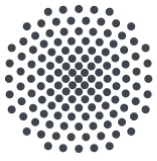
Machine Learning Based Image Analysis for a Dysprosium Quantum Gas Microscope

A Project Report Submitted by

Rana Ekmekcioglu

in partial fulfillment of the requirements for the award of the degree of

B.Sc. Physics



University of Stuttgart
Germany

Submitted to the University of Stuttgart

5th Institute of Physics

October, 2022

Ehrenwörtliche Erklärung

Hiermit versichere ich,

- dass ich meine Arbeit selbständig verfasst habe,
- dass ich keine anderen als die angegebenen Quellen benutzt und alle wörtlich oder sinngemäß aus anderen Werken übernommenen Aussagen als solche gekennzeichnet habe,
- dass die eingereichte Arbeit weder vollständig noch in wesentlichen Teilen aus dem Gegenstand eines anderen Prüfungsverfahrens besteht,
- dass das elektronische Exemplar mit den anderen Exemplaren übereinstimmt.

Ort, Datum

Rana Ekmekcioglu

Abstract

A new setup of the ultracold atoms experiment of the 5th Physical Institute at the University of Stuttgart is under construction, where it is aimed at studying the dipolar Bose and/or Fermi Hubbard model using optical lattices. Different phases are expected to emerge, which will be characterized by local and non-local order parameters. For this characterization, the correlation functions need to be known. The calculation of these functions necessitate single-site as well as single-atom resolution images of the Dysprosium atoms in the optical lattice. The challenge is to achieve this detection accuracy under the conditions when the neighbouring atoms are separated by much less than the diffraction limit of the imaging system or when the lattice is not continuously cooled during imaging etc. This thesis aims to develop a deep learning-assisted reconstruction method that works efficiently under such limiting circumstances and may improve upon the classification accuracy of threshold-based methods. Three types of artificial neural networks are implemented and trained to classify the lattice sites in a given image as occupied or unoccupied. To train, test and evaluate different models under supervision, large sets of labelled images are simulated. The result of the evaluation of each model and their comparison was that a three-layered convolutional neural network (CNN) is capable of achieving an accuracy up to 2% higher than the two simpler artifial neural networks, one of which is an analogue of the weighted sum based threshold classifiers. It was seen that the more training examples were used, the performance of the CNN improved further, reaching 83% for a figure of merit of 1.21 and 98.9% for 2.42. With further improvements of the simulations, by using larger training sets and testing other parameters, it may be possible for the CNNs to achieve higher accuracy for small figure of merit values.

Contents

1 Introduction

Based on the ideas first proposed by S. Bose, A. Einstein developed the Bose-Einstein statistics in 1924 within the newly forming field of quantum statistics [18]. By extending these ideas to massive particles, he predicted a possibility of the formation of a state of matter called Bose-Einstein condensate (BEC), which could be produced and observed for the first time in 1995 by E. Cornell and C. Wieman and shortly thereafter by W. Ketterle, which lead to a Nobel prize for the three in 2001. From that day on, BECs never ceased to be an object of interest for researchers, in particular for those who aimed to study ultracold quantum gases that serve as model systems to investigate fundamental as well as practical questions from quantum physics to condensed matter physics [10, 11, 13, 25, 36, 47, 63]. Besides from the intensely researched short range-interactions in these gases, long-range dipole-dipole interactions that bring anisotropic and long-range forces on, which can lead to self-assembled structures, bosonic and fermionic superfluids that have a supersolid character as well as other novel quantum phases with unique characteristics [8, 55], have been the focus of the Dipolar Quantum Gases project of the 5th Institute of Physics in the University of Stuttgart. After the first BEC production with chromium atoms in 2005, the element dysprosium took its place, since with the value of 10 μ_B, where μ_B is the Bohr magneton, it has the highest magnetic moment of all stable elements [49]. Therefore, with dysprosium atoms, one can obtain a dipole-dipole interaction between particles 100 times stronger than with alkali metal atoms [37]. Further research with this material has led to the discovery of a novel state of matter of droplets embedded in a BEC background, where both crystalline order and frictionless flow arise. This is associated with the breaking of two $U(1)$ symmetries and simultaneous diagonal and off-diagonal long-range order. This new state of matter has been given the name *"dipolar supersolid"* [24]. The construction of a new setup for the second generation of this experiment that will allow to produce larger BECs and thus, study more complex phenomena, started in 2017.

In addition, the new experiment aims at studying extended Bose- and Fermi-Hubbard models that primarily describe strongly correlated bosonic/fermionic gases on a lattice by exploiting the dipole-dipole interactions to establish the correlations [55]. It has been shown that the most convenient way of realizing these models is to use optical lattices resulting from the interference pattern of laser beams [30]. While this creates a periodic "light crystal" that could allow one to exploit analogies from solid-state physics and non-linear optics, optical lattices enable high control over tunable parameters encountered in the Bose-Hubbard Hamiltonian using simpler experimental techniques than in solid-state lattices [30, 55]. Some of this system's degrees of freedom are the dipole-dipole interaction strength and the localization of the wave functions of the atoms, that can be tuned by changing the lattice spacing and beam intensities, respectively [55]. From the extended Bose-Hubbard model follows also the possibility of approximating each lattice site with a micro harmonic trap and the effective PSF of the imaging system with a Gaussian [57]). Theoretical studies of this model for 2D optical lattices and dipolar quantum gases suggested the appearance of novel quantum phases, which awakens a specific interest in these settings [55].

Novel quantum gas microscopy used in this setup will enable high resolution fluorescence imaging of trapped atoms in optical lattices. However, due to the relatively small lattice constant $d = 180\,\mathrm{nm}$ as well

as the instant excitation of the atoms during the imaging process, which can lead to the displacement of some atoms from their original lattice site or hopping between sites, a single site resolution during the imaging process is challenging, yet essential for studying the correlation functions of the many-body system. The diffusion of atoms during the imaging process can be avoided by laser-cooling and pinning to some degree, both of which require a challenging implementation [45] and will unlikely be used in the present experiment.

The underlying lattice occupation in an image has to be reconstructed in a way that the inevitable diffraction and aberration effects given by the point spread function (PSF) of the imaging system, which represents the atomic density distribution, are rewinded, a technique called *deconvolution*. For this, the PSF of the optical system has to be known. Without continuous laser-cooling and pinning, atoms cannot remain localized in their excited states and their diffusive behaviour is reflected in a significant widening of the PSF.

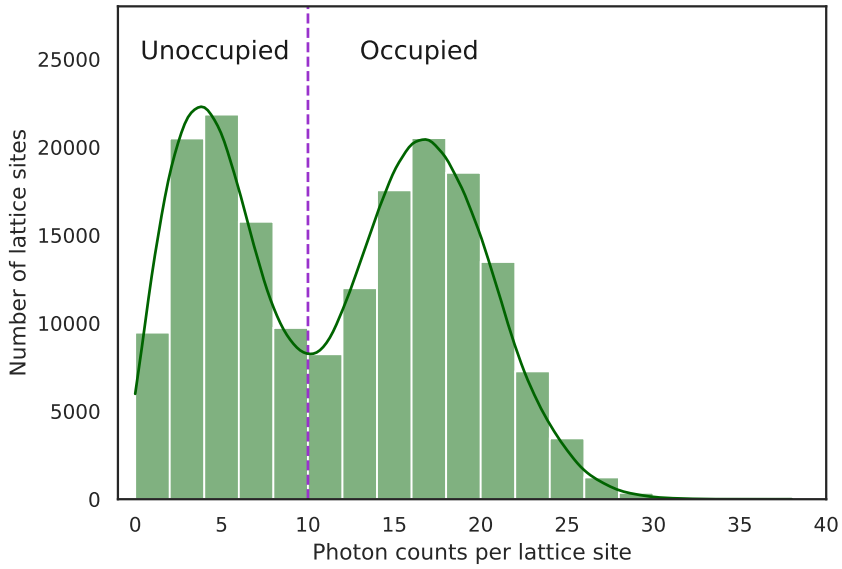


Figure 1.1: Visualization of threshold-based reconstruction. The histogram of photon counts in each lattice site in a set of simulated images that correspond to images that are deconvoluted with a PSF that has a FWHM $\approx 8\,\mathrm{px}$ while the lattice spacing is $8\,\mathrm{px}$ (see Fig. 2.3 for more information). Even though both lengths are comparable to each other, a bimodial distribution emerges, that of which threshold value can be estimated as 10 photon counts per site where the two peaks overlap. This value can then be used to classify the sites as occupied or unoccupied.

Usual intensity threshold based reconstruction (TBR) methods used in ultracold gases experiments follow the steps of (1) determining the lattice grid in a given image by extracting the phase that is connected to the lattice wave vectors by Fourier-transforming the image; (2) deconvoluting each lattice site with the known PSF so that a new brightness value is assigned to each pixel within the site, making the image sharper; (3) determining the photon counts in each site and creating a histogram demonstrating the number of sites (y-axis) that have a certain photon count (x-axis) as shown in Fig. 1.1; (4) specifying a threshold value for differentiating between the occupied and unoccupied sites by taking a value of the photon counts between the two distinct histogram peaks showing the most probable site brightness of an

occupied and unoccupied lattice site, and thus, classifying the sites accordingly [17, 39, 45]. The reliance on obtaining distinct peaks in the bimodial lattice site brightness distribution becomes unfavorable for decreasing lattice spacings, in which e.g. the density distribution of an atom in one site can extend to an empty neighboring site in a way that the latter gets labeled as occupied due to its high photon counts. Another misleading situation may arise from the overlaps of neighboring atom distributions that partly cover an empty lattice site nearby, which may again cause a false classification of the empty lattice site due to the increased intensity in that area. Moreover, low signal-to-noise ratios can also prevent a clear separation of the two peaks. Besides these, stronger excitations of atoms through longer imaging time, usage of more scattering photons or absence of cooling and pinning enhance the PSF and the overlaps of neighbouring atom densities. These scenarios pose an obstacle to high efficiency and classification fidelity of the described reconstruction methods [45].

To improve the classification fidelity in these limiting cases, in this thesis, alternative machine learning models will be investigated that aim to recognize more complex patterns in binarized noisy images with Gaussian atomic density distributions (PSF) shaped by a Poisson-distributed number of photoelectrons of the EMCCD camera used in the imaging, by using neural networks. Motivated by the work from [45], different types of neural networks from simple to more complicated are trained and tested on images that are simulated by using realistic parameters corresponding to the PSF of the imaging system and the average photon counts of the CCD camera in absence of continious cooling. A special focus is placed on convolutional neural networks (CNN) that proved to be the best fit for image classification and computer vision problems in supervised learning.

After clarifying the details about the imaging and the image simulation in section 1, the basics of machine learning are introduced in the light of the binary image classification task in section 2. The theoretical discussion of deep learning algorithms as well as their advantages and shortcomings leads to section 3 that is devoted to the implementation of these algorithms and discussion of the results, especially the test accuracy, for different physical parameters as well as hyperparameters of the neural networks. Following this, a final outlook, a summary and an appendix are included in the end.

2 Image Simulation

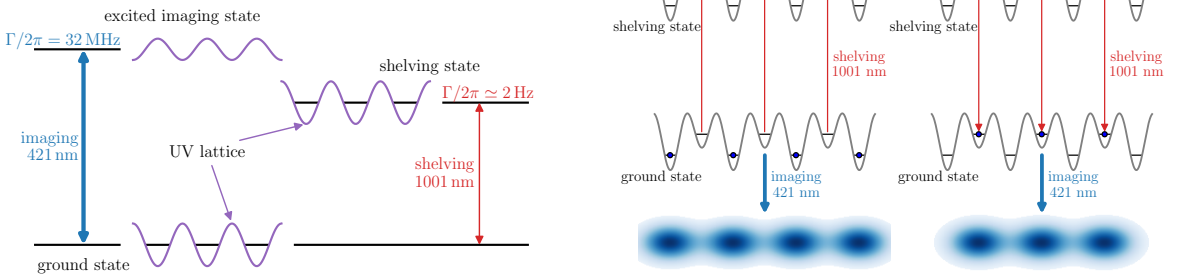
In supervised learning, neural networks are trained by feeding them a sufficiently large set of data called *training set* that familiarizes the network with the underlying patterns and traits in the given examples. Training examples are labelled, i.e. there is an expected outcome prediction for each example that is taught the neural network during the training. This way, a trained network becomes capable of making successful predictions on previously unseen data.

Before a trained model is applied to data, for which correct labels are unknown, it is tested on a separate data set called *test set* to obtain the classification accuracy of the model. This process will be discussed in more detail in section 3.

For the image classification task in our problem that focuses on the prediction of lattice site occupation, collecting training and test examples from the experiment is not possible, since the experiment is still under construction. Moreover, there is no similar experiment from which real images compatible with our system can be obtained. Besides, images that are not already labelled cannot be used in supervised learning, which is not the case for the real images. Therefore, simulated images are required. These simulations should imitate the real images by implementing a realistic noise model and fluctuations around the known mean PSF of the imaging system.

Using large images containing a great number of configurations of many atoms (for 20 atoms with a given lattice spacing the number of different configurations is 2^{20}) proves to be disadvantageous due to the fact that each unique configuration should appear multiple times in the training so that it can truly be learned, which introduces a serious storage problem. A more convenient way is to scan the whole image by using a smaller frame and focusing on its center that is either occupied by an atom or not. Thus, the learning task is simplified to the problem of detecting an atom in the center of a given partial image. By applying a chosen model to the set of small images of the same size that represent elementary regions of the main image, one can gain information about the occupation of each lattice site that is seen in the main image. It is reasonable to include the eight nearest neighbouring sites of the central site in the selected frame, since this gives the model the chance to learn, to which extent neighbour atom distributions may overlap for a given figure of merit (FOM), i.e. the ratio between the lattice spacing d_L and the width of the effective PSF of the imaging system σ. Therefore, simulated images for almost all experimental steps in this thesis are designed to contain nine lattice sites that correspond to imaginary segments. The effective PSF is calculated by convoling the PSF of the imaging system (without diffusion) and the PSF due to atomic diffusion along the spatial directions of the lattice (without optical aberration or refractive limit).

The overlap becomes larger when FOM approaches one. In the experiment, this might happen if, during the imaging, the lattice is not continuously cooled or atoms are not pinned ($\sigma \uparrow$) or the lattice spacing is deliberately reduced to increase the first-order correlations of atoms ($d_L \downarrow$). It thus becomes crucial for the model to show a good performance for smaller lattice spacing relative to the width of the effective PSF. For a broader description of the optical parameters that are relevant in the image simulation, it is necessary to have an overview of the imaging protocol in the experiment.



(a) Schematic demonstrating the idea of UV lattice and shelving state

(b) Schematic of shelving imaging protocol using a superlattice

Figure 2.1: (a) Atoms in the ground and the excited shelving state are trapped in an UV lattice at 1001 nm. The strong transition at 421 nm is used to image the atoms. (b) A shelving laser transfers every second atom in the lattice to a long-lived sheving state. After the atoms in the ground state are imaged, the ones stored in the shelving state are transferred back and imaged as well. Both images in (b) are used to reconstruct the initial occupation of the lattice. [53]

2.1 Imaging Technique

In order to enhance the magnetic dipole interaction strength between lattice sites, Dy atoms are loaded in a UV optical lattice with a periodicity of $d = 180\,\text{nm}$. However, this length is smaller than the Abbe diffraction limit $\frac{\lambda}{2} \approx 290\,\text{nm}$, where 421 nm is used for imaging as it drives a strong ground-state transition in Dysprosium [53]. This allows many photons to be scattered from each atom in a short amount of time, which is advantageous for fluorescence imaging. In order to reach single-site resolution below the diffraction limit, a principle called *shelving* can be exploited. The idea of shelving, as shown in Fig. 2.1, is to excite the atoms in selected lattice sites to a state that has a longer lifetime as compared to the imaging time; to image the atoms in the ground state and transfer the atoms in the excited state back to the ground state and image them as well. By combining the individual images, the complete lattice occupation can be reconstructed. The optical transition that is chosen for the shelving in the experiment is a narrow transition near 1001 nm with a lifetime $\tau > 87\,\text{ms}$ [44]. The atoms to be transferred to the shelving state are picked by using a superlattice that has twice the actual spacing, i.e. atoms on every second lattice site will be transferred. More details on this protocol can be found in [53].

The PSF including not only optical effects, but also diffusion of atoms due to the absence of continuous cooling and pinning is predicted to be well approximated by a Gaussian distribution with a standard deviation of $\sigma \approx 300\,\text{nm}$. Thus, the FOM for our experiment is ≈ 1.2. This motivates the study of lattice site occupation classification fidelity in an intermediate region regime FOM = 1. Because we focus on not continuously cooled lattices that facilitate atamic diffusion, hence a broadening in the effective PSF, the FOM values below 1 are not studied in the following.

2.2 Justification of Some Assumptions in the Simulation

The camera that is likely to be chosen for the detection of fluorescence photons is an electron multiplying CCD camera (EMCCD), which is particularly preferred for its additional built-in amplifying or gain register that enables the sensitive detection of very weak signals with a rapid read-out [27, 34].

In ultracold atom experiments, in which fluorescence photons are to be detected to image the present system, photon collection rate hardly exceeds 10% due to the finite NA, while some of the remaining light leads to a background noise [9,17,26,45].In order to prevent an additive technical noise (read noise) arising from the amplification stages of the camera as well as to detect fluorescence photons with a high fidelity, the photon counting mode of the EMCCD is favored. In this mode, single photons per pixel are registered with a limited intensity information, so that multiple photon incidents on a pixel are not fully distinguishable from a single photon event, which can be understood as a quasi-binarization of the pixels as bright or dark according to a threshold. This supresses the read noise of the camera without a big compromise in the detection fidelity as well as information about the lattice site occupation [9]. Based on this, the images generated in this thesis are binarized.

Another common feature of such lattice experiments is that the image of the system fails to display the initial occupation number of a given lattice site. Instead, one obtains a binarized detection output, which is given by $n_0 \bmod 2 \in \{0, 1\}$, where n_0 is the original occupation number of a site. The reason is mostly inelastic light-assisted collisions atom pairs in a single site undergo when exposed to excitation light, which energetically suffices for the atoms to leave the potential well and get lost before the exposure is even over [16]. Although this behaviour is not necessarily expected within the imaging scheme of our experiment, binarized lattice site occupation is adopted in the simulation.

2.3 Simulation

Given the explanations in the previous sections and predictions for some parameters based on past experiences, the main parameters used in simulations are the average number of fluorescence photons detected per atom $n = 25$, percentage of noise pixels in the whole image $p = 2\%$, standard deviation of the PSF $\sigma_s = 3,3$ px and the variable lattice spacing $d \in \{4, 5, 6, 7, 8\}$ px. Thus, FOM $\in \{1.21, 1.52, 1.82, 2.12, 2.42\}$ are realized and studied. The segment size is adapted to the examined lattice spacing such that the 2D-Gaussian density distributions are fully contained in the image. Photon statistics suggest a Poisson-distributed number of scattered photons that impinge on the camera sensor. Therefore, n corresponds to the expected value of photon counts. For each atom that is represented in the image, a total photon number is chosen randomly from the described distribution. The positions of these photons are then selected randomly from a normal distribution with the standard deviation σ_s. After applying the noise on an empty image and placing the specified photons, the image is binarized, resized if necessary, and ready for use. The electrons that come as camera noise have no width in the image and they are marked as single bright pixels. Examples of an isolated atom in the central lattice site simulated according to the protocol described above are demonstrated in Fig. 2.2.

