Rheinische Friedrich-Wilhelms-Universität Bonn

Mathematisch-Naturwissenschaftliche Fakultät

Fachgruppe Physik/Astronomie

Bachelorarbeit

im Studiengang Physik

zur Erlangung des akademischen Grades
Bachelor of Science

Thema:	Machine Learning-Assisted Identification of Atom Positions in Two-Dimensional Optical Lattices
Autor:	Sebastian Witt
Version vom:	February 4, 2020
1. Betreuer:	Prof. Dr. Dieter Meschede
2. Betreuer:	Dr. Thoralf Räsch

Ich versichere, die von mir vorgelegte Arbeit selbstständig verfasst zu haben. Alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten oder nicht veröffentlichten Arbeiten anderer entnommen sind, habe ich als entnommen kenntlich gemacht. Sämtliche Quellen und Hilfsmittel, die ich für die Arbeit benutzt habe, sind angegeben.

Ort, Datum *Unterschrift*

Übersicht

Im Rahmen dieser Arbeit wird eine alternative Methode zur Analyse von EMCCD-Mikroskopaufnahmen von Atomen in zweidimensionalen quantenoptischen Gittern präsentiert, die mithilfe von neuronalen Netzen die Erkennung des Besetzungszustandes jedes Gitterplatzes automatisieren soll. Der entwickelte mehrschrittige Algorithmus wird insgesamt sowie in seinen Teilen für verschiedene Werte der numerischen Apertur auf seine Genauigkeit analysiert und mit alternativen Architekturen verglichen. Zur Erstellung der Netze ist eine große Anzahl von Trainingsdaten vonnöten, deren Atomverteilung im Vorraus bekannt sein muss. Diese werden über eine Faltung der näherungsweise ermittelten Punktspreizfunktion des Abbildungssystems mit den Atomverteilungsmasken simuliert. Dabei werden die Testbilder mit bis zu 99,76 % Genauigkeit korrekt erkannt.

Abstract

This work presents an alternative method for analyzing EMCCD-microscopy images of two-dimensional quantum optical lattices, using neuronal networks to automate the recognition of lattice occupation states. We introduce a multi-step algorithm, whose overall performance as well as step-by-step performance is analyzed, and which is compared to several different architectures. Training the networks requires a large amount of training data with known lattice occupation states. These images are simulated by convolution of the experimentally estimated point spread function of the imaging system with the atomic distribution masks. The algorithm allows for an accuracy of up to 99,76 % on our simulated data.

Contents

1 Introduction

Experimentation on single atoms suspended in optical lattices plays an important role in understanding quantum information processing and simulations [1] [2] [3]. However, the confirmation of achieving the wanted atom cluster states in the grid relies on the accurate evaluation of low photon count microscopy images, which poses a challenge due to the small lattice site separation of the order of the optical lattice wavelength, combined with dominant noise effects. To this end, deterministic threshold computing algorithms of various complexity have been employed in most cases in the past, often using deconvolution with an approximated point spread function [4]. With the heightened popularity machine learning enjoys in recent years [5] [6], and it having proved to allow for significant improvements in the field of image recognition [7], the usage of such methods for the identification of single atoms in optical lattice sites stands to reason. This work deals with both the generation of suitable training data by simulation of images with the given optical setup parameters as well as the construction and evaluation of neural networks of different architectures to detect the occupation state of each lattice site.

The algorithms are designed specifically for the 2D discrete quantum simulator (DQSIM) experiment of Professor Meschede's group, which uses the hyperfine levels of neutral cesium atoms to create a polarization dependent pseudo-spin system, allowing for spin-dependent transport in the lattice. Using this on an atom in a superposition state of both spin up and spin down effectively moves the two components in different directions, and thus creates a wave function delocalized over several lattice sites. This quantum state can then be probed by measuring the atom's localization probability distribution by exciting the atoms with a laser and registering the fluorescence light of the cesium D_2-line at 852 nm with an EMCCD camera. It is those low photon count camera images which are taken to be the input of this detection scheme. Further details on the experimental setup can be found in [8] and [4].

The detection of each lattice site's state (occupied / empty) has been realized in Python, making use of the `keras` module, while the simulation of training images has been done in MATLAB.

2 Microscopy Image Simulation

2.1 The point spread function

The point spread function (PSF) of an optical system characterizes the way a pointlike source will be imaged. In the ideal case of a perfect imaging system with circular aperture, the PSF takes the form of a Bessel function of first order, whose intensity distribution is also known as the airy disc (see also figure 1), and is given by

$$I(\theta) = I_0 \left(\frac{2J_1(kr\sin(\theta))}{kr\sin(\theta)} \right)^2 . \tag{1}$$

A useful quantity to define is the Abbe radius $r_{Abbe} = \frac{\lambda}{2NA}$, which is the radial distance from the first minimum to the central maximum of the airy disc. Here, λ is the wavelength and NA is the numerical aperture, a property given by the maximal half-angle θ of the light-cone that can enter a lens system and the diffraction index n as $NA = n\sin(\theta)$.

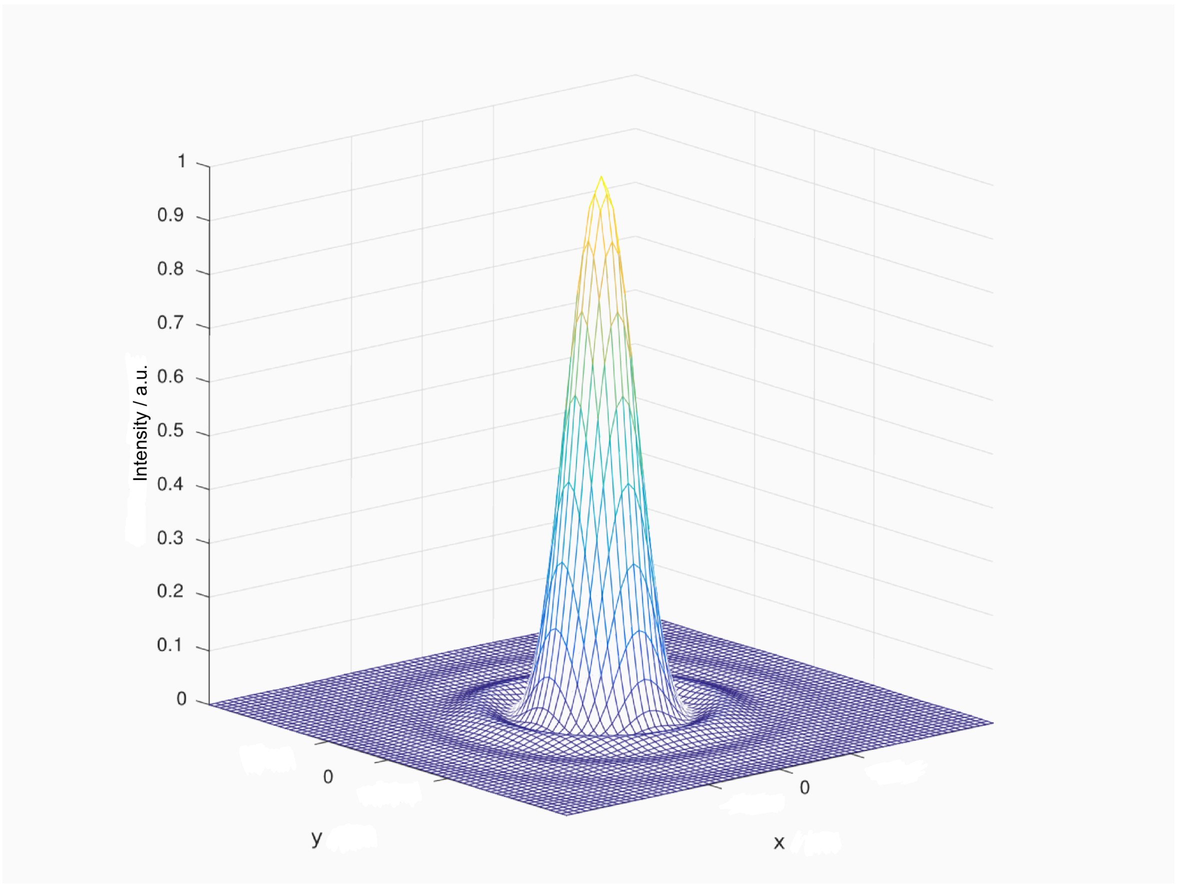


Figure 1: Intensity distribution of an Airy disc from imaging a pointlike source with an ideal imaging system [8]

2.2 Optical aberrations

In any real world setup, the imaging system suffers from optical errors, leading to a spreading of the airy disc and therefore a distorted image. These aberrations result from wavefront distortions of the electrical field at the aperture, which can be modeled using a superposition of the linearly independent Zernike polynomials Z_n^m with appropriate Zernike coefficients c_{nm} for the optical setup in question. The phase distortions expressed in this way are given by

$$W(x,y) = \sum_n \sum_{m=-n}^{n} c_{nm} Z_n^m(x,y) \tag{2}$$

[8] and can be used to compute the PSF, which is given by the absolute square of the fourier transform of the electric field $E(x,y)$ at the aperture:

$$PSF(x,y) = |\mathcal{F}(E(x,y))|^2 = |\mathcal{F}(E_0 \cdot e^{-i2\pi W(x,y)})|^2. \tag{3}$$

The Zernike coefficients for the given setup have already been analyzed in [8] and are listed in table 1, leading to the aberration effects visualized in figure 2.

Defoc	1.AstV	1.AstO	ComaV	ComaH	TreV	TreO	Sph	2.AstV	2.AstO
0.101	0.008	0.003	0.000	0.004	0.000	0.004	0.026	0.008	0.007

Table 1: Zernike coefficients[1]of the experiment as determined in [8]

2.3 Atomic distribution

Using the PSF as a kernel $P(x,y)$ to convolve with a distribution of point-like atoms $O(x,y)$, the camera image $I(x,y)$ (figure 3) is obtained by computing

$$I(x,y) = (P * O)(x,y) = \int_{-\infty}^{\infty} P(x-u, y-v) O(u,v)\, du\, dv\,. \tag{4}$$

Before doing so, the volume of the PSF is normalized to 100, which is the target number of photons captured from a single atom in the experiment.

