Deep Learning for nanotechnology: crystal growth characterization and nano-photonics inverse design

Abdourahman Khaireh-Walieh

RÉPUBLIQUE FRANÇAISE
Liberté Égalité Fraternité

Université de Toulouse

Doctorat de l'Université de Toulouse

préparé à l'Université Toulouse III - Paul Sabatier

Apprentissage profond pour les nanotechnologies : caractérisation de la croissance des cristaux et conception inverse en nanophotonique

---

Thèse présentée et soutenue, le 7 octobre 2024 par

Abdourahman KHAIREH WALIEH

École doctorale
GEETS - Génie Electrique Electronique,Télécommunications et Santé : du système au nanosystème

Spécialité
MicroNano Systèmes

Unité de recherche
LAAS - Laboratoire d'Analyse et d'Architecture des Systèmes

Thèse dirigée par
Peter WIECHA

Composition du jury
M. Guilhem ALMUNEAU, Président, CNRS Occitanie Ouest
Mme Glenna DRISKO, Rapporteure, CNRS Aquitaine
M. Demetrio MACIAS, Rapporteur, Université de Technologie de Troyes
M. Peter WIECHA, Directeur de thèse, Université Toulouse III - Paul Sabatier

Membres invités
M. Sébastien PLISSARD, CNRS Occitanie Ouest
M. Alexandre ARNOULT, CNRS Occitanie Ouest

Waxaan shahaadada doktooraaga ah u hibeeyey hooyaday Xaawo Ismaaciil iyo aabbahay Khayre Waalie.

Acknowledgment

I would like to thank all those who have been part of my doctoral journey.

My special thanks go to my thesis supervisor Dr. Peter WIECHA. Thank you for the exemplary supervision, the scientific discussions, your advice and availability which helped me to carry out my research work. It was a great chance to prepare my PhD thesis under your supervision.

I would also like to thank Dr. Alexandre ARNOULT and Dr. Sébastien PLISSARD. It was a pleasure to work with you. My thanks also go to the MPN team for their warm welcome.

My warmest thanks go to my parents and my brothers and sisters. Although you were far away from me, I can't thank you enough for all the lovely time we spent together on my holidays and those online discussions.

A big thank you to my friends. Those moments of joy and relaxation were of paramount importance for me.

I would also like to thank all the staff of my thesis preparation laboratory LAAS-CNRS, GEETS doctoral school, the degree-awarding university Université de Toulouse as well as the french ministry of higher education, research and innovation MESRI for funding my thesis.

Contents

Abstract

Deep Learning, a subset of artificial intelligence, has considerably improved the capabilities of machines in several fields. Deep Learning is based on the architecture of neural networks. This thesis introduces neural networks, some of their different architectures and fields of application, as well as the learning process, the mini-batch stochastic gradient descent. Neural network models are then employed in two applications: crystal growth characterization and nanophotonics inverse design. In the first application, three tasks are performed: the detection of substrate deoxidation using an autoencoder to compress RHEED images, followed by a convolutional neural network to classify sequences of the compressed data into oxidized and deoxidized; the classification of the surface reconstructions into (2×4) and $c(4 \times 4)$ and finally the azimuthal RHEED construction using two architectures, one to track the center of gravity of the specular spot using a semantic segmentation model and the second for determining the orientation of the crystal with respect to the incident electron beam of the RHEED system using residual neural networks. Afterwards, thin slices are trimmed across the specular point and plotted as a function of the azimuthal angle to obtain the azimuthal RHEED. In the second application, the nanophotonics inverse design problem is introduced before implementing Deep Learning architectures to circumvent this problem. The first architecture is the tandem network, a generative model involving two networks, the second one is the conditional variational autoencoder, a variant of the autoencoder and the last architecture is the so-called "neural adjoint" method, a gradient-based optimization method. Furthermore, a graph neural network surrogate model is proposed to represent the nanostructures in the form of graphs and predict the optical properties. Finally, its interpolation and extrapolation capabilities are tested, followed by fine-tuning to include the extrapolation data in the training field of the model.

Keywords:

Deep-Learning RHEED crystal growth nanophotonics inverse design Graph Neural Network

Résumé

Le Deep Learning, un sous-ensemble de l'intelligence artificielle, a considérablement amélioré les capacités des machines dans plusieurs domaines. L'apprentissage profond est basé sur l'architecture des réseaux de neurones. Cette thèse présente les réseaux de neurones, certaines de leurs différentes architectures et domaines d'application, ainsi que le processus d'apprentissage, la descente de gradient stochastique. Des modèles de réseaux de neurones sont ensuite utilisés dans deux applications : la caractérisation de la croissance des cristaux et la conception inverse en nanophotonique. Dans la première application, trois tâches sont effectuées : la détection de la désoxydation du substrat à l'aide d'un autoencodeur pour compresser les images RHEED, suivi d'un réseau de neurones convolutionnel pour classifier les séquences de données compressées en oxydées et désoxydées; la classification des reconstructions de surface en (2×4) et $c(4 \times 4)$ et enfin la construction du RHEED azimutal à l'aide de deux architectures, l'une pour le suivi du point spéculaire à l'aide d'un modèle de segmentation sémantique et la seconde pour déterminer l'orientation du cristal par rapport au faisceau d'électrons incident du système RHEED à l'aide de réseaux de neurones résiduels. Ensuite, de fines tranches sont rognées à travers le point spéculaire et tracées en fonction de l'angle azimutale pour obtenir le RHEED azimutal. Dans la deuxième application, le problème de conception inverse en nanophotonique est présenté avant de mettre en œuvre des architectures d'apprentissage profond pour contourner ce problème. La première architecture est le réseau tandem, un modèle génératif impliquant deux réseaux, la seconde est l'autoencodeur variationnel conditionnel, une variante de l'autoencodeur, et la dernière architecture est la méthode dite de "neural adjoint", une méthode d'optimisation basée sur le gradient. En outre, un modèle de réseau de neurones de graphes est proposé pour représenter les nanostructures sous la forme de graphes et prédire les propriétés optiques. Enfin, ses capacités d'interpolation et d'extrapolation sont testées, suivies d'un réglage fin pour inclure les données d'extrapolation dans le champ d'entraînement du modèle.

Mots clés:

Apprentissage profond RHEED croissance des cristaux nanophotonique conception inverse Réseaux de neurones de graphes

List of Figures

List of Tables

General introduction

Artificial Intelligence (AI) is the science of making machines do things that would require intelligence if done by human being [1]. It encompasses a broad range of technologies aimed at enabling machines to perform tasks that typically require human intelligence, such as learning, reasoning, problem-solving, perception and language understanding. Deep Learning, a subset of AI, involves neural networks with many layers (hence the term "deep") that are capable of learning representations of data with multiple levels of abstraction. Unlike traditional machine learning algorithms, which often require manual feature extraction, deep learning models can automatically discover features from raw data. This capability has led to significant advancements in various domains such as computer vision [2], natural language processing [3], speech recognition [4] and more.
At the heart of deep learning are the above-mentioned neural networks, inspired by the structure and function of the human brain. These networks consist of interconnected layers of neurons, where each neuron applies a transformation to its input and passes the result to the next layer. Training these networks involves adjusting the parameters of the connections based on the error of the neural network predictions, typically using a method called backpropagation. This training process is computationally intensive and requires large datasets, but recent advances in hardware (e.g. GPUs) and the availability of big data allowed to scale these models significantly. The impact of Deep learning is profound across various industries. For example, in healthcare, it is used for diagnosing diseases from medical images and predicting patient outcomes [5]. In automotive, it powers autonomous vehicles by enabling real-time image recognition and decision-making [6].

This technology is also being used in the manufacture of semiconductor components such as Molecular Beam Epitaxy (MBE) as a characterisation tool. Therefore, in a first step, we are employing this technique to characterize the mechanisms of crystal growth, in particular to the processes of substrate deoxidation, surface reconstruction and Azimuthal RHEED construction. In a second step, we are employing it in the field of nanophotonics by demonstrating Deep Learning inverse design methods and the use of Graph Neural Networks on the direct problem.

MBE is a controlled method for fabricating high-quality crystalline layers to create semiconductor devices. It involves the deposition of atomic or molecular beams onto a substrate in an ultra-high vacuum environment. The process allows for precise control over the thickness, composition and doping of the deposited layers, making it ideal for creating complex semiconductor structures with atomic layer precision. The MBE process begins with the preparation of a substrate, typically a wafer, which is then heated to a high temperature for purifying and to promote adhesion of the deposited atoms. Source materials, often in solid form, are

heated in separate effusion cells to produce beams of atoms or molecules. These beams travel in a straight line to the substrate, where they condense and form a crystalline layer. The growth of these layers can be monitored and controlled in real-time, allowing for the creation of structures with precise atomic configurations. A typical MBE characterization tool is the Reflection High-Energy Electron Diffraction (RHEED).

RHEED is a powerful in-situ characterization technique used in MBE to monitor the growth of thin films in real-time. In RHEED, a beam of high-energy electrons (typically 10-20 keV) is directed at the surface of the growing film at a shallow angle. The electrons interact with the surface atoms, resulting in a diffraction pattern that provides information about the surface structure and growth dynamics. RHEED patterns are sensitive to changes in the surface morphology, crystal structure and growth rate. As the film grows, the diffraction pattern evolves, allowing operators to monitor the growth process and make real-time adjustments to the deposition parameters. The analysis of RHEED patterns can reveal important information about the formation of new atomic layers, the occurrence of surface reconstructions and the crystal orientation.
The analysis of RHEED images is traditionally performed using manual or semi-automated methods, which can be time-consuming and subject to human error. Recently, deep learning techniques have been applied to automate and enhance the analysis of RHEED images. By training convolutional neural networks (CNNs) on large datasets of RHEED images, it is possible to develop models that can accurately classify different surface structures, detect defects and sample orientation and more growth parameters. Deep learning models can process RHEED images in real-time, providing immediate feedback on the growth process. This capability would enable more precise control over the MBE process, leading to more efficient fabrication and thus higher-quality devices. Additionally, the use of deep learning can uncover subtle patterns and correlations in RHEED data that might be missed by traditional analysis methods.

Deep Learning is also employed in the field of Nanophotonics. The research field of nanophotonics studies light-matter interactions on the nanometer scale. It involves the design and fabrication of nanostructures that can manipulate light. Nanophotonic devices have applications in areas such as telecommunications, sensing, imaging and quantum computing. The ability to control light at the nanoscale opens up new possibilities for creating compact and efficient photonic devices. One of the major challenges in nanophotonics is the inverse design problem, which involves determining the nanostructure that will produce a desired optical response. This is a complex and computationally intensive task with the simulation-based traditional methods. The use of generative deep learning models as well as gradient-based iterative methods offers the possibility of reducing the time and computing power required for inverse design problems.

In nanophotonics, geometric deep learning can be used to solve the forward problem, which involves predicting the optical response of a given nanostructure. Geometric deep learning is a rapidly emerging field that extends deep learning techniques to non-Euclidean data, such as graphs [7]. By training graph neural networks on datasets of nanostructures (represented as graphs) and their corresponding optical responses, it is possible to develop models that can make accurate and fast predictions. These models can be used to accelerate the inverse design process by providing rapid evaluations of candidate structures and gradients of the

optical properties. The application of geometric deep learning to nanophotonics represents a promising approach for overcoming the limitations of traditional design methods such as the slowness and the handling of complex structures.

The thesis is structured as follows:

Chapter 1 consists of an introduction to Deep Learning, explaining the basis of neural network models. Namely the artificial neuron and its activation functions, as well as different architectures handling different types of data. The chapter also explains the neural network training process and concludes with a selection of architectures for data generation (variational autoencoder and generative adversarial network), compression (autoencoder) and segmentation (U-Net).

In Chapter 2, we present the characterization of raw RHEED images with neural network models, in particular on the detection of substrate deoxidation, the categorization of surface reconstructions and the construction of Azimuthal RHEED.

Chapter 3 is about how to deal with the inverse problem in nanophotonics. We explain a number of architectures that allow to overcome the complications of inverse problem, such as the tandem network, the conditional variational autoencoder and the neural adjoint method. This chapter ends with a demonstration of graph convolutional networks (GCNs) to solve nano-photonics forward problems, illustrated by two examples, namely the electric polarization of infinitely-high nano-cylinders and the reflectivity of thin-film multilayer stacks.

Chapter 1

Introduction to Deep Learning

1.1 Introduction

To introduce this first chapter dedicated to deep learning, we will begin by situating it in the field of Artificial Intelligence (AI) which is the science of making machines do things that would require intelligence if done by human being [1]. Then comes the Machine Learning (ML), a subset of AI that focuses on the development of algorithms aimed to enable machines to learn from data and perform on some specific tasks. Deep Learning (DL) is a part of ML and characterized by the use of deep neural networks to automatically learn from data [8]. At the core of deep learning are the artificial neural networks, which are composed of interconnected layers of neurons. Each neuron processes inputs through a set of weights and a bias, applying an activation function to generate an output. A neural network architecture typically includes an input layer, multiple hidden layers and an output layer. The depth (number of layers) and the complexity of these networks enable deep learning models to capture intricate patterns and representations in the data. Neural networks are inspired by the structure and function of the human brain. Deep learning algorithms have the ability to automatically learn and extract intricate patterns and features from large datasets. Through different training methods like supervised, unsupervised and reinforcement learning, deep learning models can generalize patterns and make predictions, enabling advancements in areas like computer vision, natural language processing, autonomous driving and robotics. Deep learning has ushered in a new era of AI, providing the tools and techniques needed to tackle some of the most challenging and impactful problems across industries thanks to complex and deep artificial neural networks. The following section elucidates the concept of artificial neuron.