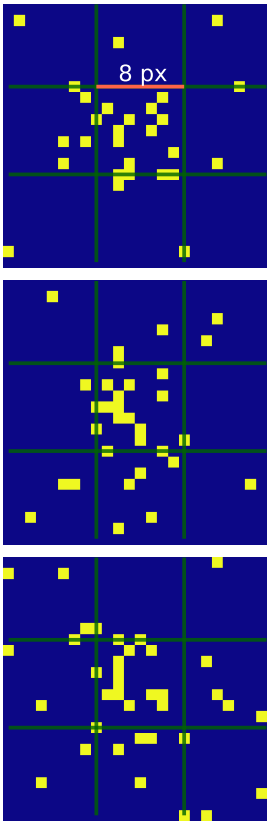
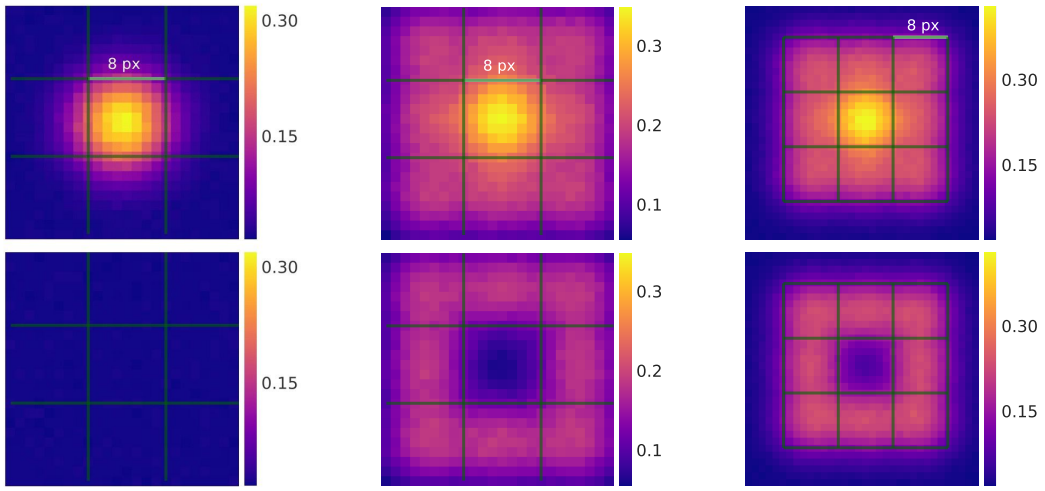


Figure 2.2: Three arbitrary samples from binarized simulated images of the density distribution of an isolated atom in the central lattice site with background noise. The images are divided into 3x3 segments that represent the 9 adjacent lattice sites of size 8 pixels ($d = 8$ px). Even though only the central lattice is occupied, the distribution of the photoelectrons as well as noise pixels partly fill some of the neighboring sites. Other parameters are elaborated in the text.

As it will be discussed in the evaluation chapter 4, the first step in the simulation is to create an image of an isolated atom in a chosen frame (here 24x24) and let the simple neural network model learn the PSF of this atom without an overlap of neighbouring density distributions. An average over 10000 such images with $d = 8$ px is shown in Fig. 2.3a. The lower image that consists of noise displays the case where the central lattice site is unoccupied.

In the next step, the nearest eight adjacent lattice sites are introduced with a 50% probability of an atom occupying a specific site, since the occupation of one site usually makes it less likely for the nearest neighboring sites to be occupied as well. To cover all 2^9 configurations and to show the neural network the effect of each configuration multiple times, a sufficiently large training set is created. The final image is resized such that only the central lattice site and its near surrounding is viewed, which does not necessarily fully contain neighboring distributions. The model learns the PSF again - this time under the effect of overlaps with neighbouring atoms. Keeping the latter partly out of the frame helps one obtain a PSF of an individual atom again, rather than fitting the whole configuration. This way, the difference between the PSF of an isolated atom and

that of an atom surrounded by its neighbours can be easily examined. This simulation step is also shown in Fig. 2.3b by means of two averaged images, the center of which is occupied or unoccupied by an atom.



(a) **Simulation step 1:** (above) isolated atom at the center, (below) only background noise.

(b) **Simulation step 2:** nearest 8 neighbouring lattice sites that are occupied by an atom with a probability of 1/2, around an (above) occupied, (below) unoccupied central lattice site.

(c) **Simulation step 3:** Simulation step 2 repeated with a larger frame that comprises all density distributions fully.

Figure 2.3: Averaged images over 10000 simulations of a lattice region with a lattice spacing of 8 px at three stages of the study. For examples of individual images used in the first step, see Fig. 2.2. The superimposed green lines indicate the lattice site boundaries. When a lattice site is occupied by an atom, the center of the atom distribution density and the center of the lattice site are assumed to be the same point. Thus, a potential atom lies at the center of one of the nine square segments. The colorbar ranges are set by the brightest and the dimmest pixel found in the upper images that contain an atom in the central lattice site while the lower images do not.

At the last stage, the goal of determining the PSF of an individual atom is enhanced to the more inclusive goal of classification of the central lattice site as part of a bigger configuration, i.e. the last model is expected to learn how to fit the given nine-neighbouring-sites system, or 3×3-segments system, to produce a label $y \in \{0, 1\}$ indicating an occupied or unoccupied central lattice site, respectively. Therefore, the segments containing all nine lattice sites and their near surroundings are included in the frame and the model works toward accomplishing the task of recognizing and accurately predicting the occupation of the central lattice cite, see Fig. 2.3c.

Simulation data is always grouped into two classes: central cite is empty $\equiv (y = 0)$ or central cite is occupied $\equiv (y = 1)$. The $(y = 0)$-set contains images with an unoccupied center. The visualised structure of the whole data set as well as the image storing technique can be found in Tab. 2.1. An average of the images produced at each stage described above can be found in Fig. 2.3.

Next chapter will be dedicated to an introduction to machine learning with a special focus on super-

Table 2.1: A view of the data structure used for storing training and testing examples. Images are stored in form of square matrices A_i with elements $(A_i)_{jk} \in \{0, 1\}$ for dark and bright pixels, respectively. The second column contains the images represented by A_i^1 that have an occupied central lattice site, while the images in the third column do not. Out of N examples carrying the label $y = 0$, the last one third, $\tilde{A}_i^0$, has only noise, while the remaining two third, A_i^0, may involve bright pixels from neighbouring atom density distributions as well. The columns are accessed at the pre-processing stage, i.e. before being fed into a network, to be combined with the class they belong to, as in $(X, y) =$ (Image, class), and shuffled to enhance the representativity of a subset of the entire set.

Examples	$y = 1$	$y = 0$
1	A_1^1	A_1^0
2	A_2^1	A_2^0
.	.	.
.	.	.
.	.	.
$\frac{2N}{3}$	$A^1_{\frac{2N}{3}}$	$A^0_{\frac{2N}{3}}$
$\frac{2N}{3}+1$	$A^1_{\frac{2N}{3}+1}$	$\tilde{A}_1^0$
.	.	.
.	.	.
.	.	.
N	A_N^1	$\tilde{A}_N^0$

vised learning with artificial and convolutional neural networks ranging from very simple architectures to multi-layered deep learning models.

3 Introduction to Machine Learning

Machine learning (ML) has become increasingly prominent in a wide variety of fields and has started playing an important supporting role in physical data processing in recent years [12]. When implemented correctly in the right place, it has a capacity to sort through large amounts of data, recognize patterns, fit and classify data and accomplish other tasks in less time than any human or computer based analysis methods with little automation are capable of, making it a rather promising tool [32].

This chapter is mostly based on Andrew Ng's lectures in The Machine Learning Specialization course [14] and the text book [6] where the reader is referred to more details. The term *machine learning* stands for a broad set of approaches to provide a machine, i.e. computer program, with artificial intelligence. One uses the term *intelligence* because the subject is not an explicitly programmed machine, but a machine that is presented with a well-defined learning problem it should master after completing its *training*, which equips it with required experience with respect to the given task. Measuring the performance of this machine on the given task shows the efficiency or success of the learning process. The prerequisite for concluding that a machine is learning is that its performance improves with experience over a certain period of time. The matter of not learning can have many reasons, such as unclear definition of the learning problem, incompatibility of the problem and the former experiences, a narrow frame of experiences for a comprehensive problem, under- or overrepresentation of some experiences etc. Identifying the reason that is blocking the learning and implementing ways to facilitate the learning process is the main task of a programmer working with ML. This can be better understood with the following example.

Three young children attend an experiment where they are prevented from communicating with anyone else other than one person who separately gives them a mandarin for them to examine and taste it. The first child is asked the question "Is this a big or a small mandarin?" while the other two children should answer the question "Is this an orange?".

Figure 3.1: A group of mandarins and oranges [1]

The first child's answer stating that it is a small mandarin is wrong according to the person asking the questions, since her or his reference mandarin is smaller than the one that was given to the child. The reason for his false prediction is merely the lack of specific information, which affects the answer, he should have been given.

The other children also give wrong answers. A little query about their encounters with mandarins and oranges shows that the second child has eaten neither a mandarin nor an orange to that day, therefore she cannot distinguish between the two. The decision she makes is unfounded and with a 50% chance, wrong. Incompatibility of the posed question and the experience this child had with these fruits leads to a false prediction. The last child has observed and eaten both fruits before. Yet the oranges he has seen

so far always had comparable sizes and textures to the mandarins he encountered. Moreover, they had a rather similar taste. The problem with this child's situation is that the type of mandarins and oranges that are similar in both aspects are overrepresented in his range of experiences while the other types that indicate clearer differences between both fruits are underrepresented. In order for the children to be able to give correct answers in the future, they should experience different types of both fruits in more depth by observing their different features like the smell, texture, taste, size etc.

Among various approaches of ML, the three broadest categories are supervised learning, unsupervised learning and reinforcement learning. In supervised learning, the correct outputs are already assigned in the training data, leaving the machine only the task of learning how to map the inputs and outputs, so that after training it can apply the learned rule to data, of which outputs are unknown, to predict those, i.e. the learning takes place under human supervision. In unsupervised learning, training is designed as a process in which the program strives to find similarities, i.e. correlations, in large unlabeled data sets which can be regarded as exploratory data analysis. Thus, it ends up organizing it in non-overlapping clusters. Reinforcement learning is based on an agent learning the most suitable behavior in the face of a problem through trial-and-error interactions with a dynamic environment to maximize the results that bring feedback analogous to rewards [31].

This thesis is devoted to testing learning algorithms within supervised learning due to the straightforward matching between our binary classification task with already known labels and the method of supervised learning. Generating a very accurately labelled training data is possible because the behaviour of the full imaging process including atomic diffusion, light collection and camera response is known well enough for a faithful simulation.

Nonetheless, unsupervised learning can be an interesting approach to test, since its lack of direction may allow it to check patterns that have not been previously considered [52], but also because a single atom in our experiment can never be prepared with 100% fidelity, i.e. obtaining absolutely accurate labels is impossible. An encouraging example of unsupervised learning applications in the physics world is diagnosing quantum states of matter studied in condensed matter physics without knowledge of the Hamiltonian among many other applications [7, 22, 46, 58].

A task in which some machine learning algorithms are particularly promising is classification of images by a set of specific features they share. The simplest case of this is binary classification, where the images are put into two separate classes. In fact, classification problems are strongly connected with the task of fitting data. A very simple example of this is fitting a suitable curve to some experimental data by uncovering a mathematical relation between e.g. the variable x and the measured, or derived from measurement, physical quantity $y(x)$. Image classification is, then, carried out by fitting a given image which corresponds to uncovering evident as well as hidden relations between pixel locations and values as well as to recognizing both global and local features in the image.

The mathematical background of how the above-mentioned tasks are tackled primarily involves the adoption of a paradigm that enables to specify, observe, evaluate and optimize the learning process and its aftermath. In the following subchapters, some of the most common elements of this paradigm are

introduced.

3.1 Gradient Descent

Learning algorithms can be considered optimization models, i.e. a framework to approach computational problems in which the task is to find the best of all possible solutions. This task can further be expressed by a minimization problem, where the object that is to be minimized is called *cost function* which is a measure of predicted output's deviation from the desired output. Given a set of m training examples $(x^{(i)}, y^{(i)})_{i\in\{1,...,m\}}$, the best function for mapping from input $x^{(i)}$ to output $y^{(i)}$ is called *hypothesis* h, so that $h(x^{(i)}) \approx y^{(i)}$. An easily comprehendible form of hypothesis is

$$h_w(x) = w_0 + w_1 x + w_2 x^2 + ... \tag{1}$$

where the *weights* w_i represent the fit parameters that are to be optimized. Depending on the features given in the input $x = (x_0, x_1, ..., x_n)^T$ and how they relate to the problem, hypothesis can take more complicated forms that involve product terms of given features.

An evident choice for the cost function is then

$$J(w) = \frac{1}{2m} \sum_{i=1}^{m} (h_w(x^{(i)}) - y^{(i)})^2 \tag{2}$$

i.e. the common squared error function with a prefactor $1/2m$, since the summation terms in a non-squared error function could cancel each other out. In many cases, finding the weights for which the minimum value of the cost function is obtained is analytically not possible. Therefore one appeals to iterative methods, the most commonly being that of *gradient descent* (**cite Alpaydin**). The idea of gradient descent is to start with some parameters w_i and keep changing them to reduce $J(w)$ until it converges to a minimum by choosing the opposite direction of the partial derivative $\partial_{w_i} J(w)$ of the cost function with respect to each parameter as the direction of change. The cost function demonstrates a surface in a space spanned by the parameters. For a linear hypothesis with only two parameters w_0, w_1 following the slope of the surface in descending direction takes one downhill to the lowest point of the valley. The step size in each iteration to reach the minimum is tuned by the *learning rate* η. The algorithm reads as shown in 1.

The number of iterations should be chosen such that the cost function converges. This parameter is related to the ML term *epoch* that denotes one complete pass over all the examples in the training set. The number of training examples used per iteration is called *batch size*. If the whole training set is used in each iteration to update the weights, i.e. single update per iteration, the algorithm is called batch gradient descent, while using a smaller number of samples per iteration, the so-called mini-batch gradient descent, enables m/b updates per iteration, where m is the number of total samples and b the batch size. Just as choosing the right number of epochs, it is important to use a batch size that catalyzes the convergence. Batch gradient descent takes too long per iteration if the training set is not small

Algorithm 1: Gradient descent algorithm for simple linear regression

Input : Data set $\{(x^{(i)}, y^{(i)})\}_{i\in\{1,...,m\}}$, initial parameters w_i^0, learning rate η, number of iterations k

Output: Values of w_i that minimize the cost function

```
w_i = w_i^0;
J(w) = J(w_i^0)
repeat k times
    gradient_i = ∂_{w_i} J(w);
    w_i = w_i − η · gradient_i;
    J(w) = J(w_i);
end
```

($m \leq 10000$) and bears a risk that the model generalizes more than desired (**cite Keskar**) alongside of occupying an immense portion of the CPU/GPU memory. It can be, however, more stable than mini-batch gradient descent that uses a very small batch size in terms of reaching the minimum without big distractions. In the latter case, the falling trend of the cost function per iteration can become very noisy, meaning that gradient descent follows a random-walk-like course when approaching the minimum due to frequent updates according to a few training samples. This can also lead to a compromise in both convergence- and computational speed, since greater number of samples in a mini-batch allows to benefit more from vectorization, i.e. turning for loops in matrix multiplications into vector operations, which boosts the computational speed as well as makes debugging easier. The common mini-batch sizes that are tested and seen to perform well range between 50 and 256 [48]. However, finding the middle ground is essentially left to trial and error.

From this point on, the focus of a programmer is to make the chosen gradient descent algorithm more efficient, i.e. to let it converge faster by pre-processing the training set, e.g. by scaling features and normalizing their mean, searching for the best initial value for the learning rate and weights, and employing more complex optimization methods introducing memory, momentum, adaptive learning rates etc. which are discussed in more detail in [48].

For classification problems linear hypothesis (linear regression) is not a good choice to map the inputs to the outputs. The threshold that must be introduced to group different cases in the problem has to shift when new examples are added, creating a risk that earlier correctly predicted outcomes become false or the new examples are predicted incorrectly because the shift in the threshold was not sufficient to adapt to the change in the data set. For a binary classification problem with the class labels "$y = 1$" and "$y = 0$" where y is the output, this issue is demonstrated in Fig. 3.2.

In order to solve this problem, one uses the *logistic* (or *sigmoid*) function

$$h_w(x) = \frac{1}{1 + e^{-w\cdot x}} \tag{3}$$

with $w = (w_0, w_1, ..., w_n)^T$ instead of the polynomial form in (1).

A very useful aspect of this choice is that sigmoid function is confined within the range [0,1] as shown in Fig. 3.3a, allowing it to be interpreted as a probability, i.e. the probability with which the condition

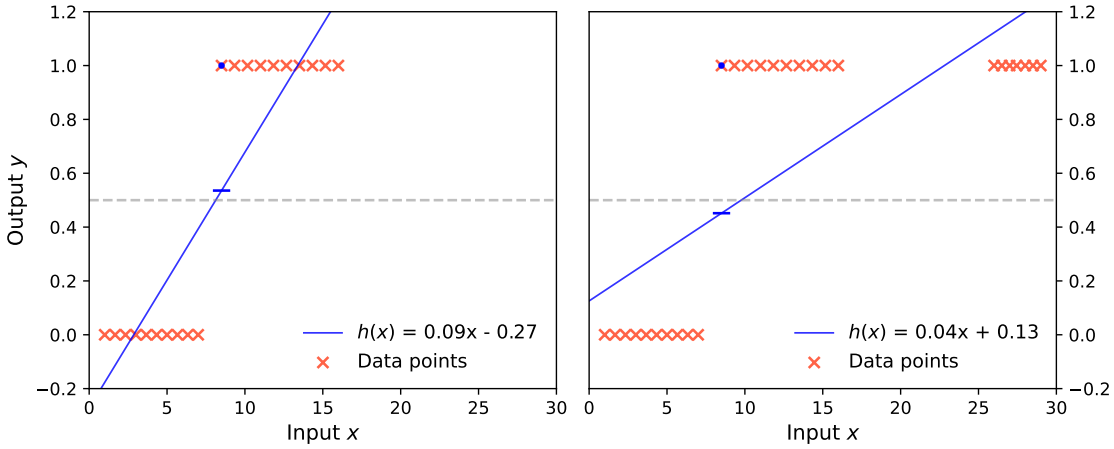
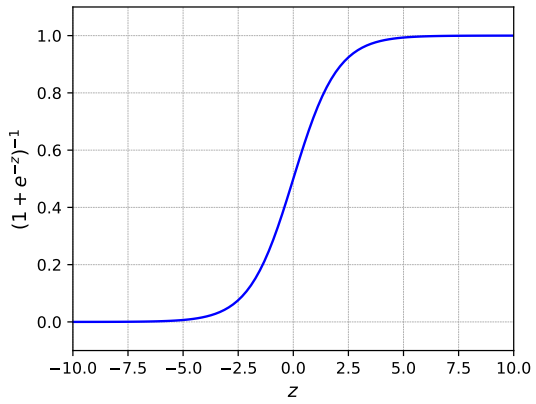


Figure 3.2: Binary classification of data points marked with orange color using linear regression with the hypothesis $h(x)$ and a threshold value of $y_{th} = 0,5$ according to the rule: label the inputs x for which $h(x) \leq y_{th}$ is fulfilled with "$y = 0$"; label the inputs x for which $h(x) \geq y_{th}$ is fulfilled with "$y = 1$". While all data points on the left figure are labelled correctly following the given rule, tracing the value of the hypothesis for the data point marked with a blue dot in both figures shows that after new data points are added, it is incorrectly given the label "$y = 0$". The threshold must be adapted to the change in the slope of $h(x)$ so that the model can keep making correct predictions.