2.4 Discreteness Problem

In numerical simulations as well as digital imaging, the discreteness of the kernel due to the discrete pixels/matrix elements can lead to information loss compared to the

[1]Defoc: Defocus
Ast: Astigmatism
Tre: Trefoil
Sph: Spherical Aberration
V: Vertical
H: Horizontal
O: Oblique

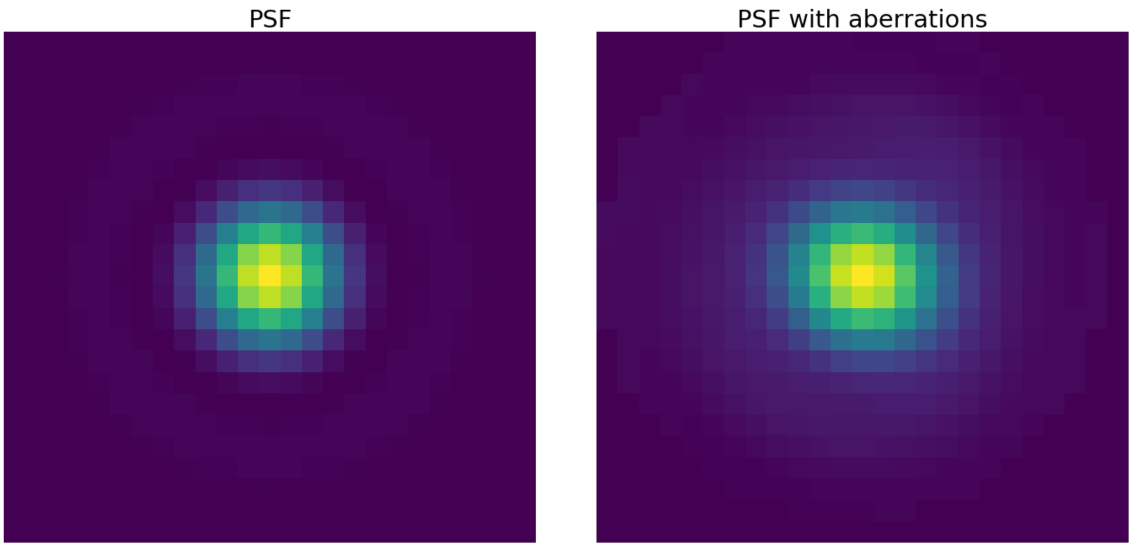


Figure 2: Simulated PSF without (left) and with (right) aberrations

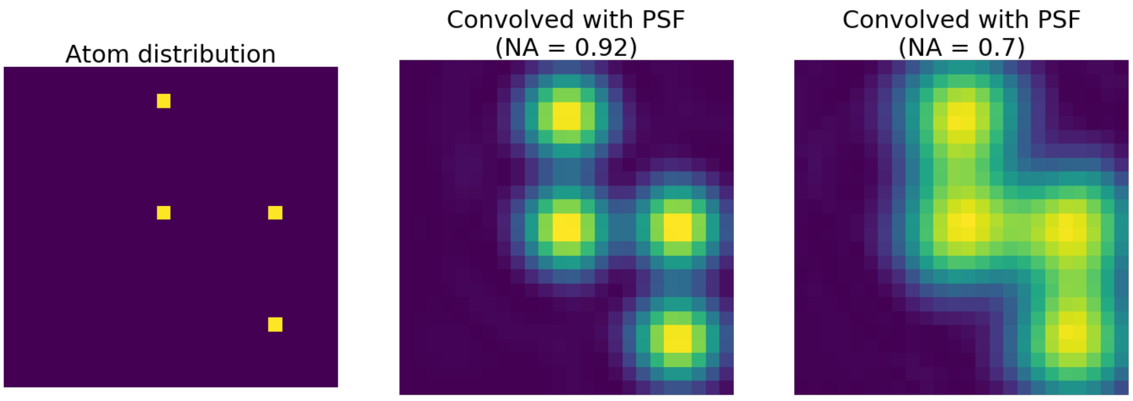


Figure 3: Exemplary atom distribution to be convolved with the PSF (left) and resulting images for different NA (middle, right)

continuous case. However, any aliasing can be avoided by obeying the Nyquist-Shannon sampling theorem, which is fulfilled if the Abbe radius r_{Abbe} is projected onto more than two pixels.

In this work, the target $NA = 0.92$ of our experiment [9] will be used as well as an intermediate, lower value of 0.7, with the wavelength set to $\lambda = 852\,\text{nm}$. The lattice constant in the image is chosen to be $a_{img} = 8\frac{\text{pix}}{\text{lattice site}}$. Knowing the physical lattice constant of $a_{phy} = 612\,\text{nm}$ and calculating the Abbe radius to $r^{phy}_{Abbe} = \frac{\lambda}{2NA} = 463\,\text{nm}$ ($r^{phy}_{Abbe} = 608\,\text{nm}$ for $NA = 0.7$)[2], the Abbe radius has to be projected onto $r^{img}_{Abbe} = 6\,\text{pix}$ ($r^{phy}_{Abbe} = 8\,\text{pix}$), as to keep the physical pixel size for the kernel and the atom distribution the same. This requires the proportionality $\frac{612\,\text{nm}}{463\,\text{nm}} \approx \frac{8\,\text{pix}}{6\,\text{pix}}$ $\left(\frac{612\,\text{nm}}{608\,\text{nm}} \approx \frac{8\,\text{pix}}{8\,\text{pix}}\right)$ to remain approximately constant, which is fulfilled for the chosen values. At this point, it is noteworthy that the image coming from the experiment will very likely not

[2]Numbers given from now on will be for $NA = 0.92$, with values for $NA = 0.7$ following in parenthesis unless otherwise stated

have a lattice constant corresponding to an integer number of pixels. However, this can be solved by choosing an upsampling factor in such a way that one gets a near-integer lattice constant to sufficient accuracy for the given number of lattice sites, as for example:

$$\begin{aligned}
\text{Lattice constant in pixels: } & 3.71833 \\
\text{Upsampling factor: } & 7 \\
& \Downarrow \\
\text{New lattice constant in pixels: } & 26.02831 \approx 26 \\
\text{Error: } \frac{26.02831 - 26}{26} & = 0{,}109\,\%
\end{aligned}$$

In this case, one would accumulate an error of less than 3 pixels over 100 sites. Within this work, the upsampling step will be skipped, instead producing the images directly in a manner that they can be used as input for the networks to follow.

2.5 Lattice drifts

Over time, a drift of the lattice with respect to the camera may occur, giving rise to the need for shifted images in the training process. To this end, the mathematical operation of convolution is used again: Shifting of an image can be achieved by multiplication with a shifted rectangular pulse in Fourier space (Fourier displacement law), or, equivalently, by convolution with the Fourier transform of this pulse in real space, which is given by a *sinc*-function, as stated by the convolution theorem:

$$w(t) = u(t) \cdot v(t) \Leftrightarrow W(f) = U(f) * V(f). \tag{5}$$

By this method, even arbitrary subpixel shifts are possible, but for our data only shifts in pixel steps are considered to avoid an exploding computing amount. To be able to include every possible shift that could occur, each simulated image is shifted by a total of up to one lattice site in both vertical and horizontal direction.

2.6 Noise simulation

Apart from the aberrations introduced by the imaging setup, information in fluorescence microscopy is lost due to different kinds of noise. The largest contribution in this experiment comes from fluorescence photon shot noise originating from photon scattering on atoms as well as limited quantum efficiency for photoelectron generation and the limited discrete resolution while imaging the PSF. Overall, this noise follows Poissonian statistics, as does the second largest source: stray light from the lattice or cooling lasers. Other noise sources are of at least 1 order of magnitude smaller and include intensity fluctuations in the imaging laser, inhomogenous sensitivity in the

CCD chip, readout noise during the analog to digital conversion and amplification and dark current noises from thermal charges [4]. For the simulation, each pixel's value is therefore replaced by a number drawn from a poissonian distribution with a mean of the original pixel value plus an average background noise term of $\sigma_b = 0.3$.
To include crosstalk from neighboring atoms while keeping the data size reasonably small, image size is limited to 3 by 3 lattice sites, with only the central site's occupation state to be determined. For this size, there are $2^9 = 512$ possible combinations of atoms. Together with $8 \cdot 8 = 64$ possible ways of shifting each image by an integer number of pixels up to the lattice constant and the fact that several images for each combination are produced to better include the contingency of the introduced noise, this amounts to several hundred thousand images (see figure 4). The image data is

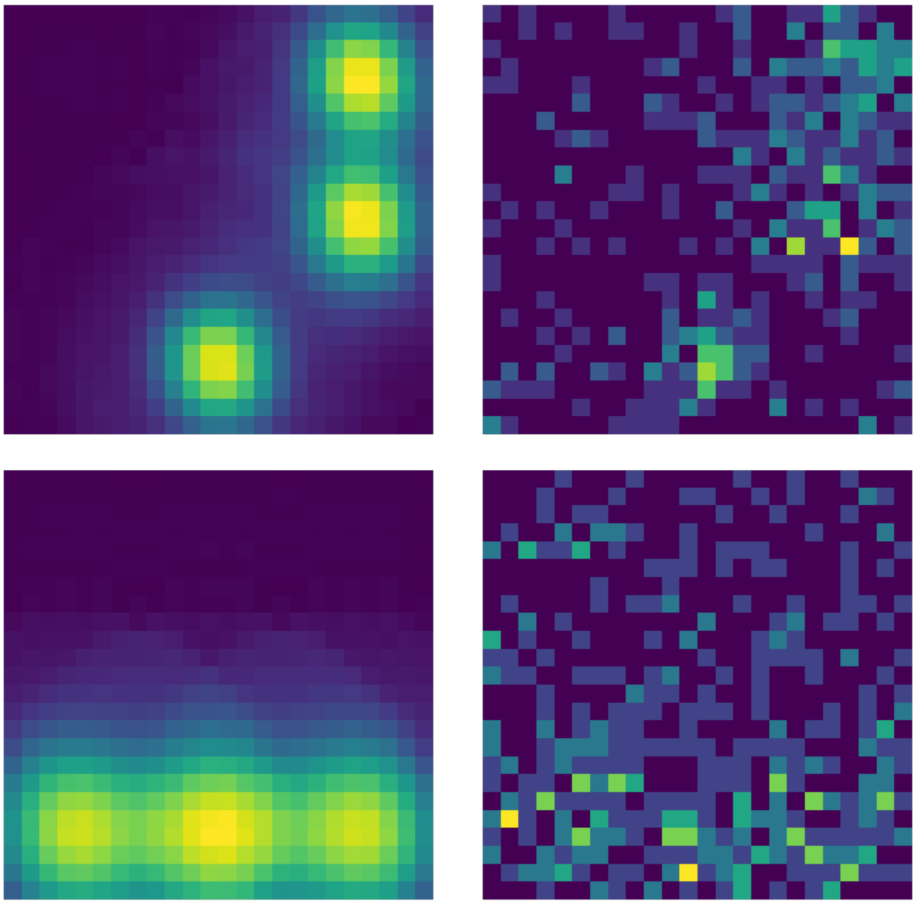


Figure 4: 3 by 3 lattice site image for $NA = 0.92$ (top) as well as for $NA = 0.7$ (bottom) without and with noise contributions

normalized and saved as a list of arrays in a `.csv`-file together with it's occupation state to be read by the following networks.

