Eberhard Karls Universität Tübingen

Mathematisch-Naturwissenschaftliche Fakultät

Institut für Angewandte Physik

BACHELOR THESIS

Alina Pleli

August 6, 2020

CLASSIFICATION AND ANALYSIS OF 2D GRAZING-INCIDENCE WIDE-ANGLE X-RAY SCATTERING DATA USING DEEP LEARNING

Supervisor:
Prof. Dr. Frank Schreiber

Contents

Eigenständigkeitserklärung

Hiermit erkläre ich, Alina Pleli,

- dass ich diese Bachelorarbeit selbständig verfasst und keine anderen als die angegebenen Hilfsmittel und Quellen benutzt habe.
- dass alle wörtlich oder sinngemäß aus anderen Werken übernommenen Aussagen als solche gekennzeichnet sind.

Diese Arbeit wurde weder vollständig, noch in wesentlichen Teilen bereits veröffentlicht. Das in Dateiform eingereichte Exemplar stimmt mit den eingereichten gebundenen Exemplaren überein.

Alina Pleli

Tübingen, 06. August 2020

Abstract

The purpose of this thesis is to analyze grazing-incidence wide-angle X-ray scattering (GIWAXS) diffraction patterns of perovskites thin films with deep learning models. Perovskites thin films are used for perovskites solar cells (PSCs). They promise high energy conversion efficiencies at a low cost of cell production. The growth of perovskites thin films depends on different parameters. This growth can be observed with real-time GIWAXS experiments, however the analysis of such experiments is very labour-intensive when done manually. In this thesis, a first step towards the automated analysis of these GIWAXS measurements is made through the creation of two neural network models, one based on ResNet for classification and one based on UNet for segmentation. The classification model is designed to separate diffraction patterns based on the type of Bragg peaks. Some domains of the thin films included preferred orientations, while others do not and as such the data is divided into three classes, one for images with preferred orientation, one without, and a third class for images of thin films both with and without preferred orientation. During training, the model learns to differentiate between these categories by receiving many meaningful examples from different pictures with the correct classification. The results show that a model trained on simulated images can accurately classify other simulated images but its accuracy on experimental data is only 35%. By training a model on experimental data, this accuracy is improved greatly to a value of 98%. In addition to the classification model, a segmentation model to automatically recognize and locate rings or diffraction spots in the images was trained. The network is designed to predict the respective features. As for the classification model, the segmentation model shows greater accuracy when it is trained with experimental images instead of simulated ones. However, the segmentation accuracy is not yet high enough for practical applications and further work is necessary to improve the performance.

Kurzzusammenfassung

Das Ziel dieser Arbeit ist es, Beugungsmuster von Weitwinkel-Röntgenstreuung bei streifendem Einfall (engl. grazing-incidence wide-angle X-ray scattering, GIWAXS) an Perowskit-Dünnschichten mithilfe von Deep-Learning-Modellen zu analysieren. Perowskit-Dünnfilme werden für Perowskit-Solarzellen verwendet. Sie versprechen hohe Wirkungsgrade der Energieumwandlung bei gleichzeitig niedrigen Kosten der Zellproduktion. Im Wesentlichen hängt das Wachstum der Perowskit-Dünnfilme von verschiedenen Parametern ab. Dieses Wachstum kann mit GIWAXS-Experimenten in Echtzeit beobachtet werden, jedoch ist die Analyse solcher Experimente bei manueller Durchführung sehr arbeitsintensiv. In dieser Arbeit wird ein erster Schritt in Richtung einer automatisierten Analyse von GIWAXS-Messungen mit zwei neuronalen Netzen getan, eines basierend auf dem ResNet zur Klassifizierung und eines basierend auf dem UNet zur Segmentierung. Das Klassifizierungsmodell ist darauf ausgelegt, Beugungsmuster auf der Grundlage des Typs der Bragg-Peaks zu trennen. Einige Dünnfilme enthielten bevorzugte Kristallorientierungen, während andere dies nicht taten. Dadurch konnten die Daten in drei Klassen unterteilt werden, eine für Bilder von Dünnfilmen mit bevorzugter Orientierung, eine ohne und eine dritte Klasse für Bilder sowohl mit als auch ohne bevorzugte Orientierung. Während des Trainings lernt das neuronale Netz, zwischen diesen Kategorien zu unterscheiden, indem es viele aussagekräftige Beispiele von verschiedenen Bildern mit der richtigen Klassifizierung erhält. Die Ergebnisse zeigen, dass ein neuronales Netz, das mit simulierten Bildern trainiert wird, andere simulierte Bilder genau klassifizieren kann, die Genauigkeit auf experimentellen Daten liegt allerdings nur bei 35%. Wenn ein neuronales Netz mit experimentellen Daten trainiert wird, verbessert sich diese Genauigkeit erheblich auf einen Wert von 98%. Zusätzlich zum Klassifikationsmodell wurde ein Segmentierungsmodell zur automatischen Erkennung und Lokalisierung von Ringen oder Beugungsflecken in den Bildern trainiert. Das neuronale Netz ist darauf ausgelegt, die jeweiligen Merkmale vorherzusagen. Ähnlich wie beim Klassifizierungsmodell, zeigt das Segmentierungsmodell eine höhere Genauigkeit, wenn es statt mit simulierten Bildern mit experimentellen Bildern trainiert wird. Die Segmentierungsgenauigkeit ist jedoch noch nicht hoch genug für praktische Anwendungen, und es sind weitere Arbeiten erforderlich, um die Leistung zu verbessern.

Acknowledgments

I would like to thank Prof. Dr. Frank Schreiber for giving me the opportunity to write my bachelor thesis on this exciting topic, for his support during my entire bachelor period and his helpful feedback. I would like to thank Vladimir Starostin for his support, patience, time and effort. I am very grateful for the opportunity to learn from him. I want to thank Alessandro Greco for his immense help with the linguistic revision of this thesis. Further, I would like to thank Dr. Alexander Hinderhofer, who was always available for questions, for his help in planning the project. Lastly, I would like to thank my family and friends for their support during my bachelor thesis.

1 Introduction

X-ray scattering is a widely used technique for the structural investigation of various materials on a molecular scale. This thesis aims to analyze grazing-incidence wide-angle X-ray scattering (GIWAXS) patterns of perovskites thin films with deep learning (DL) methods. The field of machine learning (ML) and deep learning (DL) have progressed quickly in the past decades. Today ML is a promising field with many practical applications and research topics [1]. Modern ML systems are characterized by a wide range of complex tasks, from the recognition of objects in images [2] and the recognition of natural languages [3], to the processing of speech signals [4]. These impressive achievements of ML systems, especially the DL models, demonstrate the innovative nature of this technology, which could potentially have a huge impact beyond the scientific research, and could bring about significant changes in industry and society. However, although these models are able to achieve remarkable accuracy in prediction, their interlaced non-linear structure means they lack transparency, i.e. it is not obvious exactly what kind of information in the input data effectively leads them to their decisions, and thus a detailed analysis of the DL model is necessary in order to identify possible improvements. The use of DL in scattering physics is increasing and has already produced promising results, some of which will be presented in the following section.

1.1 Analysis of scattering data with deep learning techniques

Regarding X-ray scattering experiments, several approaches using ML have been developed in order to categorize features in X-ray scattering data [5, 6, 7], such as the classification of X-ray scattering patterns [8] and thin film structure identification [9]. In addition to that a fast fitting of reflectivity data of growing thin films using neural networks was employed [10]. Liu et al. have designed a classification scheme for the categorization of several combinations of 3D nanoparticle lattice orientations [9]. ML has also been applied to a variety of other X-ray experiments, such as in the detection of variation in nanoscale grids [11] and the categorization of 3D nanoparticles on the basis of X-ray absorption spectroscopy measurements [12].
In this thesis we will focus on GIWAXS diffraction patterns of perovskite thin films. Since the data of the perovskite thin films are only used as a test case for the analysis of X-ray scattering data, the background of their advantageous optoelectronic properties is not discussed in this thesis. GIWAXS diffraction patterns contain essential information about the crystal structures of the measured samples. The ever increasing data acquisition rates prompted the need to develop automated tools for any type of preliminary analysis of measured data. Deep learning and especially computer

vision tools, are expected to become indispensable in the coming era of automated diffraction pattern analysis. These tools could help to adjust experiments based on the measurements in real time, to filter the data before the analysis, or even to classify the measured material. There are different general approaches that can be used, when solving automated analysis problems. One of the fundamental constraints on all the supervised machine learning methods is the huge amount of data which needs to be pre-analyzed (labeled) in a specific way to serve as training data. In most cases in scattering physics, it is hard to satisfy this condition for various reasons: the analysis of the data is time-consuming and there is not enough experimental data for certain important cases. Additionally, the established analysis methods may not be suitable for machine learning because they aim to solve a scientific task rather than prepare data for the machine learning algorithms. One possible solution to this issue is the use of simulated data. This approach could theoretically solve most of the problems mentioned above as well as accelerate the development of this field. However, it also has certain limitations. Some of these limitations will be discussed in later chapters. In this thesis the results of classification and segmentation of GIWAXS diffraction patterns of perovskite thin films using DL are presented. This may be a useful tool for the complex physical analysis of these GIWAXS images, which allows a structural analysis of thin films.

1.2 Structure of this thesis

As an introduction to the topic of ML, chapter 2 compares machine learning and optimization. Additionally, the working principle of an artificial neural network is described. In chapter 3 the analyzed GIWAXS images of perovskites thin films are introduced. For this purpose, the composition of perovskites thin films, the experimental method, and the basic physics behind it are briefly discussed. The applied methods and models for the classification and segmentation of the scattering data using deep learning with the respective results are described and discussed in chapter 4. In chapter 5, a conclusion concerning the neural network performance and an outlook on performance improvements of deep learning strategies for the analysis of GIWAXS data is presented.

2 Machine learning background

ML algorithms can automatically improve their performance through experience. The mathematical model is based on sample data ("training data"). The goal is to enable the algorithms to react to situations in the same way humans do, or at least similarly [13]. As shown in Figure 2.1, DL is a subfield of ML, and artificial intelligence (AI) refers to the general field of algorithms which behave intelligently [14].

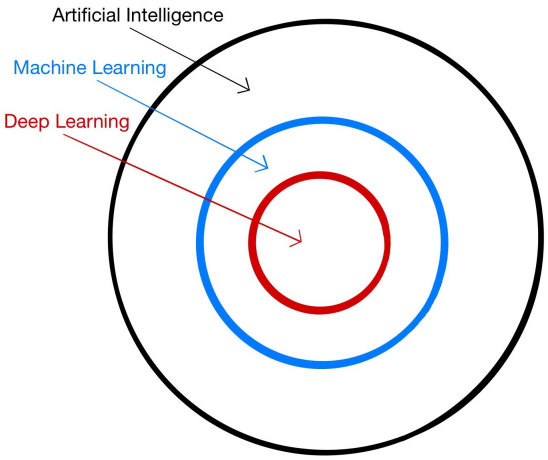

Figure 2.1: Visualization of the distinction between AI, ML and DL

ML is one of many methods that can be used to produce AI. In the following section the basics of machine learning and deep learning will be described.

2.1 Introduction to machine learning and deep learning

Machine learning is a technique used to produce AI. It can be interpreted as enabling computer systems to learn by themselves.
One useful definition of machine learning from Tom Mitchell is [15]:

"A computer program is said to **learn** from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with

experience E."

Machine Learning can be split into two main approaches, supervised and unsupervised learning:

- **Supervised learning:** The program is given the input and the required output. A popular example is the MNIST dataset for handwritten digit recognition. It is a database of labeled handwritten digits, with separate training and test sets from about 250 different writers. It contains the pictures of different digits with corresponding labels of what is shown in the image. Figure 2.2 shows 100 handwriting sample images from the MNIST dataset.

Figure 2.2: 100 samples of digits from the MNIST data set

  The goal is to construct a map between those pictures and the appropriate labels. This approach will be examined in greater detail later in this thesis.

- **Unsupervised learning:** The program is given only the input. It does not provide the program the correct output and the data is unlabeled, so that the algorithm must find relationships between the input data without being trained, and try to label the data itself. The most common tasks of this form are representation learning and clustering. This approach is not used in the thesis, therefore it will not be discussed further at this point.

If you apply the definition to the handwriting recognition learning problem, the **task T** is to recognize and classify handwritten digits in the images. The **performance measure P** is the

percentage of correctly classified digits and the **training experience E** is a dataset of handwritten digits with the right classification.

In the context of machine learning, one often speaks of **deep learning**. Deep learning, as a subset of machine learning methods (see Figure 2.1), refers to neural network models that have a relatively high number of hidden layers. This will be described in subsection 2.3.1.
Outside of DL, there are many powerful ML algorithms, e.g. the random forest algorithm (supervised learning) for classification or the k-means algorithm (unsupervised learning) for clustering. If one compares a classical ML model with only a few layers and a DL model, it is remarkable that the performance of traditional ML remains better with smaller datasets [16]. After a certain amount of data has been provided, the learning ability of traditional learning algorithms reaches a plateau. In contrast, with deep learning, the learning ability increases with the amount of data, i.e. the performance improves as shown in Figure 2.3. Since a large data input also can avoid underfitting, this relationship demonstrates the advantage of DL algorithms over ML algorithms.

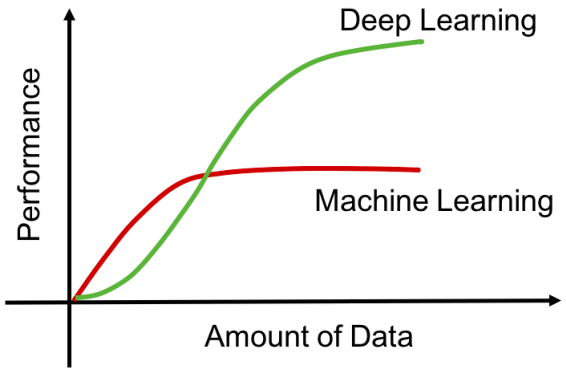

Figure 2.3: The performance as a function of the amount of data of a classical machine learning model with only a few layers and a deep learning model [16].

After this introduction to machine learning, one of the most important challenges in ML will be presented in the next section: the problem of under- and overfitting.

2.1.1 Underfitting and overfitting

A machine learning algorithm receives the training set as an input. By calculating the error of a model on the same input on which the model was trained, the training error is obtained. The model uses the training set to iteratively adjust its internal parameters in order to reduce the training error. After this process it receives an independent validation set. The expected training error should be smaller than or equal to the expected validation error. This error is estimated by measuring the performance of the model on a validation set of example inputs. The validation set is a crucial part

of the ML, as there is no other way to check its performance quality (in contrast to optimization). In theory, the training and validation sets should be representative of the unknown general data set. If a validation set is not representative, then it cannot be used to measure the quality of the prediction in general. Therefore, a lack of representative data can lead to a low quality of the trained model. The following aspects determine how well a machine learning algorithm works:

1. The training error.
2. The difference between the training and the validation error (generalization error).

This corresponds to the two key problems in ML: underfitting and overfitting.

- The model **underfits** (see the plot a) in Figure 2.4), when the training error of the model is not low enough, which means that "(...)[t]he model is incapable of capturing the variability of the data." [17]
- The model **overfits** (see the plot c) in Figure 2.4), when the gap between the training and the validation error is too large. At a certain point the model no longer improves its ability to solve problems and it learns a random regularity, which is part of the training set [17].

The plots in Figure 2.4 show a part of a cosine function using different approaches of interpolation. In the figure on the left a linear function (polynomial of degree 1) is fit to the data, and we see that it cannot capture the curves between the data points, and thus the model underfits the function. The plot (b) in Figure 2.4 shows a well-approximating function for the data, and the image to the right shows an example of overfitting with a 15^{th}-degree polynomial function. While this polynomial approximation almost passes through all the points, its shape is not a cosine function. There are more parameters than training examples. Thus, while it is possible for the polynomial approach to work, the points of the training set can be represented by many other functions, such that there is only a small chance that there is a solution, which generalizes the data well.

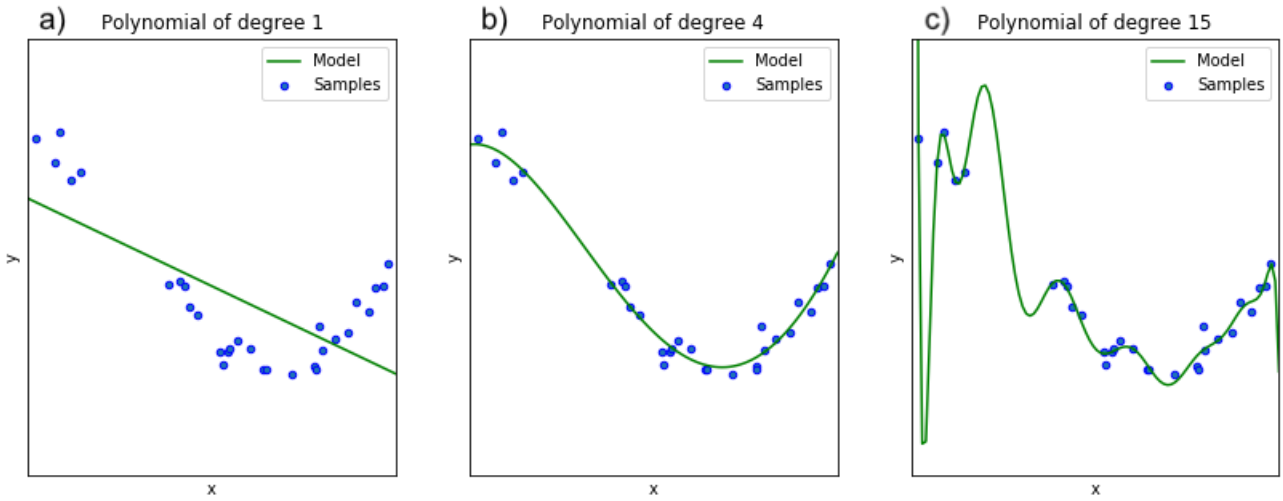

Figure 2.4: a) Underfitting of arbitrary points with a linear function (polynomial of degree 1); b) Good approximation with a polynomial of degree 4; c) Overfitting with a polynomial of degree 15.

With the capacity of a model and the amount and quality of the training data it is possible to control the under- and overfitting. High capacity models can overfit by memorizing special features of the training set, which do not help them on the validation set. Small capacity models may experience problems fitting to the training set. Models with insufficient capacity are unable to solve complex tasks, while models with high capacity can solve complex tasks, but when their capacity is higher than needed to solve the present task, they may overfit [1].