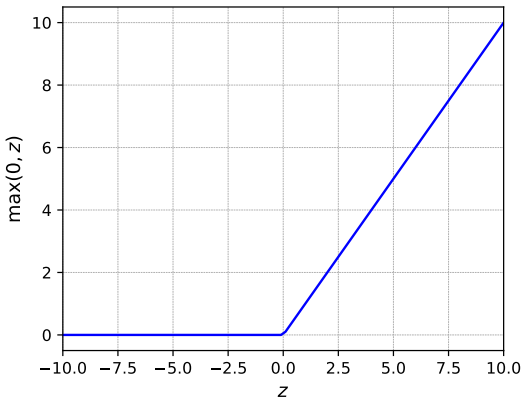
dividing the data into groups is fulfilled for an observed data point. For instance, an output $h_w(x) = 0,7$ in the binary classification problem of site occupations in our experiment means that the examined lattice site is with a 70% probability occupied by an atom. In order to label the lattice sites, this outcome is rounded to 1, just as an outcome of $0,3$ would be rounded to 0.

Even though gradient descent is an efficient optimization algorithm, it carries some inherent risks about the convergence of the cost function. One of the very common problems is *vanishing gradient*, which literally denotes the case where the derivatives of the cost function become too small, causing the weight updates to be ineffective and thus, weights get stuck in a region without reaching the sought minimum. Those areas are likely the local minima of the cost function, which does not necessarily have a simple convex form. Among plenty of solutions to this problem, a very simple one is to replace the hypothesis (later referred to as activation function) with another function that does not cause such small derivatives, like rectified linear units (ReLu) function shown in 3.3b. The opposite case of vanishing gradient is *exploding gradient* where the solutions to the first can be employed backwards.

The choice of the learning rate plays a crucial role as well. If it is too small, it may take significantly longer to reach the global minimum and in case the weights are around a local minimum, harder for them to leave it. A learning rate that is too large may cause huge weight updates, leading to large steps from one point to another, missing the global minimum. Finding the optimum value in between enables the weights to avoid divergence and getting stuck at a local minimum and allows them to converge faster.



(a) Sigmoid function. It maps to the interval (0,1) and is continually differentiable.



(b) ReLU function. It maps negative arguments to zero. Its derivative for $z < 0$ is zero, for $z > 0$ 1.

Figure 3.3: Two functions that can be used as hypothesis $h_w(x)$ depending on the requirements in the training with regard to the scale of the output, computational speed etc.

3.1.1 Bias-Variance Tradeoff

In a machine learning model there is a second but not secondary goal besides minimizing the error by letting a defined cost function converge to its global minimum: keeping the model parameters general enough to fit data beyond their training set sufficiently well, i.e. preventing them from becoming too specific, describing their training set too well. This is expressed by the interplay between two properties of a model called *bias* and *variance*. The tendency of a model to make assumptions about the relation between the features and the target outputs and thus, generalizing those relations is described by the bias of the model, while the variance of a model denotes its tendency to react to small fluctuations in the training set sensitively, which might even lead to fitting the noise in the data. High bias leads to too much a generalization, expressed by the term $underfitting$. High variance, on the other hand, causes an unwanted degree of alignment between the target outputs and predictions, expressed by the term $overfitting$. The desired behaviour of a model lies between both cases, where the model is able to capture all relevant underlying relations among the variables by neither limiting itself to the training data nor generalizing those relations too much. Improving one property inevitably leads to a compromise in the other, which is why there can be no model that performs perfectly both on the training set and on previously unseen data. Finding a middle ground is possible by using methods as regularization and feature number reduction [6]. Another important and useful way is to use a separate validation set in the training. After the processing of samples in each epoch, the model with the updated parameters is evaluated on the validation set. If one sees an overall rise in the training accuracy after several epochs despite the lack of improvement or even the beginning of a decline in the validation accuracy, it can be concluded that the model started overfitting the training set. The latest acceptable moment to stop the training to prevent overfitting and protect the weights that are able to adapt to unseen data is the point where the validation accuracy starts to decline on the average. As it will be discussed in the evaluation

chapter, there are so-called *callback functions* that can be used to automatically stop the training when a given condition as described above is fulfilled.

3.2 Artificial Neural Networks

This chapter is inspired by and based on [6, 28]. Artificial neural networks (ANN) are machine learning models that take their inspiration from the brain. With a focus on utilizing the known functionalities of the biological nervous system to solve complex problems, ANNs demonstrate abstracted models of these systems without trying to replicate them. Some of the most notable information processing characteristics of the nervous system are learning and adaptivity, its capability to generalize, its ability to handle imprecise information, fault and failure tolerance, high parallel processing, nonlinearity and robustness.

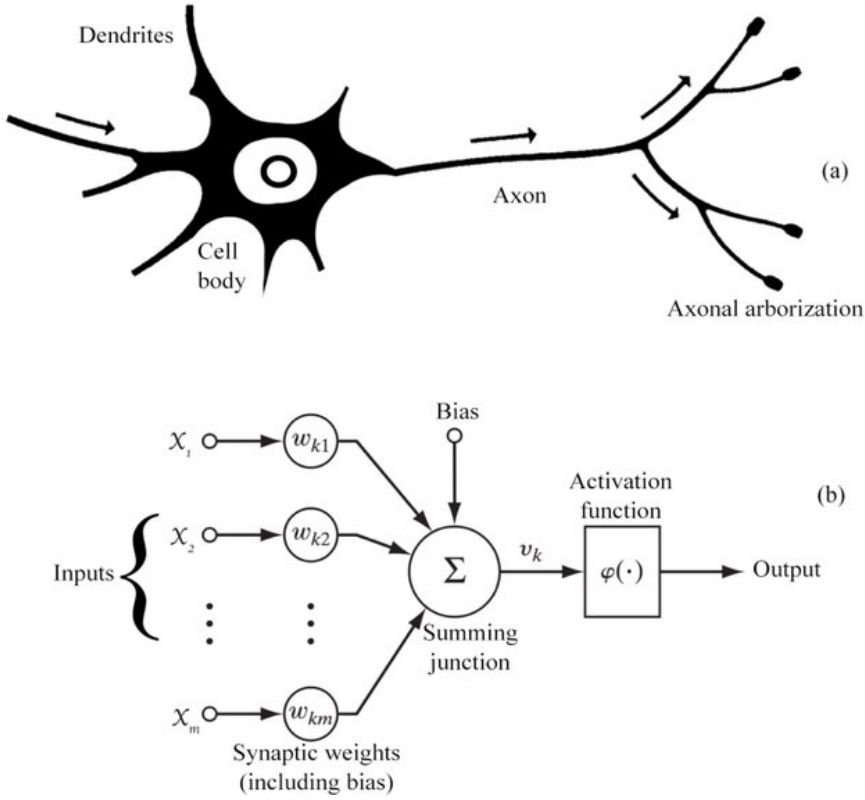


Figure 3.4: Animal brain cell (biological neuron) vs. artificial neuron and their parts are shown. The multiple extensions from the cell body of the brain cell are called *dendrites*. They receive signals from other neurons by means of their connection called *synapse*. Another extension is the *axon* that transmits a signal from the cell body to a synapse. Synapses occur between the axon of one cell and a dendrite of another cell. The axonal arborization allows the neuron to be connected with multiple targets. The artificial neuron receives its input signals weighted with a factor w_{kj} that is associated with each of them. k denotes the index of the neuron and j the index of the input signal. Weighted signals are summed and added a bias weight in the summing junction that corresponds to the cell body. An activation function φ is applied to the output v_k. Activation function can act as a threshold that decides which output signals are let through and which are inhibited. Both biological network and ANN learn by gradually adjusting the synapses' strengths or the magnitudes of the weights w_{kj}. [5]

The real-world problems are usually complex and do not allow a linear fit to the data without oversimplification. Nonlinearity in a model, on the other hand, assures a deeper learning and better fit for such problems. Processor units performing their computation parallelly, i.e. without waiting for the previous one to finish its task, enable a much faster processing. Good handling of uncertainty provides accurate results in the presence of error and noise in the data. By means of generalization, the model becomes

applicable on unseen data. Learning and adaptivity enable an unlimited modification of the model in response to changing environment and accumulating experience. Due to all these and further advantages, artificial neural networks are desirable models that keep being studied and developed with new insights in brain research as well as innovative ideas in engineering.

The parallel processing units of the brain are called *neurons*. One of the sources of brain's computational power is believed to be the connectivity of its neurons. The connection between two neurons is called *synapse*. The parts of a biological- and an artificial neuron are illustrated in Fig. 3.4. The analogy between the two structures and their function is evident.

There are numerous types of neural networks specific to some tasks or constraints. Two of them are, e.g. *convolutional neural networks*, denoted by CNN and *recurrent neural networks*, denoted by RNN. A chapter dedicated to CNNs can be found in the following pages, whereas RNNs are irrelevant to this study and therefore are not introduced in detail. In short, the outputs of the units in a recurrent network have not only connections with the next target unit like in *feedforward ANNs*, but also with the previous units or with itself, which act as a short-term memory [6]. To avoid confusion and maintain the consistency throughout this thesis, feedforward neural networks that have the most basic architecture and no additional features like convolution layers or recurrency, are denoted by ANN. Besides, the words neuron, node and unit are used interchangeably.

Before implementing large networks that consist of multiple neurons, it is important to understand how a single neuron functions.

3.2.1 The Perceptron

The simplest neural network is the *perceptron*, which is the basic processing unit of an ANN. Without an application of a threshold or other kinds of activation functions like φ shown in Fig. 3.4, the output of a perceptron is given by

$$y = \sum_{j=1}^{n} w_j x_j + w_0 = \mathbf{w}^T \mathbf{x} \tag{4}$$

where n is the number of the inputs x_j without a bias term $x_0 = 1$ that can be made the first component of the enhanced input vector $\mathbf{x}$ to enable writing the sum y as a matrix multiplication of the vector $\mathbf{w}$ containing the synaptic weights w_j, where w_0 is the bias, and $\mathbf{x}$. In this form, the perceptron with the output (4) defines a hyperplane that can divide the input space into two. Thus, it can work as a binary classifier when ascribed a threshold.

If k perceptrons with outputs y_i are employed in one layer, i.e. all of them are connected with the same input units, each of them learn a specific weight for their connections. When w_{ij} is the weight from input x_j to output y_i, the output y_i is given by

$$y_i = \sum_{j=1}^{n} w_{ij} x_j + w_{i0} = \mathbf{w}_i^T \mathbf{x} \tag{5}$$

such that the vector $\mathbf{y}$ containing all k outputs y_i becomes $\mathbf{y} = \mathbf{W}\mathbf{x}$. The weight matrix $\mathbf{W}$ has the

dimension $k \times (n+1)$. The created network is trained with gradient descent algorithm.

The things that can be done with perceptrons in a single layer are quite limited, since such a network can only approximate linear functions of the input unless it is given auxilary inputs that enable polynomial interpolation as well [6]. For nonlinearity without such a preprogramming, *hidden layers*, i.e. additional layers between the input and output layer, should be implemented.

An example of linearly separable inputs are notebooks whose length is greater than their width and vice versa. When the both features are the two coordinate axes, both classes can be separated by a straight line. However, the images of a lattice region that has or does not have an occupied center, like in the case this thesis is dedicated to, a straight line cannot be used to classify the lattice site occupation.

Nevertheless, by applying activation functions like the sigmoid and ReLu on the ouputs of the units, nonlinearity can and should be introduced in a network. Otherwise, even the hidden layers cannot contribute to the complexity of the solution that stems from nonlinearity, because then, all they achieve is to build linear combinations of linear combinations, which is another linear combination [6].

The most widely used activation function is the sigmoid function, which is plotted in Fig. 3.3a. It is a smooth, continually differentiable function that maps its argument to a value between 0 and 1. Because its derivative is also in this range, the gradients used for the updates can become too small, which can decelerate learning, if it is used in all hidden layers. In the output layer, however, it allows to interpret the result as probability, which is a quite useful feature. ReLu is shown in Fig. 3.3b. Because its value is zero for negative arguments, using ReLu leads some units to be activated while the others produce an output of zero. This accelerates the computation, while also causing some weights not to be updated, a phenomenon called “dead neurons”. ReLu is favored especially in hidden layers [54].

3.2.2 Multilayer Perceptron

Multilayer perceptrons (MLP) allow to classify objects that are not linearly separable by learning the nonlinear functions of the inputs that impact the result, and thus, the prediction. In such networks, there is a forward propagation of the information, such that the outputs at one layer become the inputs of the next layer until the last layer is reached. The weights are only updated when the output layer produced a prediction that can be compared to the true answer, after which the cost function, gradients and the updates are computed. However, an algorithm is required to trace from the output errors to the weights in the hidden layers associated with these errors and to update these accordingly. Thus, the errors should propagate from the output layer to the back to the inputs. Therefore, this algorithm, which has various versions, is called *backpropagation algorithm*. Machine learning tools (TensorFlow, Keras) that are used for this thesis already contain this algorithm and perform backpropagation automatically, which is why it is not further studied and elaborated here. For its nontrivial derivation and explanation, see [61].

Choices regarding the network architecture as well as training parameters are crucial. For instance, for too large a number of hidden neurons in a hidden layer can hinder the desired generalization by overfitting. For too small a number of hidden neurons, the underlying patterns embedded in the data might not be obtained. This and other extreme cases for different parameters are listed in Fig. 3.5.

Design parameter	Too high or too large	Too low or too small
Number of hidden nodes (NHN)	Overfitting ANN (no generalization)	Underfitting (ANN unable to obtain the underlying rules embedded in the data)
Learning rate (η)	Unstable ANN (weights) that oscillates about the optimal solution	Slow training
Momentum coefficient (μ)	Reduces risk of local minima. Speeds up training. Increased risk of overshooting the solution (instability)	Suppresses effect of momentum leading to increased risk of potential entrapment in local minima. Slows training
Number of training cycles	Good recalling ANN (i.e., ANN memorization of data) and bad generalization to untrained data	Produces ANN that is incapable of representing the data
Size of training subset (N_{TRN})	ANN with good recalling and generalization	ANN unable to fully explain the the problem. ANN with limited or bad generalization
Size of test subset (N_{TST})	Ability to confirm ANN generalization capability	Inadequate confirmation of ANN generalization capability

Figure 3.5: The consequences of extreme values of some network design parameters on network generalization and training convergence. [28]

3.3 Deep Learning

When multiple layers of perceptrons are drawn vertically and stacked in the horizontal direction, as shown in Fig. 3.6, the network architecture is considered "deep" in that direction, which gives NNs with multiple hidden layers that possess some complexity the name *deep neural networks* (DNN) [21]. Each hidden layer learns more complicated functions of the input data by combining the output values of the preceding layer. This is associated with an increasing level of abstraction happening in each hidden layer. Finally, the output layer's abstraction level becomes so high that the raw input is hardly recognizable in the end. What causes these abstractions are the gradual extraction of patterns, recurrencies and dependencies in the input data as the information flows into deeper layers [6]. A visual example of such an abstraction is given in the next subchapter regarding the convolutional neural networks.

3.3.1 Convolutional Neural Networks

This section is based on the works [40,50,59]. Convolutional neural networks are a type of ANNs that are primarily used in the field of pattern recognition with images at which they have particular advantages over classical ANNs. The input layer of a network that receives an image should have as many neurons as the pixels of the image, such that each pixel value is assigned to one neuron. If the image size is very large, there has to be a large number of neurons both in the input layer and in the sequential hidden layers to process all the information that is embedded in the image. As shown in Fig. 3.5, however, increasing the number of neurons are is connected with the risk of overfitting. Classical ANNs are not as good as CNNs at handling this computational complexity. Moreover, the weight updates in ANNs take significantly more time than in CNNs. In CNNs, locality is an essential characteristic, which is

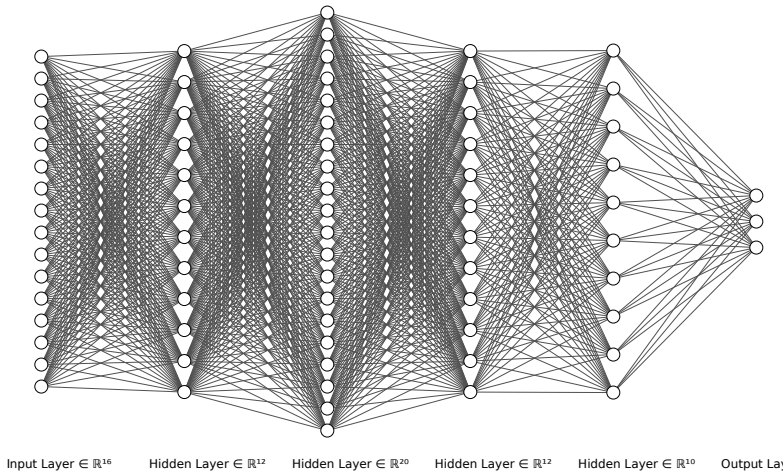
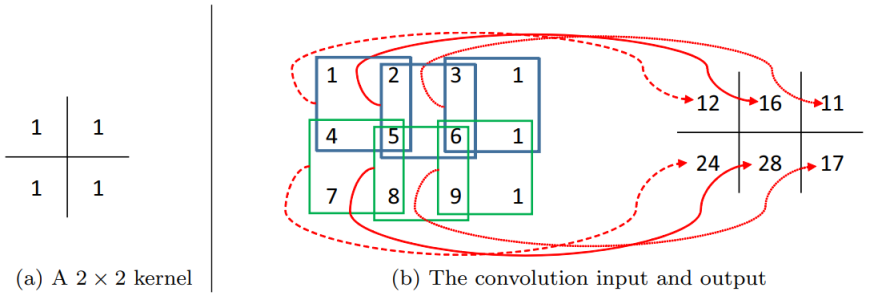


Figure 3.6: Sketch of a deep neural network architecture, created on [3]. The sizes of the six layers are given below. All nodes of one layer are connected with each of the nodes from the preceding layer, except the input nodes.

associated with the assumption that nearby points in an image have stronger correlation than points far away. Based on this, CNNs employ *filters* that have a smaller size than the input image, which glide through the image and perform certain local operations on the pixels in the overlapping area, whereby the operation of a filter is same throughout the image. This way, the same coefficients are used at different locations instead of fitting the entire image with new weights. As a result, the memory requirement of the model reduces. Another advantage of exploiting locality is that in this way, shifted patterns can still be recognized regardless on which part of the image they appear. Because fewer weights are to be updated, training a CNN requires also less time. The basic architecture of a CNN consists of four layer types:



(a) A 2 × 2 kernel (b) The convolution input and output

Figure 3.7: Illustration of a convolution operation performed by the 2x2 filter (kernel) given in (a) on the 3x4 matrix given in (b). By building the scalar product in each of the six areas, the 2x3 feature map on the right is created. [59]

convolutional layer, activation layer, pooling layer and fully connected or dense layer. The neurons of a convolutional layer are "feature detector" 2D filters that move across the image with specified step size called *stride* and compute the scalar product of their weights and the pixel values in the overlapping region of the image. The result becomes an entry of a matrix that comprises all such scalar products, which is called a *feature map*. This operation is illustrated in Fig. 3.7. The mathematical expression of

this operation is

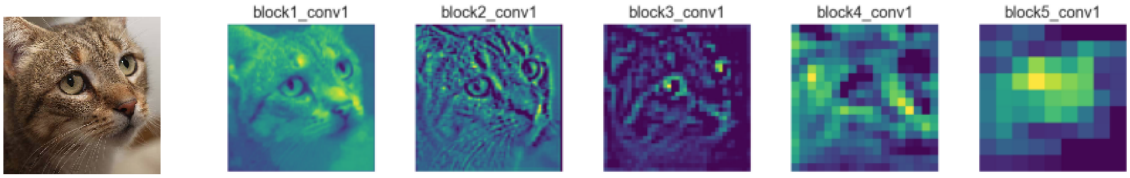
$$\begin{bmatrix} 1 & 2 & 3 & 1 \\ 4 & 5 & 6 & 1 \\ 7 & 8 & 9 & 1 \end{bmatrix} * \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} = \begin{bmatrix} 12 & 16 & 11 \\ 24 & 28 & 17 \end{bmatrix}.$$

The activation function applied to the convoluted image element-wise is given by the activation layer. Here, the most common choice is ReLu due to its promoting effect on the training speed and other advantages that are elaborated on in section 3.2.1.