3 Framework for Classification Tasks

3.1 Linear and Logistic Regression

To gain some intuition, one can look at the simplest form of machine learning, which is linear regression, commonly known as fitting a linear function $f(x) = ax + b$ to a data set. This is done by starting with random parameters a and b for the function $f(x)$, calculating the mean squared errors with respect to all data points, and then iteratively adjusting the parameters to minimize this error. The obtained function then can be used to extrapolate the given data, i.e. to predict new values following the relation extracted from the set. This procedure often works well for continuous data. However, more complicated problems, like if the data belongs to two fixed categories (binary classification problem), require a nonlinear approach. To this end, one can resort to logistic regression, which changes the linear function to the sigmoid function $f(x) = (1 + e^{-(\beta_0 + \beta_1 x)})^{-1}$. While the process stays the same - adjusting the parameters β_0 and β_1 to minimize some measure of error - the function now predicts probabilities between 0 and 1 for an input to belong to one of the two classes rather than values. This behavior can be implemented using a neural network, and all following more complex algorithms are founded on this basic thought of a fitting algorithm.

3.2 Neural Networks

In general, a neural network is nothing more than a mathematical operation, transforming an input vector, like an array of pixel values, into an output vector. Typically, these constructs are subdivided in so called layers, of which there are several kinds. In their simplest form, the so called “feedforward neural networks”, each layer is made up of an number of neurons, each being used as an input to all the neuron of the following layer. This can be written as

$$\mathbf{o}^k = \varphi(W^k \cdot \mathbf{x}^{k-1} + \theta^k) \tag{6}$$

where $\mathbf{x}^{k-1}$ represents the input array from the $(k-1)$th layer, W is the weight matrix connecting that layer with the current one and θ is a bias vector. The result of this matrix multiplication and vector addition then gets fed into the activation function φ, which, while being choosable arbitrarily, usually maps the output back to the interval $(0, 1)$. The resulting output vector $\mathbf{o}^k$ is transmitted to the next layer, constituting the input for the following neurons (see figure neuron). Some commonly used activation functions can be seen in table 2.

To get to the actual training process, the specification of a cost function $C(w)$ is needed, which encodes the difference between the estimated network output and the desired one. Each iteration, the network will evaluate and minimize this function,

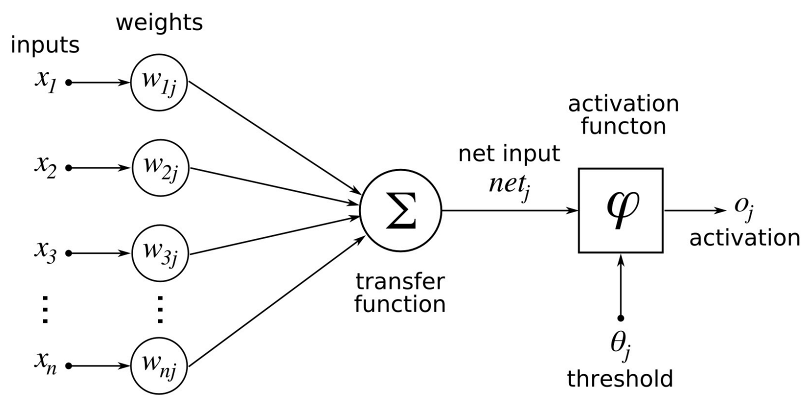


Figure 5: Scematic drawing of a single neuron, the index j counts the neurons in one layer [10]

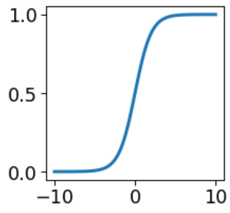
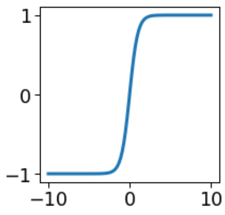
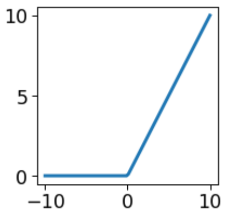
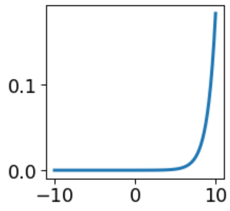
name	function	plot	comment
sigmoid	$\frac{1}{1+e^{-x_i}}$		good for binary classifiers, monotonic and differentiable, maps to (0,1)
hyperbolic tangent	$tanh(x_i)$		improved sigmoid, maps to (-1,1), maps 0 to 0
ReLu	$max(0, z)$		monotonic with monotonic derivative, maps to (0,inf)
softmax	$\frac{e^{x_i}}{\sum_i e^{x_i}}$		good for multi-class classifiers, maps to (0,1), keeps total probability at 1

Table 2: Commonly used activation functions

which can for example be the mean squared error,

$$C(w) = \frac{1}{2} \left\langle \left\| F_w(x^{(0)}) - F(x^{(0)}) \right\|^2 \right\rangle , \tag{7}$$

where $F_w(x^{(0)})$ is the output of the neural network for given input vector $x^{(0)}$ and $F(x^{(0)})$ is the desired output. The minimization is realized by a procedure called gradient descent (GD), which will update the weights according to:

$$w_* \to w_* - \eta \frac{\partial C(w)}{\partial w_*} = w_* - \eta \left\langle \sum_j \left(o_j^{(k)} - F_j(x^{(0)}) \right) \frac{\partial o_j^{(k)}}{\partial w_*} \right\rangle \tag{8}$$

where η is called the learning rate and k is the layer index. The process of updating the weights is called back-propagation, and is done for every single weight in the network. Intuitively, it would make sense to do this every time a new input has been processed. However, this would mean a huge amount of computations, so in reality, the procedure is applied only for a stochastic sample of the training inputs fed to the network at once. The decision of how much data one will feed at once is a question about the batch size. Given unlimited computing power and memory, it would make sense to use all the training data available and make a single batch from it. The drawback of this method is the need to hold and process huge amounts of data simultaneously, as well as the fact that the backpropagation step is used less often, leading to a much slower training progress. Choosing the batch size to be the whole training data is called batch gradient descent (BGD). On the other hand, one could set the batch size to 1, using backpropagation as often as possible. While the training will converge faster (in less epochs), training time will increase considerably due to a higher computation amount. This is called stochastic gradient descent (SGD). Here, an epoch describes the training process on all available data once. What is chosen in reality lies somewhere in between and is called a mini-batch gradient descent (MBGD), with a batch size to be chosen according to the available hardware by the user.

3.3 Convolutional Neural Networks

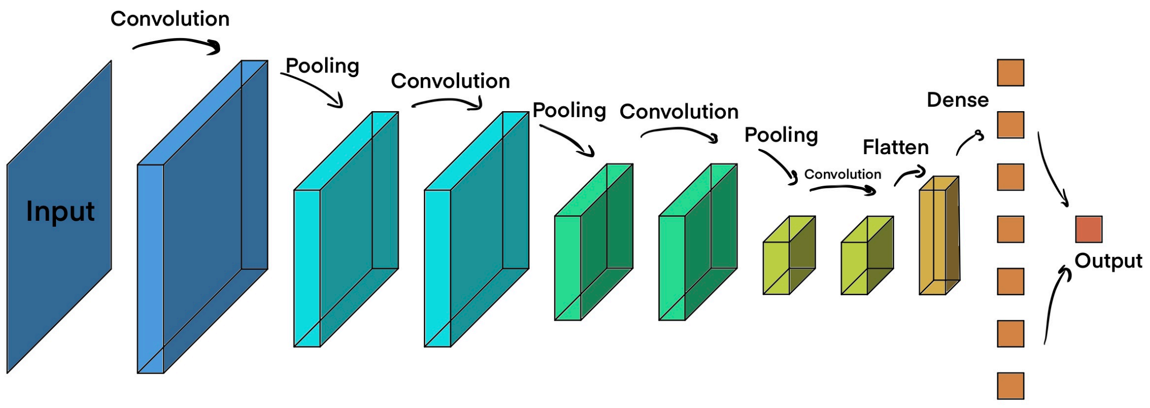


Figure 6: Exemplary CNN layer structure

Convolutional neural networks use some more advanced layers than the dense layers described in 3.2. In contrast to those, the convolutional layer uses one or multiple trainable kernel maps to iterate over small sections of the input and compute the convolution (see figure 7) of the current input region with the kernel. The parameters being changed during training are now the kernel values. Convolutional layers are especially useful, as they allow the extraction of certain features from the data. For example, a unitary matrix as a kernel results in a blur, while other kernels can be used for edge detection and more. Besides, there is the pooling operation, which takes