1.2 Artificial neuron

The artificial neuron is the fundamental building block of each artificial neural network. This section presents its constituent components and operational principle with which it transforms data.

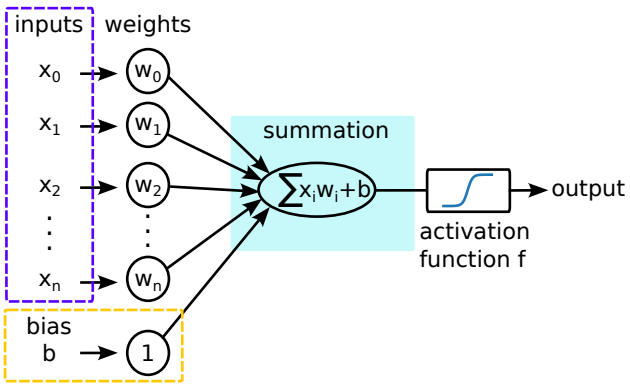

Figure 1.1: Artificial neuron. Each input x_i is associated with a weight w_i. The weighted sum is computed and added to the bias b before passing through the activation function that produces the neuron output.

1.2.1 Principle

The artificial neuron is a mathematical model that processes inputs through weighted summation and nonlinear transformation inspired by the way biological neurons work in the human brain [9]. Its ability to learn and adapt through training underpins the functionality of neural networks, driving advancements in various fields of AI and machine learning.

A single artificial neuron receives one or more inputs, typically denoted as $x_1, x_2, ..., x_i$. Each input element is a single numerical value and represents an information coming into the neuron and associated with a weight $(w_1, w_2, ..., w_i)$ which represents the importance or strength of that input element. The neuron calculates a weighted sum of its inputs by multiplying each input by its corresponding weight and then summing them up [10]. In addition to the weighted sum, a bias term (often denoted as b) is added to the result. The bias allows the neuron to adjust the threshold for the activation. The weighted sum and the bias addition make up the linear part of the neuron and can be expressed as follows:

$$\sum_{i=1}^{n}(x_i w_i) + b \tag{1.1}$$

With n representing the number of inputs, x_i is the $i-th$ input, w_i is the weight associated with the $i-th$ input and b is the bias vector.

As shown in Figure 1.1, the result of the weighted sum with bias is then passed through an activation function denoted as f. The activation function introduces non-linearity into the neuron and determines whether the neuron should produce an output signal or not. There are various activation functions used in practice, such as the *sigmoid*, *ReLU (Rectified Linear Unit)* and *softmax*. The choice of the activation function type depends on where it is in the network (in a hidden or a final layer) and the specific problem. We will break down the activation functions in the next section. The output of the neuron y represents its response or activation level. In many cases, this output is used as an input to other neurons in the network, forming the basis for complex computations in neural networks. The neuron output with an activation function f is expressed as [11]:

$$f\left(\sum_{i=1}^{n}(x_i w_i) + b\right) \tag{1.2}$$

The purpose of the weighted sum and the bias addition is to allow the neuron to learn and adapt to different patterns in the data. During the training process, the weights and biases are adjusted using optimization algorithms like gradient descent to minimize the error between the neural network predictions and the targets.

1.2.2 Activation functions

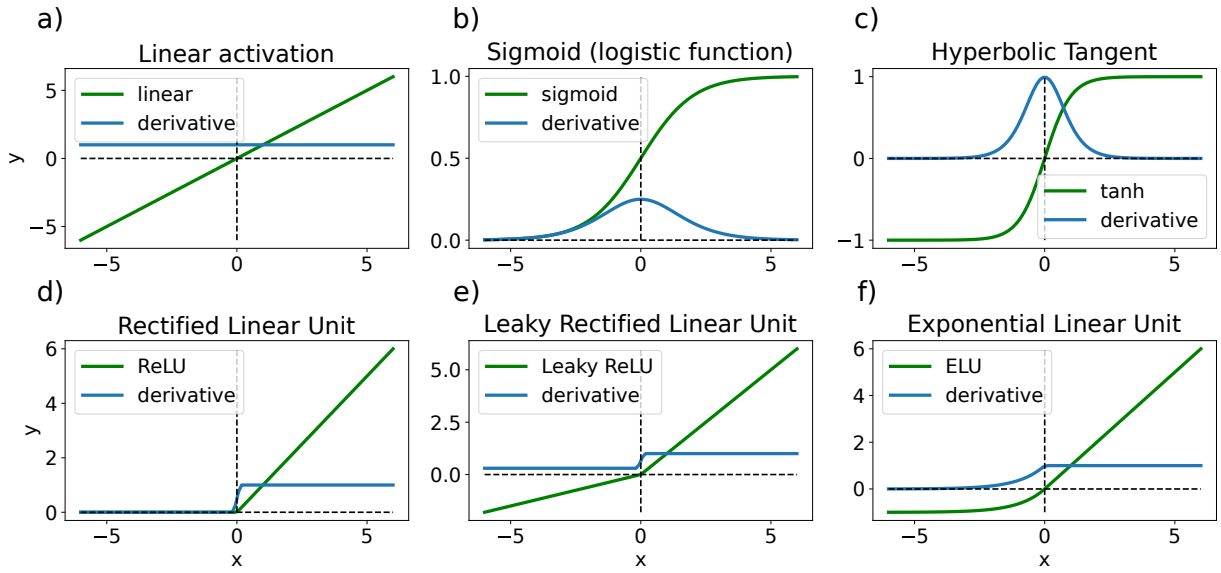

Figure 1.2: Curves of activation functions (green) and their respective derivatives (blue) where x is the function input and y its output. a) *linear*, b) *sigmoid*, c) *Hyperbolic Tangent (tanh)*, d) *Rectified Linear Unit (ReLU)*, e) *Leaky ReLU* with $a = 0.3$ and f) *Exponential Linear Unit (ELU)* with $\alpha = 1$.

In neural networks, activation functions are mathematical functions applied to the output of a neuron to determine its final output. The activation functions can be divided into two categories: linear activation and non-linear activations. Consecutive linear layers can be trivially replaced by a single linear layer. On the other hand, non-linear activations add non-linearity to the network, allowing it to approximate complex and non-linear functions [12]. Therefore, using nonlinear activation functions is crucial in any deep ANN in order to perform hierarchical feature extraction [13]. However, the neural network training process can suffer from the vanishing (very small) or exploding (very big) gradient problem that limits the network learning capabilities. The main culprit of vanishing or exploding gradient is the choice of the activation function [14]. Below is a description of the most commonly used activation functions as well as their mathematical expressions where x is the input.

***Linear activation*:** Linear activation, also known as identity activation function, has the equation of a straight line:

$$y = x \tag{1.3}$$

Linear activation function is typically used at the output layer of regression neural networks as the output in this case may have any big or small values. Examples of curves for the *linear activation* and its derivative are shown in Figure 1.2a showing that the gradient of the linear activation is constant and thus, does not cause vanishing or exploding gradient issues.

***Sigmoid (Logistic Function)*:** The sigmoid function is a non-linear activation function that maps its input from the range $]-\infty;+\infty[$ to a range in $[0;1]$. The *sigmoid* activation function can be defined as follows:

$$\sigma(x) = \frac{1}{1+e^{-x}} \tag{1.4}$$

It is particularly useful in binary classification problems as last-layer activation function as its output can be interpreted as a probability of a certain class. It has a smooth and continuous derivative as shown in Figure 1.2b, rendering it suitable for gradient-based optimisation algorithms. However, it can suffer from vanishing gradient problems in deep networks as the logistic function is flat far from the origin [15].

Softmax: The *softmax* activation function is used in the output layer of neural networks for multi-class classification problems. It is a normalized combination of multiple *sigmoid functions* and converts a vector of scores into a probability distribution over multiple classes [12].

$$softmax(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}} \tag{1.5}$$

***Hyperbolic Tangent (Tanh)*:** The *Tanh* function is a non-linear activation function similar to the sigmoid but maps its input to a range in $[-1;1]$.

$$tanh(x) = \frac{e^{2x}-1}{e^{2x}+1} \tag{1.6}$$

Like *sigmoid*, it has a smooth and continuous derivative as shown in Figure 1.2c, rendering it suitable for gradient-based optimisation algorithms. While the *TanH* function is more effective than the *sigmoid* function in mitigating the vanishing gradient problem, it is still susceptible to this issue for very large or small input values. The *TanH* activation function is especially used in the hidden layers of recurrent neural networks which is introduced in Section 1.5.

***Rectified Linear Unit (ReLU)*:** In modern neural networks, ReLU and its variants are commonly used in hidden layers. ReLU activation is computationally efficient and helps mitigate the vanishing gradient problem. It returns zero for negative inputs and linearly

scales positive inputs. Examples of curves for the *ReLU* activation and its derivative are shown in Figure 1.2d. It can be expressed as follows:

$$ReLU(x) = \begin{cases} x & \text{if } x > 0 \\ 0 & \text{else} \end{cases} \tag{1.7}$$

As *ReLU* sets all negative values to zero and since the gradient of 0 is 0, neurons arriving at large negative values cannot recover from being stuck at 0. The neuron effectively dies and the problem is known as the 'dying ReLU' problem. This can lead the network essentially to stop learning and under-perform [16]. *ReLU* function is also susceptible to the exploding gradient problem for very large positive input values.

***Leaky ReLU*:** Leaky ReLU is a variant of ReLU that allows a small non-zero slope for negative inputs in order to address the "dying ReLU" problem.

$$Leaky\ ReLU(x) = \begin{cases} x & \text{if } x > 0 \\ ax & \text{else} \end{cases} \tag{1.8}$$

Where a is a small positive constant ($a = 0.3$ in practice).

Examples of curves for the *Leaky ReLU* activation and its derivative are shown in Figure 1.2e. This variant brings non-zero gradient for negative inputs. Another *ReLU* variant is the ***Parametric ReLU (PReLU)*** with the same expression than *Leaky ReLU* except that a is a learnable parameter and determined through the neural network training process.

Exponential Linear Unit (ELU): It is also a variant of the *ReLU* function. The *ELU* activation function brings a smooth curve for the negative inputs, which serves to mitigate the issue of "dying ReLU". Examples of curves for the *ELU* activation and its derivative are shown in Figure 1.2f. It can be defined as follows:

$$ELU(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha(e^x - 1) & \text{else} \end{cases} \tag{1.9}$$

Where α is the slope of the negative section ($\alpha = 1$ in practice).

As the output layer of a neural network produces the final response, its activation function is chosen according to the task that is performed. Although the most commonly used activation functions are introduced above, the Table 1.1 recapitulates the choice of the last-layer activation functions according to the task to perform.

Now that the key component (artificial neuron) has been explained, some typical neural network architectures are presented in the following. Each architecture is aimed for specific tasks and data.

Task	Output type	Final activation function
Regression	Continuous	$linear$
Classification	Binary	$sigmoid$
Classification	Multiple class scores	$softmax$

Table 1.1: The first column presents the task to perform, the second column is the neural network output type and the third column contains the kind of activation function to use.

1.3 Multilayer perceptron

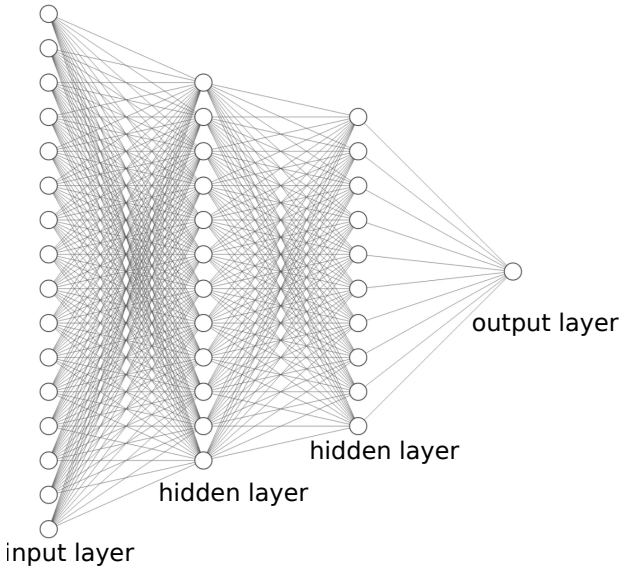

Figure 1.3: Multilayer Perceptron (MLP) architecture. This architecture contains an input layer to retrieve information, two hidden layers in which the process of transformation and extraction of the characteristics takes place and an output layer of single neuron that outputs the neural network prediction.

The MultiLayer Perceptron (MLP) is a feedforward artificial neural network architecture that consists of multiple layers of interconnected neurons. Each layer contains one or more neurons and, as shown in Figure 1.3, these neurons are organized into an input layer, one or more hidden layers and an output layer [17]. It is designed to model complex relationships between inputs and outputs and to capture patterns in data through its layered structure.

The input layer of the MLP serves as the entry point for the input data. Each neuron in the input layer corresponds to an input feature for instance a pixel value from an input image or an element from an input vector. If there are n input features, n neurons are needed in the input layer. Between the input and output layers, MLPs can have one or more hidden layers. The number of neurons in each hidden layer can vary depending on the complexity of the problem and the design of the network. Each neuron in a hidden layer is connected to every neuron in the previous layer through weighted connections. The hidden layers learn to identify and extract features and correlations from their inputs. The output layer of the MLP produces the network predictions. The number of neurons in the output layer depends

on the specific task. For instance, in binary classification, there might be one output neuron outputting 0 or 1 according to the predicted class, while in multi-class classification, there would be one neuron per class.