The pooling layer performs downsampling on the image coming from the precedent layers, i.e. feature map, in order to reduce its size and summarize the relevant features. This is also done by sliding filters over the image, building either the average of the pixel values in the overlap or taking their maximum. The first pooling type is called *average pooling*, the second *maximum pooling*.

A fully connected layer is a layer, each output of which is calculated using all input elements. It is a classical ANN layer that takes the end result of the whole feature extraction process prior to it and uses these to reach a classification decision. In this respect, a CNN consists of two main sections: feature extraction and classification. The stages of feature extraction in a CNN model that learns to detect cats in images is illustrated in Fig. 3.8.

A significant property of CNNs is that the convolutional filters are not designed or specified before the training except for their size and number per layer, but they are rather determined as part of the training. Thus, the learned patterns and rules are independent of human intervention, i.e. teaching the model to search for certain features in the image.



(a) Raw input image of a cat

(b) Five feature maps of the given image from five convolutional layers

Figure 3.8: A CNN with five convolutional layers that is trained on a large set of cat images learns multiple filters in each layer. One filter from each layer, which are shown in (b), is applied to the input image given in (a). The deeper one goes in the network (from left to right), the more abstract the representations of the original image get. Since the first convolutional layers of a CNN usually act as edge detectors, the first convoluted image is not much different from (a). [15]

4 Lattice Site Detection using Neural Networks

For the binary classification task regarding the evaluation of our experimental images, which was presented in the previous chapters, three main stages of the present study featuring different neural network architectures are discussed and compared. The starting point is a simple ANN that is trained and tested on simulated images of an isolated atom in the center of a lattice region.

The neural networks are realized, trained and evaluated with the deep learning API Keras, which is written in the programming language Python and runs on top of the end-to-end, open-source machine learning platform TensorFlow 2. With a focus on enabling fast experimentation, Keras is a powerful as well as a well-aranged tool that is very suitable for the purpose of this study [56]. An introduction to Keras and Tensorflow 2 is beyond the scope of this thesis. Nevertheless, several functions whose parameters are discussed in the following, are introduced and shortly explained. More details on working with these tools for neural network design and application can be found in [20, 23, 43].

4.1 Step 1: Simple ANN - Isolated Atom

An isolated atom in a lattice site whose effective PSF broadened by atom's motion can be approximated by a Gaussian depicts an easier learning problem than an atom surrounded by other atoms, since the intensity information of the main atom is clear of the interference of the density distributions of neighbouring atoms. Once a NN learns this PSF, it should be able to detect whether the center of a given lattice region is occupied by an atom. In the same way, the NN learns the PSF by seeing enough labeled images during the training process of the binary classification task. This is due to the nature of any binary classification problem, which necessitates fitting given data to classify it. For further elaboration on this aspect, see chapter 2. Thus, what the NN does, resembles a threshold-based reconstruction method where fitting an image corresponds to it being deconvoluted with the known PSF.

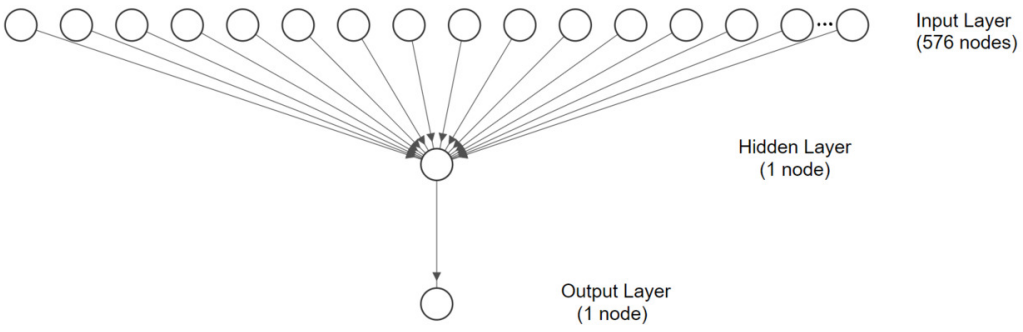


Figure 4.1: An artificial neural network with a single hidden layer that has one neuron. The input layer consists of neurons equal to the number of pixels, i.e. 576. The output layer is made of a single neuron. The arrows indicate the direction of the information flow and connections between neurons. The sketch is created on [3]

The simplest neural network that can work like a threshold reconstruction method is one with an input layer that consists of the same number of neurons as the image pixels and and output layer containing

only one neuron whose output represents the probability of the central lattice site being occupied, which is then rounded to 0 or 1 to assign a label to the input image. A hidden layer with a single neuron can also be added to the network, which has no additional impact on the output, yet makes the transition into the next stage of the study easier. Equivalent to formulating a weighted sum of the image pixels using the known PSF, the input layer of the NN acts on the pixel values given in the input vector $\mathbf{x}^{(m)}$ (flattened image, m^{th} of N input images) according to the formula

$$z^{(0)} = W^{(0)}\mathbf{x}^{(0)} + b^{(0)} \tag{6}$$

where $W^{(0)}$ is the weights matrix and $b^{(0)}$ is the bias, both specific to layer 0, i.e. the input layer. Since the output $z^{(0)}$ is the input of the hidden layer with a single node, it should be a scalar. Therefore, the weight matrix $W^{(0)}$ is a row vector of length n^2 (number of pixels in a square image of length n) and the bias $b^{(0)}$ is also a scalar. The produced output $z^{(0)}$ is transferred to the hidden layer, where it undergoes the change

$$z^{(1)} = g(W^{(1)}z^{(0)} + b^{(1)}) \tag{7}$$

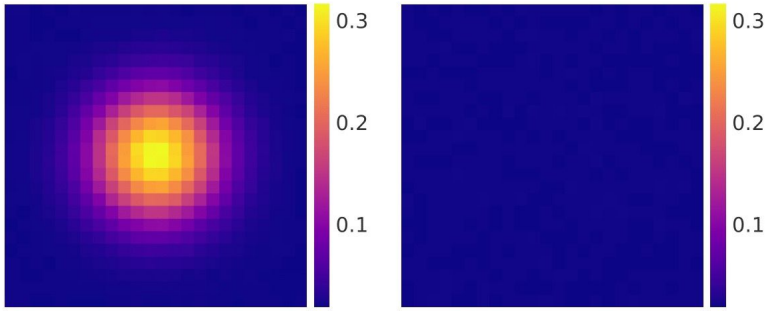
with a multiplication with the 1x1 weight matrix $W^{(1)}$, addition of the bias $b^{(1)}$, to which the sigmoid activation $g(t) = (1 + e^{-t})^{-1}$ is applied. $z^{(1)}$ is then transferred to the output layer where the same calculation (7) is repeated with the scalar weight and bias of this last layer (layer 2).

The matrix multiplication $W^{(0)}\mathbf{x}^{(0)}$ which is equal to the scalar product ${W^{(0)}}^T \cdot \mathbf{x}^{(0)}$ implicates ${{W^{(0)}}^T}_i = \text{PSF}(x_i)$ in analogy to deconvolution, where x_i is the i^{th} pixel value. Thus, by extracting the weights learned between the input and the hidden layer and reshaping them into a nxn matrix, the learned PSF of one atom can be visualized.

A data set with a structure shown in table 2.1 is created for training and evaluation. Image size is chosen as 24x24 px^2 so as to capture all fluorescence photons that are Gaussian-distributed around the center with a standard deviation of $\sigma = 3.3\,\text{px}$. The selected number of training and test images with an occupied center is 50k. With the same number of images containing only noise, the total size of the training and the test set is described by the tuple $(2, 100\,000)$ each, since each image is assigned a label $y = 0$ or $y = 1$ depending on its occupation, which gives the data sets the shape $\{(x^{(i)}, y^{(i)})\}_{i\in\{1,\dots,m\}}$. However, the 20% of the training set is split as validation set, yielding $80 \cdot 10^3$ images only for training and $20 \cdot 10^3$ images for validation. The 80:20 ratio is a common value, which is mainly derived from experience, yet can also be justified statistically [4]). An average over $50 \cdot 10^3$ images that are occupied and another $50 \cdot 10^3$ that are unoccupied is plotted in Fig. 4.2.

After designing and initializing the model that is simply given the name “nn” by declaring it as an object of the model class of Keras, which allows a model to call numerous functions defined in that class, the training is started by calling the function nn.fit(). The specified parameters of this function as well as the course of a training that can be monitored, followed and recorded is shown in Fig. 4.3.

The metrics chosen for the model to evaluate during training and testing is the accuracy, which gives the ratio of correctly classified examples to the total number of predictions. As optimizer, the Adam



(a) Isolated atom (b) Noise

Figure 4.2: Averaged images over $50 \cdot 10^3$ simulated images of (a) an isolated atom in the center of a 24x24 px^2 lattice region with background noise, (b) the same region with only background noise. The colorbars indicate the brightness of the pixels. Because noise lacks a structure, its average over a large number of images yields no consistent pattern.

```
[941] history = nn.fit(X_train, Y_train,  # save the metrics in a dictionary during the training process
            batch_size=128,
            epochs=1000,                  # using early stopping, so no real limit
            verbose=2,                    # monitor training process for each epoch
            validation_split=0.20,
            callbacks=[early_stopper])
```

```
Epoch 1/1000
313/313 - 1s - loss: 0.1282 - accuracy: 0.9622 - val_loss: 0.0558 - val_accuracy: 0.9820 - 1s/epoch - 4ms/step
Epoch 2/1000
313/313 - 1s - loss: 0.0504 - accuracy: 0.9826 - val_loss: 0.0517 - val_accuracy: 0.9808 - 855ms/epoch - 3ms/step
Epoch 3/1000
313/313 - 1s - loss: 0.0464 - accuracy: 0.9831 - val_loss: 0.0546 - val_accuracy: 0.9804 - 860ms/epoch - 3ms/step
Epoch 4/1000
313/313 - 1s - loss: 0.0453 - accuracy: 0.9834 - val_loss: 0.0535 - val_accuracy: 0.9798 - 870ms/epoch - 3ms/step
Epoch 5/1000
313/313 - 1s - loss: 0.0439 - accuracy: 0.9835 - val_loss: 0.0544 - val_accuracy: 0.9808 - 863ms/epoch - 3ms/step
Epoch 6/1000
313/313 - 1s - loss: 0.0426 - accuracy: 0.9840 - val_loss: 0.0538 - val_accuracy: 0.9810 - 856ms/epoch - 3ms/step
```

Figure 4.3: A block of code that is executed to start a training and the progress of the training on the monitor below, which is enabled by the command "verbose=2". All the monitored values are real-time stored in a dictionary that is given the name *history*. The other parameters are elaborated in the text.

algorithm is found to be the most common choice among various optimization methods for being very robust, easily tunable and well-suited to a wide range of optimization problems, in particular in solving classification problems, and requiring little memory for even large data sets [33,41]. The update rule used in a simple gradient descent algorithm to improve the i^{th} weight $w_{t,i}$ at time t is

$$w_{t+1,i} = w_{t,i} - \eta \cdot g_{t,i}$$

where η is the defined constant learning rate and $g_{t,i}$ is the derivative of the cost function with respect to $w_{t,i}$, i.e. gradient. The Adam algorithm makes use of an estimate for the first and second order moments ($\hat{m}_t$ and $\hat{v}_t$, respectively) of the gradient $g_{t,i}$ to adapt the learning rate at each step. This results in the update rule

$$w_{t+1,i} = w_{t,i} - \eta \cdot \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}$$

with a hyperparameter ϵ. Although there are two more hyperparameters used in the estimation of the

moments, their default values are mostly useful enough to tune the constant learning rate η only [48].

During the training and testing, a quantity named *loss* is used to evaluate how good the predicted results match the true labels. It returns low values for good predictions and high values for bad predictions. The loss function that is suitable for binary classification tasks is the *binary cross entropy function* [62]. When the true labels are denoted by p_i and the predicted outcomes by q_i for the i^{th} class, the cross entropy function is defined as

$$H(p,q) = -\sum_i p_i \log(q_i).$$

For the case of two classes $i \in \{1,2\}$, p_i and q_i are either 1 or 0. If $p_1 = 0$, then $p_2 = 1$. The two labels shall be given by y and $1-y$. Then, writing p_1, p_2 is simplified to writing $y, 1-y$. Analogously, q_1, q_2 can be written as $\hat{y}, 1-\hat{y}$ with the predicted labels $\hat{y}$ and $1-\hat{y}$. Thus, the binary form of this function reads

$$H(p,q) = -y\log(\hat{y}) - (1-y)\log(1-\hat{y})$$

for one example. When the loss for each example in a training set and their average is computed, it yields the function

$$J_w(y,\hat{y}) = -\frac{1}{m}\sum_j^m (y^{(j)}\log(\hat{y}^{(j)}) + (1-y^{(j)})\log(1-\hat{y}^{(j)}))$$

which becomes the negative sum of $\log(1-\hat{y}^{(j)})$ when $y^{(j)} = 0$ that is minimized by $\hat{y}^{(j)} = 0$ and the negative sum of $\log(\hat{y}^{(j)})$ when $y^{(j)} = 1$ that is minimized by $\hat{y}^{(j)} = 1$. This shows that the function $J_w(y,\hat{y})$ perfectly summarizes what is expected for the model to achieve: $y^{(j)} \stackrel{!}{=} \hat{y}^{(j)}$ for as many examples as possible. This is why it is the cost function of this problem. In other words, by specifying the binary cross entropy function as the loss function to use in the present task, the cost function to be minimized is automatically specified [62]. The average (4.1) is computed and displayed at the end of each epoch.

```
opt = Adam(learning_rate = 0.3)

nn.compile(optimizer =opt,
          loss = BinaryCrossentropy(from_logits=True),
          metrics = ['accuracy'])

early_stopper = EarlyStopping(
    monitor='val_loss',
    patience=30,
    restore_best_weights=True
    )
```

Figure 4.4: A block of code that is executed to configure several features regarding the training process as the optimizer (Adam), loss function (binary cross entropy), metrics (accuracy) and callback function (EarlyStopping).

The parameter *callbacks* is used to manipulate, in this case, to end, a training process by setting some conditions or a tolerance interval to the outputs. The EarlyStopping callback provided by Keras can be used to set a metric that should be computed after each epoch and watched until it violates one of the conditions. To avoid overfitting, it is a good way to monitor the loss function computed after each

epoch of the validation following the training. The parameter "patience" enables the callback to stop the training after the given number of epochs with no improvement in the specified metric. By setting the parameter "restore best weights" to *True*, the updated weights that resulted in the lowest validation loss will be restored.

The batch size, i.e. the number of samples per gradient update is varied until the best results are obtained. The code displaying all the discussed functions and parameters can be found in Fig. 4.3 and 4.4.

The simple NN that is described in the beginning of this chapter is trained on the simulated images several times until the best parameters are found. With a learning rate $\eta = 0.3$, batch size $b = 128$ and patience $= 30$, a 99.81% test accuracy and a 0.014 test loss (from here on accuracy and loss) are obtained. The learned weights are saved and given to the default network as initial values as an orientation. When the patience is increased to 40, the training with weight initialization leads to a slightly higher accuracy 99.83% and a lower loss 0.013. The course of accuracy and loss during the training and validation is presented in Fig. 4.5.The learned PSF is shown both in 2D and 3D in Fig. 4.7 and 4.6, respectively.

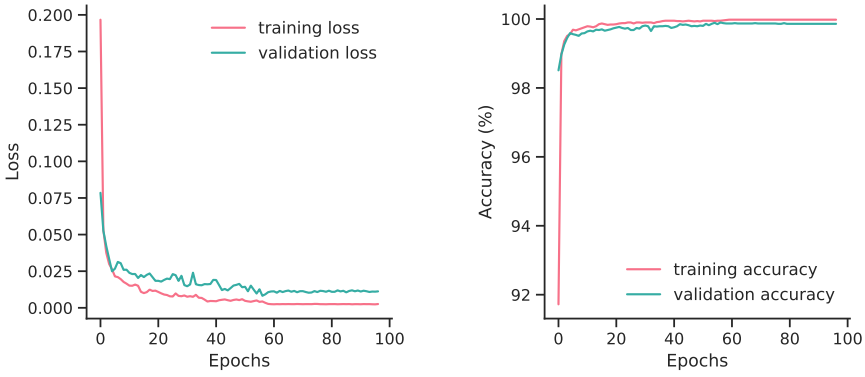
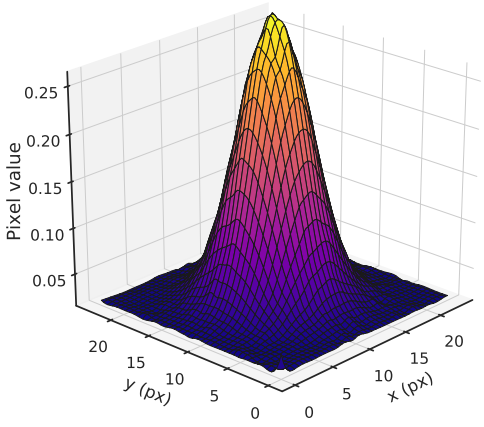
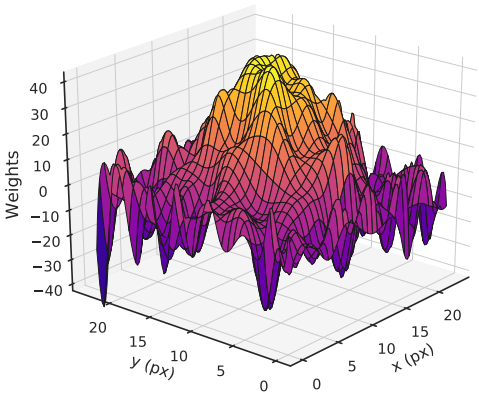


Figure 4.5: Loss and accuracy are computed for the outputs during the training and validation at the end of every epoch. The validation accuracy almost matches the training accuracy near 100%, while the validation loss is slightly above the training loss.