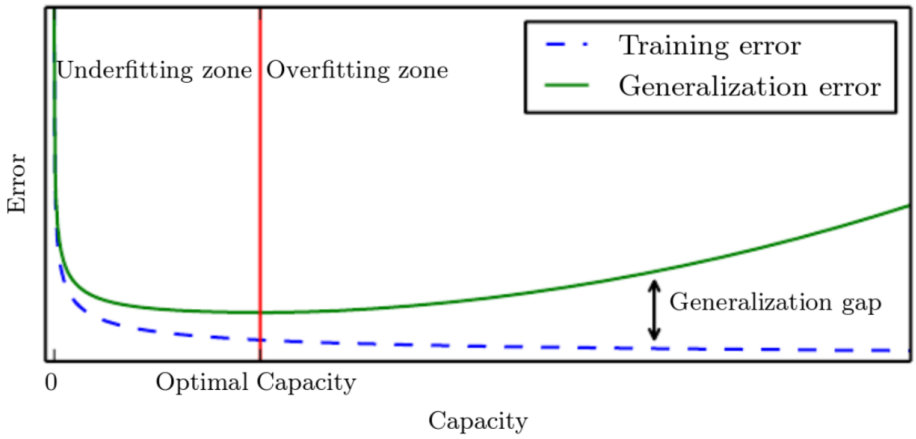

Figure 2.5: The training and generalization error as functions of the capacity [1]. The red line marks the optimal capacity and divides the zone into under- and over-fitting depending on the capacity. The difference between training error and generalization error is called generalization gap.

Figure 2.5 shows a typical relationship between the errors and the capacity of a model. In the case of neural networks, the capacity of the model determines the accuracy of the input-output mapping approximation. The models used in the classification and segmentation in this thesis operate at very high capacities. The amount of data also plays an important role, especially in regards to how well it represents a general data set. This can lead to some limitations of the models, as will be discussed in the results. There are further ways to avoid over- and underfitting, including the so-called **early stopping** which was used in the models in this thesis, and is explained in subsection 4.2.3. In order to provide for a better understanding of ML algorithms, the difference between ML and an optimization will be explained below.

2.1.2 Distinction between machine learning and a simple optimization algorithm

A simple optimization approach focuses on the extraction of model parameters and the production of set of parameters. In contrast, ML seeks to predict the most likely outputs for unseen inputs. The underlying model parameters are not the ultimate goal, but rather only an instrument used to increase the accuracy of the prediction. If one is attempting to minimize the training error, then there is an optimization problem. What distinguishes machine learning from optimization is the generalization error, which in turn differentiates a training error from a validation error [1]. One way of making the differences between an optimization algorithm and a machine learning algorithm clearer is by using **linear regression**. A linear regression by a simple optimization is for example possible using the method of least squares. Here, data points are to be approximated by a function (usually a linear function), such that the squared distances between the data points and the function values are as small as possible. The parameters then result in an optimal function that closely approximates the data. Related to linear regression, building a system which can take a vector $\boldsymbol{x} \in \mathbb{R}^n$ as input and predict the value of a scalar $y \in \mathbb{R}$ as its output, is the aim of a regression problem with ML. The output is defined as a linear function of the input:

$$y_{\text{out}} = \boldsymbol{w}^T \boldsymbol{x} \tag{2.1}$$

where $\boldsymbol{w} \in \mathbb{R}^n$ is a vector of parameters, which control the behaviour of the system, and $\boldsymbol{w}$ specifically is a set of weights that influences how each feature determines the prediction:

- $w_i > 0$: when the feature x_i is increased, the value of the prediction of this feature is increased.
- $w_i < 0$: when the feature x_i is increased, the value of the prediction of this feature is decreased.

If a feature is heavily weighted, the effect on the prediction is large, and if the weight is zero, the feature has no effect in the prediction. When this is applied to the definition of machine learning by Tom Mitchell, the task T is to predict y from x (by Equation 2.1) .
To test the performance of a model, a test set is used with m inputs x with different features i.
The performance of the model can be measured by the mean squared error (MSE) of the error on

m example test inputs:

$$\mathrm{MSE}_{\mathrm{test}} = \frac{1}{m}\sum_i (y_{\mathrm{out}}^{\mathrm{test}} - y^{\mathrm{test}})_i^2 \tag{2.2}$$

y_{out} (from Equation 2.1) is the prediction of the model based on the training set. Equation 2.2 is minimized by solving the gradient in relation to the weights w:

$$\nabla_w \mathrm{MSE} = 0 \tag{2.3}$$

When the training process is completed, a validation is performed. This requires an input set x_{val} with m_{val} events. The MSE is also calculated for the independent validation set.

$$\mathrm{MSE}_{\mathrm{test}} = \frac{1}{m_{\mathrm{val}}}\sum_i (y_{\mathrm{out}}^{\mathrm{val}} - y^{\mathrm{val}})_i^2 \tag{2.4}$$

Here y_{val} are the correct outputs and y_{out}^{val} are the predictions of the algorithm. Establishing the true behaviour of the algorithm using only those images with which the model was trained is not enough. The algorithm repeats the linear regression process until the error is small enough. The stop criterion for this depends on the task. This example shows that an optimization algorithm seeks to find the appropriate parameters for a good model. These parameters are the result of the approximation. An ML algorithm on the other hand focuses on the prediction ability, and this is the result of the approximation using ML. The parameters that led to the best prediction of the model are not of further interest, since it is very difficult to draw conclusions from them, which is certainly one of the disadvantages of ML. Even if the model can handle new data well and delivers good results, it is almost impossible to determine exactly which parameters are responsible for this. As proven in [18] an ML algorithm can approximate almost any function. An approximation is not limited to certain linear or non-linear functions. The theoretical precision of this approximation is determined by the number of nodes in the neural network, which will be discussed in more detail in section 2.2. Before proceeding to these neuronal networks, the following section will discuss various statistical conditions and theorems related to the performance of ML algorithms.

2.1.3 Statistical constraints and theorems of the machine learning approach

Let us assume that there is a model which ideally matches the distribution of a data set X. The model will still have errors in classification. This can occur for several reasons. It is possible that y may also depend on other variables that cannot be obtained from X. Or it may be that the mapping of events $x \in X$ to y is naturally already stochastic. Finally, the distribution of a dataset X may be noisy. This means that an event $x \in X$ can belong to several classes. The smallest possible error that an ideal model can have is called a **Bayesian error**. Within this limit, machine learning can only find rules that are likely to be correct for most members of the set they concern (the socalled probabilistic rules)[19].

The "**No Free Lunch Theorem**" [20] states the performance of an algorithm which has not seen the data is inherently worse than any other algorithm which is averaged over all possible data-generating distributions.
The theorem claims that although no ideal algorithm for all the tasks exists you may find an ideal algorithm for a certain task. Assumptions must always be made about the given distribution of the data. Each algorithm must therefore be specialized for a certain task. The specialized algorithm for the classification and the segmentation task are described in chapter 4. Every machine learning algorithm has to be designed for a given task. Another technique used to reduce errors by adapting a function to the given training set and avoiding over-adaptation is called **regularization**. Recalling the overfitting shown in Figure 2.4, one sees a strongly fluctuating function. During regularization, an additional penalty term is added to the error function to control the excessively fluctuating function so that the coefficients do not take on extreme values. This technique of controlling the error coefficients is called weight decay in the case of neural networks. Regularization significantly reduces the variance of the model without significantly increasing the bias. After this overview of some important machine learning constraints, we now move on to the main component of these algorithms: neural networks.

2.2 Artificial neural networks

This section describes the most common types of artificial neural networks (NN) and learning algorithms.

2.2.1 The biological neuron

A biological neuron as in shown in Figure 2.6 generally consists of a soma, some dendrites, an axon and axon terminals. A neuron receives input signals from other neurons via its dendrites. As soon as a minimum total voltage is reached, it sends its own signal via the axon, which can reach several other neurons or their dendrites due to its branching. The synapses (here axon terminals) play the role of a connector, which again attenuates the signal in different ways. The more frequently the synapse is stressed, the less it attenuates the signal, because it improves its conductivity. The synapse thus gives the signal a weight, which is greater the more frequently this synapse is stressed [21].

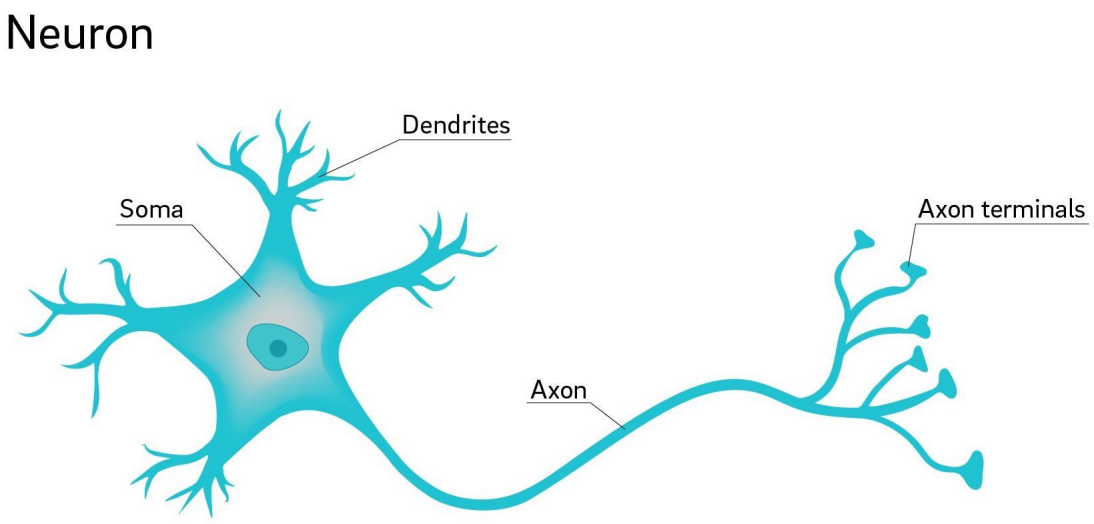

Figure 2.6: Structure of a biological neuron [22]. A neuron consists of a soma, some dendrites, an axon and several axon terminals.

2.2.2 The artificial neural network

The mathematical basis of a neural network was first provided by Kolmogorov in 1957 [23]. Kolmogorov showed for the first time that a continuous function of n variables can be mapped to a single function of a variable. His discovery has been improved upon and perfected by others in recent years [24]. It was shown that Kolmogorov proved the existence of a three-layer network for any classifiable pattern recognition problem. Thus, these theorems imply that neural networks can represent a variety of interesting functions, if they are weighted accordingly. On the other hand,

they usually do not provide a construction for the weights, but only state that such a construction is possible. Although artificial neural networks are inspired by biology, they do not operate exactly the same as biological neural networks, as will be explained below. However, the two forms of neural networks are similar on that both consist of layers of neurons, and each neuron has several inputs and one output. The following Figure 2.7 represents a general model of an artificial neural network (NN).

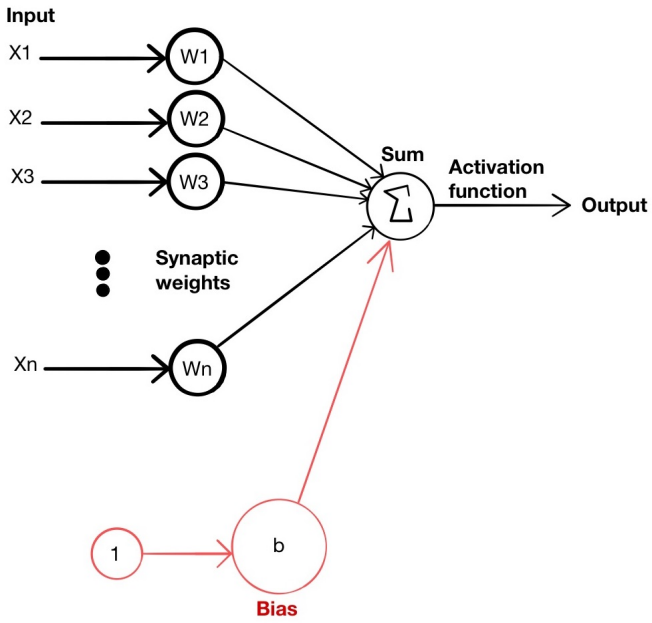

Figure 2.7: Structure of a general model of an artificial neural network (also called single perceptron). $x_1 - x_n$ are the inputs of the neurons. Each neuron is weighted with a synaptic weight $w_1 - w_n$. With the bias b, all of the weighted inputs are summed and inserted into the activation function, which calculates the output.

The input consists of n variables, each of which is multiplied by a connection weight (synaptic weight). These weights indicate the strength of a particular node. The bias value b (see the red part on Figure 2.7) can shift the activation function f_{act} up and down. In this simple case shown in Figure 2.7 the products are summed and the result is calculated by:

$$\text{output} = f_{\text{act}}(x_1 \cdot w_1 + x_2 \cdot w_2 + x_3 \cdot w_3 + ... + x_n \cdot w_n) \tag{2.5}$$

Since our models do not require a single artificial neuron with adjustable weights and a threshold value, it is also important to understand how different neurons can be connected to each other (and form what are called layers). Assume there are n different neurons. Every neuron i has an output o_i, and there is a connection from one neuron i to another neuron j, which means the function of

the synapse takes over the weighting $w_{ij} \in \mathbb{R}$. The values of the inputs are summed up to equal the network input net_i. The input of the neuron j is not only the output o_i, but it is also multiplied with the weight w_{ij}. The output is calculated with the **activation function** $f_{\mathbf{act}}$, which projects the network input onto a limited, and therefore finite, interval.
What is still missing in this consideration is a network which offers high values for the activation function and which never falls below a certain value. This would mean that a large difference in the network input of this neuron would be barely recognizable in the activation. The bias b_j is an additional parameter in the neural network which is used to adjust the output together with the weighted sum of the inputs to the neuron. This constant helps the model fit better to the given data. This can be summarized as follows:

$$\mathrm{net}_j = b_j + \sum_{i=1}^{n} o_i \cdot w_{ij} \tag{2.6}$$

$$o_i = f_{\mathrm{act}}(\mathrm{net}_j) \tag{2.7}$$

To compliment this description, the next section will present some activation functions of neural networks.

2.2.3 The activation functions

Each layer in a neural network consists of many units working parallelly, where each unit represents a vector-to-scalar function. Most of these units can be described as accepting a vector of inputs x and transform with:

$$z = W^T x + b \tag{2.8}$$

where the matrix W is the mapping from x to z with the bias b.
Then the so-called activation function $f(z)$ is applied to z element by element. Most hidden units can only be distinguished from each other through the choice of the activation function $f(z)$. The most important functions are presented below:

Rectified linear unit (ReLu)

For an activation function in neural networks the default choice is the max function:

$$f(x) = max(0, x) \tag{2.9}$$

As seen in Figure 2.8 the rectified linear unit (ReLU) activation function returns 0 as an output in the region of $x < 0$, and also returns a linear function with a slope of 1 in the region of $x > 0$. That is why ReLUs are able to be quickly optimized, since the derivative of the function is always either 0 or a positive constant value. This thus makes the gradient direction far more useful for learning than it would be with activation functions with non-vanishing and higher order derivatives.

A disadvantage of ReLUs is they can not learn with gradient based methods using examples with an activation equal to zero.

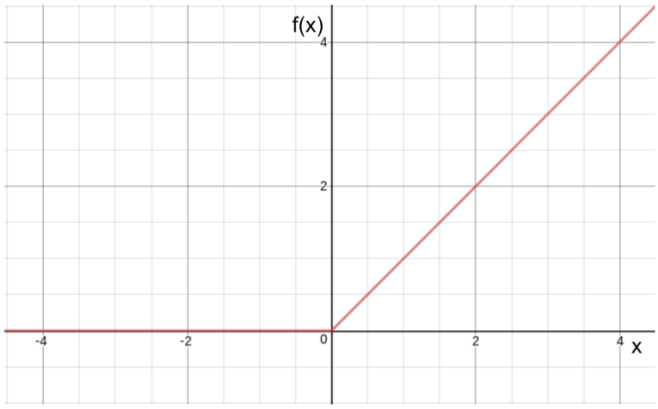

Figure 2.8: The ReLU activation function [25]. For $x < 0$ the function is always zero, for $x > 0$ the function is linear with a slope of 1.

Sigmoid

The sigmoid function describes a probability distribution over a binary variable. It is defined as:

$$\sigma(x) = \frac{1}{1 + exp(-x)} \tag{2.10}$$

As shown in Figure 2.9 the sigmoid function saturates to 0 when x is negative, and saturates to 1 when x is positive.

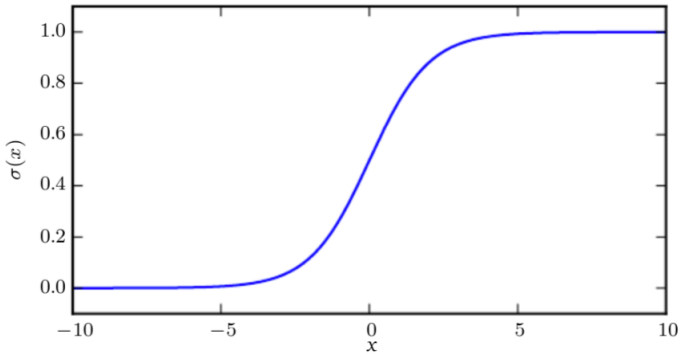

Figure 2.9: The sigmoid activation function [1]

Softmax

The softmax function of x is a probability distribution over a discrete variable with n possible values.

The function is a generalization of the sigmoid function, which is often used as the output of a classifier. The softmax function is computed using the relationship:

$$\text{softmax}(x) = \frac{exp(x_i)}{\sum_j exp(x_j)} \tag{2.11}$$

Comparison of activation functions

The softmax function differs from the sigmoid function in that the sigmoid function is used for binary classification while the softmax function is used for multivariate classification tasks. "The Softmax function is used in multi-class models where it returns probabilities of each class, with the target class having the highest probability" [26].

2.3 The artificial neural network as a system of interconnected neurons

Neural networks consist of upwards of several hundred neurons, which are structured in layers in different ways. The two most basic network structures are described in the following.

