces the molecules to change their orientation in the direction of the created field. After the field is switched off, the molecules return to their original positions. The orientation of molecules affects the refractive index causing a phase shift between the polarization components in a plane perpendicular to the propagation of light. Therefore, the polarization transformation can be controlled by the applied voltage. A single TN LC module is restricted in the range of polarization states that can prepare. To sufficiently cover the Poincaré sphere, the module must be aligned with other optical components or more TN LC modules have to be used [43].

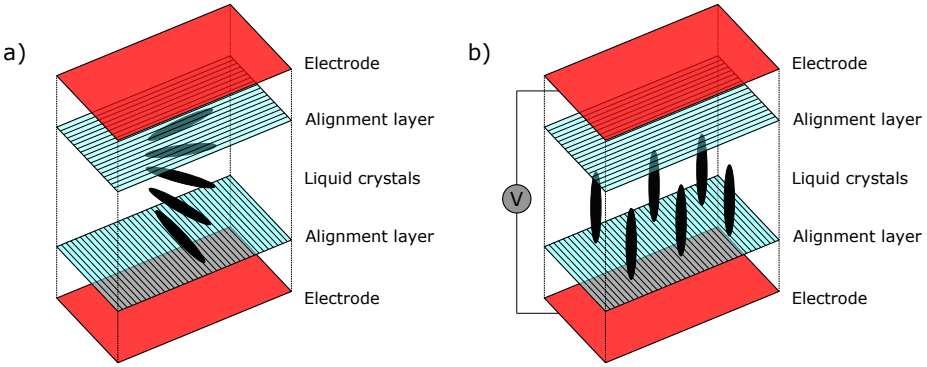

Figure 3.2: a) A structure of a TN LC module with liquid crystals forming a twist while enclosed between two perpendicular alignment layers and two electrodes. b) When a voltage is applied, the molecules of liquid crystals are forced to change orientation based on the electric field created by these electrodes. In both cases, the light propagates from the bottom to the top.

All the experimental tasks related to creating a dataset of polarization states of light were performed by Martin Bielak [43] using an optical system pictured in Fig. 3.3. The obtained dataset consists of more than 20,000 combinations of

voltages, each ranging uniformly from 0 to 10 V, labelled with their corresponding Stokes parameters.

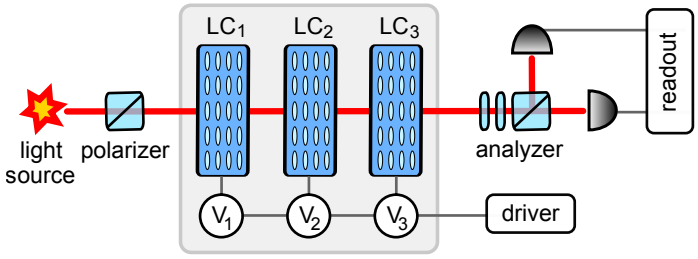

Figure 3.3: A schema of optical system assembled for preparing a dataset of polarization states of light using three TN LC devices connected to corresponding control voltages. The light beam propagates through a polarizer preparing the H state, which is then transformed in the modules by applied voltages and the analyzer based on rotating wave plates then precisely characterizes the obtained polarization state.

3.3 TN LC transformation modelling

In this section, the results obtained by using the analytical model, by linearly interpolating these data and by training a fully-connected deep neural network are presented. The dataset is randomly shuffled to contain 12,000 examples in the training set, 5,000 in the validation set and remaining approximately 3,000 in the test set, required for training and testing the neural network. The training and validation sets are used together for the interpolation method.

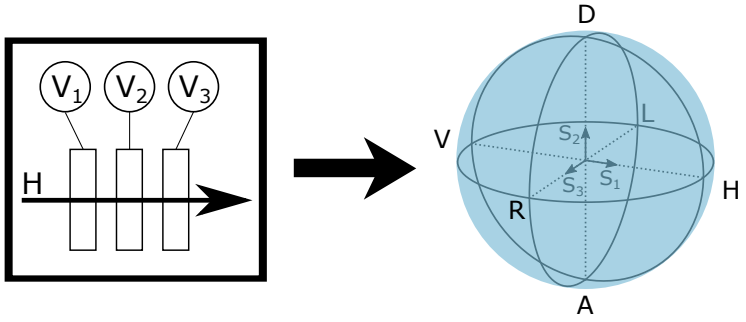

Figure 3.4: A representation of the direct network, i.e. predicting the Stokes parameters from control voltages.