(a) Average over 50 000 images containing an isolated atom at the center of a lattice region, which is approximately the PSF the model should learn.



(b) The PSF that is learned by the simple neural network with a single hidden layer node.

Figure 4.6: Figures depicting (a) the average pixel values of a simulated lattice region with an occupied center, (b) the learned PSF of an isolated atom in the center of a given region. The x-y-plane is the examined lattice site region. The z-direction shows the weight a pixel at (x,y) carries. The 3D surfaces are interpolated by using their bivariate B-spline representation. For more information, see [2]. There is a pronounced difference between the two surfaces with regard to their smoothness, which is discussed in 4.1. Nevertheless, both surfaces possess a relatively high intensity in the center, as it is expected.

Interpretation of the Results

As shown in Fig. 4.6, despite the lack of distraction from other neighbouring atoms, the PSF of an isolated atom could not be learned by the simple model such that it reflects the smooth 2D Gaussian on the left figure. However, the classification accuracy is over 99%, which indicates that fitting with the PSF in 4.6b is very successful. The first reason that comes to mind is that the noise pixels might not be fully distinguished from the fluorescence photon pixels, since both have the same intensity due to binarization and that this may have led the model to learn to fit even the noise. What is problematic with this explanation is, however, that the simple NN as an analogue of weighted sum threshold-reconstruction should be able to look past the noise over a large data set, since the noise is believed to have no structure

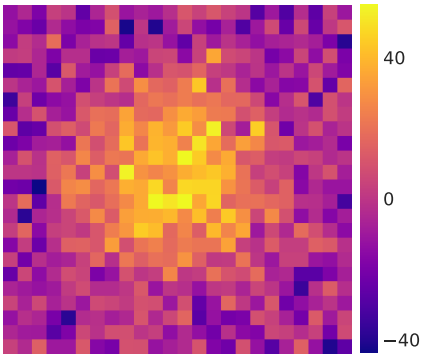


Figure 4.7: Image of the PSF of an isolated atom. Each pixel has the value of the corresponding weight given by the weight matrix $W^{(0)}$, which is in this case a row vector of length 576, that is learned by the simple neural network with a single hidden layer node. The colorbar indicates those values.

or attributes that can be learned. Sometimes there may be additional attributes hidden in noise that have not been taken into account, which may affect the learning process and predictions. Even though this is not considered likely for our case, the cause of the described observation is still not understood.

Even though a peak in the middle and a descending trend around it is visible because that area contains more bright pixels, the surface lacks a smooth, symmetric form. This aspect may be improved by using a larger training set that enables the model to generalize more as opposed to fitting the details like noise. The training can also be performed on non-binarized images in which fluorescence pixels will outnumber noise pixels, which might contribute to a better distinction between the two. Lastly, it should be kept in mind that the present model is a very simple one whose enhancement may yield more accurate results as well as a PSF that meets the expectations to a greater degree.

Testing other well-known successful weight initialization techniques can help improve the mentioned aspects as well. For a detailed explanation of the effect and different methods of weight initialization, see [35, 60].

4.2 Step 2: ANN - Atom with Nearest Neighbours

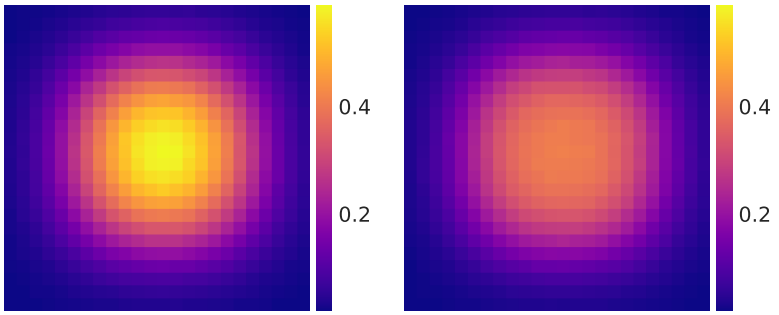
The second stage of the study is to simulate images of lattice regions that potentially contain more than one atom which is in the center of the frame and use these to train two ANNs. One of these is the 3-layer-NN introduced in the previous chapter. The second one is a 3-layer NN with a 512 hidden neurons, which are referred to as “simple NN” and “enhanced NN” in the following, respectively. The main goal is to study the effect of the nearest neighbour distributions on how good the networks can recognize and predict the occupation of the central site by varying the lattice spacing between 4 and 8 pixels, while the PSF has a standard deviation of $\sigma = 3.3\,\mathrm{px}$. These five cases are denoted by the figure of merit d/σ they describe. To introduce two classification classes describing the central site occupation to the models, the images are produced such that half of them have various atom configurations with a certainly occupied central site ($y = 1$). The other half have configurations in which the central site in unoccupied ($y = 0$). Thus, the data structure presented in Tab. 2.1 is adopted again.

The reason why an enhanced NN is contrasted to the simple NN is to test to which degree more hidden neurons contribute to the model improvement that is associated with features such as higher accuracy, lower loss, faster convergence and less memory usage. As elaborated in the theory chapter 3, a right number of hidden layers and hidden neurons are expected to enhance the accuracy, lower the loss, speed up the training but also use more storage space. In this study, the described improvements are observed and reported only for a NN with one hidden layer that has 512 neurons, which is found to be either the best or one of the best choices for all figures of merit. An alternative perspective on this number choice is also given in the subchapter 4.2.1.

The configuration of the networks as well as the training process remains the same as in the previous step. In principle, always the training parameters are mentioned that are tested among others and found to be the best. The results for both NNs are discussed in view of the visualized weights and learning

curves.

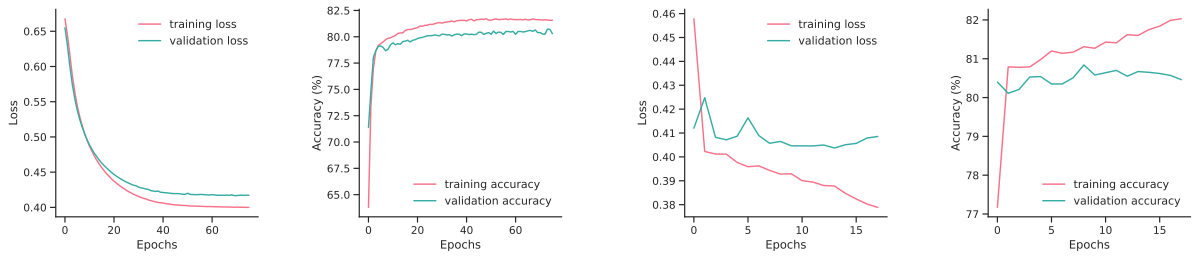
4.2.1 FOM = 1.21



(a) Center is occupied (b) Center is unoccupied

Figure 4.8: Averaged images over $50 \cdot 10^3$ simulated images with FOM = 1.21, i.e. lattice spacing $d = 4\,\text{px}$, whose central lattice site is (a) occupied, (b) unoccupied. The 8 nearest neighbours are not distinguishable from the central atom or gap due to the small lattice spacing relative to the size of their PSFs.

It is started with the most limiting case where the lattice sites are tightly confined in a small area, making it more difficult for any classification method to distinguish between the central atom and its adjacents. Even when the central lattice site is unoccupied, it is likely that the PSF-overlaps of neighbouring atoms brightening the empty space cause a confusion about the occupation of the central site. This can be seen very well in Fig. 4.8 featuring averages over $50 \cdot 10^3$ images with an occupied and another $50 \cdot 10^3$ with an unoccupied central lattice site.

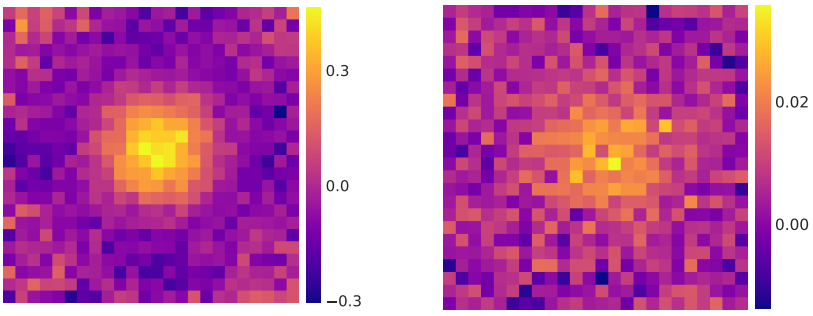


(a) Learning curves of the simple NN (b) Learning curves of the enhanced NN

Figure 4.9: Training and validation loss and accuracy computed at the end of each epoch while training the (a) simple NN, (b) enhanced NN with 512 hidden neurons for the lattice spacing $d = 4\,\text{px}$. In both cases the training accuracy is remarkably higher than the validation accuracy and vice versa for the loss values. The validation loss of the enhanced model starts to increase in very early epochs, while the validation loss of the simple model follows a rather stable course without a rise.

The training of the simple NN performed with a learning rate $\eta = 0.001$, batch size $b = 256$ and patience=4, results in an accuracy of 80.80% and a loss 0.412. Training of the enhanced NN is performed with $\eta = 0.001$, batch size $b = 300$ and patience = 4, which yields the accuracy 80.95% and loss 0.399. The additional hidden neurons bring only a 0.186% improvement in the accuracy. As shown in Fig. 4.9a, convergence of the first training happens around the 45^{th} epoch. This is rather late compared to the

second training shown in 4.9b that can be stopped already on the 10^{th} epoch to avoid overfitting. The loss of both models have a comparable trend. Due to the almost same batch size used in the training, the only difference in memory usage of both models is because of the great number of weights that are updated and stored in the enhanced model.



(a) PSF learned by the simple NN

(b) Averaged weights of the 512 hidden neurons in the enhanced NN

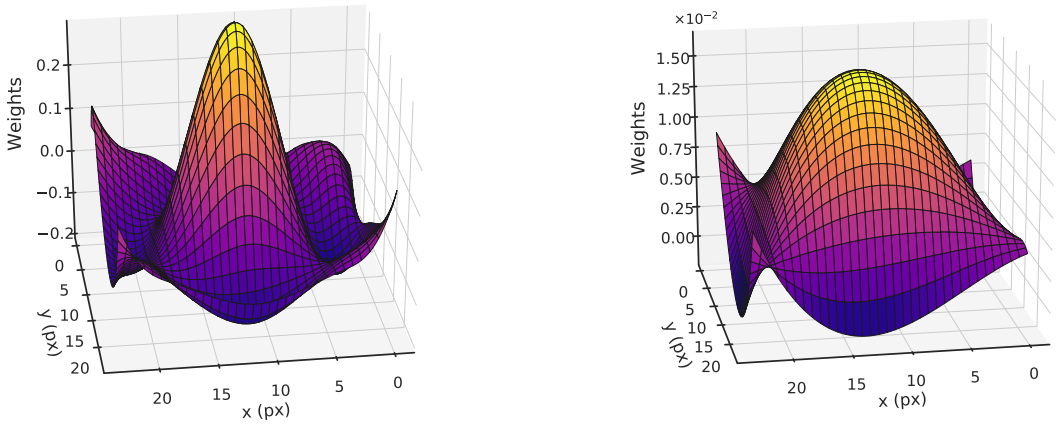
Figure 4.10: The results of the training for FOM = 1.21, i.e. lattice spacing $d = 4\,\text{px}$ presented in 2d. In (a) the PSF learned by the simple NN can be seen. The close surrounding of the central site where the nearest four lattice sites lie is darkened while the further areas are brighter. In (b) the averaged image over all 512 weights that the hidden neurons in the enhanced NN learned is given. Colorbars indicate the pixel value.

The PSF the simple model learned is visualized in 2D in Fig. 4.10a and in 3D in Fig. 4.11a. A distinct Gaussian peak in the center and a deep dark area surrounding it is a perfect sign of the model assigning negative values to the pixels constituting the neighboring lattice sites to filter the potential neighbouring atoms out. Thus, if the center is occupied, convolution with this PSF will result in a clear peak allowing the model to make a good prediction. In the same way, if the center is empty, i.e. the image pixels on the central lattice site have a near-zero value, convolution with the bright Gaussian peak of the PSF will be ineffective in terms of changing the pixel values and the center will remain dark. Since the neighbouring atom distributions will also not be able to affect the pixel values in the central lattice to a great extent, the center will be ready to be classified as unoccupied. Nevertheless, the extent to which their interference is hardly avoidable is greater than 15% of the classification accuracy. As it will be shown in the following, this extent is reducible for larger lattice spacings.

The neighbors that lie not d but $d\sqrt{2}$ far from the center of a square lattice will produce less overlaps that risk the classification of the central site. Therefore, the corners of the image containing these neighbours are allowed to be less filtered. Thus, these areas have a higher intensity than the direct surrounding of the central site.

512 Hidden Neurons, 512 PSFs?

Similar to the equivalence of the weighted sum method and fitting an isolated atom with one Gaussian (PSF) that is described in step 1, according to [45]), there is an analogy between another deconvolution-based method and a specific ANN architecture. This common method, which is also used in [38, 42], is



(a) PSF learned by the simple NN

(b) Averaged weights of the 512 hidden neurons in the enhanced NN

Figure 4.11: 3D representation of the results presented in Fig. 4.10. The x-y-plane is the examined lattice cite region. The z-direction shows the weight a pixel at (x,y) carries. The left surface possesses a clear Gaussian peak in the middle and deepened valleys where the nearest four lattice sites lie.

based on fitting a subregion of an image that comprises multiple lattice sites with a set of Gaussians of different amplitudes on each site, depending on their occupation. Subsequently, a threshold is applied to each fitted amplitude to determine which sites are occupied. This way, the signals produced by atoms on neighbouring lattice sites can be better distinguished [45]).

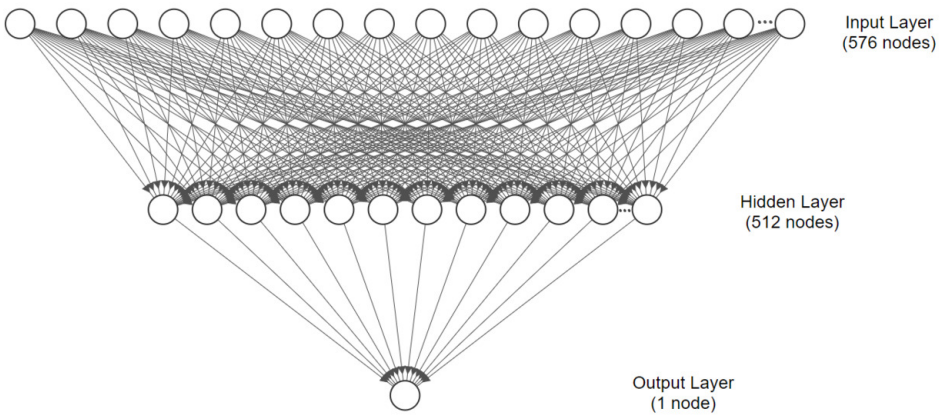


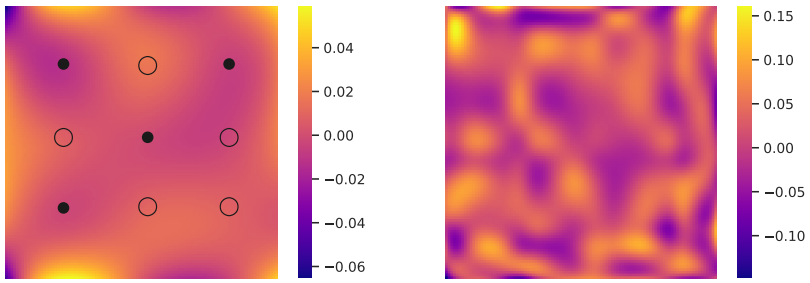
Figure 4.12: An artificial neural network with one hidden layer that consists of 512 nodes. The arrows indicate the direction of the information flow. Each neuron is connected to all neurons of the previous layer. The sketch is created on [3]

An ANN with a hidden layer that consists of 512 neurons is demonstrated in Fig. 4.12. Because there is no more a single hidden neuron that is connected with all 576 input neurons, but 512 of them, there are also 512 different weight matrices $W^{(0,i)}$. The output of the hidden layer in (6) thus becomes a vector of length 512. The 512 components of the vector $\mathbf{z}^{(1)}$ are then transferred to the output neuron after a multiplication with the weight matrix $W^{(1)}$, addition of a scalar bias $b^{(1)}$ and application of the sigmoid function to scale the result. The output neuron acts then as a judge deciding whether the central site is

occupied, which resembles labeling the sites by comparing the Gaussian fit amplitude to a threshold.

It is argued that employing such a network leads each hidden neuron to learn one of the $2^9 = 512$ configurations of 9 neighbouring lattice site occupations. Thus, each weight matrix reflects the sum of the PSFs on each lattice site in one of the 512 configurations, as it is the case for the explained deconvolution-based method. When an image is fed into the network, the hidden neurons, whose corresponding PSF has the greatest overlap with the distribution in the image, produce larger weights in the matrix $W^{(1)}$; thus, a greater contribution to the sum resulting from $W^{(1)}\mathbf{z}^{(0)}$ that determines the final neuron's output. If most of the strongly activated hidden neurons represent a distribution in which the central lattice site is occupied, the final neuron's output will very likely be $y = 1$. In the opposite case, $y = 0$.

This is a reasonable expectation given that both methods follow a way of fitting the same sort of images by utilizing the Gaussian PSF. In order to test this idea, the described NN is built and the same set of $(d = 4)$-images that are used to train the simple 1-hidden-neuron network are used to train the new NN. With a learning rate $\eta = 0.001$, batch size $b = 300$ and patience=4 the accuracy 88.95% and loss 0,399 are achieved. The train and validation loss as well as accuracy over training epochs can be seen in Fig. 4.9b. The weight matrices $W^{(0,i)}$ are extracted and plotted for many $i \in \{1, ..., 512\}$. Two of them that according to the above explained idea show the PSF of the hidden neuron number 202 and hidden neuron number 431 are presented in Fig. 4.31. The fist image is one of those in which it is somewhat possible to assign the lattice sites and predict the corresponding distribution of atoms. However, there are plentiful images like the one on the right which have a rather complex surface and do not allow such a prediction.