2.3.1 Important layer types

There exist networks whose neurons connect with themselves, with neurons within the same layer, or with any other layer. These networks are summarized under the term **feedback networks**. However, since a detailed description of such networks would go beyond the scope of this work, we will concentrate here on a more basic form of networks in which the neurons of one layer have connections only to neurons of the following layer (see Figure 2.10). These networks are summarized under the term **feed-forward networks** [27].

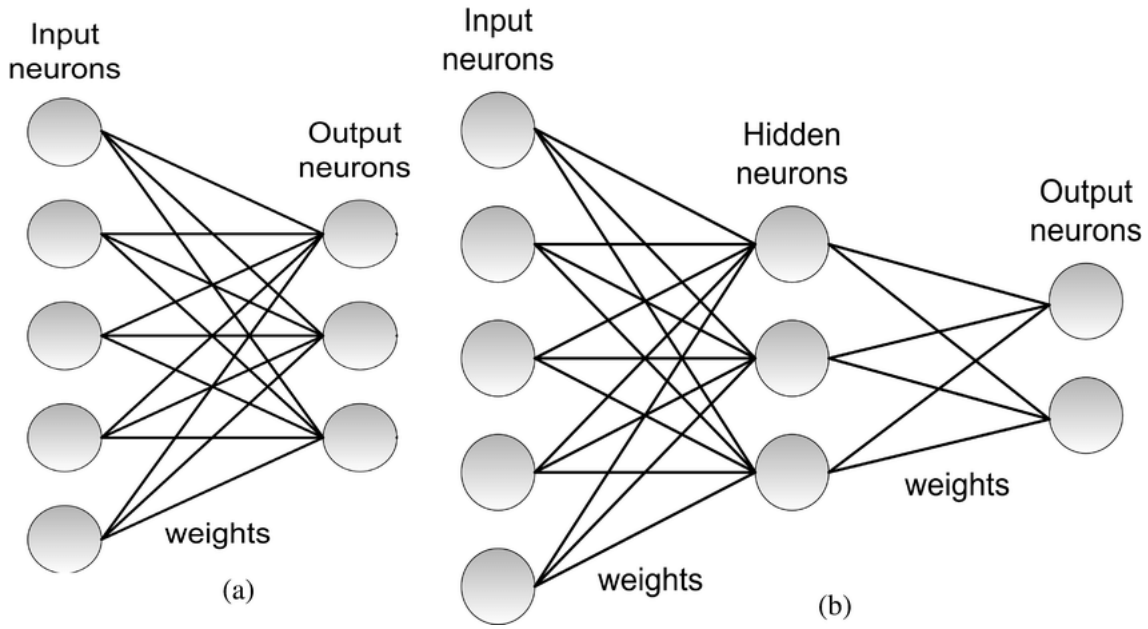

Figure 2.10: (a) Architecture of a single layer perceptron. A layer of input neurons is fully connected to a single layer of output neurons. Each connection between neurons is weighted with a certain value. (b) A multi-layer perceptron consists of more than one layer (here 3 layers) of trainable weights [28].

Single layer perceptron

This form of a feed-forward network consists of one input layer and one output layer both of which contain more than one neuron (see Figure 2.10, a)). The number of neurons is arbitrary. The term single layer indicates that there only exist connections which lead from the input to the output layer, and therefore there is only one layer of connections (weights). Accordingly, only the output neurons do any calculations, as the input neurons only feed in the data and pass them on unchanged [21]. This data represents one specific property of any object per input neuron. To divide the objects, the perceptron uses linear separation. In a classification problem each output neuron can be interpreted as assigning an object to a specific class when its activation is high, and not assigning it when its activation is too low. This creates one subspace per output neuron, defined by the activation function of the neuron, which separates the objects of a class from the rest of the class. A simple example is shown in Figure 2.11. A classification of two fictional data sets with two features (A and B), with two neurons in the input layer of a perceptron each, which must then approximate the represented separation as closely as possible. The two-dimensional coordinate space is divided into two parts by the one-dimensional dividing line as subspace (own representation). In total, this results in a differentiation between all classes. With n output neurons, (at least) n classes can be distinguished via n sub-spaces [21].

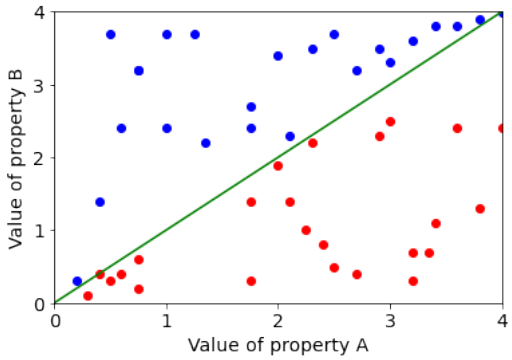

Figure 2.11: Classification of two datasets with features A (red dots) and B (blue dots) with a single layer perceptron. The perceptron uses linear separation to divide the datasets A and B (green line).

Multi-layer perceptron

There exist datasets which cannot be separated linearly (see Figure 2.12). For such data distributions a multi-layer perceptron can be used.

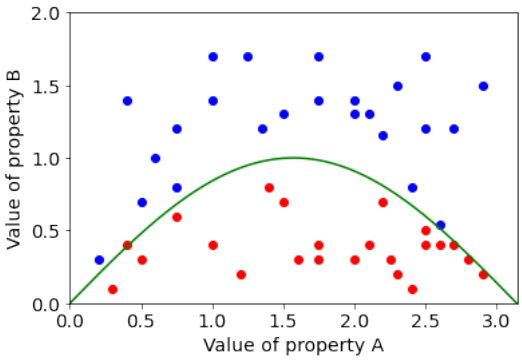

Figure 2.12: Classification of two datasets with features A (red dots) and B (blue dots) with a multi-layer perceptron. The perceptron uses quadratic separation to divide the datasets A and B (green line).

The difference here from the single-layer perceptron is that there are also one or more so-called hidden layers between the input and output layers, (see Figure 2.10 b)), which allow for a non-linear classification. Here, each hidden layer (neuron) can perform a further transformation of the original linear subspace, such as determining the position of a bend, or defining several sub-spaces to separate a single class from the rest [29]. In this thesis the neural network employed is a convolutional neural network (CNN), which will be described in detail in subsection 2.3.3.

2.3.2 Learning methods

After the introduction of important layer types in the last section, methods of learning will be discussed in more detail in the following.

Delta rule

As described in [21, p.272] the main idea of the so-called Delta rule is the change of the weights Δw_{kj} for each training dataset:

$$w_{\mathrm{kj}}^{\mathrm{new}} = w_{\mathrm{kj}}^{\mathrm{old}} + \Delta w_{\mathrm{kj}} \tag{2.12}$$

The delta rule can only be applied to single layer perceptrons. It is the aim of this rule to make an error E smaller, which is defined in the following way:

$$E_j = t_j - o_j \tag{2.13}$$

$$E_{\mathrm{ges}} = \frac{1}{2}\sum_{j=1}^{n}(E_j)^2 \tag{2.14}$$

In this case the error of the output E_j is the difference between the target value t_j and the output neuron o_j for an image of the training dataset (training pattern). The total error is the squared sum of the single errors, so that large deviations have a significant influence, while small errors remain irrelevant.
The following will explain how the change in weighting can be deduced from the error. The error can be seen as a function, a so-called cost function, which depends on the weightings. This is derived from the output o_j from Equation 2.7. This allows a training pattern to be displayed as a graph in which each weighting and calculated error is assigned to an axis. These graphs contain minima, i.e. points at which the cost function has a local minimum. These minima cannot be read out easily, as the graph only shows the cost function of a training pattern. However, the neural network should classify all patterns as belonging together. Therefore it is important to approach an "average" place of minima. In addition, there are always deviant training patterns (called noise) which deviate from the norm and affect the entire network [21, p.195],and lead to a high run time. A graph would have to be plotted for each training pattern or at least countless points would have to be calculated. Since this number obviously grows exponentially based on the number of weightings, this quickly leads to long run times. One approach to solve this problem is to **calculate the gradient** Δ. This is a vector whose individual elements are the partial derivative of a function f according to one of its variables [30, p.211 et seq.]. The following equation can be drawn up for the change of concrete weights [21]:

$$\Delta w_{kj} = -\frac{\delta E_{ges}}{\delta w_{kj}} \tag{2.15}$$

Using the chain rule (for detailed derivation see [30]) you get:

$$-\frac{\delta E_{ges}}{\delta o_j}\frac{\delta o_j}{\delta w_{kj}} = (t_j - o_j) \cdot f'_{akt}(\sum_{i=1}^{n} o_i \cdot w_{ij}) \cdot o_k \tag{2.16}$$

Batch learning and learning rate

The model's internal parameters should not been updated after each training process, but rather after a sufficient number of training images (batches). This technique, known as **batch learning** [30, p.213] requires more computing time depending on the size of each set.

$$\Delta w_{kj} = -\frac{1}{b} \cdot \sum_{r=1}^{b}(t_j^r - o_j^r) \cdot f'_{akt}(\sum_{i=1}^{n}(o_i^r \cdot w_{ij}^r) \cdot o_k^r \tag{2.17}$$

The change in weight according to the batches b, is their average value.

Finally, a so-called **learning rate** η must be introduced as a coefficient with $\eta > 0$. Since the error can move beyond the minimum, the learning rate is used to adjust the step size. As the steps become smaller, the time required to approximate a minimum increases, which is particularly noticeable in flat falling minima. Therefore, the η chosen at the beginning should be large and gradually become smaller, such that a rough approximation with subsequent fine optimization can be made. This is also known as adaptive learning rate [21, p.272]. In summary, the calculation of the weights results in:

$$w_{neu} = w_{alt} + \Delta w_{kj} = w_{alt} + \frac{\eta}{R} \cdot \sum_{r=1}^{R}(t_j^r - o_j^r) \cdot f'_{akt}(\sum_{i=1}^{n}(o_i^r \cdot w_{ij}^r) \cdot o_k^r \tag{2.18}$$

Backpropagation

The information propagates in a neural network from the input node, through the hidden layers to the output node. This is called **forward propagation**. During training, forward propagation continues until it reaches a cost point in an output node [31].
Back propagation refers to the passing of information from the cost to then flow backwards through the network to calculate the gradient. The idea of the algorithm is to compute the changes of weight layer-by-layer starting from the output layer and to compute the bias depending on its effect on the cost function and to apply them. The most commonly applied NN for analyzing visual images is the convolutional neural network. The main idea and the mathematics behind these networks will be presented in the next section.

2.3.3 Convolutional neural networks

Convolution is a mathematical operation between matrices. "(CNNs) are simply neural networks which use convolution in place of general matrix multiplication in at least one of their layers" [1].

A 2D convolution begins with a matrix of weights, called a kernel. This kernel slides over the input data and performs an element-by-element multiplication and sums up the results into a single output value.

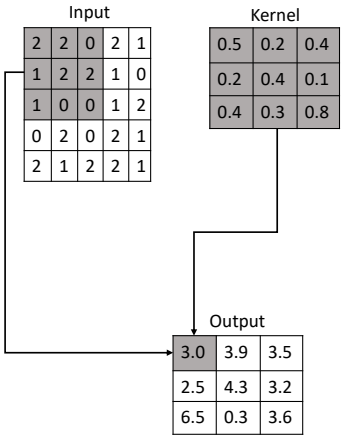

Figure 2.13: An example of 2D convolution. The input size is (5×5). The kernel (3×3) slides over the input data and returns a (3×3) matrix.

For example, in Figure 2.13 a 5×5 feature matrix is converted into a 3×3 matrix. The convolutional calculation e.g. for the first entry of the output O is:

$$O_{11} = 0.5 \cdot 2 + 0.2 \cdot 2 + 0.4 \cdot 0 + 0.2 \cdot 1 + 0.4 \cdot 2 + 0.1 \cdot 2 + 0.4 \cdot 1 + 0.3 \cdot 0 + 0.8 \cdot 0 = 3.0 \quad (2.19)$$

Before moving on to our used residual network (ResNet), it is worth looking into three more techniques, which are commonly used with convolutional layers.

Padding

During the sliding of the kernel over the input matrix the edges get cut off because the pixels are never at the center of the kernel. To avoid this, padding is added. Padding refers to adding extra ghost pixels as shown in Figure 2.14. Typically, the value of the extra pixels is set to 0.

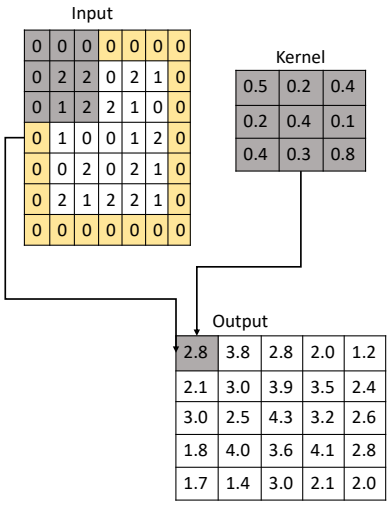

Figure 2.14: An example of 2D padding. The input size is (7×7) and is padded with zeros. The kernel (3×3) slides over the input data and returns a (5×5) matrix.

Now during sliding, the kernel reaches the entire input pixel-by-pixel at its center.
If in total p_h rows are padded and p_w columns are padded, then the output shape will be:

$$O_{size} = (n_h - k_h + p_h + 1) \times (n_w - k_w + p_w + 1) \tag{2.20}$$

where n_h, n_w are the height and width of the input matrix, and k_h, k_w are the height and width of the kernel matrix. With **same padding** (and a stride of 1) the outputs will have the same spatial dimensions as the inputs.

Pooling

When processing images, the spatial resolution of hidden representations should be gradually reduced. Information is supposed to be aggregated in such a way that the receptive field (in the input), to which each hidden node is sensitive, becomes larger the higher we go up the network.
In classification, the global question arises, e.g: Does the image contain an object? Typically, the nodes of our final layer should be sensitive to the entire input. By gradually aggregating information, which results in increasingly coarse maps, we achieve this goal of ultimately creating a global representation, while retaining all the advantages of the convolution layers in the intermediate layers of processing. In addition, we often want our representations to be somewhat translation-invariant when detecting features at lower levels, such as at the edges. If we take an image with a sharp border between a ring and the background, and move the whole image one pixel to the right, the output for the new image might look very different. The border will have shifted one pixel, and with it all activations as well.
This section introduces pooling layers which reduce the sensitivity of the convolution layers with

respect to location and spatial down-sampling.
A pooling operator consists of a window which moves across all regions in the input according to step size, with a single output calculated for each location passed through by the window. The pooling window starts at the top left of the input field and slides from left to right and from top to bottom over the input field. However, the pooling layer does not contain any parameters. Instead, pooling operators calculate either the maximum (see Figure 2.15) or the mean value of the elements in the pooling window. These operations are called **maximum pooling** or **average pooling**.

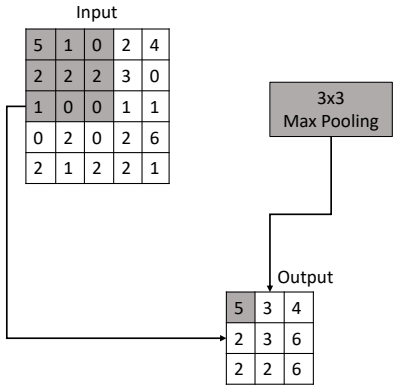

Figure 2.15: An example of 3×3 max pooling. The input size is (3×3). The (3×3) max pooling slides over the input data and returns a (3×3) matrix.

Striding

In the classification, the output of a convolutional layer should be smaller than the input. Another way of accomplishing this, in addition to using a pooling operator, is striding.
The idea of striding is to skip some positions of the kernel. The number of rows and columns that are passed through per slide is called a step. So far we have used a step of 1 for height and width in Figure 2.13 and Figure 2.14. Figure 2.16 shows a two-dimensional cross-correlation operation with a step size of 4 vertically and 3 horizontally.

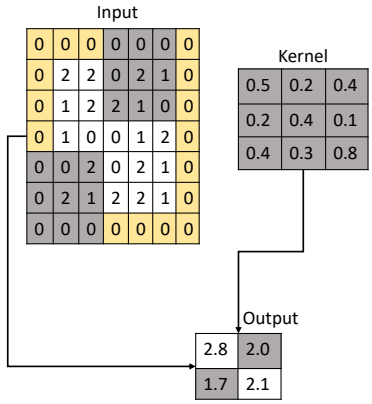

Figure 2.16: An example of 2D striding. The input size is (7×7) padded with zeros. The kernel (3×3) slides with a stride of 2 over the input data and returns a (2×2) matrix.

The behaviour of the output size O can be described with:

$$O = \frac{I + 2P - K}{S} + 1 \tag{2.21}$$

where I is the input size, P the padding, K the kernel, and S the striding.
The artificial NNs with their neurons, activation functions, learning methods, and convolutional neural networks have now been sufficiently introduced. In this thesis neural networks are used for the classification and segmentation of scattering data. However, before we come to the networks we used and the results, it is important to understand the data with which these networks were trained. Therefore, a short overview of the physics behind scattering with X-rays and the resulting scattering images will follow.

3 Experimental background and techniques

Organic-inorganic perovskites have been studied with great interest, since they can be used as absorbers for highly efficient solar cells [32]. They have been the subject of many research studies for several years (see [33, 34, 35, 36]), because they offer the promise of high energy conversion efficiencies at low cost [37]. The typical structure of perovskites solar cells (PSCs) is shown in Figure 3.1. Real-time X-ray scattering is the best way to examine the kinetic effects during structure formation. It is interesting to determine how the ions are distributed in the structure and how this affects the optoelectronic properties of the perovskite thin film [35]. Images analyzed in this thesis are taken from perovskites thin films.

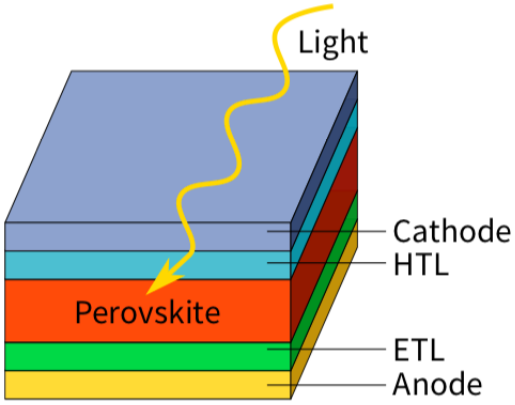

Figure 3.1: Architecture of a single-junction perovskites solar cell. The crystalline perovskite thin film is used as absorber material between two electrodes, separated by an electron transport layer (ETL) and a hole transport layer (HTL).