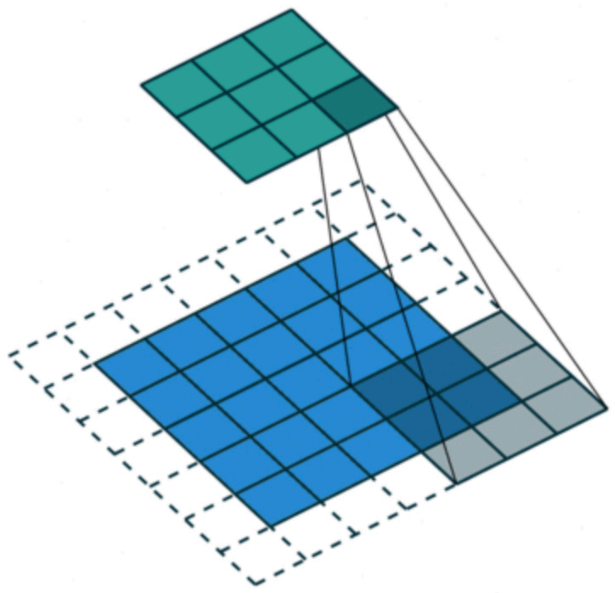


Figure 7: the convolutional layer, with the input map in blue with white padding and the kernel in grey, being convoluted to form the output map in green [11]

the average or maximum of a specified square size of pixels as new single values, thus reducing the size of the data and computing effort considerably, but at the cost of information loss concerning small details. However, not all the information in the data is useful to get to the wanted solution, or else, this type of network can never work. Thirdly, the batch normalization layers are to mention, which have been proven to be very effective in accelerating training convergence by normalizing the weights after each batch, even though it is not entirely clear where the improvement originates from [12]. Another very useful concept is the so-called dropout. This deactivates randomly selected neurons during each training epoch, therefore prohibiting networks from using only a few neurons strongly and increasing their capability to generalize.

3.4 Autoencoder

A specific type of convolutional networks is the autoencoder. It consists of two parts: an encoder, which takes an input image and reduces its size over several convolutional and pooling layers, as well as a decoder, which works the opposite way, such that the final output has the same dimensions as the input. Apart from the nonlinearity introduced by the activation function, the encoder works similarly to a principal component analysis (PCA) for dimensionality reduction. The so-called bottleneck in the middle ensures the network generalizes important features rather than memorizing the inputs and their desired outputs. The layer structure used is shown in figure 8. The fact that the autoencoder can represent input data in a lower dimensional manifold means that it can be used for example to reduce noise effects.

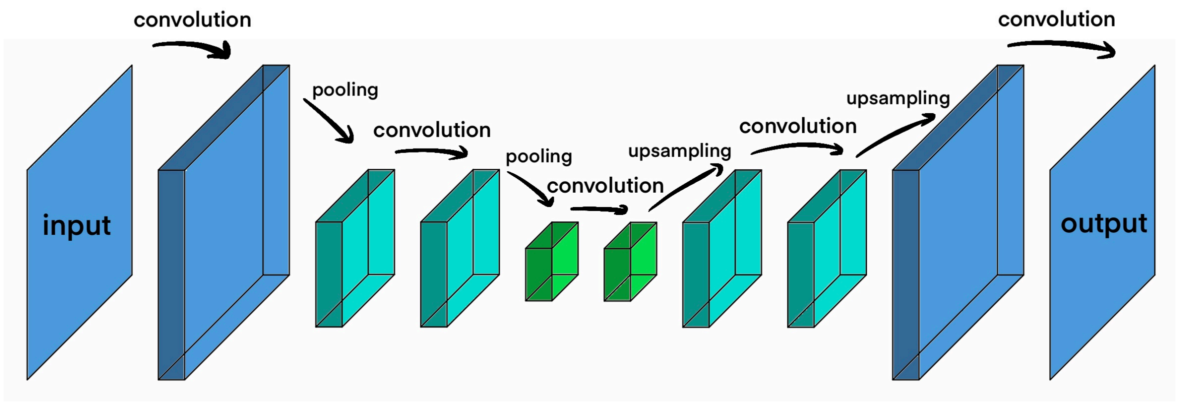


Figure 8: Exemplary autoencoder layer structure, with the encoder part (left) and the decoder part (right) forming a bottleneck

3.5 Training evaluation

While the process in 3.2 has introduced the mechanism of neural networks already, some additional details are important to ensure an analysis of the network's quality can be carried out later on. As the use-case of the network will always be to deal with prior unseen inputs, it makes sense to split up the available input data into the following subsets:

- **Training set**: Contains the biggest part of inputs, which will be divided into batches and run through the process described in 3.2 during training
- **Validation set**: Will be fed into the network after each epoch (loop through all inputs from the training set once) not to train on, but to evaluate the current network state
- **Test set**: Will be used to test the network once training is completed, to test it's generalization capability on unseen data

This splitting allows for an overview of a model's training history by plotting the value of the loss function and the prediction accuracy after each training epoch. The comparison between the evolution of the loss function for the training and the validation set during the training process can already give first clues about the quality of the model. Observing a constant or even rising validation loss together with falling training loss is a sign for overfitting, i.e. the model is fitted too close to the training data and therefore loses it's capability of abstraction. To avoid this, the model can be made more simplistic, or a dropout can be introduced. The opposite case may also occur: underfitting may be present if the training loss falls much lower than the validation loss. In this case, additional complexity can be introduced to the model as well as more training data, so that more features can be extracted.

3.6 Error metrics

Although a high accuracy / low loss is an indicator of a well functioning model, it is not sufficient to evaluate the models quality. This is especially important for imbalanced data: given a set of 99 % circles and 1 % squares, a model that only ever predicts "circle" will have an accuracy of 99 %. Therefore, for binary classifiers, one can look at the confusion matrix (see table 3), representing the following cases:

		actual	
		1	0
predicted	1	TP	FP
	0	FN	TN

Table 3: Confusion matrix

- **True Positive (TP):** An input that possesses the feature one is looking for and has been successfully recognized as such.
- **True Negative (TN):** An input that does not posses the feature and has been successfully recognized as such.
- **False Positive (FP / Type 1 Error):** An input not possessing the feature, but being identified to do so.
- **False Negative (FN / Type 2 Error):** An input possessing the feature, but not being detected as such.

From this, two useful quantities to extract are the true positive rate (TPR) and false positive rate (FPR)

$$\mathrm{TPR} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}} \qquad \mathrm{FPR} = \frac{\mathrm{FP}}{\mathrm{FP} + \mathrm{TN}}, \tag{9}$$

which can be plotted against each other for all thresholds between 0 and 1, resulting in the receiver operating characteristic curve (ROC). The threshold defines from which value on an output will be interpreted as *true*, and a perfect classifier will show as a point in the upper left corner of the ROC plot, thus maximizing the area under the curve (AUC) to 1 (see figure 9).

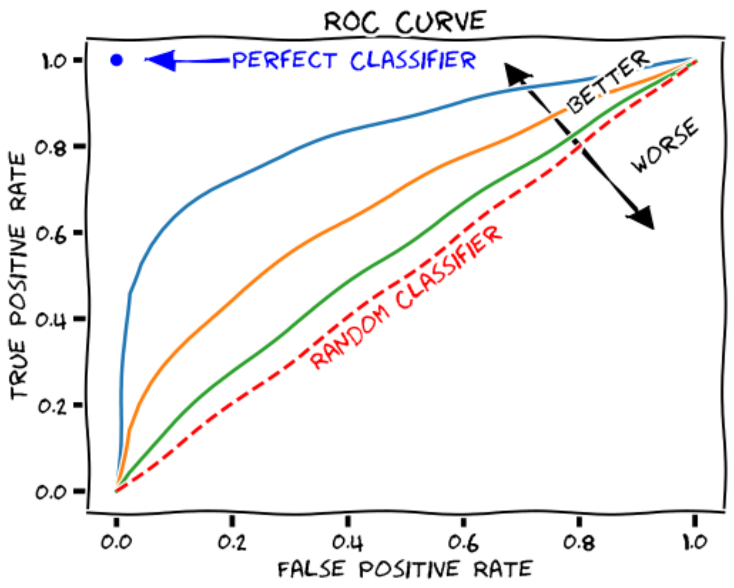


Figure 9: The binary classifier error metric ROC [13]

4 Image Classification

4.1 Prediction neglecting the image shift

Looking at the problem of atom distribution prediction, and neglecting the shift of images for now, a simple feedforward network is constructed to predict the middle lattice site state out of the simulated 3 by 3 site image segments. The network has only one hidden (i.e. between input and output) layer containing 512 neurons, which is the number of possible constellations of atoms in 9 lattice sites. Even before training, one can think of how to initialize the weights to each neuron. As suggested in [14], a Gaussian mask reflecting one of the possible atom occupation states for each hidden layer neuron is chosen. The activation function for the hidden layer is just a linear function, and the output uses the sigmoid function to map the predictions to probabilities. As the network output is basically the weighted sum of all the neurons, and the goal is to decide whether the middle lattice side is occupied or not, all those neurons representing an atom combination with no atom in the center will get a negative sign on their weights. Evaluating the data without any training at all results in an accuracy of 93,01 % (90,20 %) at the experimentally determined optimal σ of 6.9 for the Gaussian

masks used.
For the training process, the available 51 200 simulated images are split into training, test and validation set as described in 3.5. After training for 100 epochs with a batchsize of 2000 this leaves us with the evolution of the loss function as depicted in figure 10. The ROC curves shown in figure 11 are obtained, from which it is clear that even this simple model works quite well on the given data without shift. Looking at the confusion matrix, an accuracy of 99,73 % (98,55 %) is obtained, with Typ I and II errors being distributed almost evenly, as one would expect from a balanced dataset (table 4). The algorithm therefore is able to find a weight distribution which is even more useful than the one specified manually. This result can be improved even further

NA		predicted		NA		predicted	
0.92		1	0	0.7		1	0
actual	1	49,58 %	0,11 %	actual	1	49,52 %	0,59 %
	0	0,16 %	50,15 %		0	0,87 %	49,02 %