First, we will focus on modelling the prediction of polarization state from the control voltages. The analytical model advanced in [43] reaches an average relative error of Stokes parameters $\delta = 6.11$ %, which was evaluated as a mean absolute error (MAE) normalized by the range of Stokes parameters,

$$\delta = \frac{\mathrm{MAE}}{2} \cdot 100\% = \frac{1}{2 \cdot 3N} \sum_{i=1}^{N} \sum_{j=1}^{3} |y_{ij} - x_{ij}| \cdot 100\%, \tag{3.4}$$

where y_{ij} is the target j-th Stokes parameter of the i-th data example, and x_{ij} is the same configuration for the obtained output. Because we are interested in the closeness of obtained and target polarization states, the error was also evaluated as mean fidelity and reached the value $F = (96.9 - 2.6 + 1.9)$ %, where the numbers refer to the mean value, the first decile and the ninth decile respectively. This result is highly inefficient as for preparing a certain polarization state using this model, we would have to at first manually search for the correct voltages, and then we would only prepare a state with this fidelity. These two disadvantages together make the analytical model unacceptable for practical use. The alternative approach of interpolating the dataset can be performed quite sufficiently because it contains a relatively large amount of examples. Evaluating the interpolation on the test set reaches the relative error $\delta = 1.05$ %, which is equivalent to a mean fidelity $F = (99.5 - 1.3 + 0.5)$ %. As we can see, this is much better than the previous result of the analytical model, and that is including a tiny percentage of samples being extrapolated.

The third tested TN LC model is based on a DNN. The results of optimizing hyperparameters using Nomad determined optimal network with configuration 3-92(10)-3, i.e. with over 75,000 connections. Using ReLU as an activation function in hidden layers and linear function, for bipolar values, in the output layer, together with an adaptive learning rate and dropout in each layer reaches the highest score on the validation set. Once trained, this network predicts polarization state from control voltages with a relative error $\delta = 0.14$ % on the test set, which evaluated using mean fidelity is equivalent to $F = (99.996 - 0.002 + 0.04)$ %. This approach is significantly more accurate than previous methods. A visual representation of these results on the Poincaré sphere is shown in Fig. 3.5.

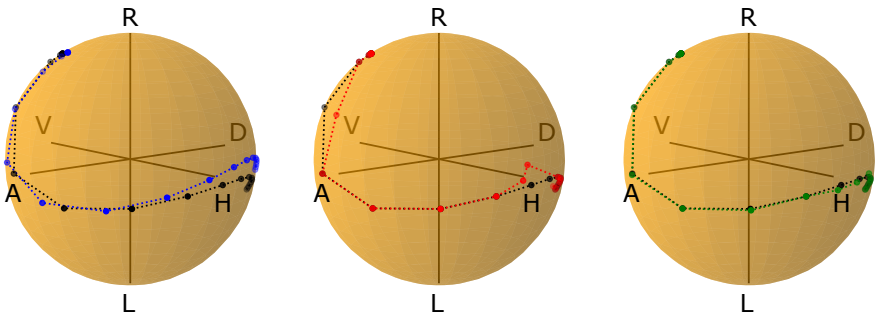

Figure 3.5: A comparison of an analytical model (blue), an interpolation method (red) and a deep neural network (green) predicting the measured polarization state (black), where the control voltage V_3 is varied. As we can see, the highest overlap with required polarization states is met by the neural network.

Modelling the inverse task, i.e. prediction of control voltages preparing the target polarization state, is more challenging, as implies the absence of any analytical model. In certain ranges, altering the control voltages causes almost no difference to the polarization state, which is one of the reasons that the model cannot be inverted. This also presents a problem for evaluating the results, as even a relatively high error in control voltages might prepare a very similar polarization state. For this purpose, the inverse and direct models are joined

together, as shown in Fig. 3.6, and the mean fidelity of the combined network is evaluated. This allows approximately estimating the mean fidelity of inverse models, as the errors propagated from direct models are, in this case, negligible compared to the errors of combined networks.

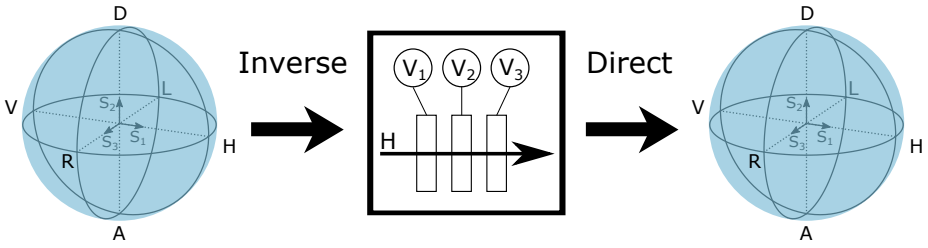

Figure 3.6: A representation of the combined network, i.e. predicting the control voltages for target Stokes parameters and then using the direct model with the predicted voltages.

As there is no analytical model, there is also nothing to evaluate for this method. On the other hand, interpolating the dataset for the combined task achieves the relative error $\delta = 2.55$ %, which is equivalent to the mean fidelity $F = (98.1 - 1.3 + 1.9)$ %. This is even better than the results of the analytical model for the direct task, which only highlights the insufficiency of the analytical model. Now moving to the DNN, the optimization for inverse transformation using Nomad resulted in a network with structure 3-104(12)-3 and the same combination of hyperparameters as previously. The combined network achieves a relative error of control voltages $\delta = 4.82$ %, which is equivalent to the mean fidelity $F = (96.2 - 5 + 3.8)$ %. As we can see, the results for the DNN are a bit worse than those of the interpolation method. In other words, interpolating the dataset presets a more reliable method for the inverse task, however, we plan to keep working on this problem even further and possibly improve the DNN performance to overcome the interpolation method.