1.4 Convolutional neural network

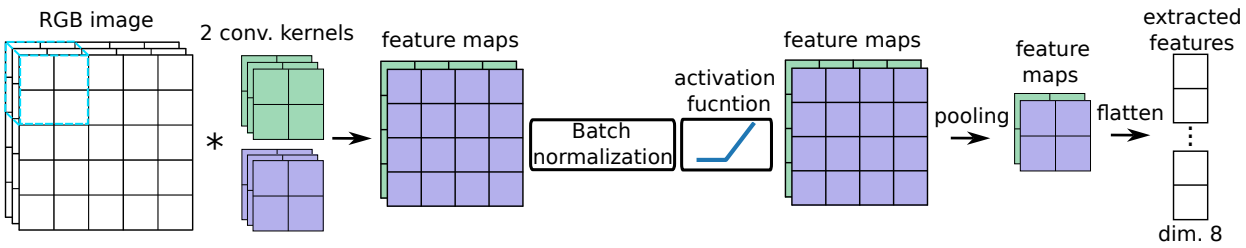

Figure 1.4: Convolutional Neural Network architecture. The input RGB image is convolved by several (here two) convolutional kernels with the same channel dimension. Each kernel dimension operates on the corresponding image dimension, the resulting three matrices (one for each input channel) are summed along the depth dimension leading to one feature map per kernel. The output feature maps are then passed through batch normalization, activation function and pooling layers for feature extraction and dimensionality reduction. The pooling layer typically keeps the highest value of each window of four elements. The extracted characteristics are then further processed, for example they can be flattened into a vector that can constitute the model prediction or fed to a MLP depending on the task.

Convolutional Neural Network (CNN) is a well-known deep learning architecture inspired by the visual cortex of mammals [18]. As its name indicates, CNN involves convolution operations and is designed to capture spatial patterns in structured grid data such as images and have achieved remarkable success in various computer vision tasks including image classification, object detection and facial recognition thanks to their ability to automatically learn and extract meaningful features from data [19].

The main elements of a convolutional neural network are illustrated on Figure 1.4. A typical block of convolution is constituted as follows:

Learnable kernels (also called filters) slide over the input data to perform convolution operations, producing feature maps. Each filter automatically specializes in detecting a particular pattern [19]. Dozens or hundreds of filters are typically used per convolution layer in a medium-sized model (a few million parameters).

After convolution, a *Batch Normalization* layer is typically applied to the features in order to keep data normalized during forward propagation. This layer speeds up the training process and prevents against overfitting. As it is presented in Section 1.7, the training data is processed in batches (a small subset of the database) and the *Batch Normalization* is based on the assumption that the statistics of the batch are similar as the total dataset statistics, therefore it is particularly helpful for large models with large available quantities of data. [13].

Then, an *activation function* is applied to the feature maps. This step introduces non-linearity into the network, allowing it to learn complex relationships within the data. *Pooling layers* are often included after activation to downsample feature maps and reduce spatial dimensions. Max-pooling is a common operation in which the maximum value within a local region of the feature map is selected, preserving the most salient information. After many convolution blocks and pooling layers, the extracted features are flattened along spatial or depth dimension and represents the prediction as such or are fed to a MLP in order to perform further processing.

1.5 Recurrent neural network

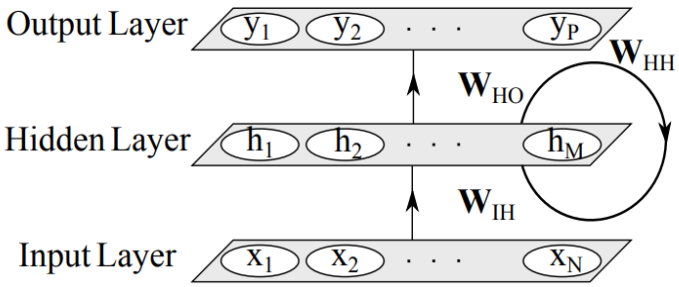

Figure 1.5: Recurrent neural network architecture. The input layer takes the input data then passes it to the hidden layer that uses in a loop its previous value and the current input both associated to weight matrices (W_{HH}, W_{IH}). The result is fed to the output layer which calculates the final value in the base of the received result associated to a weight (W_{HO}). The bias b is not taken into account to simplify the figure. Readapted from [20].

The architecture of a Recurrent Neural Network (RNN) is designed to handle sequential data [21], making it particularly useful for tasks such as natural language processing, time series prediction and speech recognition. While it will not be used in this thesis, RNNs represent an important class of Deep Learning models, therefore it will be briefly discussed in this section. A RNN is characterized by its recurrent connections that allow it to maintain a hidden state and process sequences of variable length. Recurrent neural networks have been an interesting and important part of neural network research during the 1990's [21].

We illustrate the basic RNN architecture in Figure 1.5. A RNN starts with an input sequence, which could be a sequence of words in a sentence, a time series of data or any other form of sequential data. The key feature of an RNN is the hidden state, which serves as a memory of previous inputs in the sequence. At each time step t, the RNN updates its hidden state h_t based on the current input x_t and the previous hidden state h_{t-1}. At each new input, the hidden state is updated using its previous state and the current input as expressed below [20]:

$$h_t = f(\, W_{IH} x_t + W_{HH} h_{t-1} + b_H \,) \tag{1.10}$$

Where W_{IH} and W_{HH} are weight matrices, f is an activation function (commonly the hyperbolic tangent (tanh) or the Rectified Linear Unit (ReLU)), h_{t-1} is the previous hidden state

and b_H is the bias of the hidden state.

At each time step, the RNN can produce an output y_t based on the current hidden state h_t. This output can be used for various purposes depending on the task. For example, in language modeling, it can be used to predict the next word in a sentence. The output at each time step can be computed as:

$$y_t = g(\ W_{HO} h_t + b_O\ ) \tag{1.11}$$

Where W_{HO} is the weight matrix, g is an activation function used to transform the hidden state into the desired output format and b_O is the output bias.

RNNs process the input sequence one element at a time, updating the hidden state and producing an output at each time step. This allows them to capture dependencies that span across multiple time steps. However, standard RNNs suffer from the vanishing gradient problem, which can limit their ability to capture long-range dependencies.

RNNs are trained using backpropagation through time (BPTT), a variant of the backpropagation algorithm. BPTT calculates gradients with respect to the network parameters (weights and biases) by unrolling the network in time, treating it as a deep feedforward neural network. These gradients are then used to update the parameters through optimization techniques like gradient descent [22].

Over the years, various RNN variants and improvements have been introduced to address the limitations of standard RNNs. Some of these include Long Short-Term Memory (LSTM) networks and Gated Recurrent Unit (GRU) networks, which have better capabilities for capturing long-range dependencies and mitigating the vanishing gradient problem. Because of the sequential processing, it cannot be well parallelized and therefore was mostly replaced by the attention-based architectures like transformers [13].

MLPs, CNNs and RNNs have limitations regarding the data format. Sometimes, none of the above presented architectures are well suited and data may be ideally expressed as graphs. Hence, the following section is about graph neural networks, an architecture that allows to handle graph data.

1.6 Graph neural network

The architecture of a Graph Neural Network (GNN) is designed to process and learn from data structured as graphs, which represent relationships or connections between entities. GNNs have gained significant popularity in various domains, including social network analysis, recommendation systems and biology.

The first step in using a GNN is to represent the data as a graph. A graph is composed of nodes (vertices) and edges (connections between nodes). Each node represents an entity, while edges represent relationships or interactions between entities. Graphs are the unique

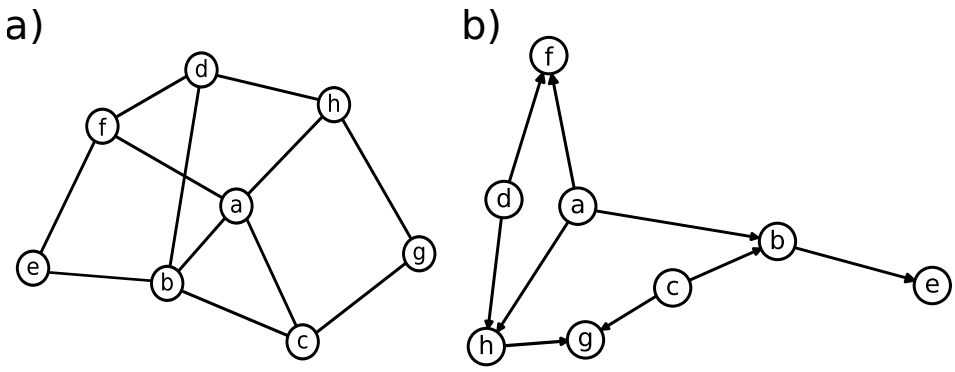

Figure 1.6: Undirected and directed graphs. An a) undirected graph have relations between nodes that are two-way where in b) directed graph the relations between nodes are one-way.

non-Euclidean data structure for machine learning as the Euclidean distances between entities of a graph do not matter [23]. Additionally, nodes and edges can have associated attributes or features which are often represented as vectors.

There are two types of graphs, directed and undirected. As shown in Figure 1.6, in undirected graphs, the relation and messages between two linked nodes are two-way while in directed graphs, the relation and messages between two linked nodes are just one-way. A graph can also be dynamic or static, dynamic when the features or the topology vary with time, this kind of graphs are used in Dynamic GNN [24]. A graph is heterogeneous when nodes and edges have different types and are handled with heterogeneous GNN [25]. These cases, however, are out of the scope of the present work. We are here focusing on GNNs that act on undirected, static and homogeneous graphs.

GNN have been successful in various applications where data is naturally represented as graphs, offering a powerful tool for tasks in node-level, edge-level and graph-level. Node-level tasks include node classification to categorize nodes into many classes or node regression to predict continuous values for nodes. Edge-level tasks include for instance link prediction to determine whether two given nodes are linked. Graph-level tasks include for example graph classification and graph regression [23].
The message passing allows nodes to communicate and aggregate information from their neighbors. At each layer of the GNN, nodes collect information from their neighboring nodes, update their own feature representations and then pass this updated information to their neighbors in the next layer through aggregate and combine stages.

The message passing operation can be represented as [26]:

$$m_v^{(k)} = AGGREGATE(\{\ h_u^{(k-1)} \forall u \in \mathcal{N}(v)\ \}) \tag{1.12}$$

$$h_v^{(k)} = COMBINE(\ h_v^{(k-1)}, m_v^{(k)}\ ) \tag{1.13}$$

Where, $m_v^{(k)}$ and $h_v^{(k)}$ represent the message vector and feature vector respectively for node v at layer k and u are neighboring nodes of v. $AGGREGATE$ is a function that aggregates

information from neighboring nodes and $COMBINE$ is a function that combines the aggregated information $m_v^{(k)}$ with the current representation $h_v^{(k-1)}$ of node v.

The final layer of the GNN produces the desired outputs based on the learned representations. The specific task at hand determines the form of the output layer. For node classification, it might involve a softmax layer for predicting node labels. For graph classification, there could be a pooling operation to produce a graph-level representation. This section ends with the recapitulation in Table 1.2 in order to summarize the above presented architectures, their respective handled data types and activation functions typically used in hidden layers. The following section explains the process of artificial neural network training with stochastic gradient descent.

Artificial neural network	**Data type**	**Hidden layers activation**
Multi-layer perceptron	Vectors	$ReLU$
Convolutional neural network	Matrices (images)	$ReLU$
Recurrent neural network	Sequential (time-dependent)	*Hyperbolic Tangent*
Graph neural network	Graphs	$ReLU$

Table 1.2: Comparison of different artificial neural networks in terms of their handled data type and commonly used activation functions in the hidden layers.

1.7 Neural network training

Task	**Output type**	**Loss function**
Regression	Continuous	$MeanSquaredError(MSE)$
Classification	Binary	$BinaryCrossEntropy$
Classification	Multiple classes	$CrossEntropy$

Table 1.3: The first column presents the task to perform, the second column is the neural network output type and the third column contains the kind of loss function to use.

Stochastic gradient descent (SGD) is an optimization algorithm used to iteratively update the neural network parameters (weights and biases) during the training phase in order to minimize the loss function, i.e. the error between the network prediction and the target, gradually [27]. As shown in Table 1.3, the loss function is chosen based on the task to perform (and so on the neural network output). Weights and biases are initialized randomly with small values, for instance samples from a normal or uniform distribution. The initialization process can have an impact on the convergence and performance of the neural network model training [28]. To elucidate the neural network training process, we cover key concepts such as gradient, chain rule, automatic differentiation and minibatch SGD in the following.