Table 4: Confusion matrices for the feedforward network trained on 3 by 3 lattice site images without shift

by employing a more complex convolutional network, with the layer structure seen in figure 6, resulting in an accuracy converging to 99,98 % (99,41 %) (see figure 10 and table 5) and a bigger area under the ROC curve (see figure 11). Pooling size has been chosen to 2x2, while the kernel size of all convolutional layers is 3x3, with the ReLu activation function. Like before, the last layer employs the sigmoid function to map the output to the interval (0,1). The amount of kernel maps in the first layer has been experimentally chosen to 16, which is doubled for every subsequent convolution. Between each of them, additional batch normalization as well as a dropout of 25 % has been introduced. As expected, the performance for the lower NA is consistently

NA		predicted		NA		predicted	
0.92		1	0	0.7		1	0
actual	1	49,88 %	0,00 %	actual	1	49,28 %	0,49 %
	0	0,02 %	50,10 %		0	0,10 %	50,13 %

Table 5: Confusion matrix for the convolutional neural network (left: $NA = 0.92$, right: $NA = 0.7$)

lower, which can be explained by the bigger overlap of the PSFs, leading to a higher difficulty in separating neighboring atoms. Regarding the indicators discussed in 3.5, no over- or underfitting can be observed for both models. The higher performance of the CNN is accompanied by a much quicker convergence, showing only very little further improvement after the first few epochs, while the feedforward network might even take some more epochs than the 100 trained to converge to a relatively stable loss. In fact, for $NA = 0.92$, the CNN model performs so well that it is found to be

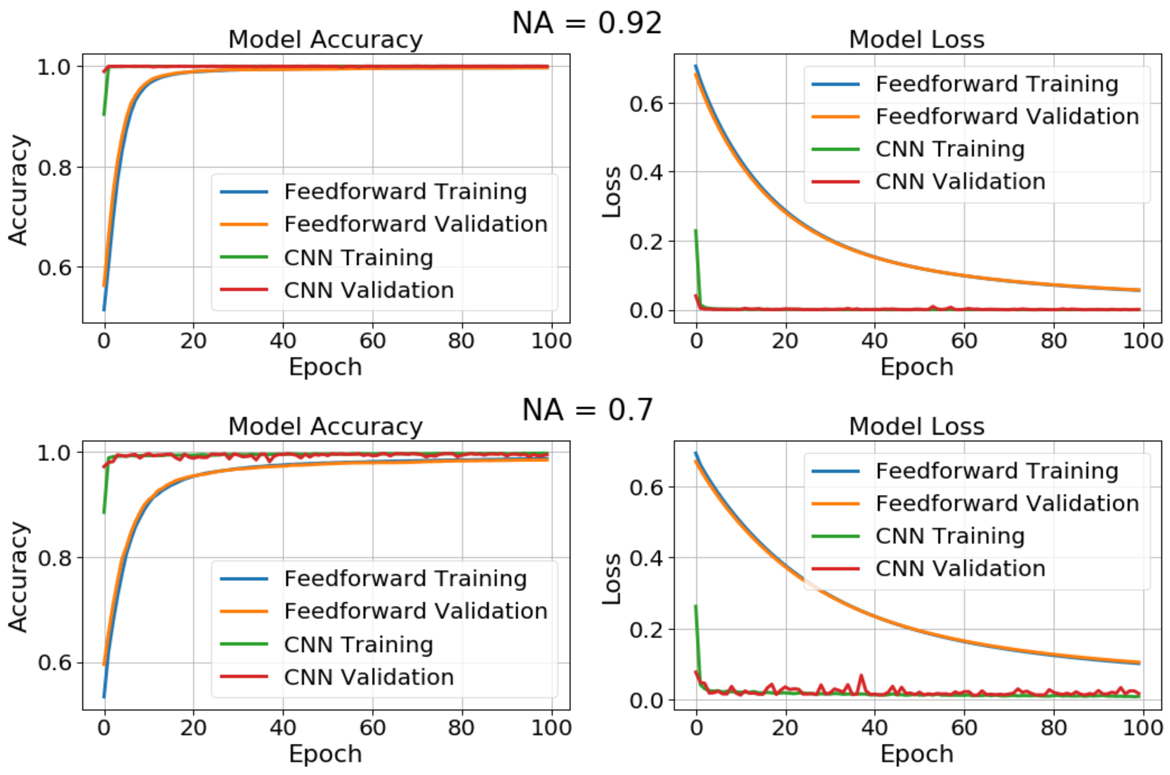


Figure 10: Development of accuracy and the cost function during training for a simple feedforward network and the CNN model, using $NA = 0.92$ (top) and $NA = 0.7$ (bottom) for nonshifted images

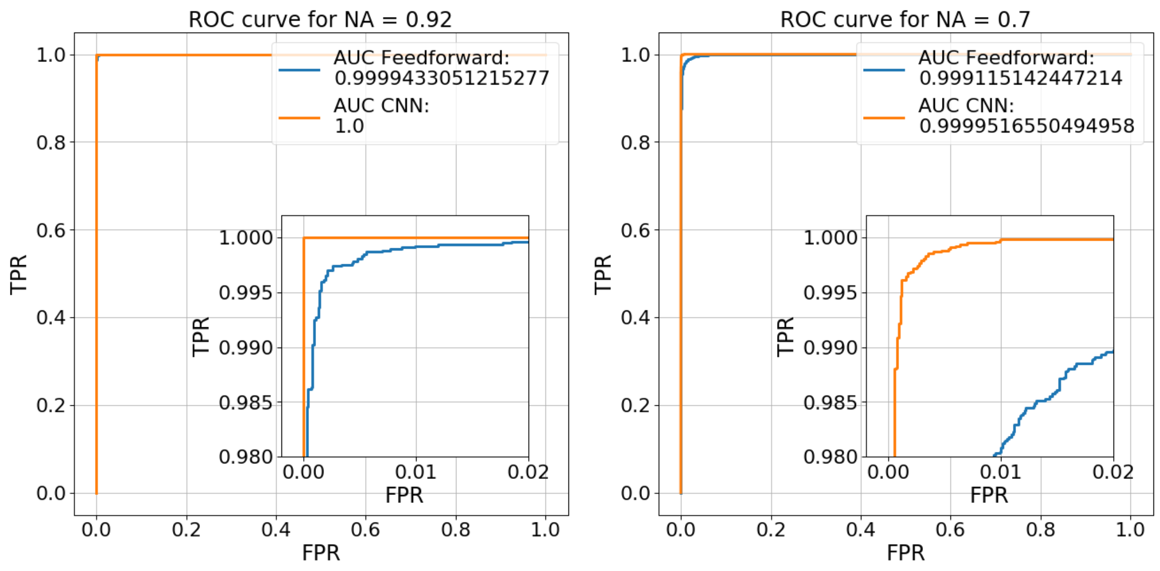


Figure 11: ROC curve for the feedforward and convolutional network for $NA = 0.92$ (left) and $NA = 0.7$ (right) for nonshifted images

a perfect classifier (AUC=1), meaning out of all the test images, not a single one was misclassified. However, this only works for the data without shift.

4.2 Prediction including the image shift

Considering the additional complication of shifted images, the models used so far suffer in performance with accuracy ratings of 92,01 % and 87,59 % for the feedforward method and 99,05 % and 96,34 % with the CNNs, for $NA = 0.92$ and $NA = 0.7$ respectively. The performance degradation can be observed already in the visualization of the training progress (figure 12), with a very notable drop in accuracy / higher loss function for the feedforward model and a considerable increase of fluctuations on the CNN validation data. This impression is emphasized further by the ROC curves depicted in figure 13, and the confusion matrix (table 6), thus giving rise to the need for a more sophisticated architecture.

After consideration and experimentation on different aspects of the detection problem, it has become apparent that the image shift is best treated in a separate step. Therefore, following scheme has been established:

NA=0.92 FF		predicted 1	predicted 0
actual	1	45,42 %	3,46 %
	0	4,54 %	46,59 %

NA=0.7 FF		predicted 1	predicted 0
actual	1	43,75 %	6,22 %
	0	6,20 %	43,83 %

NA=0.92 CNN		predicted 1	predicted 0
actual	1	49,54 %	0,46 %
	0	0,48 %	49,51 %

NA=0.7 CNN		predicted 1	predicted 0
actual	1	48,08 %	2,03 %
	0	1,63 %	48,25 %

Table 6: Confusion matrices for the feedorward (FF) and CNN model trained on the shifted data

Multistep architecture

1. The images undergo denoising, realized by an autoencoder network.
2. The denoised segments are shiftcorrected by another autoencoder.
3. The atom distribution is predicted with a convolutional neural network.

In the following, the individual steps will be explained and characterized before summarizing the results of the total algorithm in comparison with the models established so far.