Furthermore, for the direct as well as the inverse task, the average computational time for predicting one sample using a DNN is approximately $t = 50$ μs, which is almost 200 times faster than the interpolation method with the average required time $t = 9$ ms per sample. Therefore implementing a DNN with TN LC modules would speed up the preparation and measurement of polarization states even further, which is another reason for improving the fidelity of the inverse network.

3.4 Conclusion

Modelling the transformation of TN LC crystal with neural networks appears to be an effective method for predicting the Stokes parameters based on applied control voltages, as to suggest both numerical results and overlap of polarization states in Fig. 3.5. Compared to the analytical model, the network predicts polarization states with fidelity higher by three orders of magnitude, which is a significant improvement. The fidelity of this network is also higher by two orders of magnitude compared to the interpolation method. For the direct task, the DNN proved to be a very accurate model, and we plan to progress

towards its detailed experimental verification. The inverse task, i.e. predicting the necessary control voltages for preparing the target polarization state, is a challenge even for the DNN. On the other hand, there was no analytical model, to begin with, and the only other applicable method was an interpolation. As we saw in the previous section, the interpolation achieves slightly higher fidelity, but both of them are of the same order of magnitude. Based on these results, we plan to improve DNN performance even further by using Nomad for optimizing the combined network as a whole, instead of only the separated inverse part. We believe that this optimization could improve the performance of the inverse DNN and maybe even overcome the results of the interpolation method.

Chapter 4

Image deconvolution

In this chapter, convolutional neural networks enhancing a resolution and counting emitters in an image are presented. Firstly, the generation of a dataset and its relations to real-life measurements is described. Then, the deterministic deconvolution is introduced, and other possible methods are mention. The main section of this chapter is focused on the performance of an optimized convolution network and its comparison to the deterministic deconvolution.

4.1 Data generation, convolution and noise

Every optical system is affected by a diffraction limit causing imperfections of the imaging. A typical example is microscopy, where objects might be too small for the microscope to resolve them and end up being blurred. Besides the fundamental diffraction limit, the optical system is affected by optical aberrations which can be to some extent eliminated in exchange for a more complex and expensive optical system. If we know the parameters of an imaging device at least partially, we can apply deconvolution methods to improve the quality of an image, like a convolutional neural network. But to train it, a dataset of low-quality images with their corresponding improved versions is required. One way to achieve this is by collecting real-life images from experiments. In this thesis, a dataset based on the numerical simulation of the diffraction effect and noise is generated. This way, in the end, the network can be tested on real-life data to evaluate not only its performance but also the appropriateness of the generated dataset.

We start with a 28x28 pixel image with up to 20 point-like emitters. Their number and locations in the picture are generated randomly with uniform distribution so that the ideal images might look like the one in Fig. 4.1 a). Each emitter is assigned with a random number from a Poisson distribution with mean value N, characterizing its intensity by the number of detected photons. After that, the image is blurred correspondingly to the diffraction limit of the optical system, which can be simulated using a point spread function (PSF). It characterizes the imperfections caused to a point source by the optical system as a convolution of an image with this function. Typically, the real PSF of a dipole is complicated but can be approximated using a normalized Gaussian function

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{x^2}{2\sigma^2}\right), \tag{4.1}$$

where σ is a parameter characterizing the width of the function, and therefore for future references will be called the point spread function width (PSF width) [44]. The image in Fig. 4.1 b) represents the convolved version of a) using (4.1) with the PSF width $\sigma = 1$.

In a real optical system, this is not the only imperfection because we are also limited in knowing the exact value of σ. To implement this uncertainty to the data generation, instead of using a fixed value of σ, a random number is chosen from the Gaussian distribution with σ as its mean value and a certain standard deviation $\Delta\sigma$. I will refer to this deviation as the uncertainty of the point spread function width. The whole image is convolved using this particular σ, and for the next picture, a new σ is generated. The last step is adding noise, which occurs in every experiment. The type of noise considered for this data generation is a so-called shot noise caused by the discrete nature of electric charge in a camera or other detection components. The noise intensity in each pixel follows the Poisson distribution with a mean value n. This type of noise appears in a camera and, therefore, is not convolved. A signal-to-noise ratio parameter (SNR), defined as $\mathrm{SNR} = N/n$, characterizes the ratio between emitter signal intensity and the noise. In Fig. 4.1 c), a convolved example with camera noise is shown.

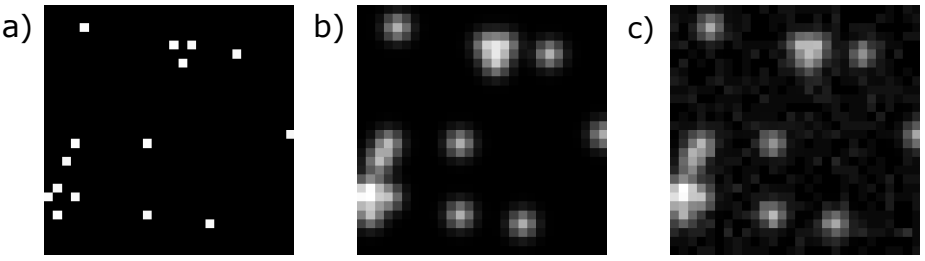

Figure 4.1: An example of a generated image with 15 point-like emitters a) before any imperfections were implemented, b) after convolution with Gaussian PSF of $\sigma = 1$, c) convolved and with added noise of SNR = 100.