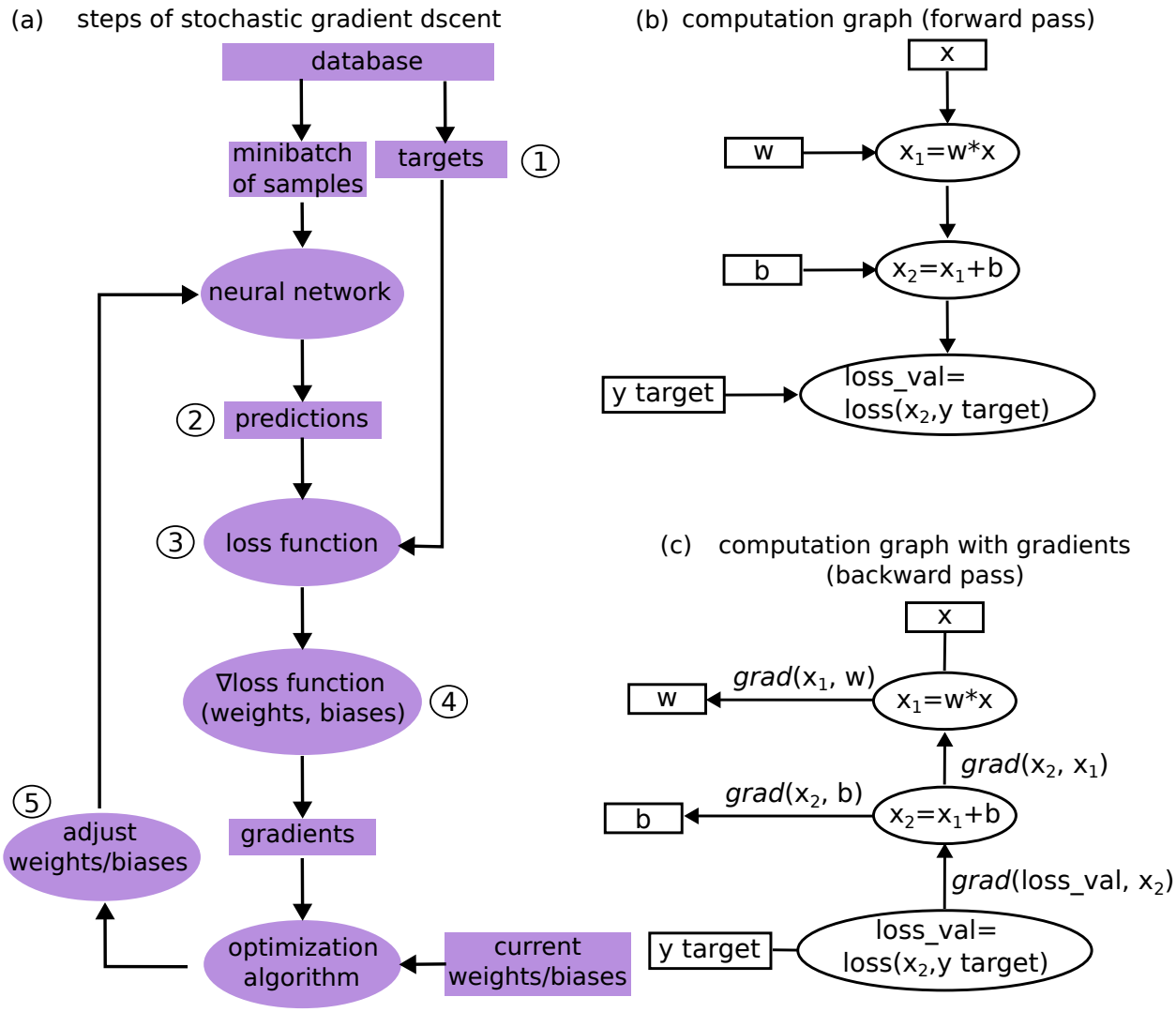

Figure 1.7: Stochastic gradient descent (SGD) and computation graph. (a) main steps of the training loop using SGD algorithm. The steps corresponding to the circled numbers are explained below. Automatic differentiation tool transforms the neural network architecture into computation graph on which it performs (b) the forward pass and (c) backward pass by computing the gradient of simple functions in the model using chain rule.

1.7.1 Gradient

During training, step by step, the model parameters (weights) are modified at once in a way that decreases the model loss. The gradient is the mathematical operator that describes how the loss varies in function of the model parameters. The derivative $\frac{\partial f}{\partial x}$ gives the slope of the tangent to the function and therefore quantifies the rate of change of the function. Moreover, the derivation can be applied to scalar functions (mapping scalar into a scalar) as well as tensor functions (mapping tuple of scalars to a scalar) and the derivative of a tensor function is called gradient. For a tensor differentiable function, the gradient represents the curvature of the multidimensional surface described by the function. All the functions (scalar product, $+$, etc) used in neural network models are differentiable and transform their inputs in a smooth and continuous way. Chaining such functions results in a bigger and still differentiable

function. In deep learning, if we know the rate of change of the loss function with respect to the weights, then we know in which direction to change the weights in order to decrease the loss and thus make the predictions better. Small changes in the model parameters results in a small, predictable change in the loss value. For instance, a model with single neuron without bias that predicts y_{pred} for an input x:

$$\begin{aligned} y_{pred} &= w \cdot x \\ loss\_value &= loss(y_{pred}, y_{true}) \\ loss\_value &= loss(w \cdot x, y_{true}) \end{aligned} \tag{1.14}$$

where $loss$ is the loss function. As x and y_{true} are constant for a given batch of data, the $loss\_value$ is just function of weights w. The gradients of the loss with respect to w represent the fundamental principle of the stochastic gradient descent. However, neural networks in practice involve complex expression by stacking many layers, each layer implementing simple mathematical functions. This chaining of multiple functions makes it difficult to compute the gradient of the whole model simultaneously. That is where the backpropagation algorithm comes in using chain rule.

1.7.2 Chain rule

Backpropagation is a way to use the derivatives of simple operations(such as addition, ReLU or tensor product) to easily compute the gradient of complex combinations of those simple operations. A neural network consists of many tensor operations chained together, each of which has generally a simple, known derivative. If a variable z depends on the variable y, which itself depends on the variable x then z depends on x via the intermediate variable y. In this case, the chain rule is expressed as (Leibniz's notation):

$$\frac{d_z}{d_x} = \frac{d_z}{d_y}\frac{d_y}{d_x} \tag{1.15}$$

1.7.3 Automatic differentiation

Nowadays, neural networks are implemented in modern frameworks, such as keras [29] and PyTorch [30], that are capable of automatic differentiation. The automatic differentiation makes it possible to retrieve the gradients of arbitrary functions in a neural network architecture [31]. Automatic differentiation is implemented with the so-called computation graph. Computation graphs are a type of graph that can be used to represent mathematical expressions. It is a directed acyclic graph of operations (tensor operations for neural network), directed because such graphs have nodes with only one-way orientation and acyclic because theses orientations never form a closed loop. Illustrations of computation graph and gradient determination using chain rule for a model comprising one layer of single neuron are shown in Figures 1.7b and 1.7c, respectively. In a first step, the automatic differentiation tool transforms the neural network structure into a graph computation in the background; and in the second step, the forward pass is performed. During the forward pass, the automatic differentiation tool calculates also the gradients of each mathematical operation along the graph. These results are then used in the chain rule during the backward pass. The backpropagation is the application of chain rule to a computation graph, it starts with the final loss value

and works backward from the top layer to the bottom layer, computing the contribution that each parameter had in the loss value. It "back propagates" the loss contributions of different parameters in a computation graph [31].

In training phase, processing the entire training dataset, by batches, constitutes an epoch. Training typically continues for multiple epochs to allow the neural network refine its parameters.

1.7.4 Minibatch stochastic gradient descent

For a given differentiable function, it is theoretically possible to identify its minimum, which is defined as a point where the derivative is equal to zero. This can be achieved by computing all the points where the derivative is equal to zero and selecting the one with the lowest value. This would involve evaluating the function for all possible values of every parameter w_i. This is inefficient for a neural network of millions of parameters. It is rather efficient to modify the model parameters little by little on the loss value for a random batch of data. Because the neural network is a differentiable function, it is possible to compute the gradient and update the weights in the opposite direction from the gradient so that the loss becomes a little less every time [31].
As schematically shown in Figure 1.7a, one efficient approach is to utilise minibatch stochastic gradient descent which operates in mainly five steps:

1. Randomly draw a batch of training samples, x and corresponding targets, y_{true}.
2. Run the model on x to obtain the predictions, y_{pred} (forward Pass).
3. Compute the *loss* of the model on the current batch.
4. Compute the gradient of the *loss* with respect to the model parameters (backward pass)
5. Move the parameters a little in the opposite direction from the gradient by computing

$$w_i^{updated} = w_i - learning\_rate \cdot \frac{\partial loss}{\partial w_i} \tag{1.16}$$

where $learning\_rate$ is a hyperparameter that controls the step size of parameter updates. The term stochastic refers to the fact that each batch is drawn randomly.

Some variants of SGD take into account the previous weight updates, rather than just looking at the current value of gradients. This is SGD with momentum. The concept of momentum addresses two issues of SGD, namely the convergence speed and local minima [32]. Momentum helps accelerate the process in the right direction by accumulating the gradients of past steps. Weight update with momentum revisits the previous equation 1.16 of weight update by adding a new term:

$$w_i^{updated} = w_i - learning\_rate \cdot \frac{\partial loss}{\partial w_i} + \eta \cdot v_i \tag{1.17}$$

η is the momentum coefficient (between 0.5 and 0.9 in practice) and v_i is the retained gradient from past updates.
The weight update is performed using an optimizer which is the algorithm through which the model updates its parameters. *Adam* optimizer [32] performs such weight update with momentum, the name *Adam* is derived from "adaptive moment estimation".

1.7.5 Overfitting and underfitting

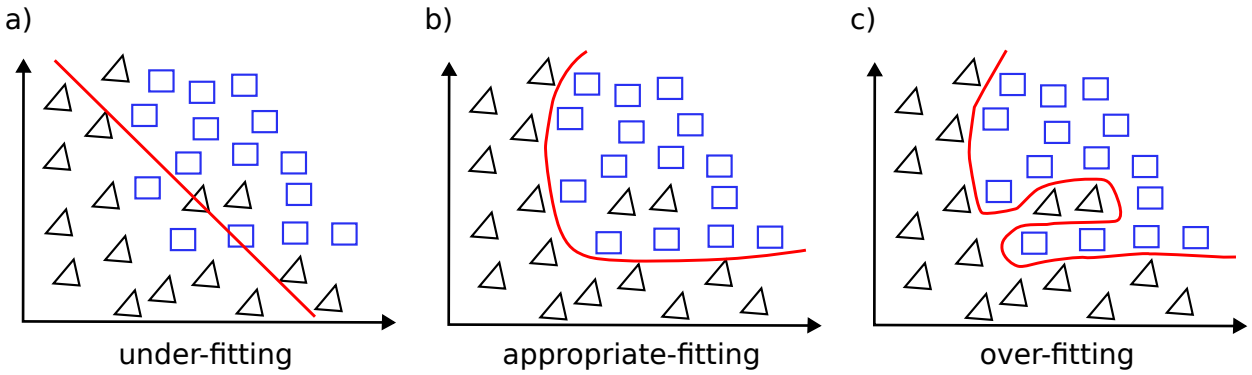

Figure 1.8: Underfitting and overfitting. The aim of the model here is to separate the squares from the triangles. The red curve shows the limit learned by the neural network in situations of under-fitting, appropriate-fitting and over-fitting.

It is also crucial to monitor the model's performance on a separate validation dataset to detect overfitting and underfitting. Overfitting is when a model learns to fit the training data too closely and it then generalizes poorly to new data and underfitting occurs when the model is trained with insufficient amount of data (e.g. learning just the mean value of the entire dataset) [33]. In Figure 1.8 is shown the effects of these problems in case of categorizing two distinct populations.

Once the training process is completed and the model has achieved satisfactory performance on the validation set, it can be evaluated on a test dataset to assess its generalization ability. Testing provides an unbiased estimate of the model's performance on unseen data.
Complex architectures of Deep Learning perform specific tasks, a selection of such neural networks is explained in the following section.

1.8 A selection of commonly used architectures

This section provides an overview of the most commonly used neural network architectures, which also serve as a basis for other more sophisticated architectures. In particular, this section presents the autoencoder, variational autoencoder VAE, U-Net and generative adversarial network GAN, covering the architecture and the operating principle.

1.8.1 Autoencoder

Autoencoders are a class of neural networks used for unsupervised learning, dimensionality reduction and representation learning. They are designed to encode data into a lower-dimensional representation and then decode it back to its original form.

As depicted in Figure 1.9, an autoencoder consists of two main parts [34]:

The ***encoder*** network compresses the input data into a lower dimension. It is made up of one or more layers of neurons, with each layer progressively downsampling the dimensions of

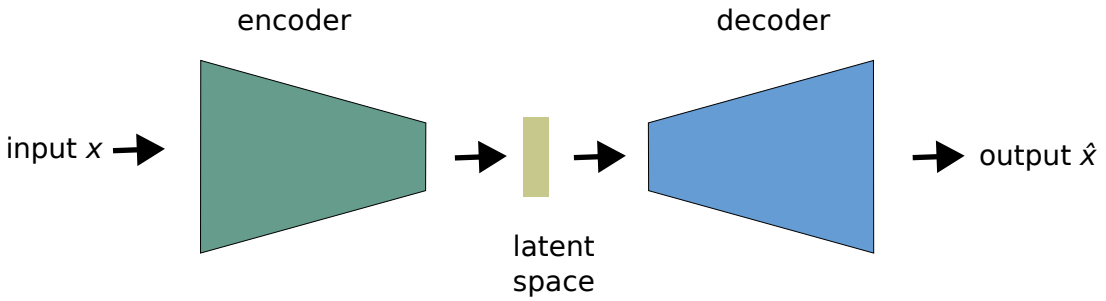

Figure 1.9: Autoencoder architecture. The encoder takes an input x and produces a latent space that is fed into the decoder to output the reconstructed data $\hat{x}$.

the data until a final representation called *latent space* or *bottleneck* which is the central layer of the autoencoder. It has the smallest number of neurons, capturing the most important features of the input data in a compressed form. In inference mode, the encoder only can be used as a compressor.