4.2.1 Noise Removal

In order to deal with the lattice shift in the images, it is crucial to reduce the otherwise dominating noise effects beforehand. To this end, an autoencoder with layer structure as seen in figure 8 is employed. Each convolutional layer operates on 128 kernels, giving

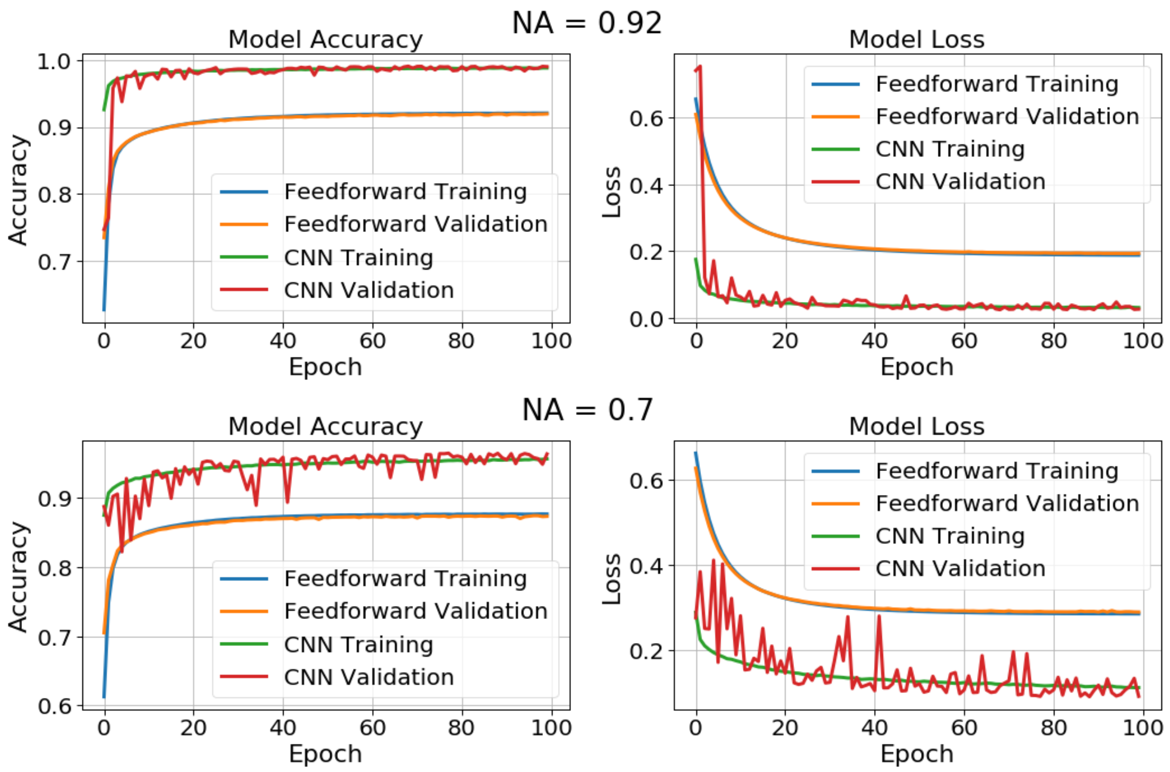


Figure 12: Change of accuracy and the cost function during training for the feedforward and CNN model trained on the shifted data

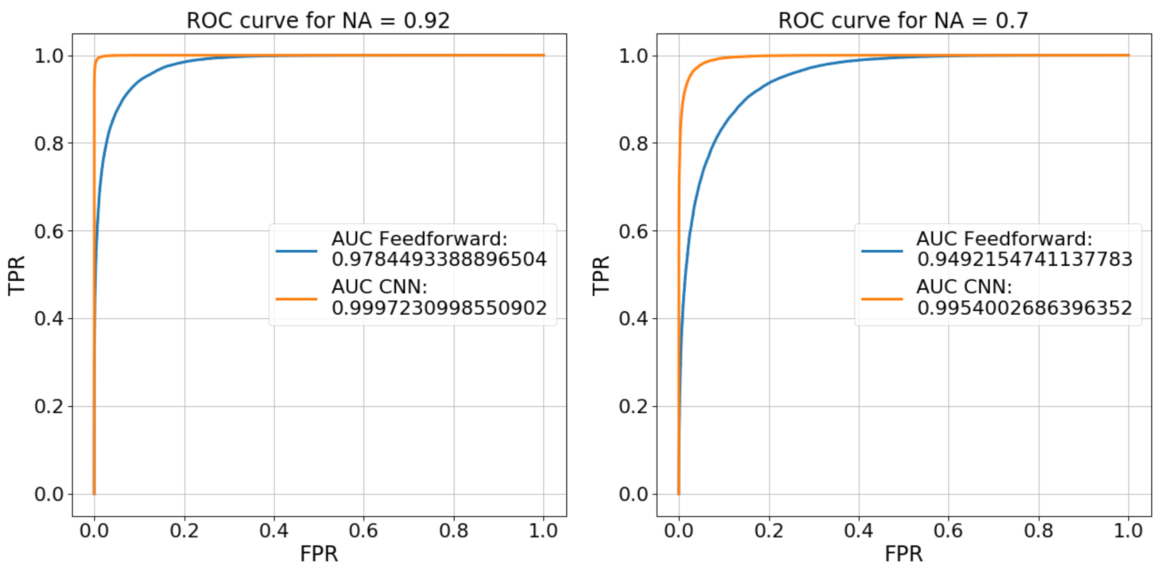


Figure 13: ROC curve of feedforward and convolutional networks on shifted input images for $NA = 0.92$ (left) and $NA = 0.7$ (right)

the network a total of over 440 000 free parameters to optimize. Similar to the CNN, ReLu is chosen as the activation function, apart from the last layer, which uses the sigmoid function. Training happens on pairs of noisy and noiseless images, leading to a training progress after 100 epochs on a total of over 300 000 images that reduces the loss function to about 0,002 (0,005) (see figure 14). While training and validation loss coincide nicely for the higher numerical aperture, the graph on $NA = 0.7$ shows some overfitting towards the end. Some examples in figure 15 show, that especially for $NA = 0.92$, the reconstruction of noiseless images works very well, with minor deviations like the leftmost atoms in the second row being shifted slightly downwards. For $NA = 0.7$, reconstruction is more challenging, and especially images with tightly packed atoms tend to blur out a bit, as shown in the last row. However, the performance of the autoencoder is found to be sufficient to go on with the second step. To get

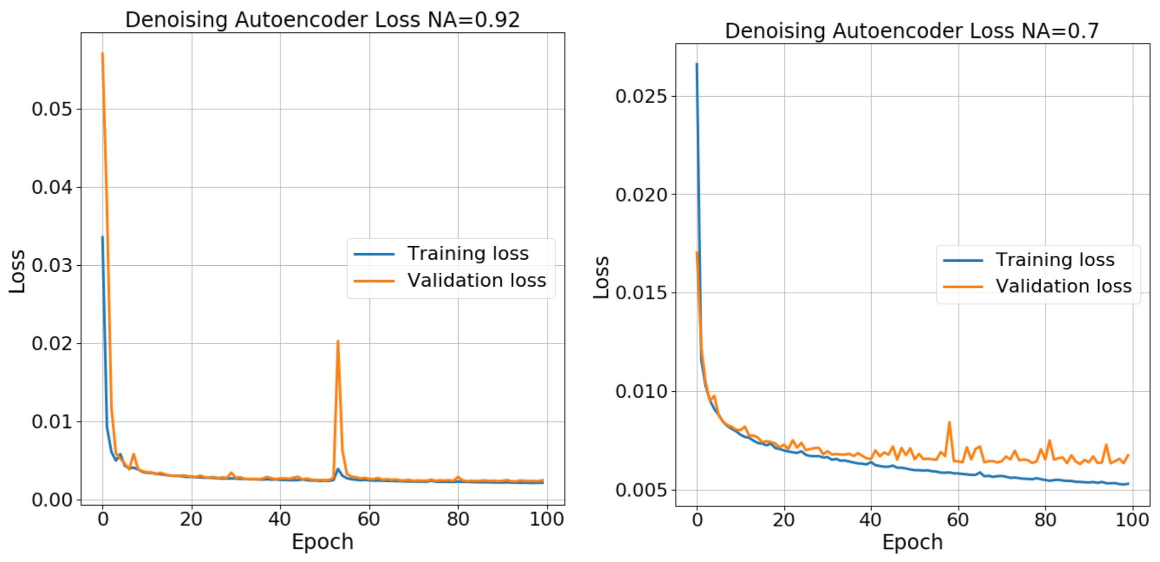


Figure 14: Developement of the loss function during training for the denoising autoencoder, using $NA = 0.92$ (left), and $NA = 0.7$ (right)

an impression of the networks robustness against changes in the input noise, some images with background noise levels of $\sigma_b = 0.1$, $\sigma_b = 0.2$, $\sigma_b = 0.4$ and $\sigma_b = 0.5$ have been produced and fed to the trained network, revealing no noticeable increase in the residuals for the lower noise levels as well as $\sigma_b = 0.4$, and some degradation in reconstruction quality for the highest noise level.

4.2.2 Shift Correction

After successfully reducing the noise, the shift can now be treated. A second autoencoder is employed for this, structured just like before, but now trained on pairs of the denoised segments with the nonshifted noiseless segments. The model is again trained for 100 epochs and its results are shown in figure 16, with a very good accordance between training and validation loss for the higher NA, while the graph for the lower value shows some overfitting in the later epochs. Figure 17 shows some examples of the autoencoders predictions, chosen to represent cases where the algorithm worked very well as well as cases where it didn't. Clearly, in the first row the shift is corrected nicely, with the only difference between desired and real output being the atom on the lower left position, which was reconstructed a little too faint. The second row exhibits almost no noticeable difference to the original. The third row is a good example for