The crystallinity and morphology of a perovskites layer determines the optoelectronic properties, and consequently also the performance of PSCs. The crystallinity and morphology are not only dependent on the chemical composition, but also on the exact conditions of film deposition.

3.1 Perovskite thin films

In general, a thin film is less than 1 μm thick and is deposited on a substrate using one of several deposition methods [38]. The structure family of perovskite is related to the mineral perovskite

$CaTiO_3$ [39]. The ideal crystal structure of a cubic ABX_3 perovskite can be seen in Figure 3.2. Most commonly, **A** can represent either methyl-ammonium (MA), formamidinium (FA), caesium (Cs^+) or rubidium (Rb^+), **B** represents (Pb^{2+}), and **X** either chlorine (Cl^-), iodine (I^-) or bromine (Br^-). Changing the composition of the perovskite films can considerably improve the growth and formation of the crystal structure and morphology.
As an experimental method the real-time grazing incidence wide angle X-ray scattering (GIWAXS) is used. Since the Perovskite thin films were only used as a test case for the analysis of X-ray scattering data, the background of their intriguing optoelectronic properties will not be discussed here. The interested reader is referred to recent reviews of this fast-moving field (see [37, 36, 35, 40]).

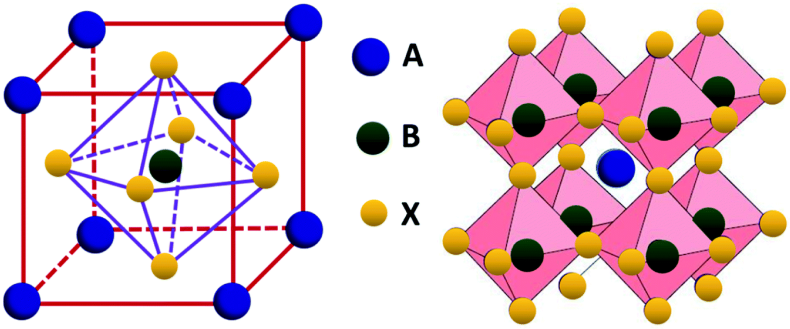

Figure 3.2: The structure of perovskites with ABX_3 formula. The B and X ions form the BX_6 octahedra with B in the middle and X in the corners. If the corners are connected, the BX_6 octahedra extends to a 3D structure [41].

3.2 Grazing-incidence wide-angle X-ray scattering

Prior to describing the experimental technique in subsection 3.2.2, the basics of X-ray scattering will be summarized below.

3.2.1 X-ray scattering

X-ray wavelengths are between 0.1 and 100 Å. X-rays resolve the atomic structure of ordered materials like crystals. Within the quantum mechanical approach, the X-rays are quantized into photons with the energy

$$E_{Photon} = \hbar\omega \tag{3.1}$$

Through the interaction of electromagnetic waves with matter, the electric field excites electrons in the atoms. These electrons then radiate their excess energy in another electromagnetic wave, in a process known as **scattering**.

Scattering vector

The relationship of the phase difference $\Delta\Phi$ between two incoming waves from a coherent and monochromatic source [42] is very important.

$$\Delta\Phi_{in}(\vec{R}) = 2\pi \cdot \frac{\vec{k}_{in} \cdot \vec{R}}{\lambda} \tag{3.2}$$

$$= kR \sin \alpha_{in} \tag{3.3}$$

The lattice vector $\vec{R}$ multiplied by the direction of the incoming wave $\vec{k_{in}}$ (with an angle of incidence α_{in}) equals the path difference. The phase difference between two scattered waves $\Delta\Phi_{sc}$ is determined analog to Equation 3.3 with opposite sign. This leads to a **total phase difference** of:

$$\Delta\Phi(\vec{R}) = \Phi_{in} - \Phi_{sc} = (\vec{k}_{in} - \vec{k}_{sc}) \cdot \vec{R} = \vec{q} \cdot \vec{R} \tag{3.4}$$

where the difference of the direction of the incoming $\vec{k}_{in}$ and the scattered wave $\vec{k}_{sc}$ represents the scattering vector $\vec{q}$. The waves interfere constructively if the total phase difference is a multiple of 2π.

Reciprocal lattice

The lattice vectors can be described as a linear combination of the basis vectors of the lattice, resulting in the Bravais lattice with:

$$\vec{R} = n_1\widehat{a} + n_2\widehat{b} + n_3\widehat{c} \tag{3.5}$$

where n_1, n_2 and n_3 are all integers. This lattice can be represented by a reciprocal lattice via a Fourier transformation. The reciprocal lattice is important in many studies of periodic structures, especially in the theory of diffraction. For example, the diffraction pattern of a crystal is used to determine the reciprocal vectors of the lattice. Through this method the atomic structure of a crystal can be inferred. As a Fourier transformation of the Bravais lattice, the reciprocal lattice can be defined with:

$$\vec{G} = h\widehat{a}^* + k\widehat{b}^* + l\widehat{c}^* \tag{3.6}$$

where $\widehat{a}^*$,$\widehat{b}^*$and $\widehat{c}^*$ are the reciprocal basis vectors and the h, k, l are the Miller indices. As previously discussed, with constructive interference the **Laue condition** can be introduced with:

$$\vec{q}\vec{R} = \vec{G}\vec{R} = 2\pi(hn_1 + kn_2 + ln_3) = n \cdot 2\pi \tag{3.7}$$

where $h, k,$ and l are all integers. If q is coincident with a reciprocal lattice vector G, the Laue condition for the observation of diffraction of a crystal lattice is thus fulfilled. Starting from the

Laue condition, and if the scattering vector q_{hkl} of a certain interference maximum is known, one can conclude the corresponding set of lattice vectors R as well as the distance of neighbouring lattice planes with:

$$d_{hkl} = \frac{2\pi}{q_{hkl}} \tag{3.8}$$

Additionally, if the crystal structure is known, the size of the unit cell in $\widehat{a}, \widehat{b}, \widehat{c}$ can be calculated. **Bragg's law**, which describes the dependency of the Laue condition on the scattering angle θ, can be derived from Equation 3.8 as:

$$n\lambda = 2d_{hkl} \sin\theta \tag{3.9}$$

whereby θ is the angle between $\vec{k}_{in}$ and $\vec{k}_{sc}$ and n is a positive integer. The scattering at a crystal is limited to certain points in the reciprocal space. From a large set of intensities of a given crystal one can derive the positions of the atoms in the unit cell. Due to the conversion of the intensities from the angular space to the reciprocal, and taking into consideration the maximum of the intensity profile $I(q)$, it is possible to determine which q fulfills the Laue condition (Equation 3.7), and to derive the conclusion what the underlying crystal lattice is.

3.2.2 Experimental methods

The images analyzed in this thesis were produced using the in situ **GIWAXS** and the in situ **grazing-incidence small-angle X-ray scattering (GISAXS)** method. The scattering geometry is shown in Figure 3.3.

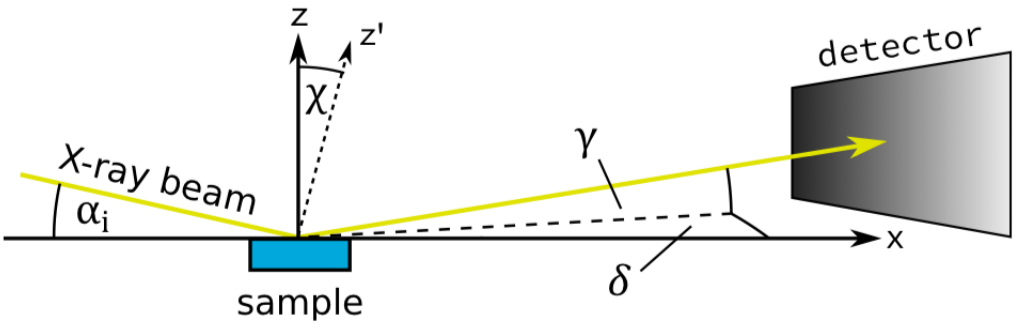

Figure 3.3: The scattering geometry of the X-ray beam path. The angle of incidence α_i is controlled by rotating the sample by an angle χ. δ and γ are the in-plane and out-of-plane scattering angles.

The X-ray beams with wavelengths close to atomic distances are directed onto a thin film at a

shallow angle of incidence α (see Figure 3.3). This angle is close to a critical angle:

$$\alpha_{cr} = \sqrt{2\delta} = \lambda\sqrt{\frac{\rho_e r_e}{\pi}} \tag{3.10}$$

Part of the intensity of the X-ray beam is converted into an evanescent wave along the surface of the thin film sample with only a little transmitted light. This has the effect of a low penetration depth ($\sim 100\ nm - \mu m$). It is ideal for analysis of thin films because possible background signals from underlying substrate are reduced.
Randomly orientated crystallites effectively reduce the $I(\vec{q})$ to a $I(q)$ profile. The scattered intensities I(q) form concentric circles with radii q_{hkl} on a 2-dimensional area detector (placed perpendicularly to the incoming X-ray beam). These are known as **Debye-Scherrer rings**.

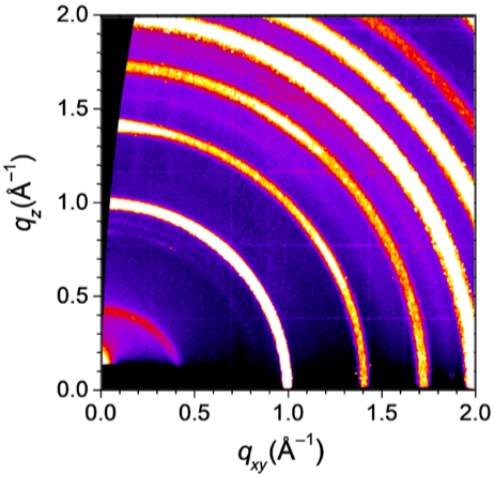

Figure 3.4: GIWAXS data of 3D perovskite thin film. The angle of incidence is 0.1°. The circles denote the structure of the thin film [40].

If the crystals are not randomly arranged, but have a certain order, peaks appear instead of rings. Instead of isotropically ordered, such a sample has a defined crystallographic orientation with respect to the substrate. In a well-oriented sample, the Bragg peaks are short diffraction spots, while in a poorly oriented sample these segments are longer. In the extreme example of a example of a perfect powder sample, the Bragg peaks become isotropic rings [43]. With these rings and segments, it is possible to investigate the structural properties of a sample.
As the background of the scattering data has now been presented, we can move on to the task of automatically distinguishing and localizing these Bragg peaks.

4 Results

In this chapter a computer analysis based on DL of diffraction patterns of perovskites thin films is described. The data was measured via the aforementioned grazing-incidence wide-angle scattering GIWAXS technique. This work addresses two different but related analysis approaches:

1. Classification of the GIWAXS patterns based on the type of Bragg peaks (presence of Debye-Scherrer rings with and without preferred orientations).

2. Segmentation of the GIWAXS patterns into areas with and without Bragg reflections.

The first task is aimed at separating diffraction patterns into two classes: those with preferred orientations and those without. Accomplishing this would accelerate the preliminary analysis of large amounts of GIWAXS data and provide information about the orientation distribution of the crystal domains. Moreover, this approach can in principle be generalized, and the insight gained from this work may be applied to other classification tasks for diffraction patterns. Segmentation is a more complex task and is aimed at localizing the Bragg peaks and separating them from the background. This issue, if solved, would have many practical applications: from data compression during measurement, to the automated peak indexing algorithms. In this thesis for both of these tasks, a supervised machine learning method is used. One of the main focuses of this research is to investigate whether or not simulated data can be used for the training process. The use of simulated and experimental data for the training process will be compared and discussed.

4.1 Data simulation

The data simulation process used in this work differs in some ways from typical physical simulations. In the case of GIWAXS simulations, the most intuitive approach would be to simulate Bragg peaks for a set of different materials, accounting for forbidden peaks by simulating intensities based on the scattering theory. Using such simulated data for training the computer vision model would allow it to learn the exact positions of these peaks for a given set of materials and crystal structures. However, due to certain considerations, a different approach was chosen for this work.
The machine learning model is supposed to distinguish peaks from background wherever peaks are located. This task may be challenging for the computer vision algorithm because of varying noisy backgrounds, different contrasts, and low Bragg peak intensities that may only be recognizable based on their shape rather than intensity levels. That is why it is important to train the model

to be able to recognize most of the Bragg peaks on the image based only on their shape, despite any noise, low concentration, or difficult backgrounds. For this reason, in the simulation process the peak locations are not constrained by any set of materials, and are allowed to appear in any place on the diffraction pattern via pseudo-random generation algorithms. For this reason there are no q-coordinates on figures with simulated images. The other important part of the simulation is the image background. Although it would be in principle possible to simulate a background based on physical understanding of its possible sources, this method would only work if there were no other missing sources of background which are difficult to formalize. Additionally, the systematic research on the set of possible background sources and their simulations is in itself a challenging task and goes beyond the scope of this thesis. Consequently, the background is simulated as a sum of noisy trigonometric functions with randomly distributed parameters to prevent the model from memorizing any background patterns. In the next section the classification of the diffraction patterns based on Bragg peak types is discussed.

4.2 Classification of the powder diffraction patterns

As mentioned above, the idea of this section is to separate the powder diffraction images into those with preferred orientations and those without. Visually, these two classes correspond to different angular distributions of Bragg peaks in the GIWAXS images. Since some of the samples are mixed (contain both types), there are three different classes in total (see Figure 4.1):

- Class 0: Rings
- Class 1: Diffraction spots (segments)
- Class 2: Rings and diffraction spots (mixed)

To analyze GIWAXS patterns, the detector images are usually converted from angular space to reciprocal space (q-space), as demonstrated in Figure 3.4. However, this work was a first attempt on image recognition on GIWAXS data using neural networks and since converting the data would add another layer of complexity to this task, only the raw data in angular space was used throughout this work. Also, since so far the tasks were to only determine if and where Bragg reflections are present in the image, no corrections for distortions or the beam center position were applied to the data. Consequently, it is only possible to show figures of the experimental and simulated data with axes in pixels rather than angles or q-coordinates. For this classification task a pretrained convolutional neural network for image recognition (a **Residual Network**) was used. The setup is described in detail below.

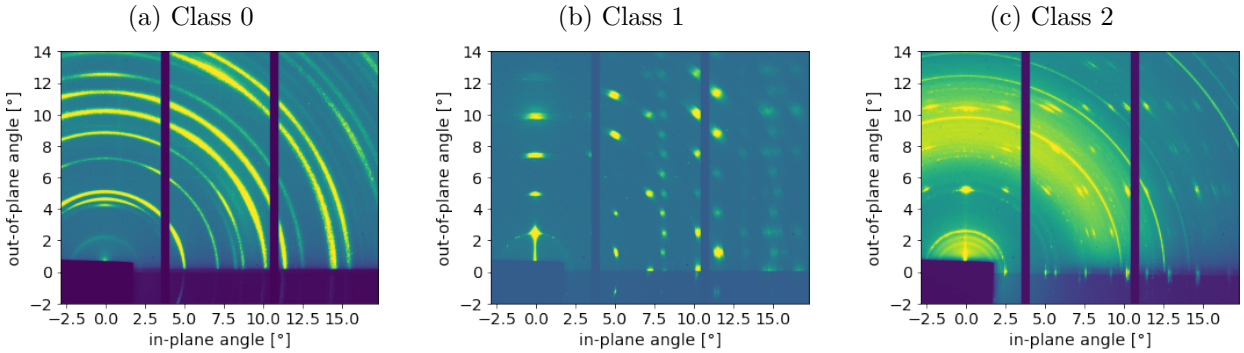

Figure 4.1: GIWAXS images of the three different classes. (a) shows an experimental image containing rings (class 0) (b) an experimental image with diffraction spots (class 1), and (c) an experimental image with rings and diffraction spots (class 2). Image recorded with a photon energy of 22 keV and a sample-detector-distance of 295 mm.

4.2.1 Setup of the neural network

The aforementioned convolutional neural networks (see subsection 2.3.3) are commonly used for image classification. For CNNs one might conclude, that the deeper they are, the better they are, because the models become more powerful. However, it has been found, that past a certain depth, the performance decreases and we encounter issues like vanishing gradients. Since an explanation of this would go beyond of the scope of this thesis, more details on the matter can be found in [44]. Residual networks solve the problem of vanishing gradients and decreasing or even saturating performance of deep neural networks. The special feature of a ResNet is the connection of the output of a layer with the input of an previous layer. If the output of a certain layer is not better than the output of the previous layer, it will skip this layer. In this way the ResNet avoids decreasing accuracy and vanishing gradients which result from the use of too many layers in a NN.

Structure of the ResNet

ResNet was designed by Kaiming He in 2015 [2]. This network makes it possible to train hundreds or even thousands of layers, and still deliver a satisfactory performance. There are different variants of ResNets, depending on the size of the individual layers and the number of layers in the model. For the classification task the ResNet-50 was used. It was pretrained using one million images from 1000 categories within the ImageNet database. The model has over 23 million trainable parameters. As shown in Figure 4.3, a ResNet consists of one convolutional and one pooling step followed by 4 blocks of layers with similar behaviour. Since the ResNet variants only differ in the number of layers, in Figure 4.3 ResNet-34 is shown. Each of the 4 layers (in different colors in Figure 4.3) performs a 3×3 convolution with a fixed feature map dimension ($F \in [64, 128, 256, 512]$). The dimensions of the width and the height are constant.
The dotted lines represent changes in the dimension of the input. This reduction is achieved by increasing the stride of the first convolution of each layer. In Figure 4.4 there is an overview of the

output size at every layer and the dimension of the kernels. Figure 4.3 shows the 4 repeating color-coded blocks. The operations of one of these blocks includes of a convolution, a batch normalization, and a ReLU activation (basic layer). Only the last operation of a block has no ReLU (bottleneck layer). These layers allow the network to remember what it has learned before. When there is nothing to learn, it applies an identity mapping weight function (see Figure 4.2), where the output is equal to the input. This is also shown in Figure 4.3 by the arrows to the right of the different layers. What the neural network has learned, is preserved here by not applying decreasing transformations. This special feature of a ResNet solves the problem of a vanishing gradient, because if the network is too deep, the gradients from which the loss function is calculated could easily fall to zero, which would cause the weights to never update their values and therefore no learning could be done.
If we look at the first operation of each block, we see that the stride of the first operation is 2 (instead of 1 as in the others). Thus the down-sampling of the volume by the network is achieved by increasing the step size, rather than by using a pooling operation. The first operation of each block reduces the dimensionality. If we skip the connection (in case of applying the identity mapping weight function, see Figure 4.2), the size of the volume also needs to be changed. This is indicated in Figure 4.3 by the dotted lines.
To train the neural network a **cost function** and an **optimizer** must be selected.