The ***decoder*** network takes the encoded representation as input and reconstructs the original input data. Like the encoder, it also consists of one or more layers but each layer upsampling the data dimension. It mirrors the encoder's structure, progressively increasing the dimensionality back to the original input size; thus, the final layer of the decoder produces the reconstructed input and the number of neurons in this layer matches the number of features in the original input data.

Autoencoders are used for various purposes, including:

Anomaly detection: Autoencoders can learn to reconstruct normal data accurately, making them effective for detecting anomalies or outliers in the data [34].

Feature learning: Autoencoders can discover meaningful features in the data, making them useful for feature extraction [35].

Several variants of autoencoders have been developed to address specific tasks like Variational Autoencoders (VAEs) which combine autoencoders with probabilistic modeling, allowing generative capabilities and better handling of continuous data [36].

1.8.2 Variational autoencoder

Variational Autoencoders (VAEs) are a type of generative model that extend the concept of traditional autoencoders by introducing probabilistic elements into the encoding and decoding processes. They are designed to learn a probability distribution over the input data, enabling the generation of new data samples that are similar to the original data [37]. VAEs have applications in various fields, including image generation and data augmentation. A VAE consists of two main components: the encoder and the decoder. Figure 1.10 shows the VAE architecture. These components are similar to those in a traditional autoencoder but with significant differences in their operation due to the probabilistic nature of VAEs.

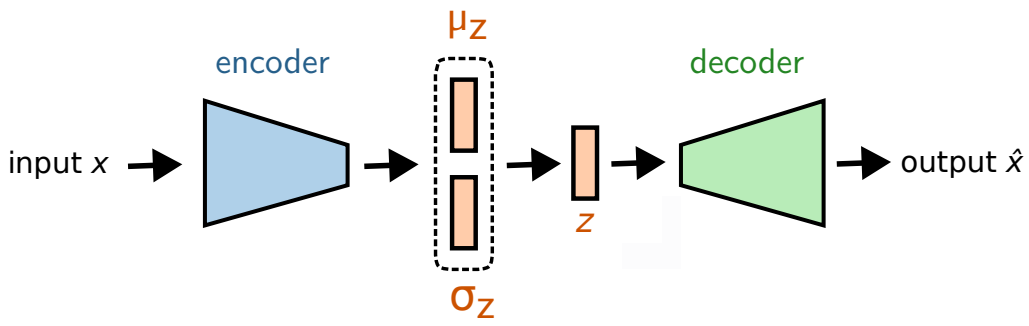

Figure 1.10: Variational autoencoder architecture. The encoder takes an input x and produces parameters mean μ_z and the standard deviation σ_z of a probability distribution. A latent variable z is computed from that distribution and fed to the decoder that outputs the reconstructed data $\hat{x}$

The ***encoder*** in a VAE maps the input data to a latent space, but instead of directly producing a point estimate like the regular autoencoder, it outputs parameters of a probability distribution. The *latent space* in a VAE is characterized by a continuous, multidimensional probability distribution. The encoder produces two vectors for the mean μ and the logarithm of the variance σ. Sampling directly from the encoder outputs (μ and σ) introduces stochasticity into the model and, in the training phase, this would require to compute gradients with respect to random elements which is problematic in the backprobagation algorithm. In order to circumvent this, a sample, typically $z = \mu + \sigma \cdot \epsilon$ with ϵ a variable of normal distribution with mean 0 and standard deviation 1, is drawn from the encoder-produced distribution [38]. This sampling is the so-called reparameterization trick enabling to make the sample z deterministic and differentiable for the backprobagation algorithm and forces the VAE to learn that similar inputs are mapped to similar regions in the latent space and, therefore, assures continuity of the latent space.
The ***decoder*** in a VAE maps the sample z back to the data space, aiming to reconstruct the input data. Thus, the output of the decoder is a reconstructed version of the input data, which can be compared to the original input to compute the reconstruction loss and in inference, the decoder only can be employed as a generative model.

During the training process, in addition to the likelihood cost function, a Kullback-Leibler loss function is optimized in order to set a compact latent space. Variants of VAE have been developed to address specific challenges and enhance their performance, for instance, the Conditional VAE (cVAE) that extends VAE by conditioning the input data on additional information [39], allowing the model to generate data based on specific conditions, such as generating images of a specific class.

1.8.3 Generative adversarial network

GANs consist of two neural networks, a generator and a discriminator which are trained together in a competitive manner. GANs have gained popularity for their ability to generate realistic data, making them valuable in various applications such as image generation, data augmentation and generative art. Figure 1.11 shows the GAN architecture.

The generator network takes a random normally distributed noise with variance = 1 as input. This makes the latent space automatically smooth and compact. It tries to produce

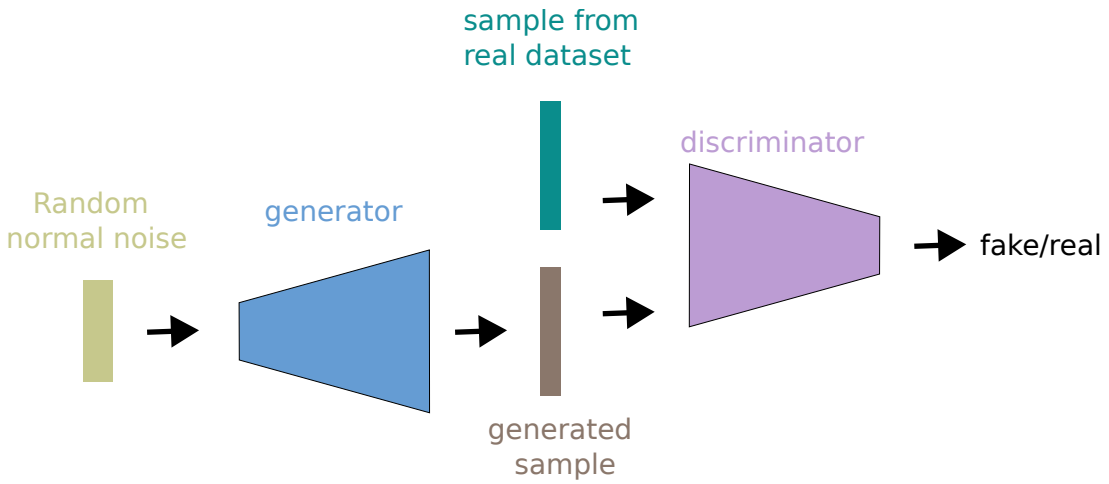

Figure 1.11: GAN architecture. The generator outputs generated from random noise. The discriminator determines whether its input is fake or real

synthetic data that is indistinguishable from real data. The discriminator network takes both real data and synthetic data as input and attempts to distinguish between them (binary classification) [40].

The training process of GANs is framed as a game, where the generator and discriminator are adversaries. The objective is to find an equilibrium where the generator produces data that is so realistic that the discriminator cannot reliably distinguish between real and synthetic samples. Generator and discriminator losses can be MSE and $BinaryCrossEntropy$ respectively.

The generator and discriminator are trained iteratively, each step of training loop involving the following:
The generator generates synthetic data samples from random noise. The discriminator evaluates both real and synthetic samples and calculates its loss. Gradients from the discriminator's loss are backpropagated to update the discriminator's parameters. The generator's loss is computed based on the discriminator's evaluation of its generated samples and gradients from this loss are backpropagated to update the generator's parameters. This process continues iteratively until the generator works satisfactory. In a way, the discriminator acts as a learned, problem-specific loss function. Training GANs can be challenging due to issues like mode collapse (where the generator produces limited diversity) and training instability. Various techniques have been developed to address these challenges such as Wasserstein GANs (WGANs) [40].

GANs have many applications in diverse fields:

Image generation: GANs can create high-quality and realistic images [41], leading to applications in art and fashion.

Super-Resolution: GANs can enhance the resolution of images, making them valuable in medical imaging and satellite imagery.

General latent representation learning: An example will be shown in the last chapter of the thesis, where a GAN is used to learn a latent representation of nanostructure geometries.

1.8.4 U-Net

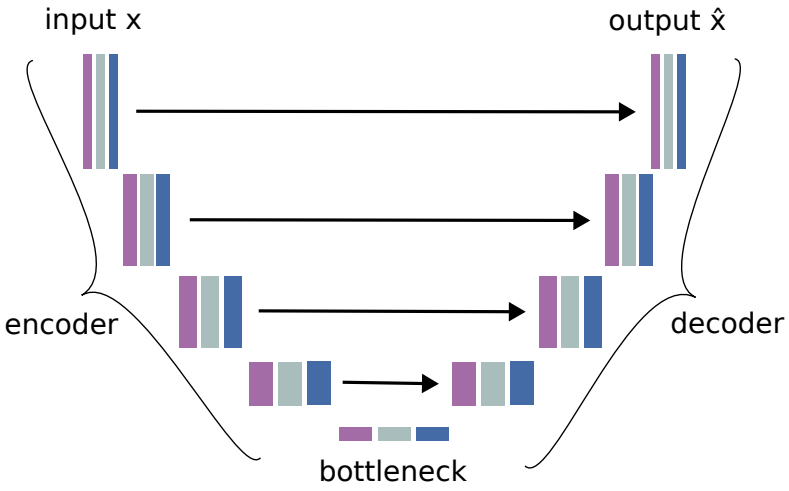

Figure 1.12: U-Net architecture. The encoder takes an input x and produces a latent space that is fed into the decoder to output the reconstructed data $\hat{X}$

Even though it will not be used in this thesis, U-Nets represent an important class of Deep Learning models, thus it will be briefly discussed in this section. U-Net is a convolutional neural network (CNN) architecture initially designed for biomedical image segmentation tasks. It has become a widely used model due to its effectiveness in segmenting images with high spatial resolution. U-Net derives its name from its U-shaped architecture, shown in Figure 1.12, which consists of an encoder path and a corresponding decoder path with skip connections between them.
The ***encoder*** path of U-Net resembles a typical convolutional network [42]. It consists of multiple convolutional layers followed by max-pooling layers. Each convolutional layer is responsible for extracting features from the input data while reducing its spatial dimensions through downsampling while expanding the depth dimension.
The ***decoder*** path is designed to reconstruct the segmented output from the encoded features. It consists of upsampling layers followed by convolutional layers. Each upsampling step increases the spatial dimensions until they are back to the original size of the input image.

Importantly, skip connections are established between corresponding layers of the encoder and decoder paths. These skip connections concatenate feature maps from the encoder with those from the decoder at the same spatial resolution. This allows the network to localize and preserve fine-grained details during the upsampling process [43]. The final layer of U-Net typically consists of a convolutional layer followed by a softmax activation function (for multi-class segmentation) or a sigmoid activation function (for binary segmentation). This layer generates the final segmentation mask, which predicts the class label or probability for each pixel in the input image.

1.9 Conclusion

In conclusion, the core of neural network architectures is the artificial neuron that computes the weighted sum of its inputs and potentially adds a bias vector and applies an activation function in order to introduce non-linearity into the network. Different types of deep learning models in the field of deep neural network serve for specific roles. Convolutional neural networks excel at processing grid-like data, making them ideal for tasks like image classification and object detection. They leverage convolutional layers to extract spatial features, making them a cornerstone of computer vision. Graph neural networks are designed for graph-structured data, allowing them to model complex relationships and dependencies. They have applications in many fields like social networks and recommendation systems. Multilayer perceptrons are versatile and often used for general-purpose deep learning tasks. With fully connected layers, they are effective for tasks like regression and classification on data with high level features and are the foundation of deep learning. Recurrent neural networks specialize in sequential data, for instance in natural language processing and time series analysis. Their recurrent connections enable them to capture temporal dependencies.

These data-driven models are trained with optimization algorithms such as minibatch stochastic gradient descent. We presented a selection of complex architectures. Autoencoders involve a encoder-decoder architecture with a latent space (bottleneck) in between. Variational autoencoders are close to the autoencoders except that the latent space corresponds to a probability distribution whereas generative adversarial networks have a generator and a discriminator that distinguishes generated and real data during training. U-Net is used for image segmentation which consists of classification at pixel level.

Chapter 2

Deep Learning for automatization of RHEED patterns monitoring

2.1 Introduction

Molecular Beam Epitaxy is used to create systems where materials with dissimilar properties are joined at an atomically sharp interface [44]. MBE can be seen as a refined form of vacuum evaporation in which directed neutral thermal atomic and molecular beams impinge on a heated substrate under ultra-high vacuum (UHV) conditions according to [45,46]. MBE is the gold-standard for the growth of epitaxial materials; this is due to the fact that MBE enables a high-level of control over the growth process [44]. This level of control is rooted in how the constituent elements are delivered to the growing surface of the thin film. Each element is individually heated to a temperature where evaporation/sublimation takes place, which creates a beam of atoms or molecules. These beams are directed towards a substrate crystal with an atomically flat surface. There, the elements adsorb, undergo diffusion on the surface and finally chemically bond. The temperature of the substrate can be tuned to an optimum value which, in a simplistic view, maximizes surface diffusion to confine nucleation and growth at the proper atomic sites. Furthermore, MBE growth takes place under ultrahigh vacuum (UHV) $< 1 \times 10^{-9}$ $Torr$ (for comparison, atmosphere is 760 $Torr$). The atomic beams suffer no scattering in route to the substrate, which enables highly uniform growth while minimizing both thermal leakage among the cells and elemental cross-contamination [47]. To monitor the material growth in MBE, Reflection High-Energy Electron Diffraction (RHEED) became an essential tool.