(a) Weight matrix $W^{(0,431)}$ (b) Weight matrix $W^{(0,202)}$

Figure 4.13: Visualized weight matrices $W^{(0,431)}$ and $W^{(0,202)}$ with a color code that reflects their elements' values. The left image can be assigned nine neighbouring lattice sites by analyzing the dark and bright areas and marking their estimated center. Their occupation is also determined based on the brightness of the areas around the marked centers. Black dots stand for unoccupied sites, empty circles stand for occupied sites. Unoccupied Images are interpolated to smoothen the surfaces and make their interpretation easier.

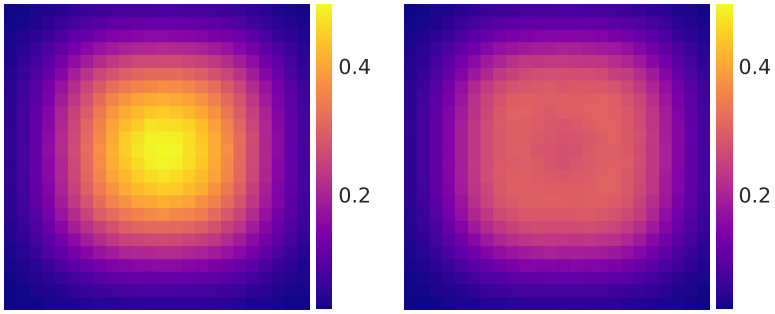
From this small experiment it can be concluded that there is no such rule for the given NN and 9-segment images as "each hidden neuron learns a specific PSF that describes one configuration". This implies that it is not a small set of hidden neurons that describe the real distribution the best that dominates the decision-making regarding the occupation of the central lattice site, but a non-generalizable combination of hidden neurons. In case such a generalization is considered possible, a suitable test with

the explained NN and images should be performed, which is not done in this thesis since this was not the main focus of the study.

It is found, however, that the average over all weight matrices yield a matrix that resembles the PSF learned by the simple NN. This comparison is shown in 2D in Fig. 4.10 and in 3D in Fig. 4.11. This is an unsurprising result, since averaging over 512 matrices corresponds to replacing 512 hidden neurons with a single one, resulting in the same architecture as in Fig. 4.1. Nevertheless, the differences are remarkable because the built arithmetic average does not consider the vector $\mathbf{z}^{(0)}$, which is the input of the second layer, whose multiplication with the weight matrices leads to a strong activation of some neurons and a weaker activation of the others. This visual comparison in 2D, 3D or in both will be presented in the following subchapters as well. However, its detailed discussion is avoided since this would only cause a repetition of the explanation that is given above.

4.2.2 FOM = 1.52

The next lattice spacing that is studied is $d = 5\,\text{px}$, corresponding to a figure of merit 1,52. The average of simulated images can be found in Fig. 4.14. As compared to the lattice spacing $d = 4\,\text{px}$, an unoccupied central lattice site for $d = 5\,\text{px}$ seems more recognizable.



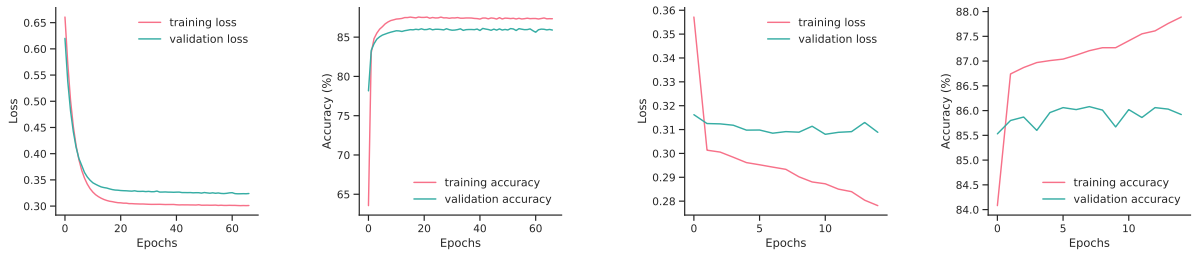
(a) Center is occupied (b) Center is unoccupied

Figure 4.14: Averaged images over $50 \cdot 10^3$ simulated images with FOM = 1.52, i.e. lattice spacing $d = 5\,\text{px}$, whose central lattice site is (a) occupied, (b) unoccupied.

In the training of the simple model the learning rate $\eta = 0.001$, batch size $b = 64$ and patience = 4 is used. The achieved accuracy and loss are 86.37% and 0,318, respectively. The highest accuracy and the lowest loss achieved with the training of the enhanced NN are 86.51% and 0,303, respectively. The used training parameters are unchanged. The improvement in accuracy by using a larger network with 512 neurons is only 0.16%, which is smaller than for $d = 4\,\text{px}$.

Fig.4.15 shows that the necessary number of iterations in the first training is again larger than in the second training, which indicates a faster convergence to the searched minimum of the cost function for the enhanced model. The difference between the validation and training loss and the validation and training accuracy in both trainings are comparable.

Fig. 4.16a and 4.17a display the PSF the simple model learned in 2D and 3D, respectively. The darkened nearest lattice sites, brighter corners and the sharp Gaussian peak in the center confirm the

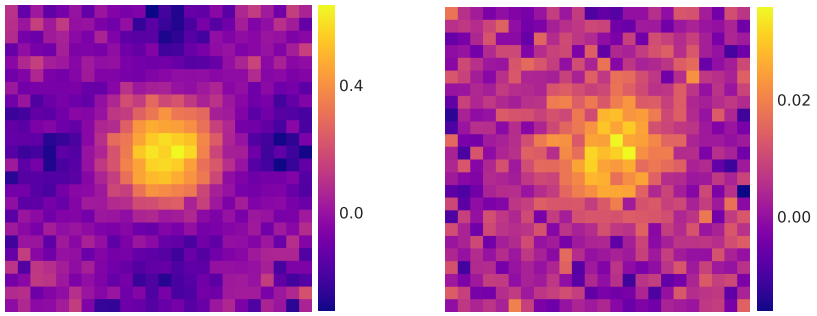


(a) Learning curves of the simple NN

(b) Learning curves of the enhanced NN

Figure 4.15: Training and validation loss and accuracy computed at the end of each epoch while training the (a) simple NN, (b) enhanced NN with 512 hidden neurons for the lattice spacing $d = 5\,\mathrm{px}$.

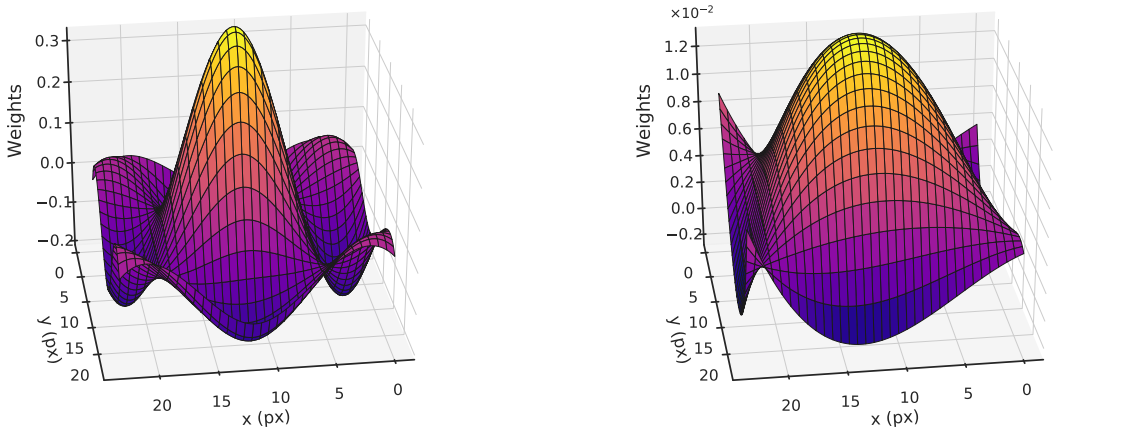
interpretation expressed in the previous subchapter 4.2.1.



(a) PSF learned by the simple NN

(b) Averaged weights of the 512 hidden neurons in the enhanced NN

Figure 4.16: The results of the training for FOM = 1.52, i.e. lattice spacing $d = 5\,\mathrm{px}$ presented in 2D. In (a) the PSF learned by the simple NN is shown. The close surrounding of the central site where the nearest four lattice sites lie is darker than the further areas. In (b) the averaged image over all 512 weights that the hidden neurons in the enhanced NN learned is given.



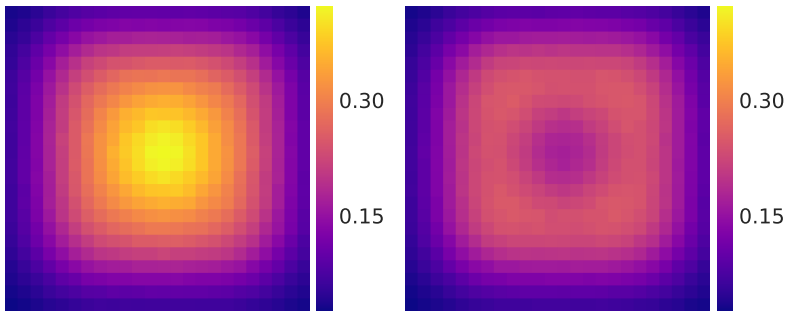
(a) The PSF that is learned by the simple NN

(b) Averaged weights of the 512 hidden neurons in the enhanced NN

Figure 4.17: 3D representation of the results presented in Fig. 4.16. The z-direction shows the weight a pixel at (x,y) carries.

4.2.3 FOM = 1.82

As the lattice spacing is increased, the bright pixels that stem from the neighbouring atom distributions no longer cover the unoccupied central lattice site, see Fig. 4.18. One expects both models to perform better than for the previous lattice spacings.



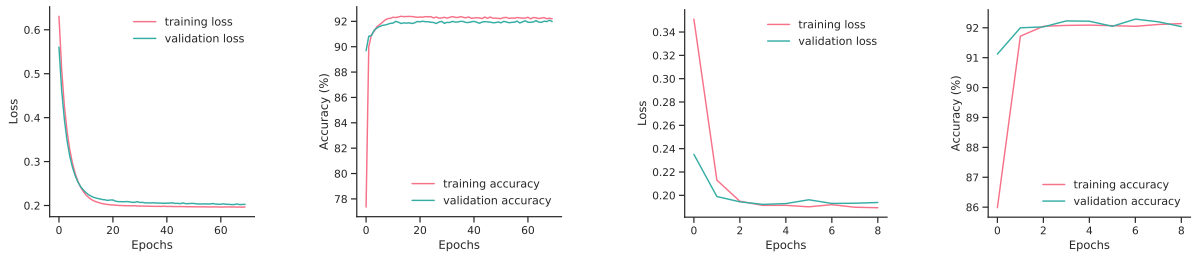
(a) Center is occupied (b) Center is unoccupied

Figure 4.18: Averaged images over $50 \cdot 10^3$ simulated images with FOM = 1.82, i.e. lattice spacing $d = 6\,\text{px}$, whose central lattice site is (a) occupied, (b) unoccupied.

The simple model is trained with a learning rate $\eta = 0.001$, batch size $b = 64$ and patience = 5, which yields a 91.7% accuracy and 0,210 loss. The enhanced model is, on the other hand, trained with a lowered learning rate $\eta = 0.005$, while the other parameters are remained the same. One obtains the accuracy 91.86% and loss 0.200. The accuracy improves about 0.18% when the network involves 512 hidden neurons.

In both trainings, the gap between the validation and training loss and validation and training accuracy becomes smaller. The second training is as always faster, requiring even less than 4 iterations. The

matching between these metrics in both trainings is a good sign that the networks are neither over- nor underfitting.

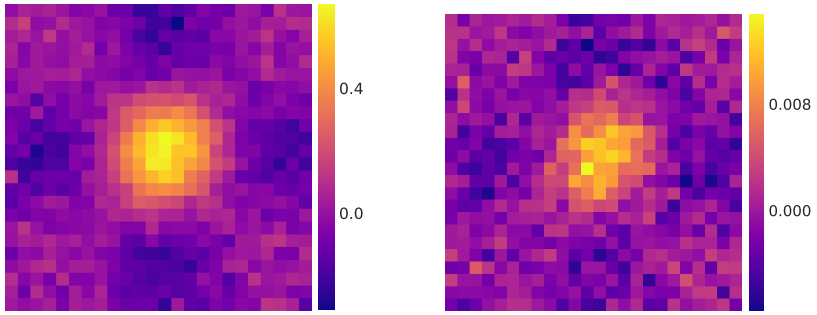


(a) Learning curves of the simple NN

(b) Learning curves of the enhanced NN

Figure 4.19: Training and validation loss and accuracy computed at the end of each epoch while training the (a) simple NN, (b) enhanced NN with 512 hidden neurons for the lattice spacing $d = 6\,\text{px}$. There is a nice alignment between the training and validation metrics in each training.

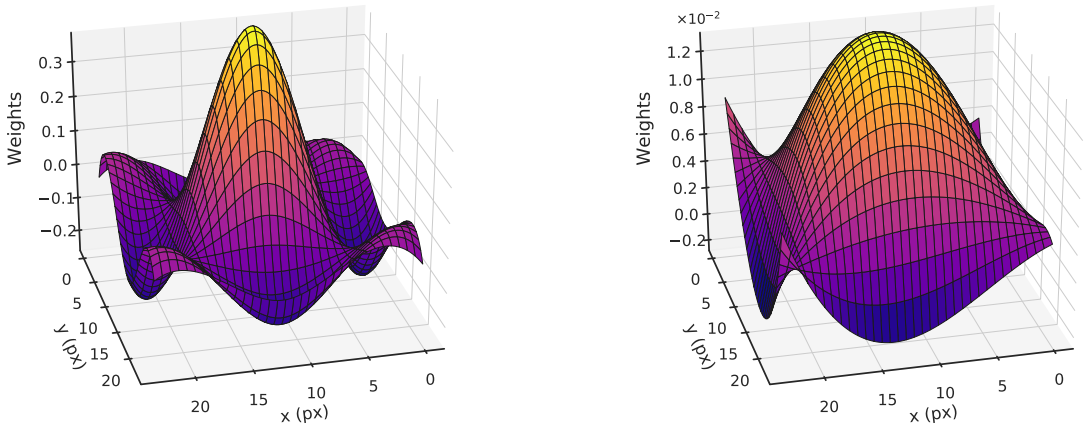
Fig. 4.20a and 4.21a display the PSF the simple model learned in 2D and 3D, respectively. The form that is seen at the previous two lattice spacings continue to appear in the PSF. Furthermore, it becomes slowly visible that the model filters the neighbouring lattice sites more weakly. This is because the overlaps due to the tight spacing slowly loose their harming effect on the classification. The model does not require to control them as strictly as it had to in the previous cases.



(a) PSF learned by the simple NN

(b) Averaged weights of the 512 hidden neurons in the enhanced NN

Figure 4.20: The results of the training for FOM = 1.82, i.e. lattice spacing $d = 6\,\text{px}$ presented in 2D. In (a) the PSF learned by the simple NN is shown. In (b) the averaged image over all 512 weights that the hidden neurons in the enhanced NN learned is given.



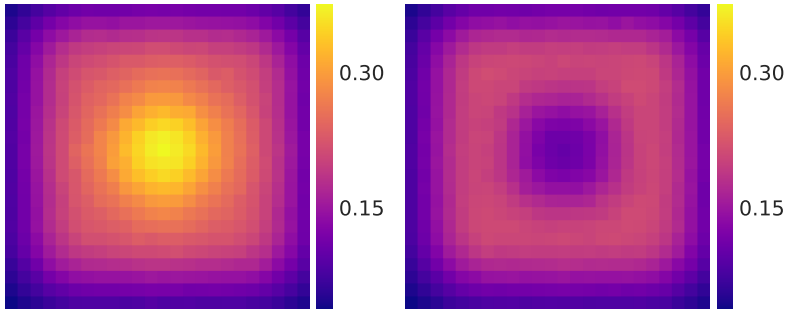
(a) The PSF that is learned by the simple NN

(b) Averaged weights of the 512 hidden neurons in the enhanced NN

Figure 4.21: 3D representation of the results presented in Fig. 4.20.The x-y-plane is the examined lattice cite region. The z-direction shows the weight a pixel at (x,y) carries.

4.2.4 FOM = 2.12

The averaged simulated images with a lattice spacing $d = 7\,\text{px}$ are shown in Fig. 4.22. The clear separation of the neighbouring atoms is an advantage for the classifiers, since it makes it easier to distinguish between neighbouring sites.

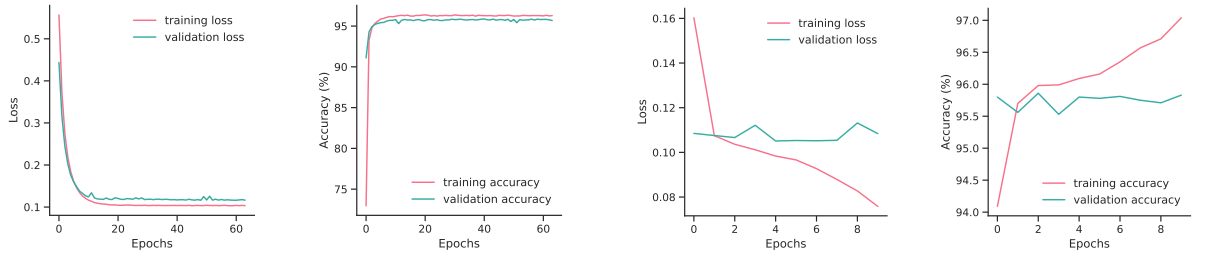


(a) Center is occupied (b) Center is unoccupied

Figure 4.22: Averaged images over $50 \cdot 10^3$ simulated images with FOM = 2.12, i.e. lattice spacing $d = 7\,\text{px}$, whose central lattice site is (a) occupied, (b) unoccupied.

The simple model is trained with a learning rate $\eta = 0.03$, batch size $b = 128$ and patience = 15, while for the enhanced network the parameters $\eta = 0.001$, batch size $b = 64$ and patience = 5 are selected. The accuracy and loss obtained from the first training are 95.97% and 0.114, respectively. The second training yields the accuracy 96.0% and loss 0.103. Thus, the accuracy improves about 0.03%. That the improvements by increasing the number of hidden units become smaller, can be an indicator of some inherent limits the classifiers face, both in terms of their learning capacity and the features of the simulated images.

The same conclusions that are made for the learning curves of the previous implementations apply to the learning curves in Fig. 4.23. The fast rise of the training accuracy and fall of the training loss of the enhanced model can be viewed as a warning against the tendency of the model towards overfitting. Therefore, the training should be executed for a small number of iterations.