With a fully-known point spread function width, a direct approach for deconvolving an image is deterministic deconvolution based on the Fourier transform of convolution. The process of implementing imperfections can be described as

$$y = h * x + \omega, \tag{4.2}$$

where y is the convolved image created by a convolution of the input image x with a PSF h and adding noise ω. By applying a Fourier transform, we get

$$Y = HX + \Omega, \tag{4.3}$$

where Y, H, X and Ω are the Fourier transforms of y, h, x and ω respectively. We can divide this expression by H to obtain

$$X = \frac{Y}{H} - \frac{\Omega}{H}. \tag{4.4}$$

As we can see, without any noise, this would allow perfectly reconstructing the original image. Unfortunately, the term $\frac{\Omega}{H}$ can create a divergence, and the uncertainty of PSF width makes h inaccurate, which together significantly lowers the ability of deterministic deconvolution to restore the original image.

Because our data are represented by matrices, we have to use the discrete version of convolution. To implement the convolution numerically, we multiplied the original matrix x from left and right with a 28x28 symmetric Toeplitz matrix T, whose row and column values correspond to a two-dimensional Gaussian function, see Fig. 4.2 a),

$$y = TxT + \omega, \tag{4.5}$$

where ω represents noise and y is the convolved image. Therefore, the deterministic deconvolution can be implemented by multiplying the convolved matrix y from each side by the inverse Toeplitz matrix, shown in Fig. 4.2 b). As we can see, the same noise limitation holds also for the discrete deconvolution.

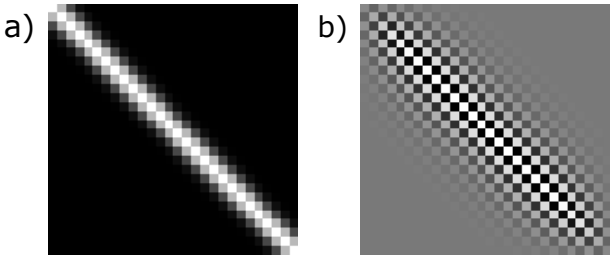

Figure 4.2: An example of a) a symmetrical Toeplitz matrix, b) an inverse Toeplitz matrix.

As the performance of deterministic deconvolution is strictly limited by the mentioned imperfections, more advanced approaches of deconvolving an image have been developed. For example, various types of regularization techniques might improve the performance of deterministic deconvolution, as they aim to decrease the destructive effect of noise [45]. In the future, we plan to use one of these methods to compare its results to the convolutional neural network. For now, the CNN approach is compared to a simple deterministic deconvolution.

4.2 Deconvolution by CNN

The convolutional network is optimized with a dataset containing $25,000$ examples in the training set and $10,000$ in both validation and test set. These data were generated using an emitter signal with the mean number of photons $N = 10^4$, and the PSF width $\sigma = 1$ with the uncertainty $\Delta\sigma = 0.02$. Because the mean value of noise photons is set to $n = 1$, the SNR $= N$. The final structure of the CNN is shown in Fig. 4.3. The convolutional part of the network is responsible for both deconvolution and noise reduction, whereas the two dense layers serve as a classifier counting emitters in the image.

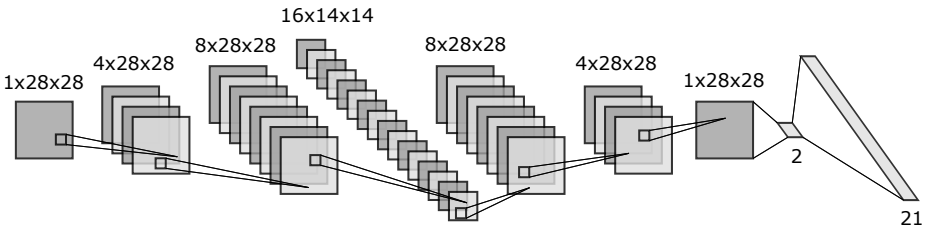

Figure 4.3: The architecture of the convolutional neural network trained for the parameters described above. The first part of this network performs deconvolution as well as noise reduction and outputs enhanced images, in which the dense layers determines the number of emitters as a classification task.

The trained network predicting the number of emitters from convolved images achieves an accuracy $A_C = 99.28$ % on the test set, which was evaluated as

$$A = \frac{\text{Number of correctly predicted samples}}{\text{Number of all samples}}, \tag{4.6}$$

where the samples in our context refer to the test set. This result is significantly higher than the accuracy of the deterministic deconvolution $A_D = 61.41$ %, whose low performance is caused by the presence of the camera noise and the uncertainty of PSF width, as described in Sec. 4.1. A Fig. 4.4 shows a confusion matrix of the mentioned network.