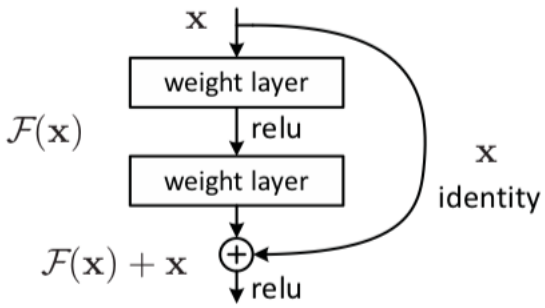

Figure 4.2: A building block with identity mapping [2]. The output of a layer is connected with the input of an earlier layer. A layer can be skipped via the identity skipping connection.

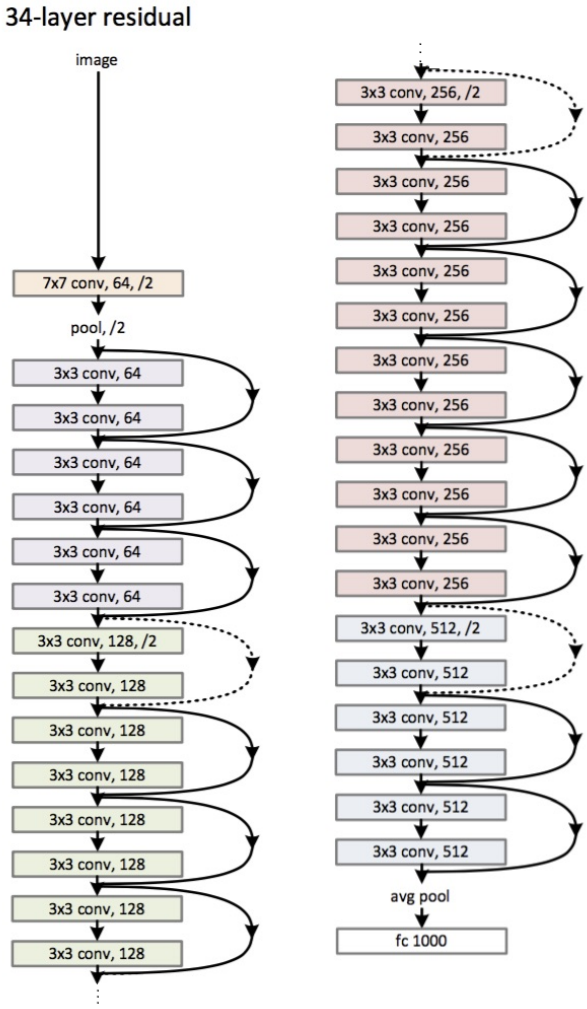

Figure 4.3: Architecture of a residual network with 34 parameter layers. [2]

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix} \times 2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix} \times 3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix} \times 2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix} \times 4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix} \times 8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix} \times 2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix} \times 6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix} \times 6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix} \times 23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix} \times 36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix} \times 2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix} \times 3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

Figure 4.4: Output sizes and convolutional kernels for ResNets [2]

Cost function

For the cost function the **cross-entropy loss** was chosen, because it measures the performance of a classification model whose result is a probability value between 0 and 1. **Cross-entropy loss** is defined as:

$$C = -\frac{1}{n}\sum_{x}[y\ln(a) + (1-y)\ln(1-a)] \tag{4.1}$$

with the neuron output a, the total number of training data n, and the corresponding desired output y. The C value is always positive, and if the neuron's actual output is close to the desired output x, then C will be close to zero. C increases when the predicted probability differs from the actual label.

Optimizers

Furthermore, the choice was had to be made between the stochastic gradient descent (SGD) optimizer and the adaptive moment estimation (adam) optimizer. Since a more detailed explanation of the functionality of these gradient descent optimization algorithms would go beyond the scope of this thesis, reference is made here to paper [45].

Grid search

Hyperparameter optimization is a significant part of deep learning. To see if the results are robust enough to withstand possible parameter changes, a grid search is useful. In the neural network there are many parameters which can influence the performance of the model including the batch size, the learning rate, the cost function, the optimizer, and how often all of the training samples passed through the learning algorithm. This quantity is called an epoch. And there are many more hyperparameters. Due to spatial constraints, only the change in the learning rate and the optimizer will be discussed in this thesis. A deep pretrained network and well-chosen optimizers and cost-functions are not sufficient on their own. As mentioned above, it is necessary to have a representative dataset to train the network. For this purpose, the following section will explain which preprocessing steps were necessary for the simulated and experimental images.

4.2.2 Data preparation

The classification task was todeled with two networks trained with a simulated and a experimental dataset, respectively. Before discussing the results of the two trained networks, it is important to point out what preparatory work of the data was necessary.

Simulated Data

The images were shaped to size (224×224) and were normalized with the maximum intensity of the respective image. Figure 4.5 shows one simulated image of each class. 5000 simulated images

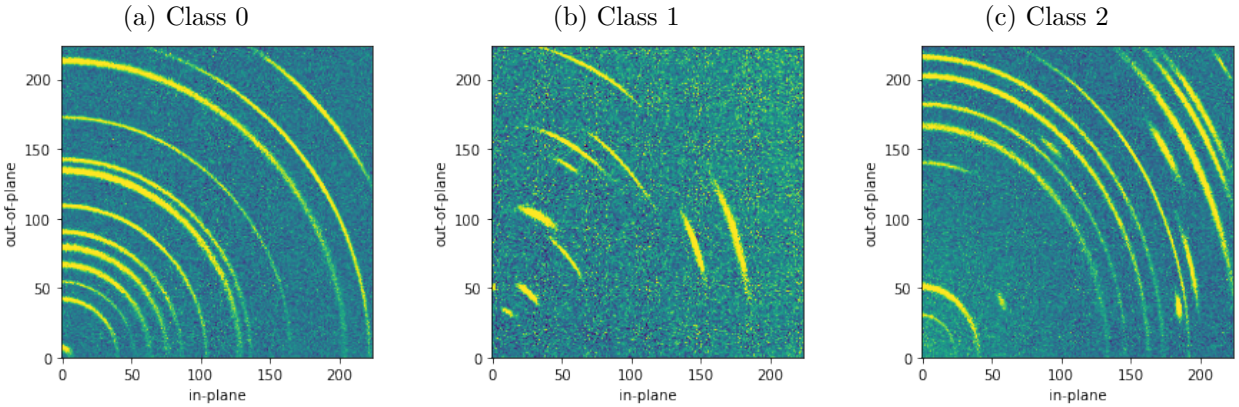

Figure 4.5: Simulated images of each class. In figure (a) the image only contains rings (class 0), in figure (b) the image contains only segments (class 1) and figure (c) contains both (class 2). The image dimensions were 224×224 pixels.

were created and divided into 3 subsets as follows:

- Training dataset: 3250 images
- Validation dataset: 750 images
- Testing dataset: 1000 images

Experimental Data

The experimental data was obtained from different GIWAXS measurements with various perovskites thin films:

- NAIB-JM1-5: adamantyl-based layered hybrid perovskites ($A_2CH_3NH_3Pb_2I_7$)
- NAIB-meso-8, 12, 14, 16: formamidinium-based Dion-Jacobson hybrid perovskites
- NAIB-S1, S2, S3: 3D-CH_5N_2PbI3
- NAIB-S10, S11: 2D-butylamine and 2D-$(CH_3)2CH$-CH_2-NH_2

Before the experimental data could be used for training, each of the used images had to be labeled manually with this corresponding class. In Figure 4.1 there is one example image from each class. Every image can be assigned to a class. The following categorization was made:

- Class 0 (rings): 154 images
- Class 1 (segments): 44 images
- Class 2 (mixed): 110 images

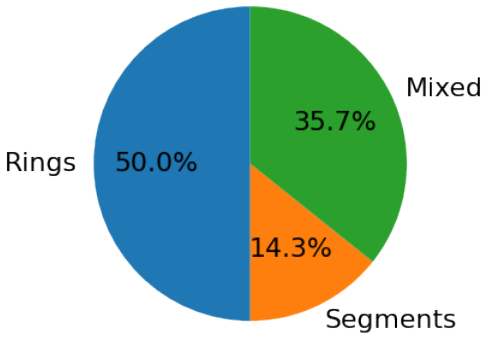

Figure 4.6: Class distribution in the experimental GIWAXS data used.

Since the images are not distributed equally between the classes, there is a resulting data imbalance (see Figure 4.6).

In order to ensure that the network can be trained well, no class should be underrepresented. Thus it was necessary to create more data which belonged to classes 1 and 2. This was done by flipping and rotating the images of the two classes. Just as in the case of simulated images, the images were resized from 484×619 to 224×224 pixels. It was necessary to increase the contrast of the experimental images through clipping, the minimum value of each was set to:

$$\min_{\text{img}} = \overline{v}_{\text{img}} - \sigma_{\text{img}} \cdot f \tag{4.2}$$

and the maximum value was set to:

$$\max_{\text{img}} = \overline{v}_{\text{img}} + \sigma_{\text{img}} \cdot f \tag{4.3}$$

where $\overline{v}_{\text{img}}$ is the mean value and σ_{img} is the standard deviation of the image values. f is an empirically determined factor (here: $f = 5$). In Figure 4.7, the effect of clipping is shown. Most of the rings and segments are easily distinguishable from the background whereas some are only visible after clipping.

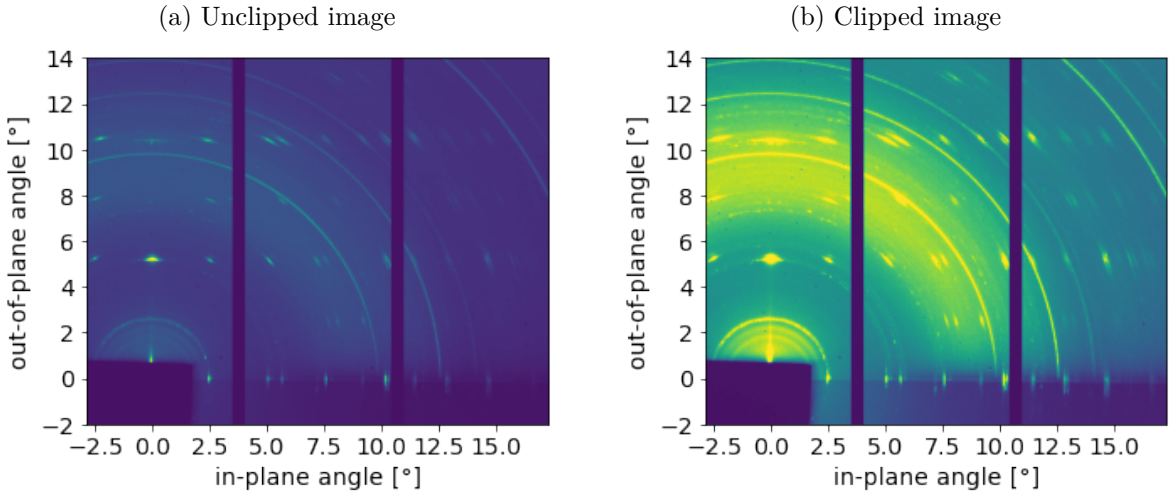

Figure 4.7: Comparison of the contrast between an unclipped (a) and a clipped (b) experimental GIWAXS image. Image recorded with a photon energy of 22 keV and a sample-detector-distance of 295 mm.

Costly the images were split into a set of training data, validation data and test data:

- Training dataset: 300 images
- Validation dataset: 69 images
- Test dataset: 60 images

4.2.3 Results of the network trained with simulated data

In order to find the best learning rate and optimizer for the model, grid search was performed using different learning rates and the two optimizers SGD and adam. The results are shown in Table 4.1. With these values of accuracy, it can be stated the model is stable under changing parameters.

Learning rates	0.01	0.02	0.03
Accuracy SGD	94%	93%	89%
Accuracy adam	94%	95%	92%

Table 4.1: Accuracies of the SGD/adam optimizer and different learning rates

For further classification with this model, the adam optimizer and a learning rate of 0.02 was used since it yielded the highest accuracy. The model was trained for 75 epochs. It has to be taken into account, as already mentioned in subsection 2.1.1, that too much training would cause the model to overfit to the training dataset. This problem was avoided by using the **early stopping** method, with which the network is trained until the performance on the validation dataset starts to decrease. The checkpoint for early stopping was set at the 54th epoch. Figure 4.8 shows the training and the validation loss of the adam optimizer as a function of the epochs.

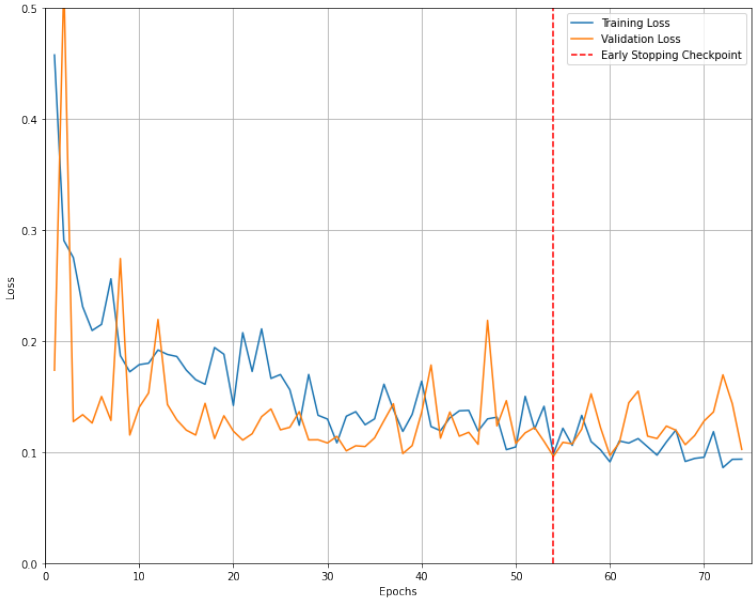

Figure 4.8: Training and validation loss using the adam optimizer as a function of the trained epochs.

With the **simulated test dataset**, the following accuracy rates were achieved:

Accuracy of class 0:	96%	(325/338)
Accuracy of class 1:	100%	(330/330)
Accuracy of class 2:	91%	(303/332)
Accuracy (overall):	95%	(958/1000)

Overall 95% of the simulated test dataset were classified correctly.

When the trained model was tested on the **experimental dataset**, the general accuracy was about 35%:

Accuracy of class 0:	0%	(0/154)
Accuracy of class 1:	0%	(0/44)
Accuracy of class 2:	100%	(110/110)
Accuracy (overall):	35%	(110/308)

However, the neural network always predicted the same class which corresponds to 35% of the test set (class 2). The fact that it always predicts only one type for any experimental images indicates that it cannot distinguish between different types of experimental images. This could possibly be due to the simulated data not being similar enough to the experimental data for this particular task. To explain why this happens, one has to examine exactly how the model makes a prediction of a certain class. As discussed above, a CNN model uses a hierarchical structure of simple kernels ("feature filters"). As a result of the layer-by-layer convolution of the input image with these kernels,

the model extracts certain "higher-order" objects if they are present on the image. The result of the classification depends on whether a set of certain kernels is activated on the image, i.e., if the model detects certain particular objects on the image. In other words, the classification model pays attention to some features on the image and predicts accordingly. This attention depends on the training process, and during the model process the model memorizes dependencies between the target labels (classes) and the features present on each corresponding image.
For some reason the model can memorize wrong features very confidently. This is a well-known phenomenon usually related to the lack of variability of the training data. For instance, if a certain type of fish appears in an image dataset only as a dead fish in the hands of a fisherman, the model can memorize fisherman's hands or his clothes instead of certain unique features of this type of fish, and this is more likely to occur when a fisherman appears only on this certain class of images. The problem here is that the model works perfectly, it has to classify a new dataset of images where a fisherman appears on every photo, as in this case, the model will classify every image as an image containing some particular type of fish.
The situation described above provides insight on what could go wrong with the classification of experimental data. Most likely there is a certain feature in the mixed type simulated images that is also present in all the experimental images. This also shows that the resulting metrics cannot be interpreted correctly without a deeper understanding of how the model works. The fact that the model correctly classifies the mixed type of images does not necessarily mean that the model works correctly (pays attention to the correct features) for these images. It is possible to visualize the model attention and change the simulated data correspondingly to solve the problem. However, this would require undertaking a time-consuming repetitive process aimed at improving the simulation process. Another possible solution is to provide additional information to a model concerning which part of an image it should pay attention to. This can be done via the object detection approach, in which each Bragg peak would be detected independently. This would eliminate the problem of giving attention to background features unrelated to the task.

4.2.4 Results of the network trained with experimental data

In this particular case it seems to be a more optimal approach to train the neural network with experimental data instead of simulated data. Since the experimental data is easy to divide into one of three classes, and it eliminates most of the potential problems relating to the representation of the training data. After the preprocessing work (described in section 4.2.2), the ResNet-50 was trained with experimental images. Just as in the training process of the previous model, a grid search was done. In the grid search Table 4.2, the accuracy of the model trained with experimental data is summarized. The model is stable under changing parameters with accuracies between 95 and 98%.

Optimizer	Learning rate	Accuracy
Adam	0.01	95 %
Adam	0.02	96 %
SGD	0.01	98 %
SGD	0.02	95 %

Table 4.2: Grid search results of training the ResNet-50 with adam optimizer and SGD optimizer with different learning rates.

The SGD optimizer with a learning rate of 0.01 was used, because it had the highest accuracy for the test dataset. Figure 4.9 shows the training loss and the validation loss of the SGD optimizer as a function of the epochs. Using early stopping, the model was trained for 100 epochs. To avoid over- and underfitting, the training loss always has to be slightly larger than the validation loss. The training and validation losses decrease after fluctuations in the first 20 epochs, until they are close to zero. The checkpoint for early stopping was set around the 90th epoch.