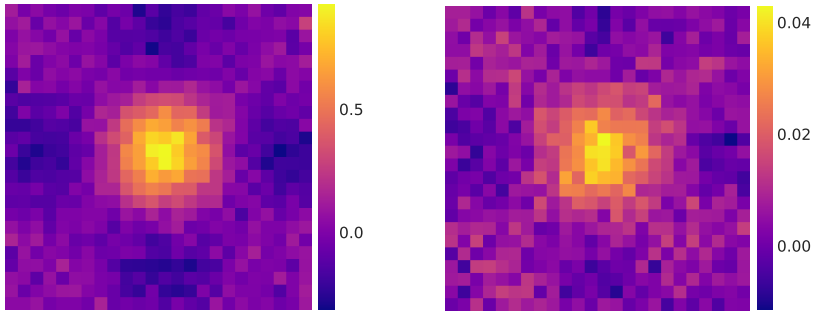


(a) Learning curves of the simple NN

(b) Learning curves of the enhanced NN

Figure 4.23: Training and validation loss and accuracy computed at the end of each epoch while training the (a) simple NN, (b) enhanced NN with 512 hidden neurons for the lattice spacing $d = 7\,\text{px}$.

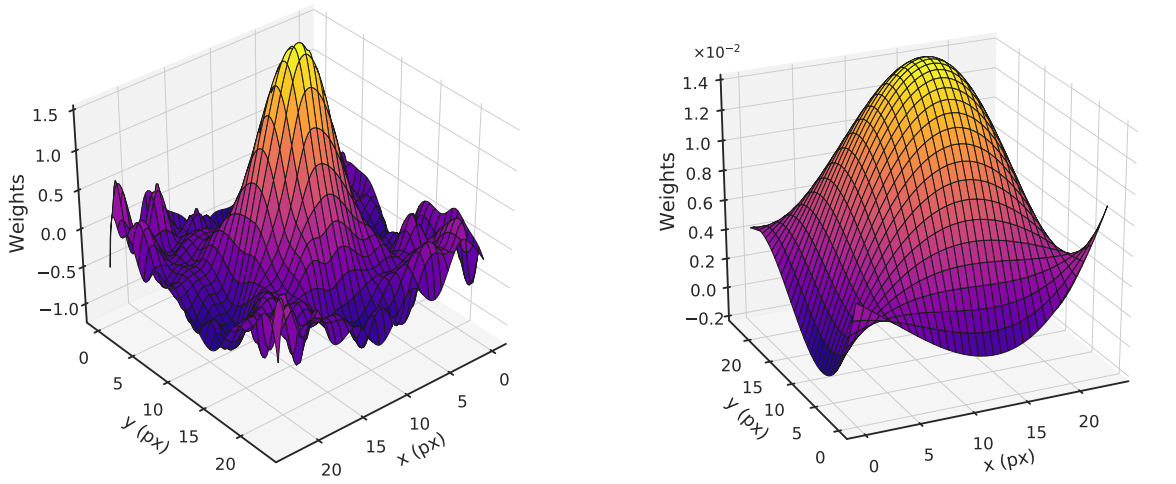
Fig. 4.24a and 4.25a illustrate the PSF the simple model learned in 2D and 3D, respectively. In the 3d representation it is noticeable that the "foothills" of the Gaussian peak in the middle are wavier than the previous ones. The reason for this can be that the noise pixels close to the center become more influential as the lattice spacing increases, i.e. PSF overlaps decrease, and are reflected in the learned PSF due to their contribution to the fitting.



(a) PSF learned by the simple NN

(b)

Figure 4.24: The results of the training for FOM $= 2, 12$, i.e. lattice spacing $d = 7\,\text{px}$ presented in 2D. In (a) the PSF learned by the simple NN is shown. In (b) the averaged image over all 512 weights that the hidden neurons in the enhanced NN learned is given.



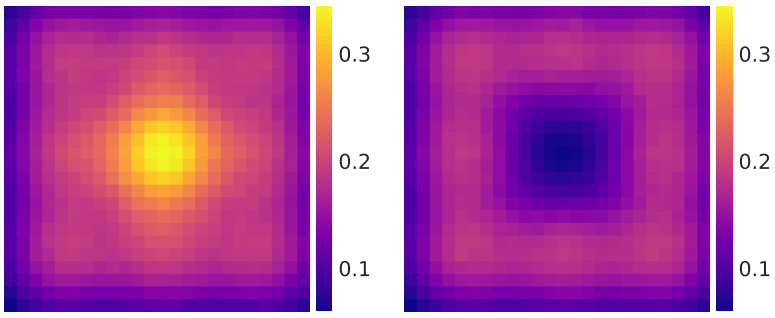
(a) The PSF that is learned by the simple NN

(b) Averaged weights of the 512 hidden neurons in the enhanced NN

Figure 4.25: 3D representation of the results presented in Fig. 4.24. The x-y-plane is the examined lattice cite region. The z-direction shows the weight a pixel at (x,y) carries.

4.2.5 FOM = 2.42

Lastly, the lattice spacing $d = 8\,\text{px}$ is examined with the simulated images whose averages are shown in Fig. 4.26. The best results are expected from this setting because it should be easier for the classifier to distinguish between neighboring sites.

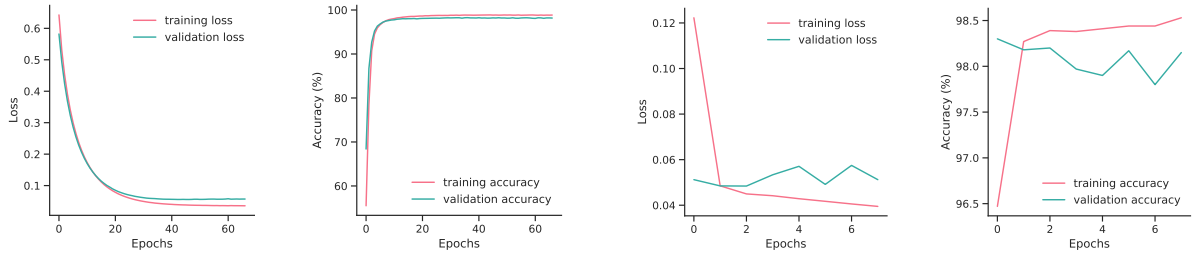


(a) Center is occupied (b) Center is unoccupied

Figure 4.26: Averaged images over $50 \cdot 10^3$ simulated images with FOM = 2.42, i.e. lattice spacing $d = 8\,\text{px}$, whose central lattice site is (a) occupied, (b) unoccupied.

Training the simple model with the parameters $\eta = 0.01$, batch size $b = 128$ and patience = 20 gives the result 98.0% for the accuracy and 0.0558 for the loss. Training the enhanced model with $\eta = 0.001$, batch size $b = 128$ and patience = 5 yields 98.19% for the accuracy and 0,048 for the loss. This means an improvement of 0.19% in the classification accuracy. This shows that the predicted reason for the small improvement in the accuracy of the previous case is not coorect. Before coming to conclusions like this, it is reasonable to take the stochastic nature of the gradient descent into consideration, which can lead

to very good and less good results for the same set of parameters. Therefore, computing averages over the training and testing results after a certain number of repetition with the same parameters may allow a better estimation of the model success.

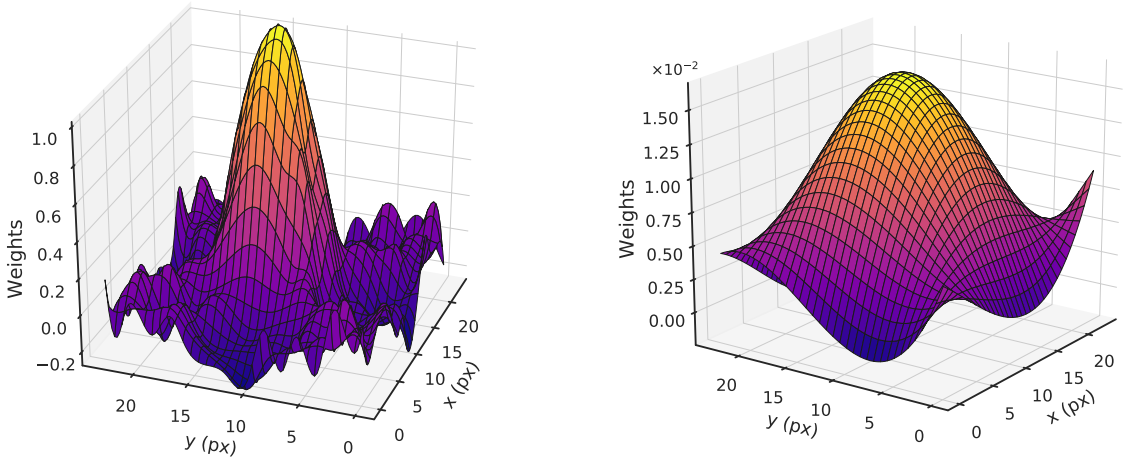


(a) Learning curves of the simple NN

(b) Learning curves of the enhanced NN

Figure 4.27: Training and validation loss and accuracy computed at the end of each epoch while training the (a) simple NN, (b) enhanced NN with 512 hidden neurons for the lattice spacing $d = 8\,\text{px}$.

The 3D representation of the PSF learned by the simple model as well as the averaged weight matrix of the enhanced model are illustrated in Fig. 4.28.



(a) The PSF that is learned by the simple NN

(b) Averaged weights of the 512 hidden neurons in the enhanced NN

Figure 4.28: Figures depicting (a) the average pixel values of a simulated lattice region with a lattice spacing of $d = 8\,\text{px}$ that has an occupied center, (b) the learned PSF of an isolated atom in the center of the given region. The x-y-plane is the examined lattice cite region. The z-direction shows the weight a pixel at (x,y) carries.

4.3 Step 3: CNN - Atom with Nearest Neighbours

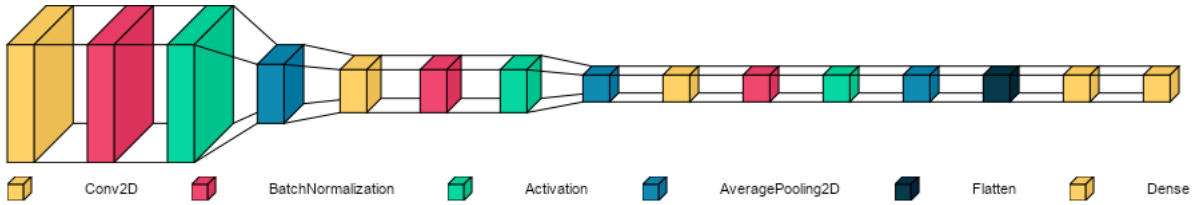


Figure 4.29: The architecture of the 3-layered convolutional neural network employed in the study. Below, the names of the color-coded network layers are given. The first layer consists of one 2D-convolutional, one batch normalization and one activation (ReLu) layer of the same size. After applying 2D-average pooling, the output is transferred to the next layer that consists of the same type of layers in the same order with a reduced size due to the previous pooling. The sizes of the layers depend on the size of the input image, which is a variable that depends on the FOM. An example is shown in Fig. 4.30. The number of filters used in convolutional layers vary depending on the observed FOM, as well. The mostly preferred option is 20-30-40 filters for the 1.-2.-3. convolutional layer, respectively. After the third average pooling, the output is flattened and fed into a (fully connected) dense layer with 16 neurons that are activated by the ReLu function. Lastly, the dense output layer with a single neuron is connected to the previous dense layer and applied the sigmoid activation function. The activation of the last two layers are not shown separately. The sketch is created using the *visualkeras* tool provided by [19]

Due its remarkable performance and advantages over the deep ANNs, as explained in section 3.3.1, CNNs are a suitable candidate for the present image classification task, which are experimented in the third step of this study and reported on in this section. A CNN is composed of four main layer types: convolutional, activation, pooling and dense layer. Since the input images are two dimensional, the convolution operation is also in 2D, shortly denoted by Conv2D. The number of the filters to use in these layers and their sizes are determined in the design of the network architecture. The patterns the filters are to detect are, however, not specified because learning these is a part of the training of a CNN. As activation function in the activation layer, ReLu is used due to its computational speed. The pooling layers can perform average or maximum pooling, which is varied in the experiment. A fully connected layer with multiple neurons is employed at the second to last layer of the network to capture all the extracted features and higher-level abstract representations of the input image to utilize in the classification. A last dense layer consists of a single neuron that decides on the binary label of the image. The neurons in the second to last layer are activated by the ReLu function, whereas a sigmoid function is applied to the last layer's output. The activations used in dense layers are not usually shown as additional activation layers in network sketches of CNNs.

Another layer type that can be employed to improve the training performance is *batch normalization*. It is proposed in the paper [29] and is used in many deep learning applications to exploit its advantages such as enabling the usage of higher learning rates and reducing the need for careful initialization, making the training faster and stabler [51].

Inspired by the network architecture choice in the paper [45] as well as based on common CNN architectures [40], the CNN to be used in this study is designed as illustrated in Fig. 4.29. Overall, 2-and 4-layered CNNs with a similar architecture are tested along the 3-layered CNN and the latter is found to be the most suitable structure.

The training and architecture parameters in this model are the number and sizes of convolutional filters in the three Conv2D layers, the size, stride and type of pooling layers, thee number of neurons in the dense layer before the output layer, the learning rate that is given to the optimizer Adam as a parameter and the size of the training set. For each FOM value that is studied in the previous step, the mentioned parameters are varied for different fixed number of training samples to find the best candidates for making the model efficient. The performance metric used to evaluate the training for each parameter set is the test accuracy, like before. For each FOM, simulated images are created with a size that enables the image frame to contain all atom distributions, such that the nine-lattice-site subregion in which the central atom is observed in presence of its nearest eight neighbours in 2^9 possible configurations, is embedded in the total image without leaving a large space empty. An example of such simulated images is shown in the simulation section 2 in Fig. 2.3c for $d = 8\,\mathrm{px}$. The length of the square matrix in which an image with $d \in \{4, 5, 6, 7, 8\}$ reads $n \in \{26, 28, 30, 32, 34\}$, respectively.

The summary of a training can be monitored on Keras, such that each layer is indicated with its output's shape and the number of weights that are learned in that layer. The training for $d = 8\,\mathrm{px}$ with the best parameters yields the summary in Fig. 4.30. The best parameters read

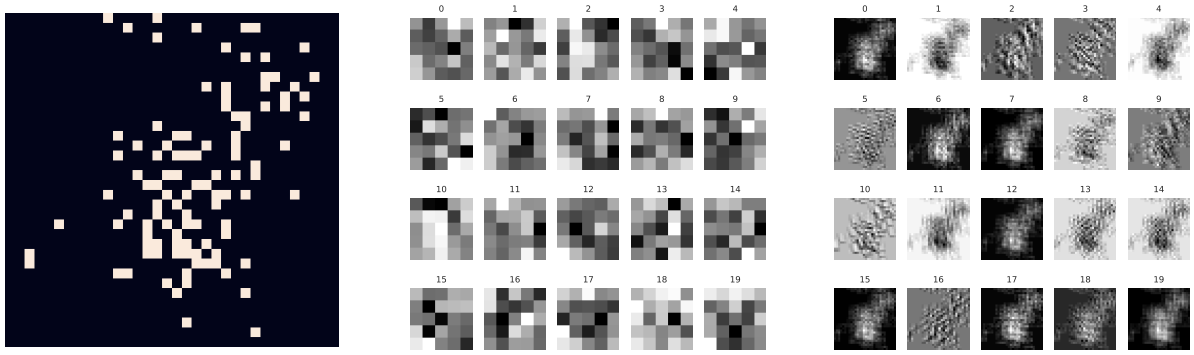
- number of filters in three Conv2D layers: 20,30,40
- filter sizes in in three Conv2D layers: 5,4,3
- pooling types in three pooling layers: average, average, average
- pooling filter sizes in three pooling layers: 1,2,2
- stride: 1
- learning rate: 0.05
- number of hidden neurons in the secon to last dense layer: 16

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 30, 30, 20)        520

batch_normalization (BatchN  (None, 30, 30, 20)        80
ormalization)

activation (Activation)      (None, 30, 30, 20)        0

average_pooling2d (AverageP  (None, 30, 30, 20)        0
ooling2D)

conv2d_1 (Conv2D)            (None, 27, 27, 30)        9630

batch_normalization_1 (Batc  (None, 27, 27, 30)        120
hNormalization)

activation_1 (Activation)    (None, 27, 27, 30)        0

average_pooling2d_1 (Averag  (None, 13, 13, 30)        0
ePooling2D)

conv2d_2 (Conv2D)            (None, 11, 11, 40)        10840

batch_normalization_2 (Batc  (None, 11, 11, 40)        160
hNormalization)

activation_2 (Activation)    (None, 11, 11, 40)        0

average_pooling2d_2 (Averag  (None, 5, 5, 40)          0
ePooling2D)