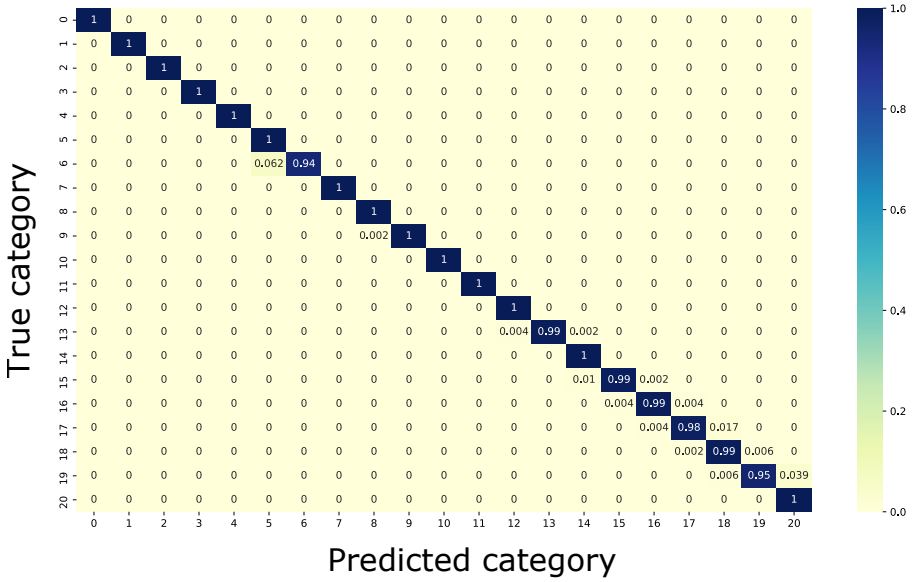

Figure 4.4: A confusion matrix of the network with accuracy $A_C = 99.28$ %.

To explore the dependence of the network architecture on the mean value of emitter photons, new networks with the same combination of hyperparameters were trained on datasets with a different mean value of emitter photons, but without any uncertainty of PSF width. Then they were tested for SNR ranging

from 1 to 10^5, and the results are shown in Fig. 4.5 a). From this picture, we can see that the network trained with $N = 10^4$ can efficiently work even for lower SNR until its performance rapidly decreases. On the other hand, the deterministic deconvolution starts decreasing at SNR higher by two orders of magnitude. We can also see that networks trained on lower SNR can work relatively well in the corresponding regions.

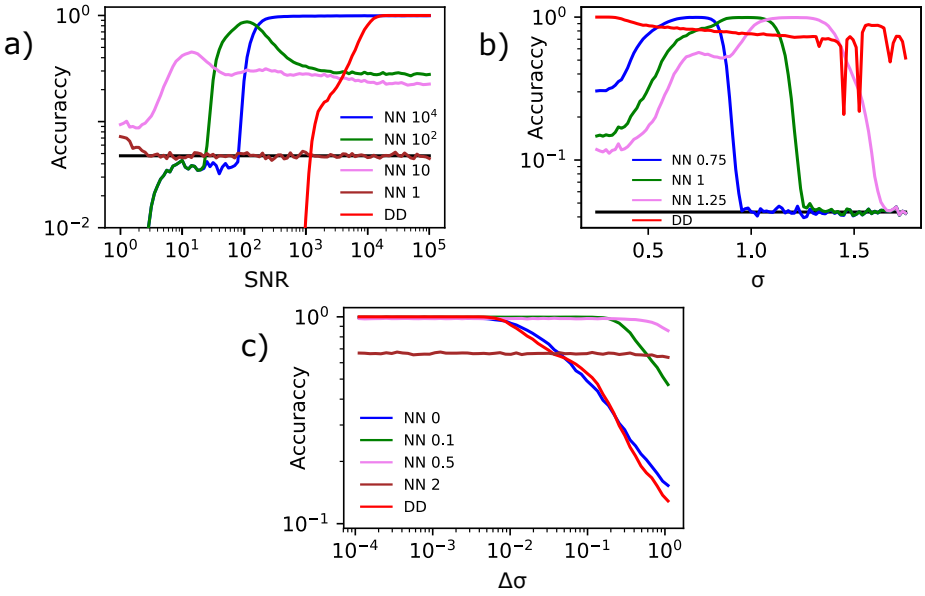

Figure 4.5: Dependence of the accuracy of neural networks trained on a dataset: a) with different emitter intensities, i.e. SNR, b) with different PSF width values, c) with different uncertainties. The black line represents random guessing, numbers in a legend stand for a value used for training the corresponding neural network, and the DD stands for deterministic deconvolution.

The same idea is behind the graphs shown in Fig. 4.5 b) for networks trained with different values of a PSF width and c) its uncertainties, only without any noise. As we can see in the second graph, networks trained with a certain fixed value of PSF width are efficient in the corresponding regions, just as in the case of intensities. This behaviour suggests that the same network could be trained and used even for a slightly different value of PSF width and still keeps its accuracy. All the curves show better performance in the left part than in the right part of the graph. This is caused by higher σ values corresponding to more blurred images, and therefore a more complex task. From the third graph, we can see that the network trained on a dataset with a fixed $\Delta\sigma = 0$ and no noise has approximately the same accuracy as deterministic deconvolution. On the other hand, if it sees a small $\Delta\sigma$ during training, it can keep its high accuracy even for larger uncertainties in exchange for a slightly worse performance for lower values, whereas training with too big uncertainty significantly lowers the performance for all its values. From this knowledge, we can deduce that with the correct value of $\Delta\sigma$, we can achieve an optimal performance of the network.