RHEED is a widely used in situ control method for MBE [48–51]. RHEED diffraction patterns provide information about the crystal surface with atomic resolution and since the ultrahigh vacuum in typical growth chambers allows an easy integration of electron beam systems in MBEs, RHEED has become a standard in situ characterization instrument in MBE, enabling unprecedented accuracy in monitoring the crystal growth, an overview illustration of the MBE-RHEED system is provided in Figure 2.1. RHEED is highly sensitive to several key MBE parameters such as the growth rate, the crystal structure, the lattice parameter and strain, etc [52–56]. However, RHEED images can be difficult to interpret, since the diffraction patterns produce information in the Fourier space. Furthermore, the actual recorded patterns are very sensitive to calibration and often also dynamic variations in the patterns over several time scales contain valuable information, rendering even more challeng-

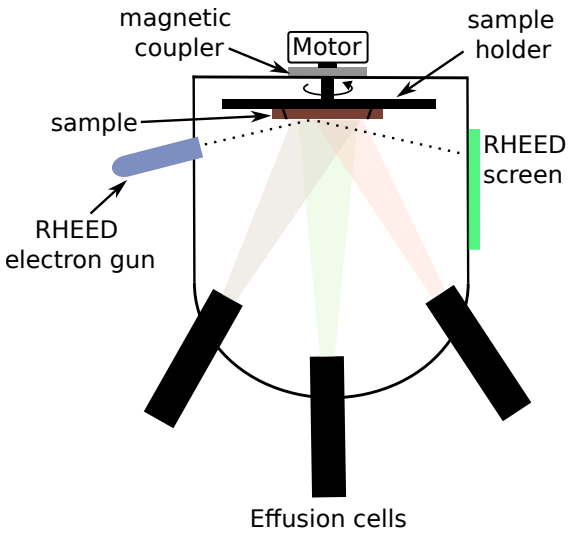

Figure 2.1: MBE and RHEED illustration. Growing sample, rotating around the surface normal, under constituting material beams with an electron beam arriving at a very small angle. The electrons are then diffracted by the surface atoms into the RHEED screen, where produced surface patterns are monitored.

ing. Real-time exploitation of RHEED data is therefore often limited to easily accessible information such as the deposition rate. Sophisticated analysis is usually done a posteriori, on recorded RHEED images or videos. Due to the complexity of the task, RHEED interpretation usually requires experienced operators, possessing years of machine-specific training.

This chapter on RHEED characterization using Deep Learning is structured as follows, section 2.2 is about the state-of-the-art focusing on the two most commonly used techniques (PCA and Deep Learning) for RHEED automatization monitoring, in section 2.3 we present our published work on substrate deoxydation detection by Deep Learning; the commercial substrate has a thin oxide film on top that needs to be removed to allow crystalline surface for the MBE growth. Section 2.4 contains our work on surface reconstruction monitoring. Surface reconstruction refers to the rearrangement of surface atoms of the material in growth affecting the surface morphology and electronic properties of the material. Typical examples of images from $c(4\times4)$ and (2×4) surface reconstructions can be found in Figure 2.20. Furthermore, the Azimuthal RHEED, map of RHEED intensity as a function of azimuthal angle enables detecting long periodicities and epitaxial orientations, we propose a Deep-Learning method determining the azimuthal angle from raw RHEED patterns in section 2.5.3.

2.2 State-of-the-art

In recent years, the crystal growth characterization witnessed significant advancements in the analysis of RHEED images using statistical analysis methods such as Principal Component Analysis (PCA) and Deep Learning. The complexity and richness of RHEED data present challenges in extracting meaningful information. To address this, researchers have turned to advanced statistical techniques to unlock deeper insights into crystalline structure, surface morphology and growth dynamics. However, as of the moment of writing this thesis, only a

few works have reported RHEED analysis with statistical or machine learning methods. In this section, we therefore discuss the most relevant works in this area, to give an idea of the state of the art on the automatization of RHEED patterns monitoring and focus on PCA which the principle is explained below and Deep Learning that has emerged as a cutting-edge approach for RHEED image analysis.

2.2.1 Principal component analysis

Principle

Principal Component Analysis (PCA) is a dimensionality reduction technique used to transform high-dimensional data into a lower-dimensional space while preserving the most important information [57, 58]. The fundamental principle of PCA is to identify and extract the directions (principal components) along which the data varies the most. By projecting the data onto these principal components, one can capture the most significant variability and reduce the dimensionality of the dataset. PCA achieves this by performing eigenvalue decomposition (EVD) or singular value decomposition (SVD).

Eigenvalue decomposition. The eigen-decomposition applies to squared matrices. Let X be a $(n \times n)$ data matrix. The covariance matrix is calculated from the centered data. It represents the pairwise covariances between different features. For two variables A and B, the covariance is given by [59]

$$cov(A, B) = \frac{1}{n-1} \sum_{i=1}^{n} (A_i - \bar{A})(B_i - \bar{B}) \tag{2.1}$$

Where $\bar{A}$ is the mean of feature A.
Then the unit eigenvectors v and eigenvalues λ of the covariance matrix C are calculated.

$$Cv = \lambda v \tag{2.2}$$

The eigenvectors represent the directions of maximum variance and the corresponding eigenvalues indicate the magnitude of the variance along those directions. The eigenvectors are ranked based on their corresponding eigenvalues. The eigenvector with the highest eigenvalue represents the direction of maximum variance and subsequent eigenvectors capture decreasing amounts of variance [59]. The first p selected eigenvectors (principal components) form the feature vectors, a matrix of those principal components in columns. The dataset is projected onto the selected principal components to obtain the transformed data in a lower-dimensional space.

$$TransformedData = FeatureVector \times CenteredData \tag{2.3}$$

Where $FeatureVector$ is the transpose of the principal components matrix so that the eigenvectors are now in the rows, with the most significant eigenvector at the top and $CenteredData$ is the mean-adjusted data transposed [59].

Singular value decomposition. As we cannot do an eigen-decomposition on data of arbitrary shape, because its matrix representation (all data-samples as vectors stacked together) is in general not square and invertible, the SVD is a generalized form that can be applied to rectangular matrices [60]. Let X $(n \times m)$ be a data matrix with n observations and m variables and $n > m$, two orthogonal matrices U $(n \times n)$ and V $(m \times m)$ exist, as well as a diagonal matrix $S = diag(\sigma_1, ..., \sigma_n)$ $(n \times m)$ with $\sigma_1 \geq \sigma_2 ... \geq \sigma_n \geq 0$ such that [60, 61]

$$X = U \times S \times V^T \tag{2.4}$$

The column vectors of $U = [u_1, ..., u_n]$ are called the left singular vectors of X, the column vectors of $V = [v_1, ..., v_m]$ are the right singular vectors of X and represent the eigen vectors of the variables, the diagonal values of S are the singular values representing the square root of the eigenvalues and the dimensionality reduction is accomplished by discarding the components that correspond to the lowest eigenvalues [62]. The data transformation is performed as in equation (2.3), a linear transformation of vector/matrix multiplication. On the other hand, Deep learning uses non-linear activation functions, so it is more expressive in the sense that it can develop a non-linear decomposition and hence has the potential to describe data with less components than PCA. In the following sections, we present the most relevant works that handle RHEED patterns analysis with machine learning and Deep Learning algorithms so far.

Classification of in situ reflection high energy electron diffraction images by principal component analysis [63]

This study proposes an unsupervised learning for RHEED image classification during the MBE growth of GaAs by two algorithms. The first one extracts features from the data and the second one performs the clustering of those features into different categories. PCA is used for feature extraction and the density-based spatial clustering of applications with noise (DBSCAN) method to group the data with similar features into the same group. The RHEED images were collected during the MBE growth of GaAs on an n-type GaAs (001) substrate. The (2×4) and $c(4 \times 4)$ pattern images were taken at substrate temperatures of 580 $°C$ and 480 $°C$, respectively. The collected images were 1024×1024 pixel uncompressed 14 bit grayscale images. The exposure time of all collected images was 23 ms. The images were obtained in the [110], [$1\bar{1}0$], [$\bar{1}\bar{1}0$] and [$\bar{1}10$] directions while rotating the substrate at 12 rpm. A data set of 219 images of the (2×4) pattern and 46 images of the $c(4 \times 4)$ pattern was used for dataset preprocessing, image resizing and vectorization. Each 1024×1024 pixel image was resized to 512×512 pixel and converted to a 1×262144 matrix for PCA.

Figure 2.2(a) shows a scatter plot of the coefficients of the two most significant principal components PC1 and PC2. One point in the plot indicates the coefficient of each PC. These coefficients represent the size of the basis vectors PC1 and PC2 reflected in a data set image. The symbol of each plot point is expressed in four types according to the temperature and direction at which each point is measured. In the upper left part and upper right part, the RHEED images were obtained in the [110] direction and [$1\bar{1}0$] direction of the $c(4 \times 4)$ pattern, respectively. Meanwhile, other RHEED images in the lower left part and lower right part were obtained in the [110] direction and [$1\bar{1}0$] direction of the (2×4) pattern, respectively.

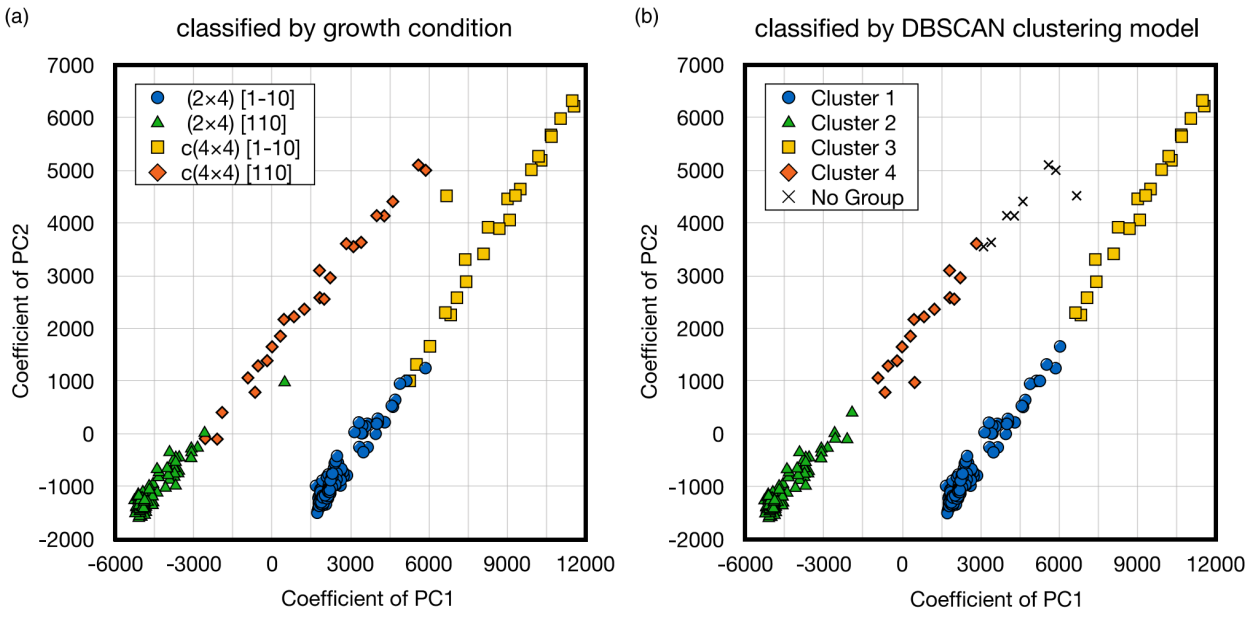

Figure 2.2: PCA and DBSCAN methods for RHEED patterns clustering. Clustering of the principal component analyzed RHEED patterns classified by (a) growth condition and (b) DBSCAN clustering model. Readapted from [63].

Figure 2.2(b) shows the result of clustering the scatter plot shown in Figure 2.2(a) by the DBSCAN method. The clustering parameter ε and the minimum number of points were 970 and 5, respectively. The clusters 1, 2, 3 and 4 corresponded to $(2 \times 4)[1\bar{1}0]$, $(2 \times 4)[110]$, $c(4\times4)[1\bar{1}0]$ and $c(4\times4)[110]$, respectively. The detailed results are displayed in the confusion matrix shown in Table 2.1 where two patterns (2×4), which had a lot of data and high density, displayed relatively high accuracy (over 99%). In contrast, the $c(4 \times 4)$ pattern had a relatively low accuracy. However, if the No group points, which were not included in any cluster, were included in the nearest cluster, cluster 4, an accuracy of more than 87% could be obtained in the $c(4 \times 4)$ pattern.

Directions	Cluster 1	Cluster 2	Cluster 3	Cluster 4	No group
$(2 \times 4)[1\bar{1}0]$	100.0%	0.0%	0.0%	0.0%	0.0%
$(2 \times 4)[110]$	0.0%	99.1%	0.0%	0.9%	0.0%
$c(4 \times 4)[1\bar{1}0]$	13.0%	0.0%	87.0%	0.0%	40.3%
$c(4 \times 4)[1\bar{1}0]$	0.0%	4.3%	0.0%	60.9%	30.4%

Table 2.1: Confusion matrix for the classification model. Readapted from [63].

Big-Data Reflection High Energy Electron Diffraction Analysis for Understanding Epitaxial Film Growth Processes [64]

An unsupervised learning approach on RHEED images is proposed in this study using principal components analysis, k-means clustering and Fast Fourier Transform to identify the areas in the RHEED pattern with the most statistical variance and identify transitions in the growth mode. Here we will take the two first machine learning approaches. As a test

case, the authors of [64] analyze a RHEED data set acquired for growth of a 25 unit cell-thick film of $La_{3/8}Ca_{5/8}MnO_3$ ($LCMO$) on an etched (001) $SrTiO_3$ substrate by Pulsed Laser Deposition (PLD).