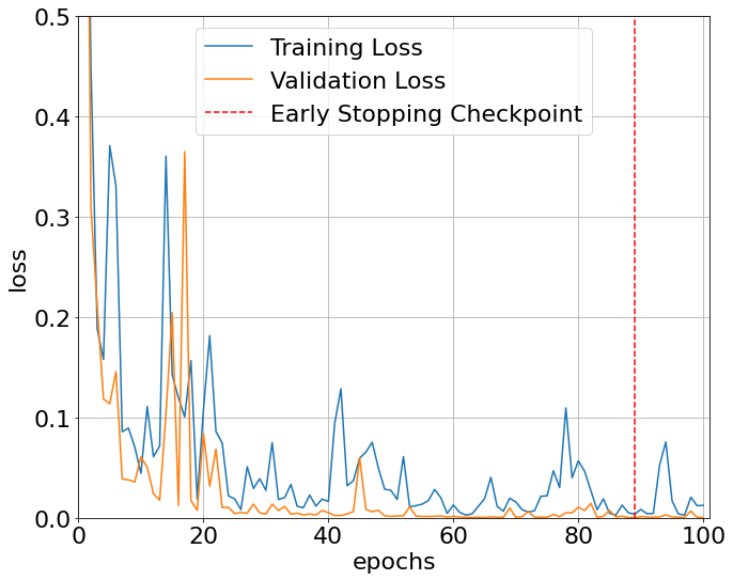

Figure 4.9: Training and validation loss using the SGD optimizer as a function of the trained epochs.

To evaluate the accuracy of our NN model, it was tested with the test dataset of 60 experimental images. With the test dataset, the following accuracies were reached:

Accuracy of class 0: 100% (31/31)
Accuracy of class 1: 100% (6/6)
Accuracy of class 2: 95 % (22/23)
Accuracy (overall): 98 % (59/60)

In contrast to the model which was trained with simulated data (see subsection 4.2.3), almost all experimental images from each class are classified correctly with the model trained with the experimental data.
Even if the neural network had never seen these images of the test dataset, it should be noted that the images come from the same 14 different experiments as the training and validation data. Within one experiment the images were very similar to each other. It is important to ensure that the neural network is also likely to correctly classify images from other experiments, which is why the neural network also was tested with 22 images from another GIWAXS experiment with perovskites thin films (NAIB-1LI-2). These images were not part of the training, validation or test dataset and were all part of class 0. The model classified all the images correctly as rings (class 0). One of the images with the appropriate classification is shown in Figure 4.10.
These results show that the approach of training the model using the experimental data is quite promising. The testing accuracy is significantly higher (98 %) compared to the model trained on simulated data (35%), and the performance is robust to withstanding different hyperparameters and the optimizer. Furthermore, it was shown that the trained model can also correctly classify the images from other experiments. However, the classification would probably need to be tested on more experimental images from different experiments and beam-lines to investigate its robustness and generalization error. If the images of an experiment differ strongly from those tested thus far, it is not certain that the neural network would be able to correctly recognize such images. Such an investigation is left for future research.
In summary, training with the experimental data seems to be the most optimal and robust approach for these types of classification tasks. It avoids all the problems with the data representation that can appear when training on the simulated data. Furthermore, this method does not require a lot of data for training, and the labelling process is not time-consuming.
However, using the simulated data for these types of classification would allow the algorithm to perform more complex classifications in the future without needing to find the appropriate experimental data for training. The current results show that the simulated data cannot be used so far for this task. It is important to investigate the underlying reasons for this in order to be able to improve the simulation process in the future. A few potential approaches to investigating this were already discussed previously. In the following, a simple approach to evaluate the quality of the simulation is presented. For that purpose, the simulated data via the model successfully trained on the experimental data was classified.

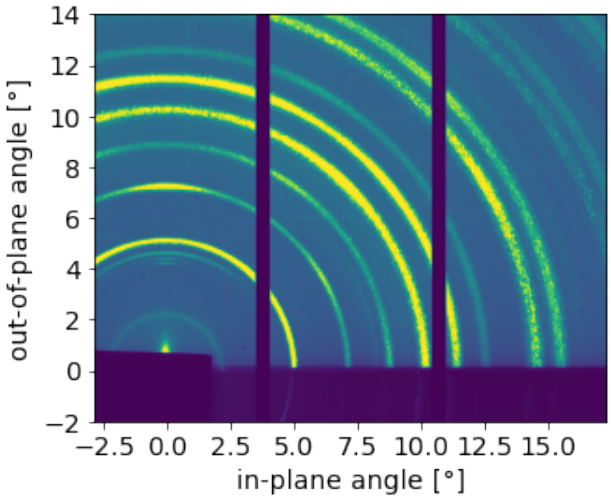

Figure 4.10: Classification result of the model trained with experimental data for an experimental image with rings (class 0) with an accuracy of 100%. Image recorded with a photon energy of 22 keV and a sample-detector-distance of 295 mm.

4.2.5 Model trained with experimental data tested on simulated data

To classify the simulated images, the ResNet model trained with experimental data was used. The following accuracies were reached:

Accuracy of class 0:	99%	(326/327)
Accuracy of class 1:	56%	(190/338)
Accuracy of class 2:	0 %	(0/335)
Accuracy (overall):	51%	(516/1000)

Nearly all the images from class 0 were classified correctly. This in turn shows the simulated images with rings (class 0) are sufficiently similar to the experimental ones, as the model was able to detect them very well. Only 56% of the images with segments (class 1) were classified correctly. The trained model classified approximately half of the simulated images with segments (class 1) as class 0 (rings). The mixed simulated images (class 2) were all classified as rings (class 0). However, it can be concluded that the simulated images with segments (class 1), and especially the images with rings and segments (class 2), are not similar enough to the experimental images.
With this test of the trained model on simulated data one can conclude that simulation can be improved, especially for the images from class 1 and 2. This could have positive effects on the classification performance of experimental data with the model trained on simulated data.

4.2.6 Summary

The goal was to create a NN model that is able to classify GIWAXS images based on three different classes (rings, segments or both). The possible use of simulated data for training the NN was investigated. The results indicate the existence of certain fundamental problems of such an approach.

The difference between simulated and experimental data may dramatically affect the results because of the uncontrolled "attention" mechanism that determines the output of the classification model. In this case, some visual features presented on the mixed-type simulated data as well as on all the experimental images distorted the performance of the model. The deeper investigation of these features in order to improve the simulations is necessary, and it is left for the future researches. In contrast, the network trained with the experimental data was shown to correctly recognize and classify experimental images from the three different classes. Except for very small deviations, which are often unavoidable in image recognition, the developed neural network provides good results for classification at 98 % accuracy.

The question still remains whether or not the model is capable of classifying GIWAXS diffraction patterns of other experiments. This problem is common for machine learning applications with limited amounts of data for testing. The excellent results of the classification on the testing dataset indicate that the model can be successfully applied at least to a subset of GIWAXS images. However, a more systematic testing may be required.

The next section is devoted to the segmentation task. Unlike in the classification approach, the segmentation task requires explicit information about the Bragg peaks positions, so that the model would can learn the correct features on the simulated data. Hence, there is a possibility that the simulated data is more suited for segmentation.

4.3 Segmentation of the powder diffraction patterns

In addition to the classification task, part of this work was to train a neural network model that is able to detect and localize Debye-Scherrer rings and ring segments. Dividing images this way into information-rich and information-poor regions is commonly called "segmentation". Typically, CNNs are used for classification tasks where the output of an image is a single class label, as described in section 4.2. However, for many visual tasks, especially in physical image processing of scattering data, the desired output should include a localization, i.e. each pixel should be given a class label. The algorithm for this segmentation task was designed to create a binary mask of a scattering image as shown in Figure 4.11.

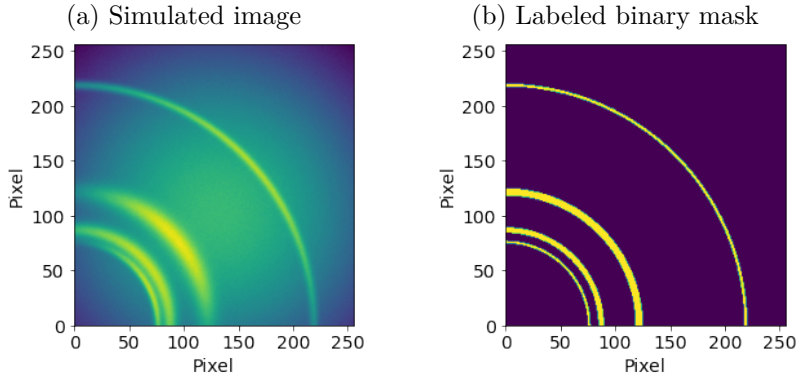

Figure 4.11: Segmentation of simulated scattering image. (a) shows the input image and (b) the binary mask. The images dimensions were 256×256 pixels.

In addition to the physical application of scattering data, there is great interest in segmentation in the biomedical field of image recognition. One of the most used segmentation models developed for biomedical purposes is a fully convolutional network [46] known as U-Net due to its unique U-shape architecture (see Figure 4.12). The main idea and the architecture of this network will be described in the following section.

4.3.1 Setup of the neural network

As mentioned before, the U-Net is a modified fully convolutional network (FCN) [46]. The main idea behind it is to extend a common contracting network into successive layers in which the resolution of the output is increased. This is possible by replacing pooling operators with upsampling operators. High-resolution features are combined from the contraction path with the upsampled output which enables localization. The learning process includes the ability of a successive convolution layer to assemble a more precise output with this information.

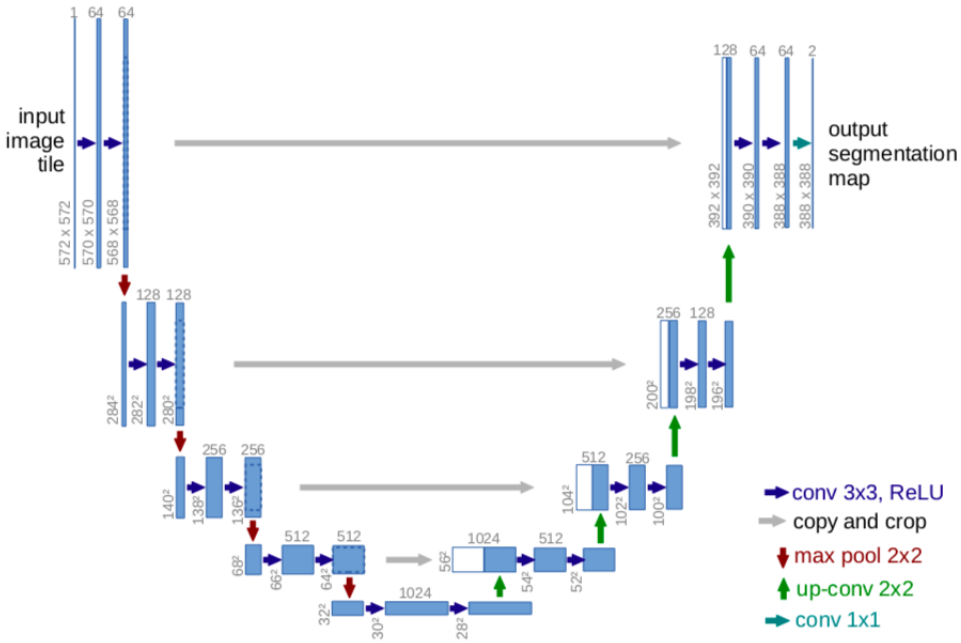

Figure 4.12: Architecture of the U-Net [47]. Each blue box represents a multi-channel feature map. The number of channels is noted at the top of the box. The x-y size is specified at the bottom left of the box. White boxes are copied feature maps. The coloured arrows mark the different operations.

The U-Net architecture is shown in Figure 4.12. It consists of a contraction part (left side) and an upsampling/expansion part (right side). In total, the network has 23 convolution layers and about 29 million trainable parameters. On one side of the network, the **contraction path** follows the typical convolution network structure described in subsection 4.2.1 and contains a pretrained ResNet-18. There are many feature channels that allow the network to propagate context information into higher resolution layers. The network includes a repeated application of two 3×3 convolutions without padding, each followed by a ReLU and a 2×2 maximum pooling operation with step 2 for downsampling. Each downsampling step doubles the number of feature channels. In this part, the previous steps increase the "what" and reduce the "where".
On the other side of the network is the **expansion path** which is responsible for creating a high-resolution segmentation map. It contains an upsampling of the feature map, followed by a 2×2 convolution (up-convolution), which halves the number of feature channels. There is also a concatenation with a high-resolution feature map from the contracting path and two 3×3 convolutions, each of which are followed by a ReLU. Cropping is necessary here, because the edge pixels are lost in each convolution. A 1×1 convolution is applied to the last layer. This ensures that each 64-component feature vector is mapped to the desired number of classes. As a result, the two paths are more or less symmetric and result in a U-shaped architecture.
The network does not contain fully connected layers. The segmentation map contains only pixels

for which the full context is available in the input image. This allows the segmentation of images of any size using an overlap-tile strategy. Since pixels at the edge of the image can also be relevant, the missing context is extrapolated by mirroring the input image. This allows the network to be applied to larger images.
Moreover, data supplementation was employed by applying elastic deformations to the available training images. Doing this allows the network to learn invariance in deformations without having to see these transformations in the corresponding image. Another challenge that U-Net has taken on is the separation of touching objects of the same class. For this purpose, the use of a weighted loss is used, where the separated background labels between touching objects are given a large weight in the loss function.

Loss function

In the segmentation task there are two different classes. For the loss the binary cross entropy (BCE) loss and the dice loss is used. The loss is thus calculated as:

$$\text{loss} = \text{bce} \cdot \text{bce}_{\text{weight}} + \text{dice} * (1 - \text{bce}_{\text{weight}}) \tag{4.4}$$

where bce is the BCE loss (see Equation 4.1), $\text{bce}_{\text{weight}}$ is the weighted positive label, and dice is the dice coefficient. The dice is a statistical instrument used to measure the similarity between two data sets, and is possibly the most widely used tool in the validation of image segmentation. In sum, the dice measures the overlap between two objects. When the dice is 1 it denotes a complete overlap. It can be calculated as:

$$\text{dice} = \frac{2|A \cap B|}{|A| + |B|} \tag{4.5}$$

where $|A \cap B|$ represents the elements common between the datasets A and B, and $|A|$ (or $|B|$) the number of elements in the set A (or B). The $|A \cap B|$ is approximated by summing the element-wise multiplication between the prediction and the target mask. Since our target mask is binary, we effectively eliminate all pixels that are not "activated" in the target mask from our prediction. If the prediction for a pixel is low, it will be weighted less.
An experimental image contains just a few objects as about 70% of the image is background. When a segmentation model predicts a mask and does not find any rings or segments, the information imbalance will lead to an accuracy of about 70%.

Optimizers and grid search

Furthermore, the choice was again between the SGD optimizer and the adam optimizer. As with the grid search for classification, only the change of learning rate and optimizer will be discussed for the segmentation task. In order to train the network with the simulated and experimental images a preprocessing of the images is helpful.

4.3.2 Data preparation

As in the classification task in section 4.2, first an attempt was made to train the network using simulated data. The preprocessing of both simulated and experimental data will be described.

Simulated Data

5100 simulated images were generated with dimensions of 256×256 pixels and a corresponding mask was created for each image (see Figure 4.11). Similar to the simulation process of the classification task, peak locations are not constrained by any set of materials, thus there are no q-coordinates on figures with simulated images. The whole dataset was divided into the following subsets:

- Training dataset: 3315 images
- Validation dataset: 765 images
- Test dataset: 1020 images

The contrast in the images was increased through clipping (see Equation 4.2.2). The images were reshaped to a uniform size of 224×224 pixels and normalized by the maximum intensity of the image.

Experimental data

The experimental data was obtained from GIWAXS measurements of different perovskite thin films:

- Project 7 TUD (MK1 - MK6): Data shows formation of $CH_3NH_3PbI_3$ perovskite structure and exhibits domain orientation of perovskite structure and PbI_2.
- RJ series (mixed cation): perovskites thin films $CH_5N_2Cs_{0.1}PbI_{x+2}Br_{0.1}$ (with $x = 0.7 - 1.6$).
- A-MA series: adamantyl-based layered hybrid perovskites ($A_2CH_3NH_3Pb_2I_7$).
- Project 7 2D/3D/2D-3D: GIWAXS data of 2D, 3D and 2D-3D perovskites thin films.

The images were clipped (see Equation 4.2.2) and normalized by the maximum intensity. Figure 4.13 shows the pixel sizes of the experimental data were different, but the aspect ratios were similar (1.2 - 1.4). The images were resized to 224×224 pixels. Due to this transformation the distortion of the experimental images is assumed to be small enough not to have an effect on the neural network performance. Examples of different experiments are shown in Figure 4.13.