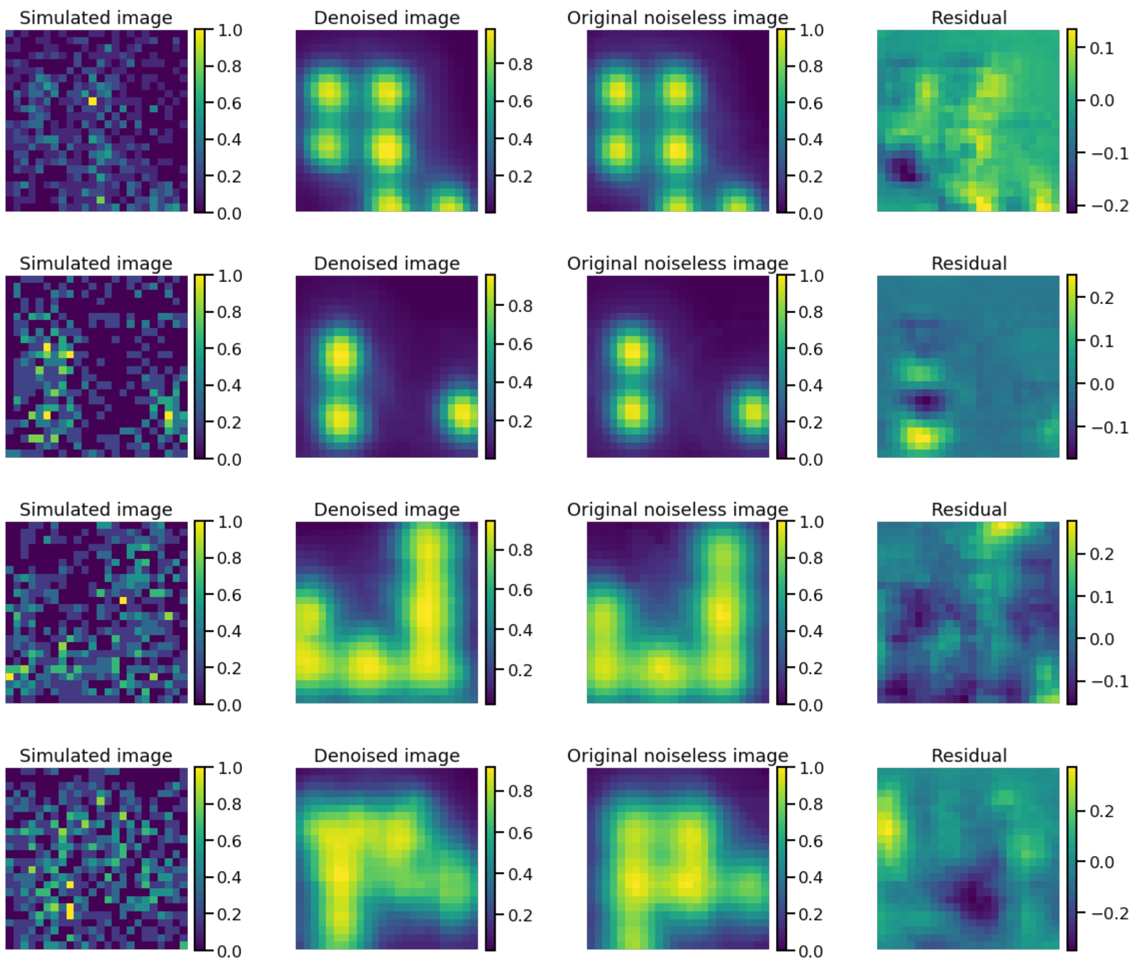


Figure 15: LR: Input image segment, output of the autoencoder network, desired output, deviation of output from desired output. The first two rows are for $NA = 0.92$, the last two for $NA = 0.7$

$NA = 0.7$, there is only a slightly too bright patch in the middle. In the last row, an example of bad reconstruction can be seen, where one atom has been blurred to the point that it is not anymore distinguishable from the background. Still all in all, most of the atoms are reconstructed on their original positions. The treatment of the shift has posed the biggest challenge during the course of this work, and different approaches have been considered. While this method does not make use of the fact that the shift is a global property of each big experimental image, but treats it individually for each segment, and does not return any information on the direction and magnitude of the shift itself, it still has proven to be the most accurate of the approaches. The evaluation of the correctness of the shift detection can only be inferred from the residuals to the original images and the evolution of the loss function. A more quantitative information will be obtained by looking at the whole process in 4.2.3.

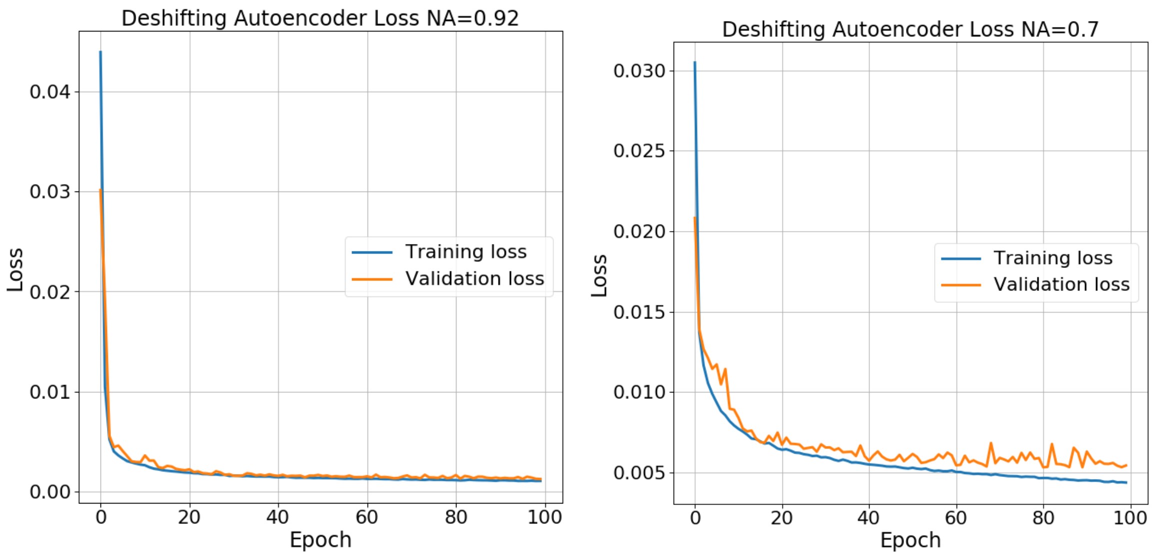


Figure 16: developement of the loss function during training for the shiftcorrection autoencoder, using $NA = 0.92$ (left), and $NA = 0.7$ (right)

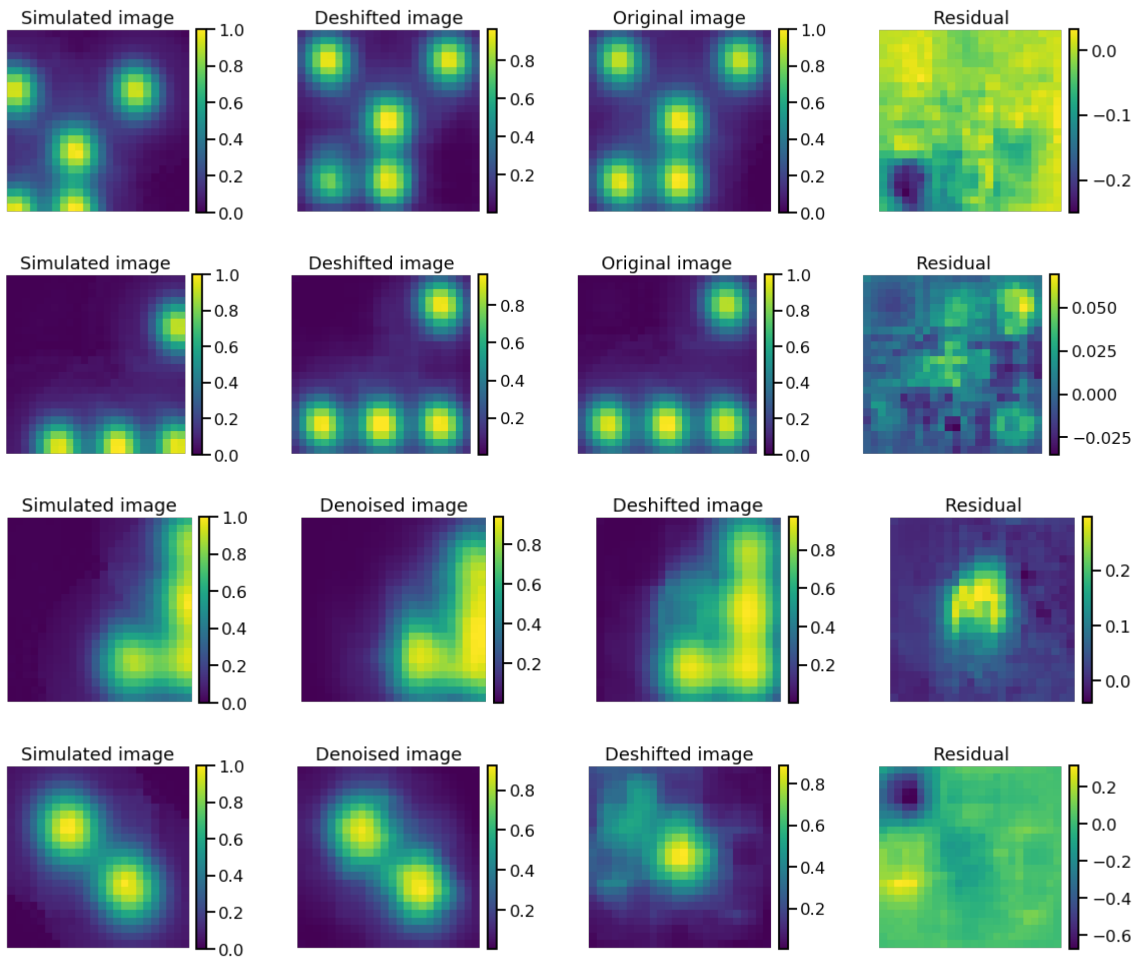


Figure 17: LR: Simulated noiseless image segment, output of the denoising autoencoder network, output of the deshifting network, deviation of output from desired output. The first two rows are for $NA = 0.92$, the last two for $NA = 0.7$

4.2.3 Position Prediction

In the last step, to predict the occupation state of the central lattice site of the now denoised and shift corrected segments, the same architecture as shown in 6 is employed. The training progress is depicted as the development of accuracy and the loss function over 100 epochs in figure 18 and the binary classifier metric ROC can be seen in figure 19. No over- or underfitting has been observed, and from the areas under the ROC curves of $AUC = 0.9994$ ($AUC = 0.9985$), an improvement over the previous model can already be inferred. The total accuracy can be read from the confusion matrix in

NA 0.92		predicted 1	predicted 0	NA 0.7		predicted 1	predicted 0
actual	1	49,85 %	0,15 %	actual	1	49,39 %	0,61 %
	0	0,10 %	49,90 %		0	0,37 %	49,63 %

Table 7: Confusion matrix for the multistep architecture (left: $NA = 0.92$, right: $NA = 0.7$)

table 7 to be 99,76 % (99,03 %), which is an improvement of 0,71 % (2,69 %) over the simpler architecture in 4.1 using the CNN only (see table 6).