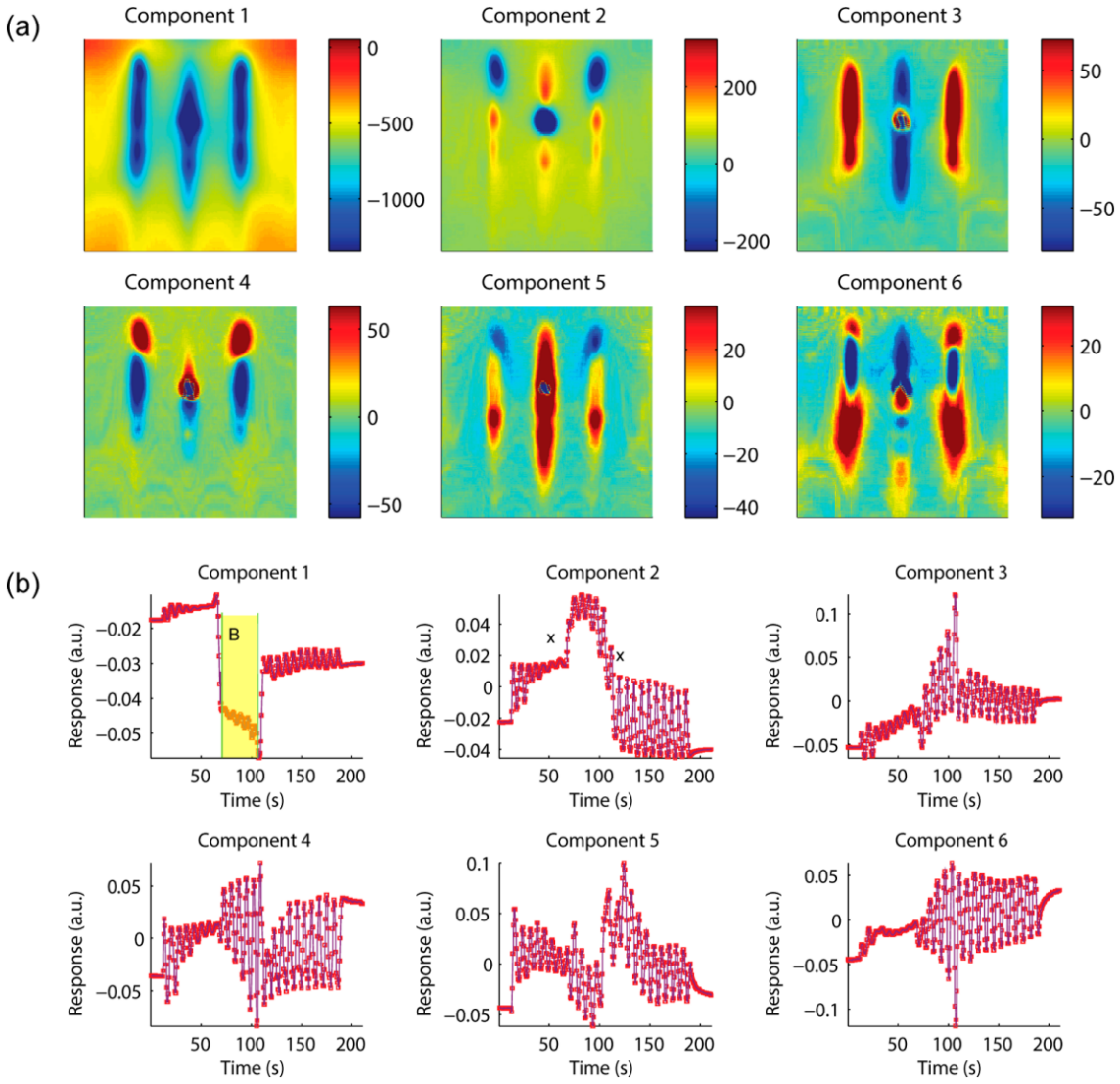

Figure 2.3: PCA on RHEED data. The PCA of the entire acquired data set is shown above with the (a) eigenvectors and (b) eigenvalues plotted. The decomposition results in a set of eigenvectors (images), which when multiplied by the respective time-dependent eigenvalues, can reproduce the entire movie. The intensity of the RHEED beam was altered twice during the acquisition; these times are marked with x and are clearly visible in the plotted eigenvalues. A small segment of the data set is highlighted in yellow in (b) and data in this time window were reanalyzed by PCA with the results shown in Figure 2.4.

PCA. The eigenvectors (components) in this decomposition are 2D images. Figure 2.3 shows the (a) first six eigenvectors and (b) the time-dependent eigenvalues. The first six eigenvectors (out of 1054) account for 75.7% of the statistical variance in the data set. Limiting the analysis to these 6 PC thereby represents a very large decrease in data size while still retaining the majority of the information. The eigenvalues indicate periodic oscillations, due

to the growth, as well as two large changes in baselines indicated by the "x" marks on the component 2 plot in Figure 2.3b, where the intensity of the RHEED beam was manually adjusted, at $t =\sim 65$ s it was increased and at $t =\sim 110$ s it was decreased. The first eigenvector represents a mean intensity profile, while the second component appears to show some additional spots more suggestive of a 3D or imperfect 2D growth. The third component appears to be mostly pure streaks.

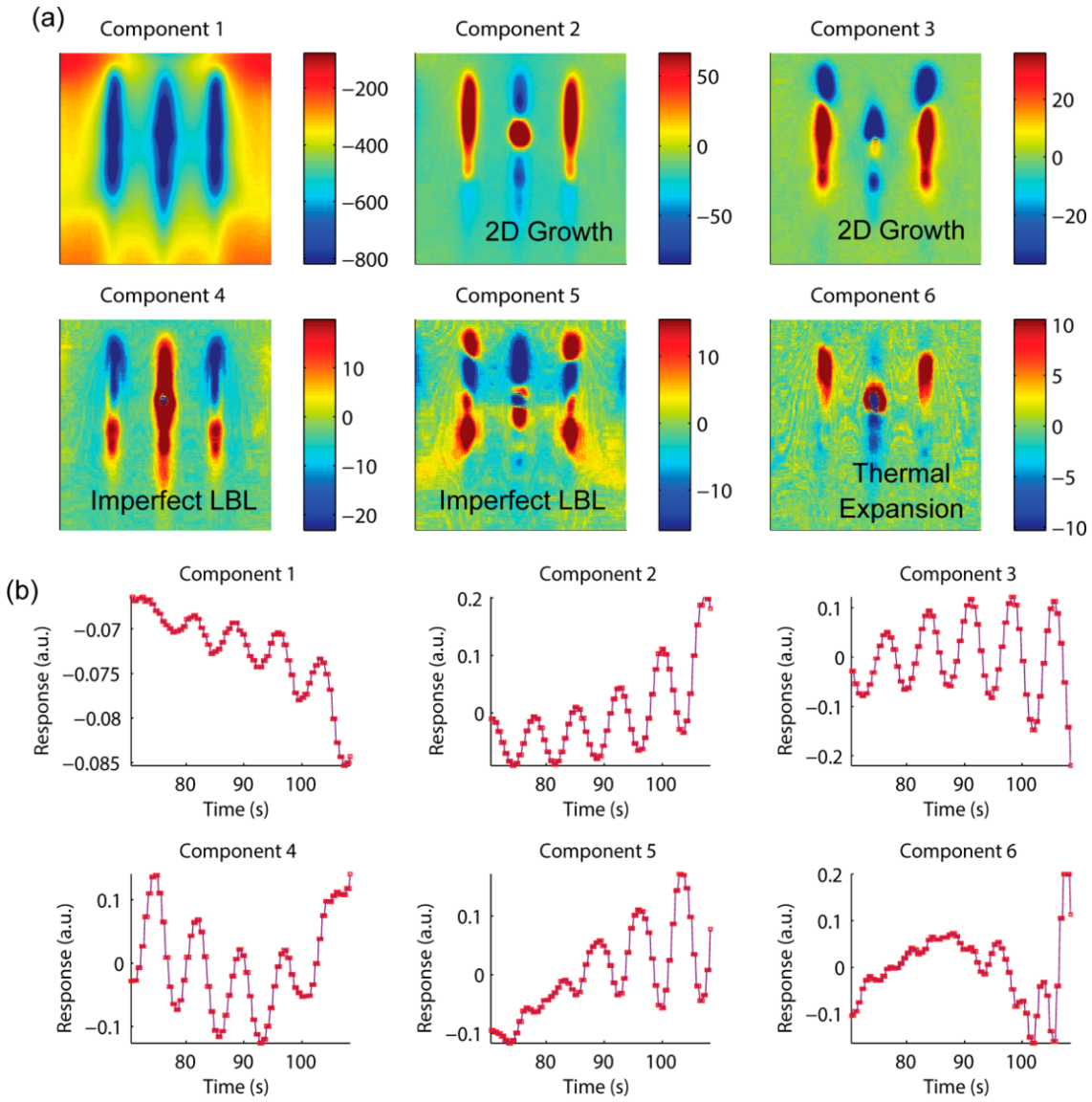

Figure 2.4: PCA on RHEED, segment B. The smaller time-series segment marked B in Figure 2.3 was reanalyzed with PCA; the results are plotted above with the (a) eigenvectors and (b) eigenvalues. Components 4 and 5 show extra spots characterizing them as imperfect layer-by-layer (LBL) growth modes.

In order to reduce the effect of the manual intensity change on the decomposition, the data were analyzed by PCA restricted to time windows $t = 70$ s and $t = 108$ s, referred to as "segment B" and outlined in the component 1 plot in Figure 2.3b. In this second decomposition, shown in Figure 2.4, it is interesting to note that there is a little periodicity in the

sixth component and in the eigenvector the two side spots are clearly shifted down, while the specular spot is shifted up, with respect to the other components. In the used PLD setup there is a thermal expansion of the substrate suspension system over time through heating, which effectively changes the incident angle of the beam toward more gliding angles, so the authors estimate that component 6 may be partly a result of an adjustment due to this thermal expansion. Components 4 and 5 appear to show extra transmission spots and may be related to either 3D growth modes, or imperfect layer by layer growth while Components 2 and 3 are indicative of 2D growth. Here we note the ability of PCA to identify the relevant behaviors and modes that are present in the data set, as well as in reducing the data set to a manageable size.

K-means clustering. K-means clustering is a vector quantization method in which the RHEED image sequence is partitioned into K clusters based on statistical similarity [65]. It is less evident from the PCA analysis whether there are any significant changes in the growth modes occurring in the film, even though there appear to be several components related to different types of growth. To explore this aspect the authors use the k-means clustering approach to split the RHEED image sequence into k clusters as depicted in Figure 2.5. Each observation (image) belongs to the cluster with the nearest mean.
A dendrogram is a hierarchical grouping diagram used to organise data into a tree structure based on similarities [66]. A hierarchical clustering is shown in Figure 2.5a using a dendrogram in order to determine the optimum cluster number and indicates that this RHEED image sequence can be grouped into 10 clusters. The 10 groups are shown boxed in blue in the dendrogram. The k-means approach was applied to the RHEED image sequence and the data set was divided into 10 distinct clusters. The mean of all members of the individual clusters was computed and is plotted in the upper panel in Figure 2.5b, while the temporal distribution of the clusters is shown in the lower panel in Figure 2.5b. Three important clusters between times 70 s and 110 s are bounded by blue dashed rectangles in the upper and lower panels and they indicate a streaky pattern reminiscent of a more disordered type of surface. The four clusters bounded by green dashed rectangles appear to show more spot-like diffraction patterns and are less streaky, indicating significantly less disorder in the film once it is growing at these thicknesses. This second set of cluster begins appearing after $t \sim 115$ s based on the timeline in the lower panel in Figure 2.5b. This suggests that at $t \sim 115$ s there is a crossover to a more ordered growth pattern. The authors confirmed these results with Scanning Tunneling Microscope (STM) topography images. It is notable that k-means clustering can readily identify the presence of growth mode transitions and provides a statistical method to determine the transition point.