4.3 Universal network

Based on results from the previous section, the goal now is to come up with a network universal across intensities, whose accuracy would be comparable to the best results of networks shown in Fig. 4.5 a). For this purpose, the new network was trained under the same circumstances as in Fig. 4.5 a). Instead of using a fixed value of the SNR, the dataset of approximately 425,000 samples containing SNR values ranging from 1 to 10^6 was generated. The distribution of those values was linearly increasing under a logarithmic scale, and approximately 17.5 % of samples did not contain any noise. The dataset was a result of optimizing the distribution to achieve the highest accuracy across the SNR range. This dataset was divided into 290,000 samples for the training set, 70,000 for the validation set and approximately 65,000 for the test set.

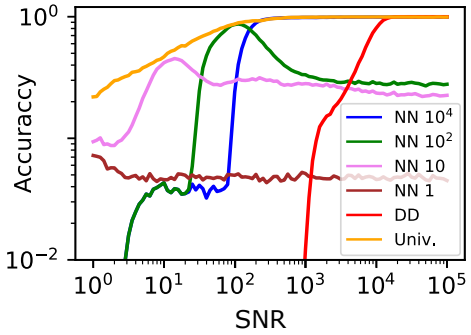

Figure 4.6: The accuracy of the universal network across the SNR range, together with accuracies of relevant previous models.

As we can see in Fig. 4.6, the universal network successfully achieves results comparable to previous individual networks, reaching the mean accuracy over the flat region $A_U = 99.34$ %. Fig. 4.7 shows an example deconvolved by the universal network compared to the deterministic deconvolution. Furthermore, the average computational time for predicting one sample by the CNN is approximately $t_C = 80$ μs, which is almost four times higher than the average time required by deterministic deconvolution $t_D = 18$ μs. However, this difference in the computational speed does not compensate the extreme difference in their accuracy.

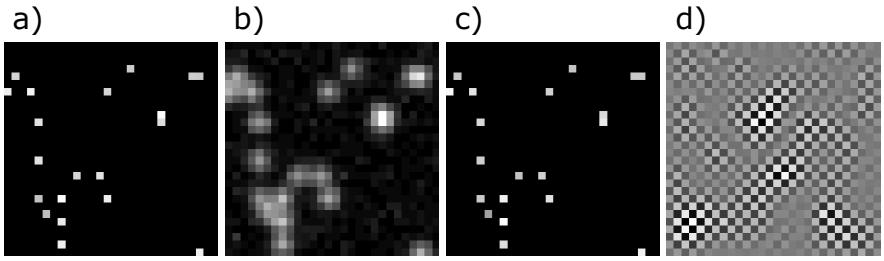

Figure 4.7: a) An example of an original image, b) convolved with a Gaussian PSF of width $\sigma = 1$ and SNR = 100 for camera noise, c) deconvolved by the universal network, d) deconvolved by the deterministic deconvolution.

4.4 Conclusion

In this chapter, image enhancement and emitter counting as an application of convolutional networks were discussed. The goal was to train a network capable of successfully deconvolving an image together with counting the point-like emitters across a certain range of SNR values. First, the performance of individual networks working only for a fixed intensity was compared to deterministic deconvolution and proved to be more accurate in their predictions. These results served as a foundation for the universal network, which achieved relatively high accuracy across a broad range of SNR values, as shown in Fig. 4.6. Based on these results, convolutional neural networks appear to be an effective method for this task, and in the future, we plan to improve their performance even further and under more challenging conditions. Our next goal after implementing regularization techniques to the deterministic deconvolution is to explore the possibility of training a network capable of effectively deconvolving images not only independently of the point-like emitter intensity, but also the value of the PSF width.

Chapter 5

Conclusion

In this thesis, the goal was to introduce artificial neural networks and their optics related applications. At the beginning of the work, the structure and processes of a deep neural network are introduced. After that, the methods for hyperparameter optimization and overfitting reduction are discussed. Also, a specific type called convolutional neural network is introduced together with its benefits to imaging applications.

The first application of a deep neural network in this thesis is modelling the transformation of the polarization state of light after passing through twisted nematic liquid crystal modules, which cannot be accurately modelled analytically. A fully-connected neural network can predict polarization states from control voltages that compared with the actually measured states have a mean fidelity of 99.996 %, which is by three orders of magnitude more accurate than the analytical model. For the inverse task, i.e. prediction of necessary control voltages to prepare the target polarization state, no analytical model exists, and a neural network achieves a relative error of 6.5 % of control voltages. The direct network proves to be a more efficient method of modelling twisted nematic liquid crystal modules even compared to interpolating the dataset, which we now aim to achieve also for the inverse network.

The second application of deep neural networks deals with enhancing the resolution of images with several point emitters and counting them using a convolutional neural network. At first, a dataset based on a physical understanding of diffraction limit and shot noise is modelled. Then, convolutional neural networks for deconvolving the images and counting the emitters are trained for certain combinations of data imperfection parameters. After this, the performance of a single universal network that could work across a broad range of emitter intensities is explored. This universal network achieves sufficiently high accuracy compared to the previously mentioned networks.