flatten (Flatten)            (None, 1000)              0

dense (Dense)                (None, 16)                16016

dense_1 (Dense)              (None, 1)                 17

=================================================================
Total params: 37,383
Trainable params: 37,203
Non-trainable params: 180
```

Figure 4.30: The summary of the CNN model. The first column shows the layer type in the same order as the layers are connected, the uppermost layer being the input layer that receives the image to be classified. The second column indicates the resulting tensor shape after each layer. The first dimension of each output denoted by *None* is the variable batch size which does not need to be fixed prior to the training. The second and third dimensions correspond to the height and width of the tensor. The fourth dimension gives the depth of the tensor, which is controlled by the number of filters in convolutional layers. The last column shows the number of parameters learned by the end of each layer. Since activation, pooling and flattening are applications that do not require learning, their entries are zero. The total number of parameters and how many among them are trainable is given below the table. Some parameters are in the non-trainable group, because although they are computed and used during the training, they are not updated using gradient descent, like the mean and standard deviation of the activations used in the batch normalization.

where the filters are squarish and their sizes are given by their width or height. Some of these parameters like the learning rate, pooling filter sizes and the pooling layer types change slightly depending on the FOM value or the training set size. The other parameters prove successful in all cases.

Due to the large number of parameters in sequential convolutional layers (9630, 10840 and 16016, see Fig. 4.30), accessing the filters of the first convolutional layer with 520 filters and visualizing them is practically easier. In Fig. 4.31, the 20 5x5 filters learned by the first convolutional layer and the result of their application on a raw binarized image from the category $d = 8\,\text{px}$, i.e. feature maps, are presented. As the very first level of abstractions in the deep neural network, the feature maps in Fig. 4.31c are still quite similar to the input image. In the deeper layers, they are expected to become hardly recognizable.



(a) 32x32 sized image to be classified (Input)

(b) 20 5x5 filters learned by the first convolutional layer

(c) 20 30x30 feature maps of the given filters on the given image

Figure 4.31: The 20 filters (b) of the first convolutional layer of the CNN are learned in the training with 32x32 sized images of lattice subregions containing nine neighboring lattice sites with a spacing of 8 px, like the image in (a). The trained model is applied to the image in (a) to predict its central site occupation. The feature maps of this image after applying each of the shown filters are visualized in (c).

Since training a model with the same set of parameters can yield slightly different results at each attempt, the test accuracy is averaged over several trainings and presented with the associated error bar, which is determined by the maximal deviation seen from the average value. These results for different FOMs expressed by the corresponding lattice spacing d can be found in Fig. 4.32.

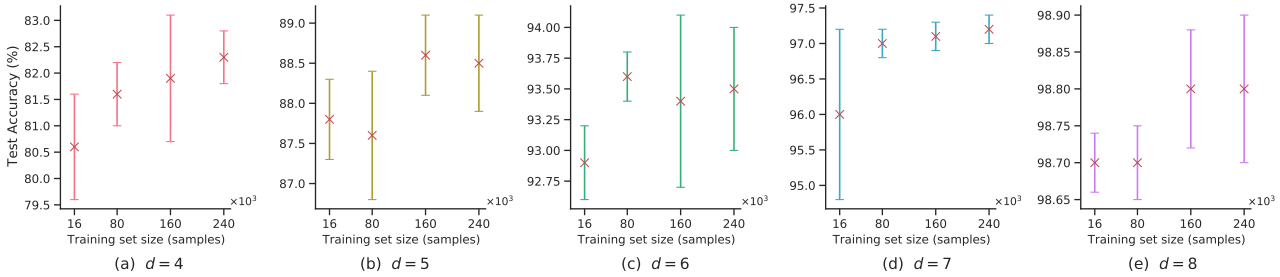


Figure 4.32: Test accuracy in percent dependent on the training set size, i.e. number of training samples for different lattice spacings $d \in \{4, 5, 6, 7, 8\}$ px. The points marked with a red $\times$ sign are the average values of test accuracy over several trainings (from 4 to 10). The widths of the error bars are determined by the maximal deviation from the average values. The results are interpreted in the text.

Fig. 4.32 shows that increasing the training set size has a improving effect on the test accuracy, which

means a higher training performance, since the network can perform well on the unseen test data if it learned to generalize. However, this improvement does not seem to exceed 3%. Due to the limited GPU capacity, training sets of a larger size could not be used to train the model. However, the trend that is seen in each subfigure indicates a progress that may continue up to very large number of training samples, where a convergence of the accuracy is anticipated [45]. Moreover, the greatest overall improvement of the accuracy upon increasing the training set size can be seen in (a), i.e. for the smallest spacing, even though its largest value is under the lowest value of the accuracies in other cases. This makes it more tempting to study this limiting case further by using more data.

5 Summary and Outlook

The main purpose of this thesis was to experiment with and compare certain basic feedforward neural network architectures that may be a better alternative to usual threshold-based image reconstruction techniques that aim to achieve a high-accuracy lattice site detection in limiting cases that necessitate a smaller lattice spacing or lead to a broadening of the effective PSF of the imaging system. Both make it difficult to classify a single lattice site as occupied or unoccupied, since in both cases, the density distributions (PSF) of some neighbouring atoms extend to adjacent lattice sites, build overlaps and complicate the detection of an atom or its absence in a given lattice site. In order to employ supervised learning, labelled images of 3x3-segment lattice subregions were simulated. The system's figure of merit (FOM), which is the lattice spacing d divided by the width of the effective PSF σ, is used as a parameter in trainings by keeping the representative PSF width $\sigma = 3.3$px constant and varying d. For $d \in \{4, 5, 6, 7, 8\}$px, FOM $\in \{1.21, 1.52, 1.82, 2.12, 2.42\}$ could be realized. The results can be used to make predictions about any combination of lattice spacing and effective PSF width that yields one of these ratios.

Three types of neural networks, all of which were given the task to classify the central lattice site in a given image, i.e. assign a label $y = 0$ or $y = 1$ to them, are built and tested. The first model was a simple neural network that consists of an input layer with neurons equal to the number of image pixels, a hidden layer and an output layer, both of which have a single neuron. Trained on $80 \cdot 10^3$ simulated images of an isolated atom in the central lattice site and of an empty central lattice site, the model was able to learn the PSF of the system, which corresponded to its weight matrix. When applied on test images, the model fitted them with the PSF it learned in the training and achieved a classification accuracy of 99.81%. Mathematically, it was shown that this model behaves no differently than a threshold-based classifier. Therefore, the next two models were compared with this model in terms of the progress they make compared to a classical reconstruction method.

The simple network as well as its enhanced version that contains 512 hidden neurons were trained on $80 \cdot 10^3$ images that include neighbouring atoms with a 50% chance for each of the eight sites for the above mentioned five values of FOM. The highest classification accuracy the enhanced model achieved for FOM = 1.21 was 80.95%, about 0.19% better than the simple model. Up to FOM = 2.42, the accuracy both models achieved kept increasing until it reached 98.15% for the enhanced model, again 0.19% better than the simple model. The training parameters η (learning rate), b (batch size) and patience used in these stages were varied between 0.001 and 0.005, 64 and 300, 4 and 20, respectively. It was noticed that the models were flexible enough to yield similarly good results for different sets of parameters. Lastly, a 3-layered convolutional neural network (CNN) was employed. It consisted of 2D-convolutional, batch normalization, activation, pooling and dense layers. The last dense layer had a single neuron that produced the label of a given image, while the deep layers were responsible for extracting hidden features of the image, regularizing and accelerating the training and protecting the model from overfitting. This model was trained on data sets of four different sizes from $16 \cdot 10^3$ to $240 \cdot 10^3$ for the five values of FOM. The accuracy in dependency of the number of training samples can be viewed in Fig. 4.32. The

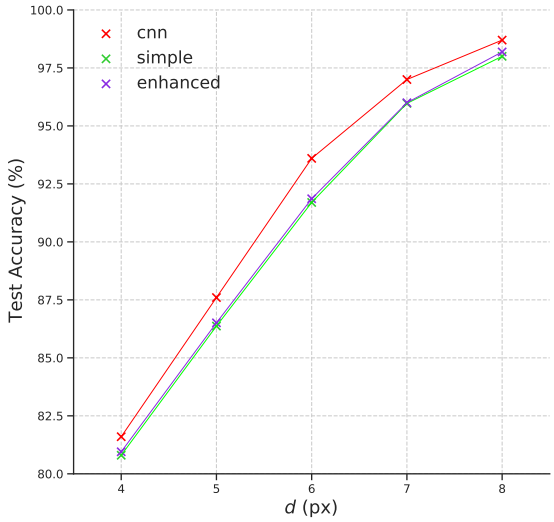


Figure 5.1: Comparison of the three implemented neural networks with regard to their test accuracies for the five lattice spacing values d given in pixels. The associated FOM values are 1.21, 1.52, 1.82, 2.12 and 2.42. For each model, the best training parameters specific to it are used. The training and test set size are $80 \cdot 10^3$ and $100 \cdot 10^3$, respectively. Biggest improvement by switching from simple ANN to CNN is seen for $d = 6\,\text{px}$, i.e. FOM=1.82, with $\approx 1.9\%$. For larger training sets and smaller d, the accuracy of CNN increases by up to 3%, while the improvement is more limited for larger d.

biggest improvement in the accuracy was observed for FOM = 1.21, rising from 79.5% to nearly 83% by using larger training sets. The accuracy for FOM = 2.42, on the other hand, improved less than 0.25%. Nevertheless, it seems possible for these values to improve more, if trained with even larger data sets. The architectural parameters of the CNN were 20,30,40 filters with the sizes 5,4,3 in the three sequential convolutional layers, average pooling filters with the sizes 1,2,2 that were applied with a stride of 1, 16 hidden neurons in the second to last dense layer and a learning rate of 0.05.

A comparison of the three models for the same size of the training set are shown in Fig. 5.1. The CNN proved to be the most accurate of the three networks. In particular, for FOM = 1.52, 1.82 and 2.12, it showed a noticeable improvement of classification accuracy around 1.4%, 1.9% and 1.4%, respectively. According to these results, CNN has a promising potential to reach high accuracies for small FOM ratios that represent the technically challenging limits, e.g. when the neighbouring atoms are separated by much less than the diffraction limit of the imaging system. Its architecture is simple, training is more automatized and the choice of parameters allows flexibility. By using larger training sets and performing more rigorous simulations that involve many relevant parameters the real images would have, the proposed CNN can be improved and eventually be used on the real images in the future.

List of Figures

List of Tables

References

[1] Photo by maria mileta on pexels, 08.10.2022.

[2] scipy.interpolate.bisplrep — scipy v1.9.1 manual, 27.08.2022.

[3] Nn svg, 28.02.2022.

[4] Afshin Gholamy, Vladik Kreinovich, and Olga Kosheleva. Why 70/30 or 80/20 relation between training and testing sets: A pedagogical explanation. 2018.

[5] Ergün Akgün and Metin Demir. Modeling course achievements of elementary education teacher candidates with artificial neural networks. *International Journal of Assessment Tools in Education*, pages 491–509, 2018.

[6] Ethem Alpaydın. *Introduction to machine learning*. Adaptive computation and machine learning. MIT Press, Cambridge, Massachusetts and London, England, third edition edition, 2014.

[7] Deepinder Jot Singh Aulakh, Steven B. Beale, and Jon G. Pharoah. A generalized framework for unsupervised learning and data recovery in computational fluid dynamics using discretized loss functions. *Physics of Fluids*, 34(7):077111, 2022.

[8] M. BARANOV. Theoretical progress in many-body physics with ultracold dipolar gases. *Physics Reports*, 464(3):71–111, 2008.

[9] Andrea Bergschneider, Vincent M. Klinkhamer, Jan Hendrik Becher, Ralf Klemt, Gerhard Zürn, Philipp M. Preiss, and Selim Jochim. Spin-resolved single-atom imaging of li6 in free space. *Physical Review A*, 97(6), 2018.

[10] Immanuel Bloch and Markus Greiner. The superfluid-to-mott insulator transition and the birth of experimental quantum simulation. *Nature Reviews Physics*, 2022.

[11] B. Bylicka, D. Chruściński, and S. Maniscalco. Non-markovianity and reservoir memory of quantum channels: a quantum information theory perspective. *Scientific reports*, 4:5720, 2014.

[12] Giuseppe Carleo, Ignacio Cirac, Kyle Cranmer, Laurent Daudet, Maria Schuld, Naftali Tishby, Leslie Vogt-Maranto, and Lenka Zdeborová. Machine learning and the physical sciences. *Reviews of Modern Physics*, 91(4), 2019.

[13] R. A. Carollo, D. C. Aveline, B. Rhyno, S. Vishveshwara, C. Lannert, J. D. Murphree, E. R. Elliott, J. R. Williams, R. J. Thompson, and N. Lundblad. Observation of ultracold atomic bubbles in orbital microgravity. *Nature*, 606(7913):281–286, 2022.

[14] Coursera. Machine learning — coursera, 09.10.2022.

[15] Arden Dertat. Applied deep learning - part 4: Convolutional neural networks. *Towards Data Science*, 08.11.2017.

[16] E. T. Owen. Towards an imaging lattice for magnetically trapped atoms. 2017.

[17] G. J. A. Edge, R. Anderson, D. Jervis, D. C. McKay, R. Day, S. Trotzky, and J. H. Thywissen. Imaging and addressing of individual fermionic atoms in an optical lattice. *Physical Review A*, 92(6), 2015.

[18] A. Einstein. Quantentheorie des einatomigen idealen gases. 1924.

[19] Paul Gavrikov. visualkeras, 2020.

[20] Aurélien. Géron. *Hands-on machine learning with Scikit-Learn, Keras and TensorFlow: Concepts, tools, and techniques to build intelligent systems.* O'Reilly Media, Sebastopol, 2e ed. edition, 2019.

[21] Andrew Glassner. *Deep Learning: A Visual Approach.* No Starch Press, Erscheinungsort nicht ermittelbar, 2021.

[22] Aldo Glielmo, Brooke E. Husic, Alex Rodriguez, Cecilia Clementi, Frank Noé, and Alessandro Laio. Unsupervised learning methods for molecular simulation data. *Chemical reviews*, 121(16):9722–9758, 2021.

[23] Antonio Gulli, Amita Kapoor, and Sujit Pal. *Deep learning with TensorFlow 2 and keras: Regression, ConvNets, GANs, RNNs, NLP, and more with TensorFlow 2 and the Keras API, second edition.* Packt Publishing and iG Publishing, Inc, Birmingham and [Singapur], 2nd ed. edition, 2020.

[24] Mingyang Guo, Fabian Böttcher, Jens Hertkorn, Jan-Niklas Schmidt, Matthias Wenzel, Hans Peter Büchler, Tim Langen, and Tilman Pfau. The low-energy goldstone mode in a trapped dipolar supersolid. *Nature*, 574(7778):386–389, 2019.

[25] Yudan Guo, Ronen M. Kroeze, Brendan P. Marsh, Sarang Gopalakrishnan, Jonathan Keeling, and Benjamin L. Lev. An optical lattice with sound. *Nature*, 599(7884):211–215, 2021.

[26] Elmar Haller, James Hudson, Andrew Kelly, Dylan A. Cotta, Bruno Peaudecerf, Graham D. Bruce, and Stefan Kuhr. Single-atom imaging of fermions in a quantum-gas microscope. *Nature Physics*, 11(9):738–742, 2015.

[27] Ken Holmes, Phil Harris, and Marcus Elkington. *Clark's essential physics in imaging for radiographers*. Clark's companion essential guides. CRC Press, Boca Raton, second edition edition, 2021.

[28] I.A Basheer and M Hajmeer. Artificial neural networks: fundamentals, computing, design, and application. *Journal of Microbiological Methods*, 43(1):3–31, 2000.

[29] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift.

[30] D. Jaksch, C. Bruder, J. I. Cirac, C. W. Gardiner, and P. Zoller. Cold bosonic atoms in optical lattices. *Physical Review Letters*, 81(15):3108–3111, 1998.

[31] L. P. Kaelbling, M. L. Littman, and A. W. Moore. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4:237–285, 1996.

[32] Ku Chhaya A. Khanzode and Ravindra D. Sarode. Advantages and disadvantages of artificial intelligence and machine learning: A literature review. *International Journal of Library & Information Science (IJLIS)*, 9(1):3, 2020.

[33] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization.

[34] Ulrich Kubitscheck, editor. *Fluorescence microscopy: From principles to biological applications*. Wiley-VCH, Weinheim, Germany, second edition edition, 2017.

[35] Siddharth Krishna Kumar. On weight initialization in deep neural networks.

[36] Liang Liu. Exploring the universe with matter waves. *Nature*, 562(7727):351–352, 2018.

[37] Thomas Maier. *Interactions in a quantum gas of dysprosium atoms*. Dissertation.

[38] Martin Miranda, Ryotaro Inoue, Naoki Tambo, and Mikio Kozuma. Site-resolved imaging of a bosonic mott insulator using ytterbium atoms. *Physical Review A*, 96(4), 2017.

[39] Ahmed Omran, Martin Boll, Timon A. Hilker, Katharina Kleinlein, Guillaume Salomon, Immanuel Bloch, and Christian Gross. Microscopic observation of pauli blocking in degenerate fermionic lattice gases. *Physical Review Letters*, 115(26):263001, 2015.

[40] Keiron O'Shea and Ryan Nash. An introduction to convolutional neural networks. *ArXiv e-prints*, 2015.

[41] A.Pasumpon Pandian. *Computer Networks, Big Data and IoT*, volume 66. Springer Singapore, [S.l.], 2021.

[42] Maxwell F. Parsons, Florian Huber, Anton Mazurenko, Christie S. Chiu, Widagdo Setiawan, Katherine Wooley-Brown, Sebastian Blatt, and Markus Greiner. Site-resolved imaging of fermionic ˆ6li in an optical lattice. *Physical Review Letters*, 114(21):213002, 2015.

[43] Daniel Slater Gianmario Spacagna Ivan Vasilev Valentino Zocca Peter Roelants. *Python Deep Learning - Second Edition: Exploring deep learning techniques and neural network architectures with PyTorch, Keras, and TensorFlow, 2nd Edition*. Packt Publishing, 2019.

[44] N. Petersen, M. Trümper, and P. Windpassinger. Spectroscopy of the 1001-nm transition in atomic dysprosium. *Physical Review A*, 101(4), 2020.

[45] Lewis Picard, Manfred Mark, Francesca Ferlaino, and Rick Bijnen. Deep learning-assisted classification of site-resolved quantum gas microscope images, 2019.

[46] Nicholas O. Ralph, Ray P. Norris, Gu Fang, Laurence A. F. Park, Timothy J. Galvin, Matthew J. Alger, Heinz Andernach, Chris Lintott, Lawrence Rudnick, Stanislav Shabala, and O. Ivy Wong. Radio galaxy zoo: Unsupervised clustering of convolutionally auto-encoded radio-astronomical images. *Publications of the Astronomical Society of the Pacific*, 131(1004):108011, 2019.

[47] S. L. Rolston and W. D. Phillips. Nonlinear and quantum atom optics. *Nature*, 416(6877):219–224, 2002.

[48] Sebastian Ruder. An overview of gradient descent optimization algorithms.

[49] S Hensler, A Greiner, J Stuhler, and T Pfau. Depolarisation cooling of an atomic cloud. *Europhysics Letters (EPL)*, 71(6):918–924, 2005.

[50] Samer L. Hijazi, Rishi Kumar, and Chris Rowen. Using convolutional neural networks for image recognition by. 2015.

[51] Shibani Santurkar, Dimitris Tsipras, Andrew Ilyas, and Aleksander Madry. How does batch normalization help optimization. *Advances in neural information processing systems*, 31, 2018.

[52] R. Sathya and Annamma Abraham. Comparison of supervised and unsupervised learning algorithms for pattern classification. *International Journal of Advanced Research in Artificial Intelligence*, 2(2), 2013.

[53] Jan-Niklas Schmidt. *Density fluctuations in a dipolar quantum gas - from superfluids to supersolids*. Dissertation, Verlag Dr. Hut, München.

[54] Siddharth Sharma, Simone Sharma, and Anidhya Athaiya. Activation functions in neural networks. *International Journal of Engineering Applied Sciences and Technology*, 04(12):310–316, 2020.

[55] T Lahaye, C Menotti, L Santos, M Lewenstein, and T Pfau. The physics of dipolar bosonic quantum gases. *Reports on Progress in Physics*, 72(12):126401, 2009.

[56] Keras Team. Keras documentation: About keras, 29.09.2022.

[57] C. Trefzger, C. Menotti, B. Capogrosso-Sansone, and M. Lewenstein. Ultracold dipolar gases in optical lattices. *Journal of Physics B: Atomic, Molecular and Optical Physics*, 44(19):193001, 2011.

[58] Lei Wang. Discovering phase transitions with unsupervised learning. *Physical Review B*, 94(19), 2016.

[59] Jianxin Wu. Introduction to convolutional neural networks. 2017.

[60] Jim Y.F. Yam and Tommy W.S. Chow. A weight initialization method for improving training speed in feedforward neural network. *Neurocomputing*, 30(1-4):219–232, 2000.

[61] Xinghuo Yu, M. O. Efe, and O. Kaynak. A general backpropagation algorithm for feedforward neural networks learning. *IEEE Transactions on Neural Networks*, 13(1):251–254, 2002.

[62] Iffat Zafar. *Hands-on convolutional neural networks with TensorFlow: Solve computer vision problems with modeling in TensorFlow and Python*. Packt Publishing, Birmingham, UK, 1 edition, 2018.

[63] Leifeng Zhang, Yanming Che, Jibiao Wang, and Qijin Chen. Exotic superfluidity and pairing phenomena in atomic fermi gases in mixed dimensions. *Scientific reports*, 7(1):12948, 2017.