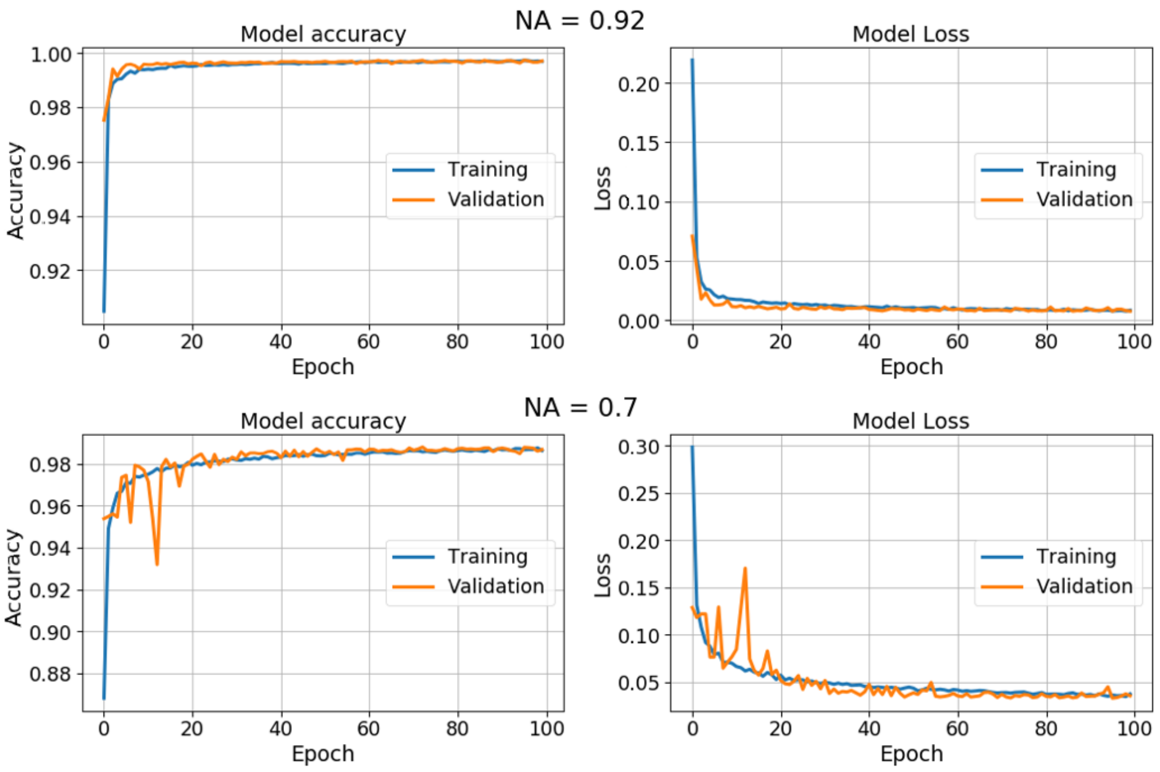


Figure 18: Evolution of the accuracy and loss function during training for the position prediction CNN in the multi-step algorithm, using $NA = 0.92$ (top), and $NA = 0.7$ (bottom)

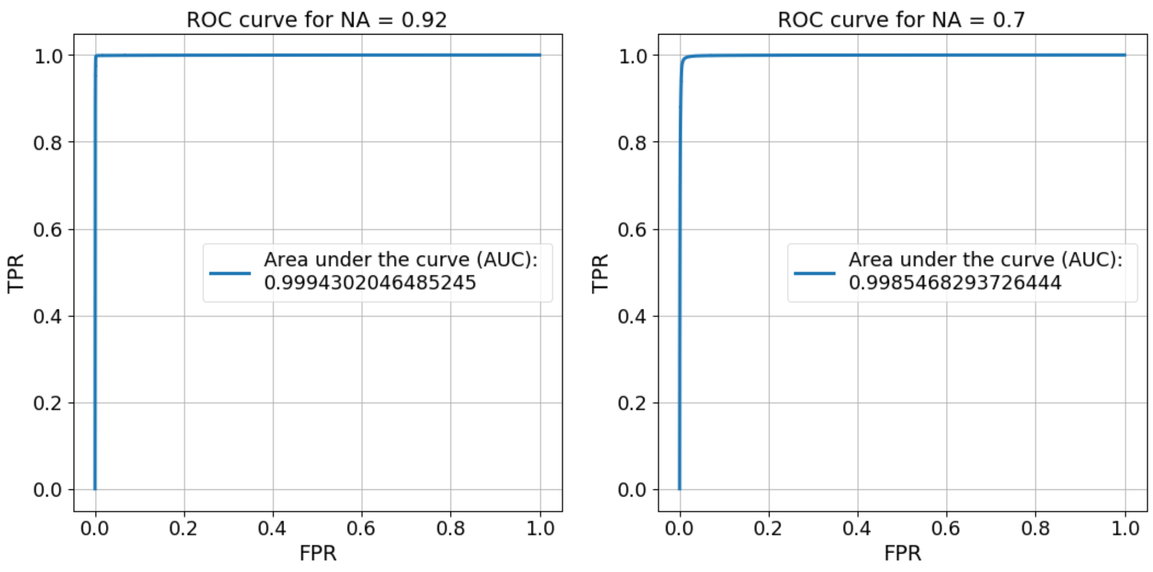


Figure 19: ROC curve for the position prediction CNN of the multistep algorithm for $NA = 0.92$ (left) and $NA = 0.7$ (right)

5 Conclusion and Outlook

In this work, a simulation algorithm for atoms in two-dimensional optical lattices for variable numerical aperture has been developed, being capable of producing square grids of arbitrary size and filling fraction. Besides, it takes into account imaging noise and lattice drifts with respect to the camera. The obtained training data has been used to train and compare machine learning algorithms of different complexities and architectures with the aim of predicting the occupation state for each lattice site. It has been shown, that for images without shift, even the most simplistic networks are able to recognize atoms to a high accuracy. The biggest challenge for the algorithms is posed by the lattice shift. A more complex approach has been developed, dealing with this complication separately, and has been proven to reduce recognition errors down to under 0,3 % (1 %).

To apply the algorithm to real world data, the simulations should be refined to fit the final parameters of the experiment as closely as possible. A first step would be to chose a more complex model for the simulation of noise, to accurately represent the different sources identified in [4]. Also, some parameters might have to be adjusted once the experiment reaches it's final state. Besides, it might be worth investigating the aberrations characterized by the Zernike coefficients for a position dependence on the lens, and including those effects in the simulation. Next, the preparatory step of a fitting upsampling should be implemented, which determines the final lattice site separation in the input images and therefore is crucial to train the network. Besides, as images from the experiment will include significantly more lattice sites, an appropriate algorithm cutting them into smaller segments needs to be introduced.

List of Figures

List of Tables

Bibliography

[1] Christian Gross and Immanuel Bloch. "Quantum simulations with ultracold atoms in optical lattices". In: *Science* 357(6355).2 (2017), 995–1001. ISSN: 1361-6501. DOI: 10.1088/1361-6501/ab44d8. URL: http://dx.doi.org/10.1088/1361-6501/ab44d8.

[2] Chuanwei Zhang, S. L. Rolston, and S. Das Sarma. "Manipulation of single neutral atoms in optical lattices". In: *Physical Review A* 74.4 (2006). ISSN: 1094-1622. DOI: 10.1103/physreva.74.042316. URL: http://dx.doi.org/10.1103/PhysRevA.74.042316.

[3] J. H. Lee et al. "Robust site-resolvable quantum gates in an optical lattice via inhomogeneous control". In: *Nature Communications* 4.1 (2013). ISSN: 2041-1723. DOI: 10.1038/ncomms3027. URL: http://dx.doi.org/10.1038/ncomms3027.

[4] Andrea Alberti et al. "Super-resolution microscopy of single atoms in optical lattices". In: *New Journal of Physics* 18.5 (2016), p. 053010. ISSN: 1367-2630. DOI: 10.1088/1367-2630/18/5/053010. URL: http://dx.doi.org/10.1088/1367-2630/18/5/053010.

[5] Alexander Koryagin et al. *Seismic horizon detection with neural networks.* 2020. arXiv: 2001.03390 [cs.CV].

[6] Horace He. *The State of Machine Learning Frameworks in 2019.* [Online; accessed January 7, 2020]. 2019. URL: https://thegradient.pub/state-of-ml-frameworks-2019-pytorch-dominates-research-tensorflow-dominates-industry/.

[7] Xiaolong Liu, Zhidong Deng, and Yuhan Yang. "Recent progress in semantic image segmentation". In: *Artificial Intelligence Review* 52.2 (2018), 1089–1106. ISSN: 1573-7462. DOI: 10.1007/s10462-018-9641-3. URL: http://dx.doi.org/10.1007/s10462-018-9641-3.

[8] Weiqi Zhou. "Analyse der Punktspreizfunktion des Abbildungssystems vom DQSIM-Experiment anhand der Fluoreszenzaufnahmen". Bachelor's Thesis. Rheinische Friedrich-Wilhelms-Universität Bonn, Aug. 2016.

[9] F. Kleißler. "Assembly and Characterization of a High Numerical Aperture Microscope for Single Atoms". Rheinische Friedrich-Wilhelms-Universität Bonn, 2014.

[10] Chrislb. *Artificial Neuron Model.* [Online; accessed January 7, 2020]. 2005. URL: https://en.wikipedia.org/wiki/Backpropagation#/media/File:ArtificialNeuronModel_english.png.

[11] Sumit Saha. *Convolution Operation with Stride Length = 2*. [Online; accessed January 7, 2020]. 2018. URL: `https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53`.

[12] Sergey Ioffe and Christian Szegedy. *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift*. 2015. arXiv: `1502.03167 [cs.LG]`.

[13] MartinThoma. *ROC CURVE*. [Online; accessed January 7, 2020]. 2018. URL: `commons.wikimedia.org/wiki/File:Roc-draft-xkcd-style.svg`.

[14] Lewis R B Picard et al. “Deep learning-assisted classification of site-resolved quantum gas microscope images”. In: *Measurement Science and Technology* 31.2 (2019), p. 025201. ISSN: 1361-6501. DOI: `10.1088/1361-6501/ab44d8`. URL: `http://dx.doi.org/10.1088/1361-6501/ab44d8`.