We plan to keep working with convolutional neural networks and extend their application in imaging. At first, we want to implement regularization methods to the deterministic deconvolution to improve its performance and compare the network with a more advanced and robust technique than the straightforward deterministic deconvolution. We aim to train a convolutional network independent not only of the emitter intensity but also of the point spread function width. After that, the new goal will be implementing another type of noise to the data representing a stray light or substrate fluorescence.

The difference between this noise and the already implemented camera noise is that the stray light is convolved by the imaging system. The stray light would make the deconvolution significantly more challenging in the case of a very low SNR, as it would be hard to decide whether the detected photon comes from an emitter or stray light. We plan to incorporate other statistical methods to facilitate reliable emitter counting.

Bibliography

[1] A. Kalinovsky and V. Kovalev, "Lung image segmentation using deep learning methods and convolutional neural networks," in *XIII International Conference on Pattern Recognition and Information Processing*, Oct. 2016.

[2] J. Bernal, K. Kushibar, D. S. Asfaw, S. Valverde, A. Oliver, R. Martí, and X. Lladó, "Deep convolutional neural networks for brain image analysis on magnetic resonance imaging: a review," *Artificial Intelligence in Medicine*, vol. 95, pp. 64–81, Apr. 2019.

[3] N. Abiwinanda, M. Hanif, S. T. Hesaputra, A. Handayani, and T. R. Mengko, "Brain tumor classification using convolutional neural network," in *IFMBE Proceedings*, pp. 183–189, Springer Singapore, May 2018.

[4] L. Scotti, H. Ishiki, F. M. Júnior, M. da Silva, and M. Scotti, "Artificial neural network methods applied to drug discovery for neglected diseases," *Combinatorial Chemistry & High Throughput Screening*, vol. 18, pp. 819–829, Sept. 2015.

[5] H. Chen, Y. W. O. Engkvist, M. Olivecrona, and T. Blanchke, "The rise of deep learning in drug discovery," *Drug discovery today*, vol. 23, pp. 1241–1250, Jan. 2018.

[6] V. Goncalves, K. Maria, and A. B. F. da Silv, "Applications of artificial neural networks in chemical problems," in *Artificial Neural Networks - Architectures and Applications*, InTech, Jan. 2013.

[7] R.-D. Lasseri, D. Regnier, J.-P. Ebran, and A. Penon, "Taming nuclear complexity with a committee of multilayer neural networks," *Physical Review Letters*, vol. 124, Apr. 2020.

[8] R. Stivaktakis, G. Tsagkatakis, B. Moraes, F. Abdalla, J.-L. Starck, and P. Tsakalides, "Convolutional neural networks for spectroscopic redshift estimation on euclid data," *IEEE Transactions on Big Data*, pp. 1–1, 2019.

[9] A. Davies, S. Serjeant, and J. M. Bromley, "Using convolutional neural networks to identify gravitational lenses in astronomical images," *Monthly Notices of the Royal Astronomical Society*, vol. 487, pp. 5263–5271, May 2019.

[10] J. Fluri, T. Kacprzak, A. Refregier, A. Amara, A. Lucchi, and T. Hofmann, "Cosmological constraints from noisy convergence maps through deep learning," *Physical Review D*, vol. 98, Dec. 2018.

[11] R. Carrasco-Davis, G. Cabrera-Vives, F. Förster, P. A. Estévez, P. Huijse, P. Protopapas, I. Reyes, J. Martínez-Palomera, and C. Donoso, "Deep learning for image sequence classification of astronomical events," *Publications of the Astronomical Society of the Pacific*, vol. 131, p. 108006, Sept. 2019.

[12] A. Kimura, I. Takahashi, M. Tanaka, N. Yasuda, N. Ueda, and N. Yoshida, "Single-epoch supernova classification with deep convolutional neural networks," in *2017 IEEE 37th International Conference on Distributed Computing Systems Workshops (ICDCSW)*, IEEE, June 2017.

[13] J. Gao, L.-F. Qiao, Z.-Q. Jiao, Y.-C. Ma, C.-Q. Hu, R.-J. Ren, A.-L. Yang, H. Tang, M.-H. Yung, and X.-M. Jin, "Experimental machine learning of quantum states," *Physical Review Letters*, vol. 120, June 2018.

[14] Z. A. Kudyshev, S. Bogdanov, T. Isacsson, A. V. Kildishev, A. Boltasseva, and V. M. Shalaev, "Rapid classification of quantum sources enabled by machine learning," 2019.

[15] T. Xin, S. Lu, N. Cao, G. Anikeeva, D. Lu, J. Li, G. Long, and B. Zeng, "Local-measurement-based quantum state tomography via neural networks," *npj Quantum Information*, vol. 5, Nov. 2019.

[16] A. M. Palmieri, E. Kovlakov, F. Bianchi, D. Yudin, S. Straupe, J. D. Biamonte, and S. Kulik, "Experimental neural network enhanced quantum tomography," *npj Quantum Information*, vol. 6, Feb. 2020.

[17] G. Torlai, G. Mazzola, J. Carrasquilla, M. Troyer, R. Melko, and G. Carleo, "Neural-network quantum state tomography," *Nature Physics*, vol. 14, pp. 447–450, Feb. 2018.