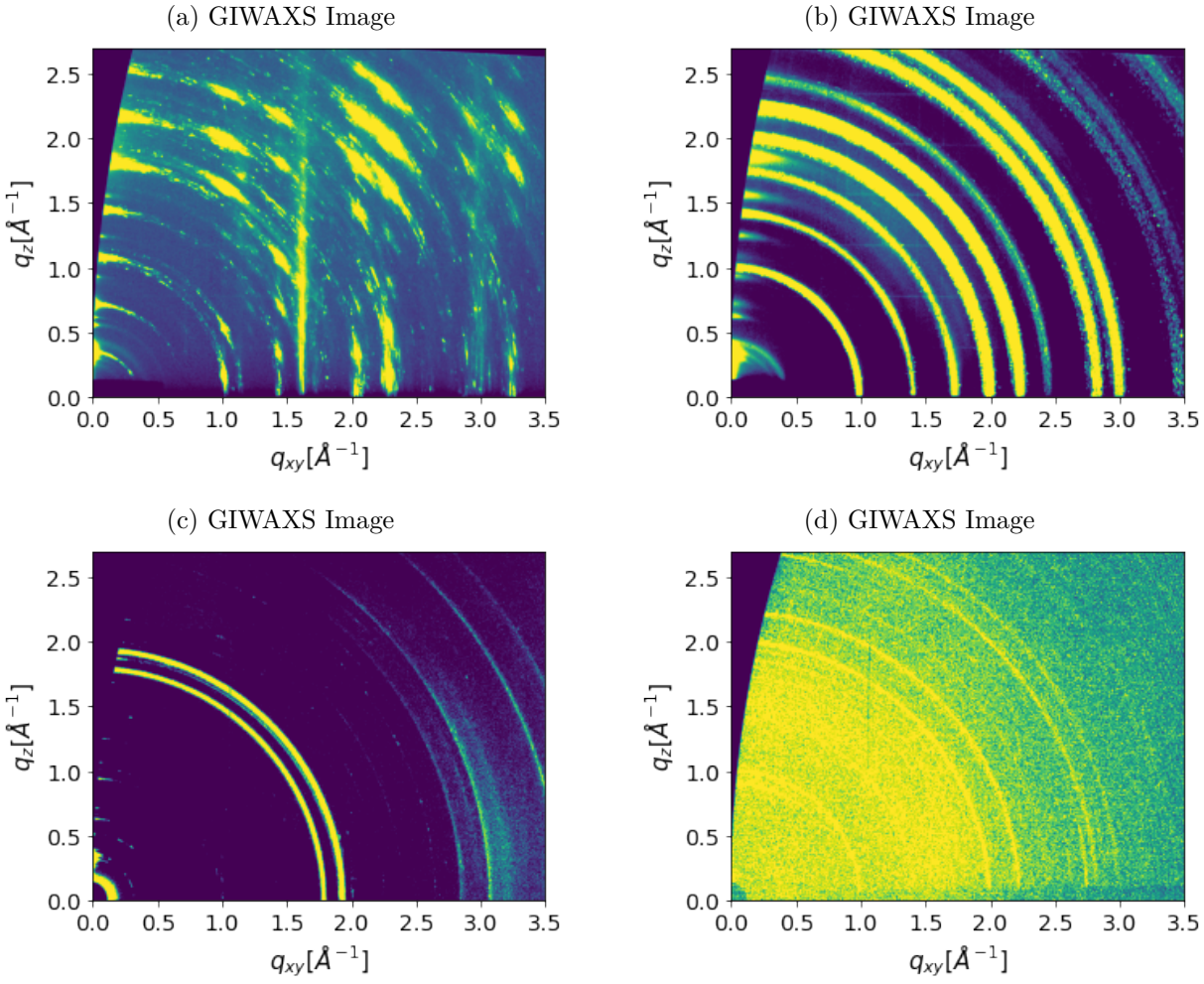

Figure 4.13: (a) shows an image of a GIWAXS experiment with 2D perovskites (Project 7 2D/3D/2D-3D). (b) is a GIWAXS image of 2D-3D perovskites thin film (Project 7 2D/3D/2D-3D). (c) shows a GIWAXS image of adamantyl-based layered hybrid perovskites (A-MA series) and (d) is a GIWAXS image of the RJ series.

Labelling the experimental data

For training the U-Net model, the experimental images have to be labeled (see Figure 4.14) which means the rings and segments all have to be marked. To accomplish this, a Gaussian fit was applied to the radial intensity distribution of each ring or ring segment, and then a mask was created from the marked areas and the background. Through this labelling process, a total of 199 masks for different images were created. Images from different experiments were labeled when possible. Since the result of the segmentation of the U-Net depends strongly on the quality of the labeled data, the rings and segments were marked as accurately as possible.

(a) GIWAXS image

(b) Labeled binary mask

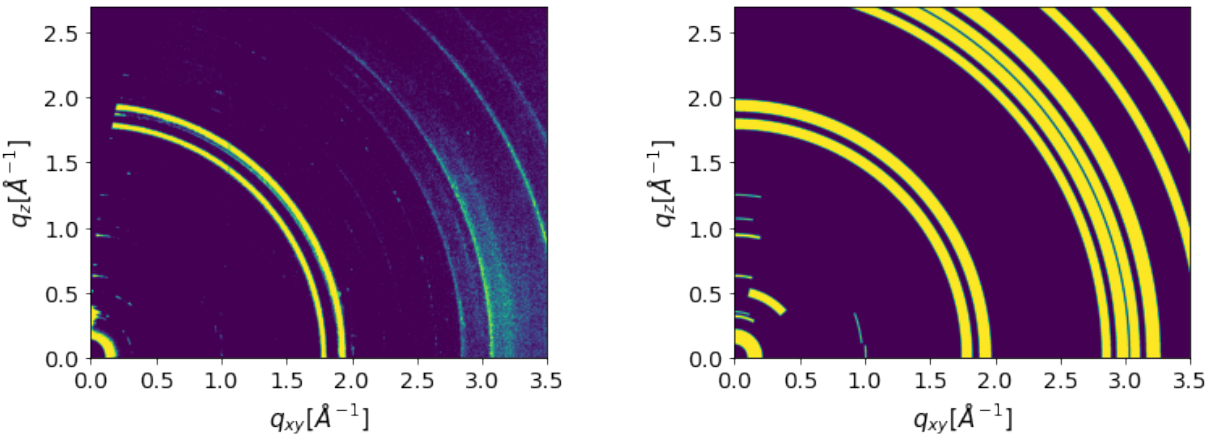

Figure 4.14: (a) shows an experimental scattering image of the A-MA series and (b) shows the labeled binary mask of this image.

The dataset was divided into three parts:

- Training dataset: 130 images
- Test dataset: 40 images
- Validation dataset: 29 images

4.3.3 Results of the network trained with simulated data

After the preprocessing of the simulated data, the U-Net was trained for about 50 epochs with these simulated images. The Figure 4.15 shows the training and the validation loss of the adam optimizer as a function of the epochs. The training loss is slightly larger than the validation loss. The best value of the validation loss was 0.426.

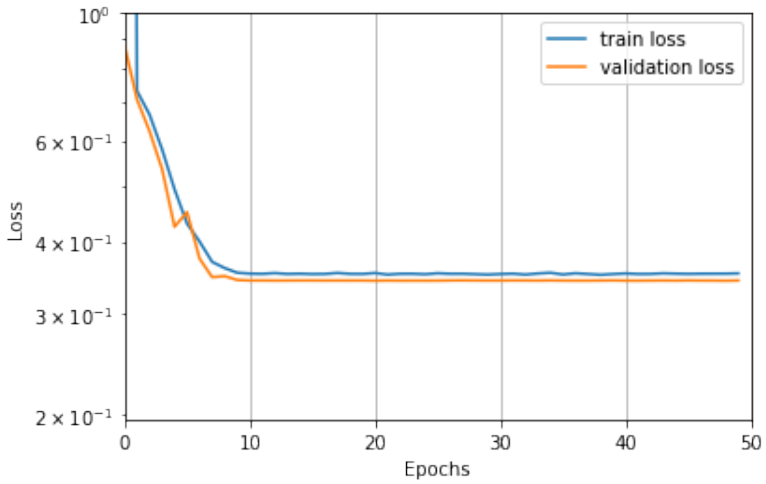

Figure 4.15: Training and validation loss of the adam optimizer as a function of epochs.

To evaluate the accuracy of the model, it was tested with the test dataset of simulated images. The labeled and predicted mask of a simulated images are shown in Figure 4.16. The predicted binary mask (c) in Figure 4.16) look similar to the labeled mask (b). The performance of mask prediction is high for the simulated data.

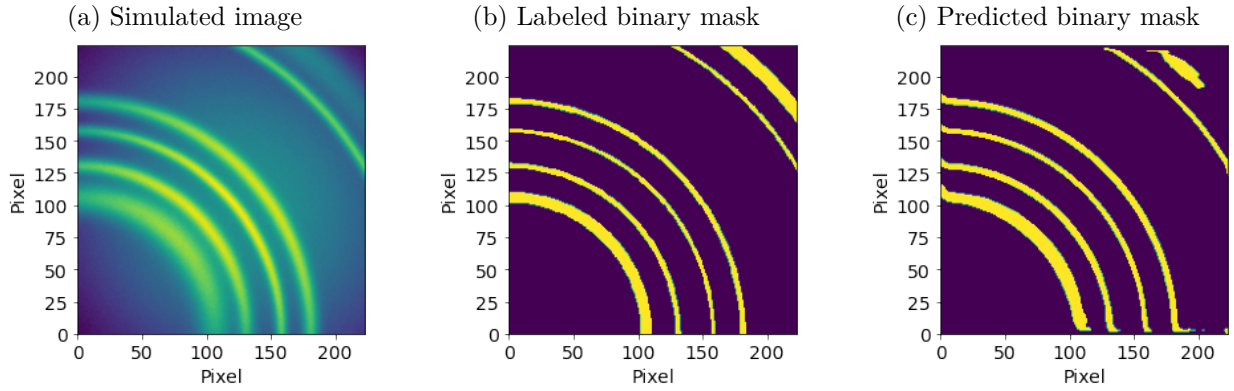

Figure 4.16: Segmentation results of the U-Net model trained with simulated data. (a) shows a simulated image, (b) a labeled binary mask of the simulated image and (d) shows the predicted binary mask of the simulated image. The image dimensions were 224×224 pixels.

Since the model is supposed to segment experimental images, it is necessary to test it on experimental images. As shown in Figure 4.17, the predicted mask (c) is not similar to the labeled mask (b) of the experimental image (a). In this case it can be assumed the simulated images are not similar enough to the experimental ones. For some of the images the model interpreted some of the rings properly. Although many of the rings were missing, the rings with the highest intensity, which are most distinct from the background, were still identified by the model. If the background is not clearly distinguishable from the objects, the model has issues detecting the scattering dots and rings. This may be due to the fact that in all simulated images the rings and dots are much more distinct from the background. The model had not seen such images before and thus treated the speckled background as many small scattering dots. However, making predictions for this type of image is important. The network's performance with these images demonstrates, that it is not sufficiently trained and the performance is not yet good enough to detect other structures in experimental scattering images.

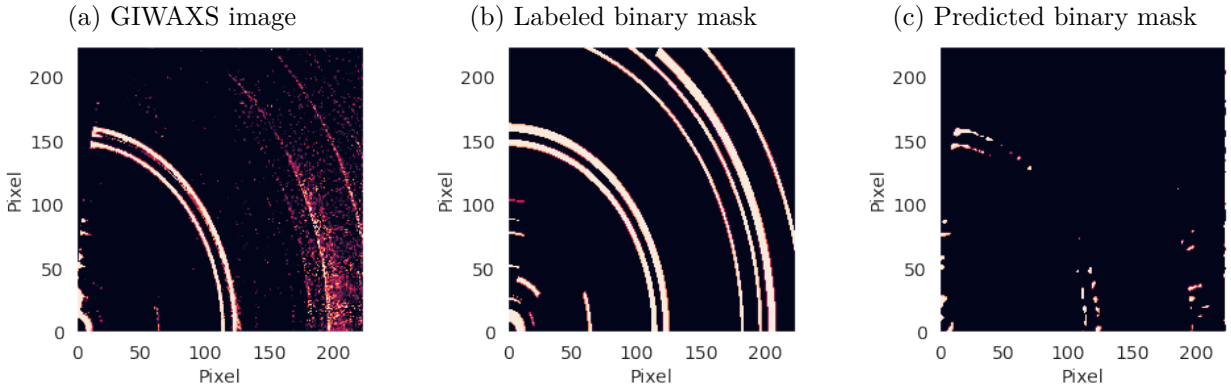

Figure 4.17: Segmentation results of the U-Net model trained with simulated data. Figure (a) shows an experimental GIWAXS image, (b) a labeled binary mask of the experimental image and (c) shows the predicted binary mask of the experimental image. The images dimensions were 224 × 224 pixels. The maximum value of q_{xy} is 3.5 Å^{-1} and the maximum value of q_z is 2.7 Å^{-1}.

It would be possible to improve this trained network by adding more images with noisy backgrounds and with rings closer together to the simulated dataset. Since the performance rate on experimental images was low, no further quantitative analysis of this trained model is presented.
To achieve better prediction masks of experimental images, the network was trained with experimental data. The results will be described in the next subsection.

4.3.4 Results of the network trained with experimental data

The U-Net was trained with the labeled experimental images over 50 epochs with the adam optimizer and a learning rate of 0.01. If one looks at the dice and the BCE loss separately (see Figure 4.18), it is noticeable that the dice loss is larger. This separation is useful for setting the value of $\text{bce}_{\text{weight}}$ to get the lowest loss. The lowest value of validation dice loss is 0.59. The validation BCE loss is decreased to a value of 0.53. In our case the BCE loss is weighted with $\text{bce}_{\text{weight}} = 0.6$ more than the dice loss ($\text{dice}_{\text{weight}} = 0.4$). After the initial epochs, the training losses are slightly larger than the validation losses, and the model does not overfit the data.

Figure 4.18: A live plot during training the neural network. The dice and BCE loss as a function of epochs with the training and the validation loss of the adam optimizer.

To verify the accuracy of the model, it was tested with the test dataset of experimental scattering images. As shown in Figure 4.19 (c) (f) and (i) the predicted binary masks of the images are similar to experimental images (a) (d) and (g). If the experimental images contain rings, the model can recognize most of them. While the model worked well for this kind of data, if the background is noisy as seen in Figure 4.19 (a) and (g), the model's prediction capability decreases. Even though such images were included in the training set for this trained network, the model was yet unable to properly segment such images.

Quantitative results of the performance of the model using experimental data

The number of correct predicted pixels increased to 78 % when the model was trained with experimental data instead of simulated data. However, as previously stated, it is not enough to look at the accuracy of the prediction. When evaluating a segmentation model, we usually classify our predictions into four categories: true positives, false positives, true negatives, and false negatives. A confusion matrix describes these as follows:

	Predicted = 0	Predicted = 1
True label = 0	True Negative	False Positive
True label = 1	False Negative	True Positive

Table 4.3: Confusion matrix describing the relationship between the true and false predicted labels.

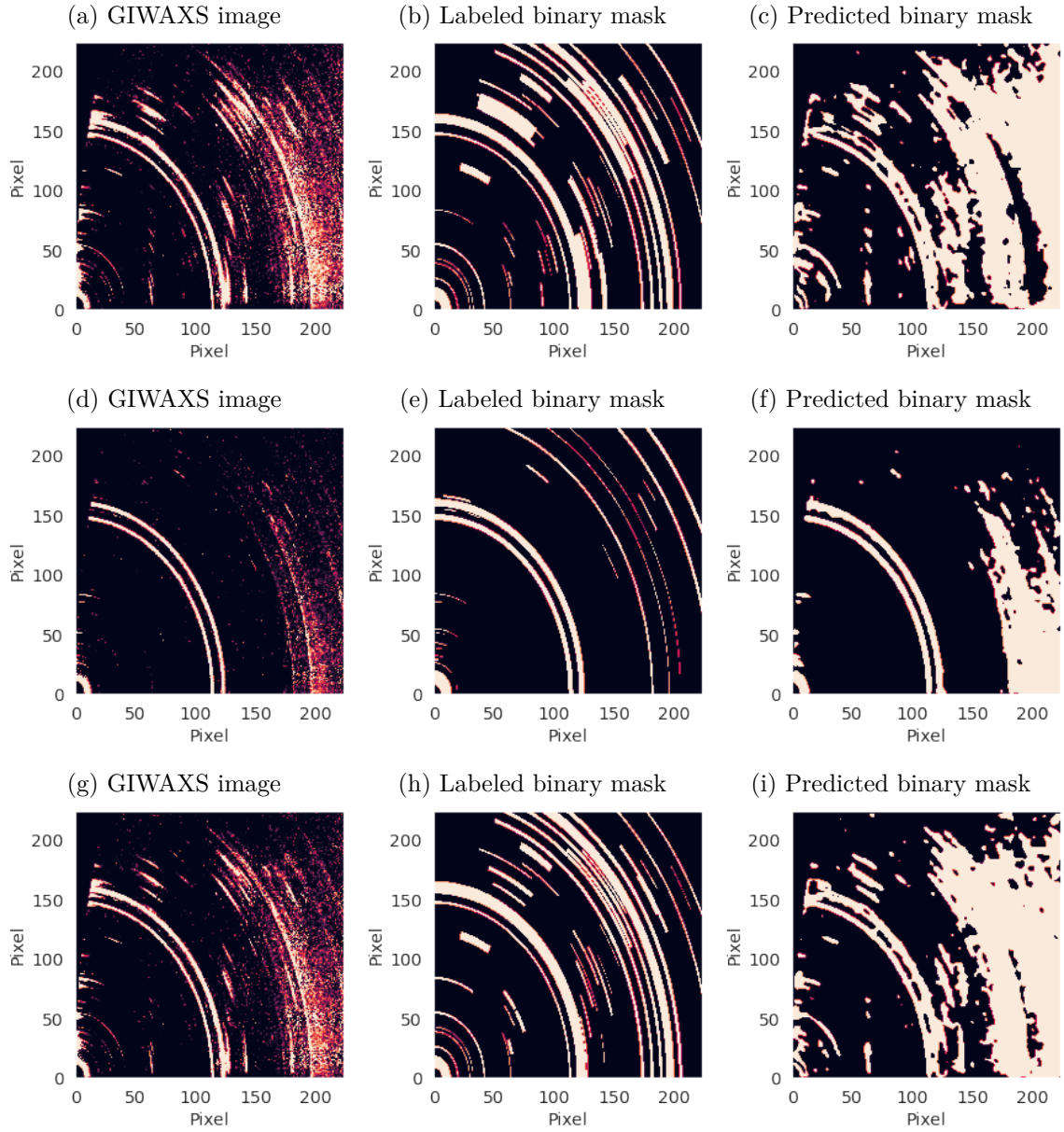

Figure 4.19: Segmentation results of the U-Net model trained with experimental data. (a) (d) (g) show experimental images and (b) (e) (h) show labeled binary masks of the experimental images (a) (d) (g) respectively. (c) (f) (i) show the predicted binary masks of experimental images. The image dimensions were 224×224 pixels. The maximum value of q_{xy} is 3.5 Å^{-1} and the maximum value of q_z is 2.7 Å^{-1}.

With the correlations from section 4.3.4 different metrics are introduced:

- **Recall** indicates the fraction of the positive objects (pixels in this case) which are predicted correctly.

$$\text{recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{4.6}$$

- **Precision** indicates how many of all positive predictions are true positive predictions.

$$\text{precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{4.7}$$

- **Pixel accuracy** is defined as:

$$\text{pixel accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}} \tag{4.8}$$

  and simply reports the percent of pixels in the image which were classified correctly.

These metrics are calculated for the training, validation and test datasets of the experimental data. The accuracy for all datasets was 78% (see Figure 4.20), but the recall rate was approximately 30%. This low recall indicates that many of the scattering dots and rings were not recognized by the model. This recall must be increased in order to improve the network, which could be accomplished by weighting the positive levels higher or to further reduce the dice loss. The precision for the dataset was 54%, which means that slightly more than half of the positive predictions were truly positive predictions. It is likely that the neural network performance could be improved by further refining the labelling process.