[18] R. Soleti, L. Cantini, F. Berizzi, A. Capria, and D. Calugi, "Neural network for polarimetric radar target classification," in *Proceeding of the 14th European Signal Processing Conference*, Sept. 2006.

[19] H. Y. D. Sigaki, E. K. Lenzi, R. S. Zola, M. Perc, and H. V. Ribeiro, "Learning physical properties of liquid crystals with deep convolutional neural networks," *Scientific Reports*, vol. 10, May 2020.

[20] P. Caramazza, O. Moran, R. Murray-Smith, and D. Faccio, "Transmission of natural scene images through a multimode fibre," *Nature Communications*, vol. 10, May 2019.

[21] N. Borhani, E. Kakkava, C. Moser, and D. Psaltis, "Learning to see through multimode fibers," *Optica*, vol. 5, pp. 960–966, Aug. 2018.

[22] Y. Li, Y. Xue, and L. Tian, "Deep speckle correlation: a deep learning approach toward scalable imaging through scattering media," *Optica*, vol. 5, pp. 1181–1190, Sept. 2018.

[23] C. Dong, C. C. Loy, K. He, and X. Tang, "Image super-resolution using deep convolutional networks," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 38, pp. 295–307, Feb. 2016.

[24] E. Nehme, L. E. Weiss, T. Michaeli, and Y. Shechtman, "Deep-STORM: super-resolution single-molecule microscopy by deep learning," *Optica*, vol. 5, pp. 458–464, Apr. 2018.

[25] W. Ouyang, A. Aristov, M. Lelek, X. Hao, and C. Zimmer, "Deep learning massively accelerates super-resolution localization microscopy," *Nature Biotechnology*, vol. 36, pp. 460–468, Apr. 2018.

[26] W. Xie, J. A. Noble, and A. Zisserman, "Microscopy cell counting and detection with fully convolutional regression networks," *Computer Methods in Biomechanics and Biomedical Engineering: Imaging & Visualization*, vol. 6, pp. 283–292, May 2016.

[27] S. Helgadottir, A. Argun, and G. Volpe, "Digital video microscopy enhanced by deep learning," *Optica*, vol. 6, pp. 506–513, Apr. 2019.

[28] K. Hornik, "Approximation capabilities of multilayer feedforward networks," *Neural Networks*, vol. 4, no. 2, pp. 251–257, 1991.

[29] X. Glorot, A. Bordes, and Y. Bengio, "Deep sparse rectifier neural networks," in *Proceedings of the 14th International Conference on Artificial Intelligence and Statistics*, Oct. 2011.

[30] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, pp. 533–536, Oct. 1986.

[31] Y. Dauphin, R. Pascanu, C. Gulcehre, K. Cho, S. Ganguli, and Y. Bengio, "Identifying and attacking the saddle point problem in high-dimensional non-convex optimization," in *Advances in neural information processing systems 27*, June 2014.

[32] A. Choromanska, M. Henaff, M. Mathieu, G. B. Arous, and Y. LeCun, "The loss surfaces of multilayer networks," in *Conference on AI and Statistics*, Nov. 2014.

[33] N. S. Keskar, D. Mudigere, J. Nocedal, M. Smelyanskiy, and P. Tang, "On large-batch training for deep learning: Generalization gap and sharp minima," in *The International Conference on Learning Representations*, 2017.

[34] F. Chollet, *Deep Learning with Python*. Manning Publications Co., 2018.

[35] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *The Journal of Machine Learning Research*, vol. 15, p. 1929–1958, Jan. 2014.

[36] M. Belkin, D. Hsu, S. Ma, and S. Mandal, "Reconciling modern machine-learning practice and the classical bias–variance trade-off," *Proceedings of the National Academy of Sciences*, vol. 116, pp. 15849–15854, July 2019.

[37] G. Valle-Pérez, C. Q. Camargo, and A. A. Louis, "Deep learning generalizes because the parameter-function map is biased towards simple functions," in *The International Conference on Learning Representations*, Apr. 2019.

[38] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *The International Conference on Learning Representations*, 2015.

[39] A. Eiben and J. Smith, *Introduction to Evolutionary Computing*. Springer Berlin Heidelberg, 2015.

[40] S. L. Digabel, "Algorithm 909," *ACM Transactions on Mathematical Software*, vol. 37, pp. 1–15, Feb. 2011.

[41] C. Audet, A. L. Custódio, and J. E. Dennis, "Erratum: Mesh adaptive direct search algorithms for constrained optimization," *SIAM Journal on Optimization*, vol. 18, pp. 1501–1503, Jan. 2008.

[42] Y. LeCun, "Generalization and network design strategies," in *Connectionism in perspective* (R. Pfeifer, Z. Schreter, F. Fogelman, and L. Steels, eds.), Elsevier, 1989.

[43] M. Bielak, "Polarization state generation, measurement and control using liquid-crystal modulators." Master's Thesis, Palacký University, 2019.

[44] S. Stallinga and B. Rieger, "Accuracy of the gaussian point spread function model in 2d localization microscopy," *Optics Express*, vol. 18, pp. 24461–24476, Nov. 2010.

[45] G. Peyré, "The numerical tours of signal processing - advanced computational signal and image processing," *IEEE Computing in Science and Engineering*, vol. 13, no. 4, pp. 94–97, 2011.