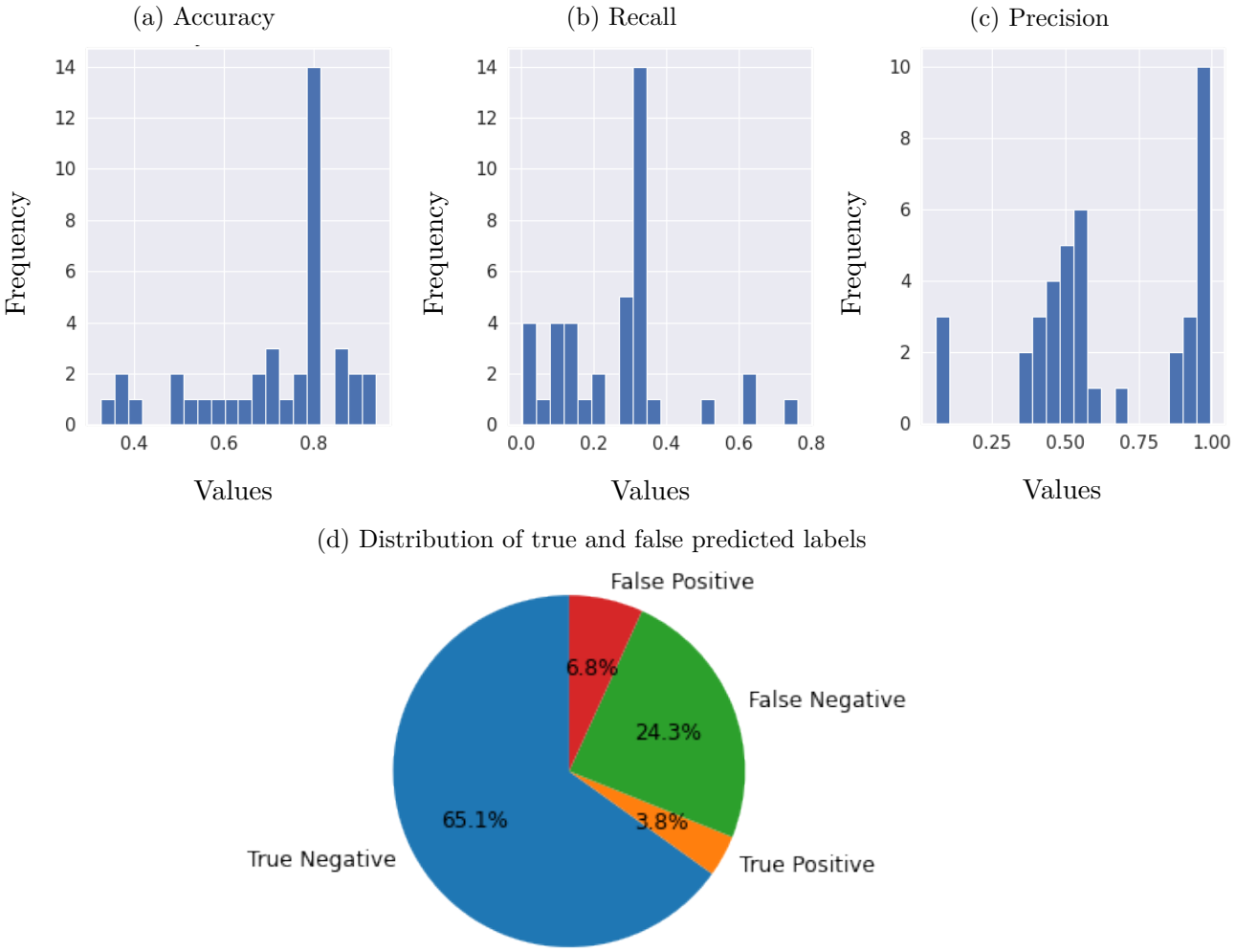

Figure 4.20: Metrics of the experimental test dataset (44 GIWAXS images). The frequency as a function of the value between 0 and 1: Histogram of accuracy (median is 78%), of recall (median is 30%) and of precision (median is 54%). (d) gives a visual overview of the true and false predicted labels and the not predicted labels.

4.3.5 Summary and discussion of the segmentation results

The segmentation task was much more complex than the classification performed in the previous section, partially due to the size of the output. Unlike the classification task where the output dimension was equal to the number of different classes (3 in this case), with the segmentation task the size of output was equal to the number of pixels per image: $224 \times 224 = 50176$. Hence, it is not surprising that it is much harder to achieve high performance for segmentation. The results show that some principal changes are required to improve the performance.
First of all, the model trained on the experimental data still has a relatively low performance. There are several possible reasons for this, including:

- An insufficient number of images available for training. The labelling process is time-consuming, and this problem is inherent to such an approach.
- The data was grouped into several different experiments which are very similar within the group in terms of noise, contrast, etc, but vary significantly between the groups. Given that size of each group is relatively low, it is hard to extract some general rules. This is also related to the previous point.
- A low capacity of the model. To rectify this, and one could try using a deeper model with more parameters.
- Data labelling quality. The labeling process is difficult and time-consuming, and it is hard to maintain the desired quality. Due to this, some of the segments are not labeled and some of the labels do not correspond to real segments. As a result, the data is very difficult to learn from due to the lack of stable visual features, and the only possible way to learn it would be by memorizing every pixel in every image. Combined with the previous point, when the model capacity is relatively low for such a task, the model only learns the basic features which are consequently labeled correctly.
- Perhaps another neural network model is better suited.

In conclusion, the segmentation task is complex and requires a lot of training data which is not easy to prepare. The lack of data would normally lead to overfitting through the memorization of the training set, but it is highly unlikely that the model could memorize over $50176 \times 130 \approx 6.5$M pixels since it only has approximately 23M parameters. Instead, during the training the model learns the stable and easy to recognize features. However, the most incorrectly predicted pixels do not lead to any stable training since the corresponding parameter changes are mutually exclusive (partly due to imperfect labelling and partly due to the grouped diversity of data). Thus, the achieved loss is a stable local minimum which corresponds to the basic features which were labeled correctly or to certain experiments which were easier to learn. The last point could be tested through calculating the metrics for each experiment separately. However, this is left for future research.

5 Conclusion and outlook

5.1 Conclusion

In this thesis two problems related to automated analysis of GIWAXS diffraction patterns were addressed. First, the classification into patterns with Debye-Scherrer rings, with diffraction spots or with both and second the segmentation of the images into areas with and without Bragg reflections. The classification task was carried out with very promising results after it was trained on experimental data, with an accuracy of 98% for the experimental data. Nevertheless, further systematic testing on a wider set of experimental data should be performed to generalize this result. The open question, whether the model is capable of classifying data from other GIWAXS experiments, is a common one for machine learning applications. The limited amount of data for testing is likely the main bottleneck. Based on our results, however, it can be concluded that the classification can achieve satisfactory results on a subset of GIWAXS data. In general, the automated separation of diffraction patterns based on the type of Bragg peaks was achieved. However, the simulated data was not well-suited for this task most likely due to the uncontrolled attention mechanisms in classification, and also because of the dissimilarities between the simulation and the experimental data.
The segmentation task is generally a more complex and, thus, more ambitious approach. The results achieved in our experiment were still limited, but they helped to identify some crucial points that have to be considered when improving this approach. The main focus should be put on the data which is used for training as it is important that one has the proper amount of data, that the variability of the data is not grouped into similar looking experiments, and that the data is properly labeled. The trained model was not yet able to predict binary masks of the experimental data with high enough accuracy for practical applications. However, the performance was generally better when the model was trained on experimental data instead of simulated data. It is possible to further improve this NN, by optimizing the labeling process to obtain more and precise labels or by using a deeper model with more parameters. In addition to these possible solutions, one could also focus on the optimization of simulation process.

5.2 Outlook

The field of analyzing scattering data using deep learning methods is rapidly developing, and some of the results reported in this thesis show a huge potential in exploiting computer vision tools for analyzing experimental data. A few fundamental challenges were discussed in the thesis, including

that the simulated data should be more similar to the experimental data and the issue of uncontrolled attention, which could be solved via visualization of the attention of the model and changing the simulated data accordingly. Another way to solve this problem would be to provide additional information about which part of the image the model should pay attention to. The dissimilarities between simulated and experimental data can also be studied via machine learning tools. This may solve the problem of satisfying some requirements for the experimental data which are hard to identify otherwise. The segmentation performance could be improved through the separate calculation of metrics for each experiment or by using the other ideas described in subsection 4.3.5. On the one hand one can see that it is possible to achieve excellent performance using experimental data through classification with DL models. On the other hand, the results of the segmentation task also demonstrated that there are several limitations to this approach. A few potential solutions to this issue were carried out and others, which remain open for further research, were discussed in this thesis.

6 List of Acronyms

adam adaptive moment estimation

AI artificial intelligence

BCE binary cross entropy

CNN convolutional neural network

DL deep learning

FCN fully convolutional network

GISAXS grazing-incidence small-angle X-ray scattering

GIWAXS grazing-incidence wide-angle X-ray scattering

ML machine learning

MSE mean squared error

NN neural network

PSCs perovskites solar cells

ReLU rectified linear unit

ResNet residual network

SGD stochastic gradient descent

Bibliography

[1] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[2] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *ArXiv:1512.03385*, 2015.

[3] Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *ArXiv:1406.1078*, 2014.

[4] Li Deng, Geoffrey E. Hinton, and Brian Kingsbury. New types of deep neural network learning for speech recognition and related applications. *IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 8599–8603, 2013.

[5] B. Wang, K. Yager, D. Yu, and M. Hoai. X-ray scattering image classification using deep learning. In *2017 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 697–704, 2017.

[6] Clément Douarre, Richard Schielein, Carole Frindel, Stefan Gerth, and David Rousseau. Transfer learning from synthetic data applied to soil–root segmentation in x-ray tomography images. *Journal of Imaging*, 4:65, 2018.

[7] Woon Bae Park, Jiyong Chung, Jaeyoung Jung, Keemin Sohn, Satendra Pal Singh, Myoungho Pyo, Namsoo Shin, and Kee-Sun Sohn. Classification of crystal structure using a convolutional neural network. *International Union of Crystallography*, 4(4):486–494, 2017.

[8] Hiroyuki Ikemoto, Kazushi Yamamoto, Hideaki Touyama, Daisuke Yamashita, Masataka Nakamura, and Hiroshi Okuda. Classification of grazing-incidence small-angle X-ray scattering patterns by convolutional neural network. *Journal of Synchrotron Radiation*, 27(4):1069–1073, 2020.

[9] Shuai Liu, Charles N. Melton, Singanallur Venkatakrishnan, Ronald J. Pandolfi, Guillaume Freychet, Dinesh Kumar, Haoran Tang, Alexander Hexemer, and Daniela M. Ushizima. Convolutional neural networks for grazing incidence x-ray scattering patterns: thin film structure identification. *Materials Research Society Communications*, 9(2):586–592, 2019.

[10] Alessandro Greco, Vladimir Starostin, Christos Karapanagiotis, Alexander Hinderhofer, Alexander Gerlach, Linus Pithan, Sascha Liehr, Frank Schreiber, and Stefan Kowarik. Fast fitting of reflectivity data of growing thin films using neural networks. *Journal of Applied Crystallography*, 52(6):1342–1347, 2019.

[11] Nouamane Laanait, Zhan Zhang, and Christian M. Schlepütz. Imaging nanoscale lattice variations by machine learning of x-ray diffraction microscopy data. *Nanotechnology*, 27(37):374002, 2016.

[12] Janis Timoshenko, Deyu Lu, Yuewei Lin, and Anatoly Frenkel. Supervised machine learning-based determination of three-dimensional structure of metallic nanoparticles. *The Journal of Physical Chemistry Letters*, 8:5091–5098, 2017.

[13] A. Artasanchez and P. Joshi. *Artificial Intelligence with Python: Your Complete Guide to Building Intelligent Apps Using Python 3. X and TensorFlow 2, 2nd Edition.* Packt Publishing, 2017.

[14] Charu C. Aggarwal. *An Introduction to Neural Networks.* Springer International Publishing, Cham, 2018.

[15] Tom M. Mitchell. *Machine Learning.* McGraw-Hill, New York, 1997.

[16] Anurag Bhardwaj Di, Wei and Jianing Wei. *Deep Learning Essentials: Your hands-on guide to the fundamentals of deep learning and neural network modeling.* 2018.

[17] Haider Khalaf Jabbar and Rafiqul Zaman Khan. Methods to avoid over-fitting and under-fitting in supervised machine learning (comparative study). *Computer Science, Communication and Instrumentation Devices*, pages 163–172, 2014.

[18] Michael A. Nielsen. *Neural Networks and Deep Learning.* Determination Press, 2015.

[19] Kagan Tumer and Joydeep Ghosh. Bayes error rate estimation using classifier ensembles. *International Journal of Smart Engineering System Design*, 5:95–109, 2003.

[20] David H. Wolpert. The lack of a priori distinctions between learning algorithms. *Neural Computation*, 8(7):1341–1390, 1996.

[21] Wolfgang Ertel. *Grundkurs Künstliche Intelligenz: Eine praxisorientierte Einführung.* Springer Vieweg, Wiesbaden, 4th edition, 2016.

[22] David Baillot. Why are neuron axons long and spindly? *Medical Xpress*, 2018.

[23] Andrei Nikolaevich Kolmogorov. On the representation of continuous functions of many variables by superposition of continuous functions of one variable and addition. *Doklady Akademii Nauk SSSR*, 114(5):953–956, 1957.

[24] Robert Hecht-Nielsen. Kolmogorov's mapping neural network existence theorem. In *Proceedings of the international conference on Neural Networks*, volume 3, pages 11–14. IEEE Press New York, 1987.

[25] Abien Fred Agarap. Deep learning using rectified linear units. *ArXiv:1803.08375*, 2018.

[26] Chigozie Nwankpa, Winifred Ijomah, Anthony Gachagan, and Stephen Marshall. Activation functions: Comparison of trends in practice and research for deep learning. *ArXiv:1811.03378*, 2018.

[27] Andreas Kroll and Ralf Mikut. Computational intelligence. 56(7):335–338, 2008.

[28] Luis Camuñas-Mesa, Bernabé Linares-Barranco, and Teresa Serrano-Gotarredona. Neuromorphic spiking neural networks and their memristor-cmos hardware implementations. *Materials*, 12:2745, 2019.

[29] H. Ramchoun, M. A. Janati Idrissi, Y. Ghanou, and M. Ettaouil. Multilayer perceptron: Architecture optimization and training with mixed activation functions. In *Proceedings of the 2nd International Conference on Big Data, Cloud and Applications*, BDCA'17, New York, NY, USA, 2017. Association for Computing Machinery.

[30] Uwe Lämmel. *Künstliche Intelligenz*. Carl Hanser Verlag GmbH Co. KG, 2008.

[31] D. Rumelhart, G. Hinton, and R Williams. Learning representations by back-propagating errors. *Nature*, 323:533—536, 1986.

[32] Henry J. Snaith. Perovskites: The emergence of a new era for low-cost, high-efficiency solar cells. *The Journal of Physical Chemistry Letters*, 4(21):3623–3630, 2013.

[33] Yaoguang Rong, Yue Hu, Anyi Mei, Hairen Tan, Makhsud I. Saidaminov, Sang Il Seok, Michael D. McGehee, Edward H. Sargent, and Hongwei Han. Challenges for commercializing perovskite solar cells. *Science*, 361(6408), 2018.

[34] Martin Green, Anita Ho-Baillie, and Henry Snaith. The emergence of perovskite solar cells. *Nature Photonics*, 8:506–514, 2014.

[35] Alessandro Greco, Alexander Hinderhofer, M. Ibrahim Dar, Neha Arora, Jan Hagenlocher, Andrey Chumakov, Michael Grätzel, and Frank Schreiber. Kinetics of ion-exchange reactions in hybrid organic–inorganic perovskite thin films studied by in situ real-time x-ray scattering. *The Journal of Physical Chemistry Letters*, 9(23):6750–6754, 2018.

[36] Neha Arora, M. Ibrahim Dar, Alexander Hinderhofer, Norman Pellet, Frank Schreiber, Shaik Mohammed Zakeeruddin, and Michael Grätzel. Perovskite solar cells with cuscn hole extraction layers yield stabilized efficiencies greater than 20%. *Science*, 358(6364):768–771, 2017.

[37] Neha Arora, M. Ibrahim Dar, Alexander Hinderhofer, Norman Pellet, Frank Schreiber, Shaik Mohammed Zakeeruddin, and Michael Grätzel. Perovskite solar cells with cuscn hole extraction layers yield stabilized efficiencies greater than 20%. *Science*, 358(6364):768–771, 2017.

[38] James Thewlis. *Concise dictionary of physics and related subjects.* New York, Pergamon Press, 1973.

[39] Mats Johnsson and Peter Lemmens. *Crystallography and Chemistry of Perovskites.* 2007.

[40] Yuhang Liu, Seckin Akin, Linfeng Pan, Ryusuke Uchida, Neha Arora, Jovana V. Milić, Alexander Hinderhofer, Frank Schreiber, Alexander R. Uhl, Shaik M. Zakeeruddin, Anders Hagfeldt, M. Ibrahim Dar, and Michael Grätzel. Ultrahydrophobic 3d/2d fluoroarene bilayer-based water-resistant perovskite solar cells with efficiencies exceeding 22%. *Science Advances*, 5(6), 2019.

[41] Zijun Yi, Najib Haji Ladi, Xuxia Shai, Hao Li, Yan Shen, and Mingkui Wang. Will organic–inorganic hybrid halide lead perovskites be eliminated from optoelectronic applications? *Nanoscale Adv.*, 1:1276–1289, 2019.

[42] Jens Als-Nielsen and Des McMorrow. *Elements of modern X-ray physics.* John Wiley & Sons, 2011.

[43] Rafael Quintero-Bermudez, Aryeh Gold-Parker, Andrew Proppe, Rahim Munir, Zhenyu (Kevin) Yang, Shana Kelley, Aram Amassian, Michael Toney, and Edward Sargent. Compositional and orientational control in metal halide perovskites of reduced dimensionality. *Nature Materials*, 17:900—907, 2018.

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

[45] Sebastian Ruder. An overview of gradient descent optimization algorithms. *ArXiv:1609.04747*, 2016.

[46] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. *ArXiv:1411.4038*, 2014.

[47] O. Ronneberger, P.Fischer, and T. Brox. U-net: Convolutional networks for biomedical image segmentation. In *Medical Image Computing and Computer-Assisted Intervention (MICCAI)*, volume 9351 of *LNCS*, pages 234–241. Springer, 2015.