Machine-learning-assisted analysis of transition metal dichalcogenide thin-film growth [65]

In this study, a ML-assisted in situ RHEED analysis, including PCA and K-means clustering, is performed to understand the epitaxial growth of $ReSe_2$ thin films, with different thicknesses, on graphene. The authors prepared $ReSe_2$ thin films, with varied thicknesses, on graphene substrates. Figure 2.6a shows the atomic structure of the distorted 1T (1T') $ReSe_2$. Figure 2.6b–d show the schematic models of the graphene substrate and ReSe2 thin films with 0.3 and 3 unit cells (UC), respectively. Initially, the bilayer graphene substrate was prepared, yielding a sharp RHEED pattern as shown in Figure 2.6e. After 4 min of film

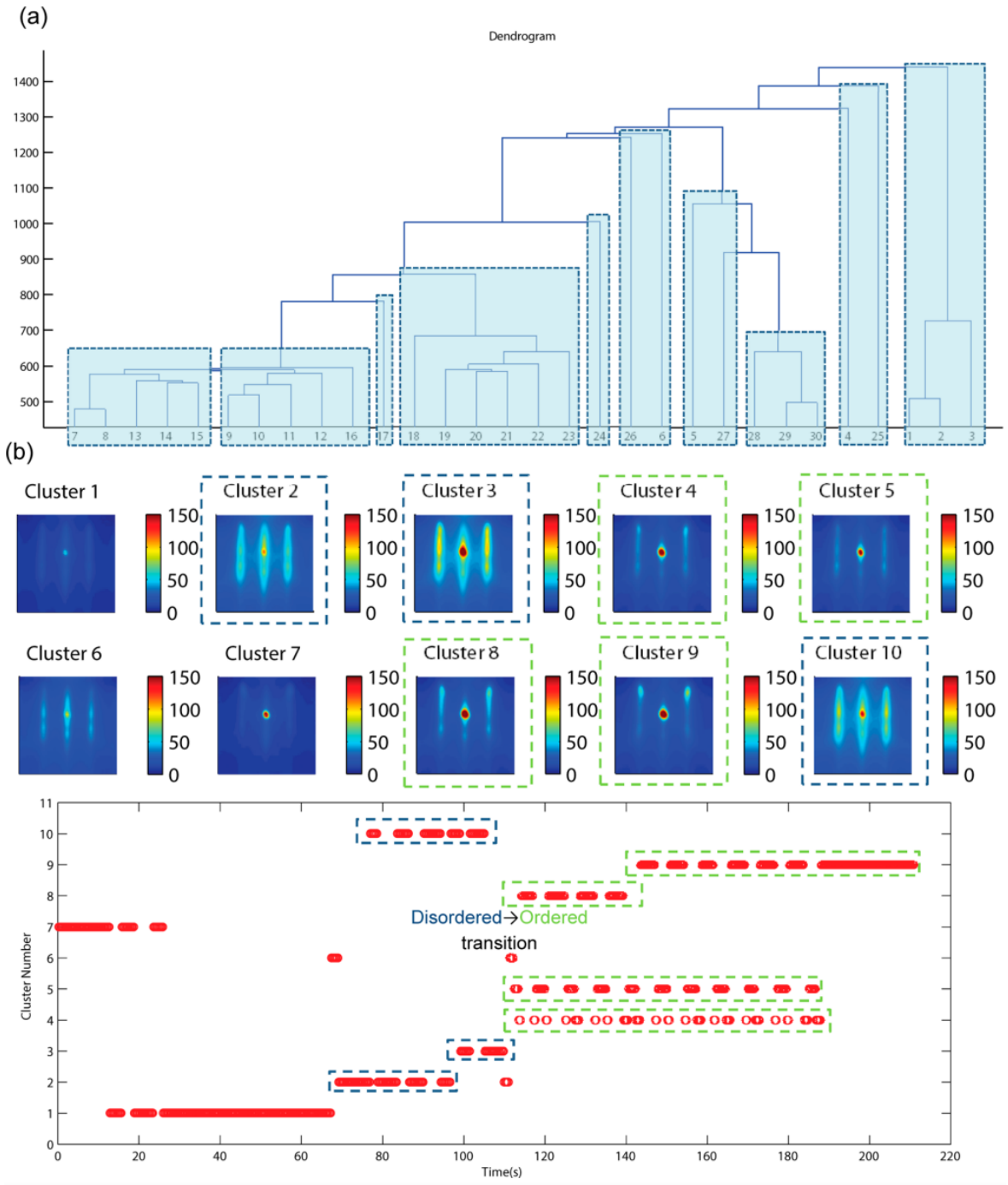

Figure 2.5: k-Means clustering on RHEED data. (a) Dendrogram of the RHEED image sequence, with leaves grouped into 10 clusters. (b) The result of k-means clustering of the entire data set is shown in the upper and lower panels in this portion of the figure. The upper panels reveal the image obtained by averaging the members of each cluster, while the timeline in the lower panel indicates the temporal dependence of the clusters. At about 115 s, the pattern changes from being dominated by streaks (indicating a more disordered three-dimensional growth) to a clear 2D growth mode dominated by specular and side spots.

growth, additional streaks of the $ReSe_2$ lattice emerged in the RHEED pattern, indicated by red arrows in Figure 2.6f. After 62 min of deposition, the RHEED pattern of graphene

completely disappeared, leaving only the $ReSe_2$ streaks, as shown in Figure 2.6g.

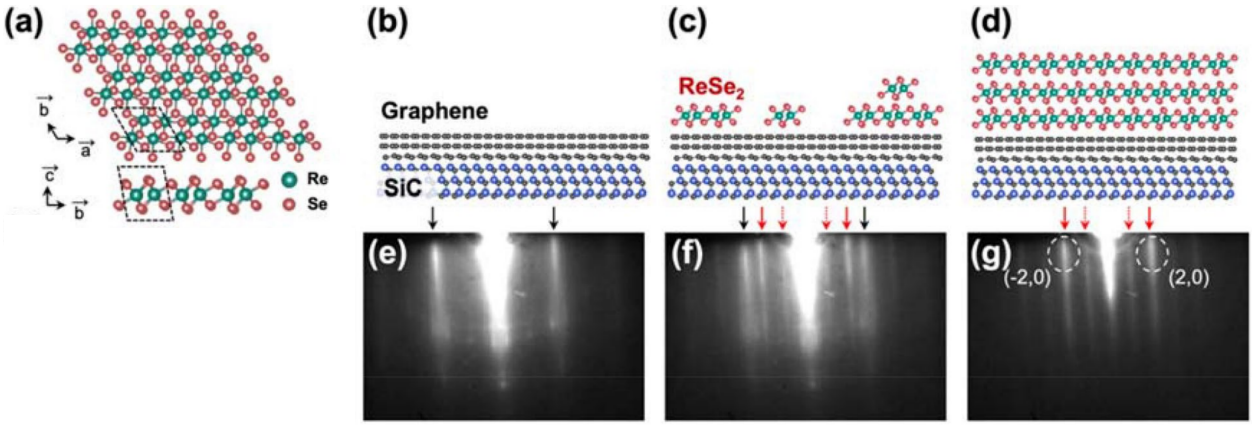

Figure 2.6: Growth and characterization of $ReSe_2$ thin films. a) Crystal structures of 1 T' $ReSe_2$. b–d) Schematic models of the graphene substrate and $ReSe_2$ thin films with 0.3UC and 3UC. e–g) RHEED images; and the black and red arrows in the RHEED images indicate the bilayer graphene substrate and $ReSe_2$ diffraction streak, respectively. Readapted from [65].

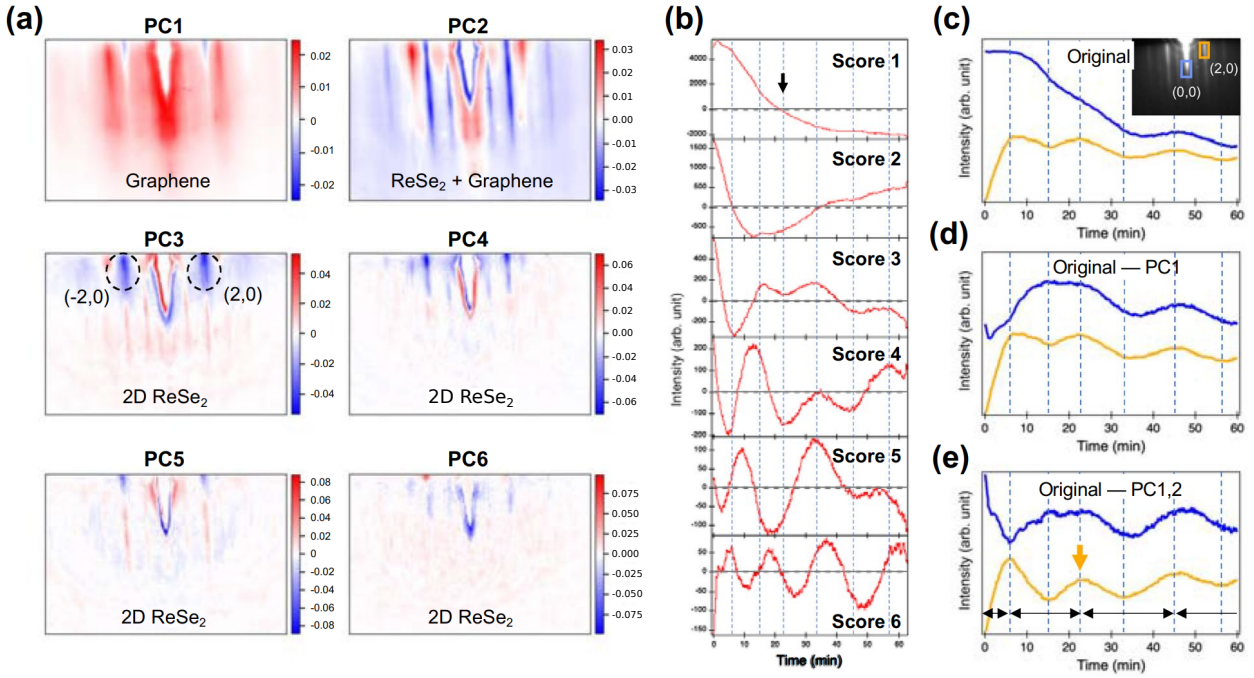

Figure 2.7: PCA results: a) Six PCs of the RHEED video for the 3UC-thick $ReSe_2$ thin film and b) the corresponding score plots. Component 1 (PC1) shows the diffraction signal of graphene, while component 2 (PC2) contains the signals of both the graphene and $ReSe_2$ layers. Component 3–6 (PC3-6) show the signal of only the 2D growth of $ReSe_2$ layer. c–e) The intensity plots of the (c) original RHEED video and d, e) modifed RHEED video. Blue and orange lines denote the (0,0) and (2,0) diffraction streaks of the $ReSe_2$ thin film, respectively. Readapted from [65].

PCA. First, the RHEED video of the $ReSe_2$ film is analyzed by PCA. Figures 2.7a,b show the first six principal components (PCs) and their corresponding score values. The six components add up to 98.95% of statistical variance in the dataset, implying most of the dataset

can be represented by a few components and scores. Especially, PC1 has the most variation (91.98%) in the RHEED video. The PC1 in Figure 2.7a shows two major characteristics. First, the positive (red) area well matches the graphene pattern shown in Figure 2.6e. On the contrary, the negative (blue) area matches with the (-2,0) and (2,0) diffraction points of $ReSe_2$ (surrounded on Figure 2.6g). The score 1, defined as the change of PC1 over time, decreases gradually and undergoes a sign change from positive to negative near the third dashed line in Figure 2.7b. This result implies that in the initial RHEED video, a gradually decreasing trend of the graphene signal is primarily observed. The second component, PC2, dominates the $ReSe_2$ streaks and minor diffraction points on the graphene and SiC substrates. The negative value of PC2 represents the epitaxial 2D growth of the $ReSe_2$ thin film, which is evidenced by the similar RHEED pattern of $ReSe_2$ in Figure 2.6g. The positive (red) region of PC2 includes the graphene diffraction streaks and several additional spots in the middle. The initial decrease in score 2 (Figure 2.7b) indicates that the substrate pattern disappears and the $ReSe_2$ pattern begins to emerge, corresponding to the first dashed line. Conversely, PC3–6 contain the (-2,0) and (2,0) diffraction signals of the 3UC (3 Unit cells) $ReSe_2$ layers. The corresponding scores 3–6 exhibit an oscillating behavior (Figure 2.7b). In the MBE growth, the oscillating behaviors of specular or diffraction spots are used to estimate the film thickness and to analyze the growth modes [67]. Although, the contribution of PC3–6 to the entire RHEED signal is $< 2\%$, they contain physical meaning about the film thickness and its growth mode. In Figure 2.7c, the (0,0) peak intensity gradually declines, representing the graphene contribution, which is well correlated to score 1. To separate the weak $ReSe_2$ signal from the original video, the authors made the modifed RHEED data using mPCA (modified Principal Component Analysis), it consists of consecutively subtracting graphene-related components (PC1 and PC2) from the raw RHEED video ("original - PC1,2" as mentioned in Figure 2.7d,e). Figure 2.7d,e show the intensity plot of the (0,0) (blue lines) and (2,0) (orange lines) streaks obtained from the mPCA video sets. In Figure 2.7d, the subtraction of PC1 mainly changes the intensity plot within the initial period up to the third dashed line (23 min). This change indicates the signal transition from graphene to $ReSe_2$, consistent with the sign change in score 1 (indicated with an arrow in Figure 2.7b). In Figure 2.7e, further subtractions of PC1 and PC2 result in stable oscillations for both blue and orange curves. Such oscillatory behaviors of the (0,0) and (2,0) streaks are likely linked to the layer-by-layer film growth, as mentioned in [51]. The consistent oscillating behaviors of the blue and orange curves in Figure 2.7e provide accurate information about the film thickness such that the resulting film thickness of 3UC is consistent with the illustration presented in Figure 2.6c.

K-means clustering. For comparison with the PCA results, the authors of [65] analyzed the same RHEED dataset by categorizing the images using K-means clustering, employing a different number of clusters (K=2–6). A cluster can be represented by its centroid, which is the arithmetic mean of the data in the cluster. Figures 2.8a,b show the time-dependent clustering for each K value and the corresponding centroids. As K is increased from 2 to 6, more divided sections appear for the initial growth time (i.e. <35 min), implying that the major pattern change mostly occurs at the initial duration. The boundaries between the clusters show good alignment with the vertical dashed lines for K=5 and 6 (Figure 2.8a). As shown in Figure 2.8c, the cost function (the accumulated differences between the clusters and the original data) is used to determine the valid number of clusters. The cost function is saturated when K>4. To investigate the evolution of the centroids in detail, the difference between the adjacent centroids ($\Delta C_{i(i+1)}$) is plotted as shown in Figure 2.8d by subtracting

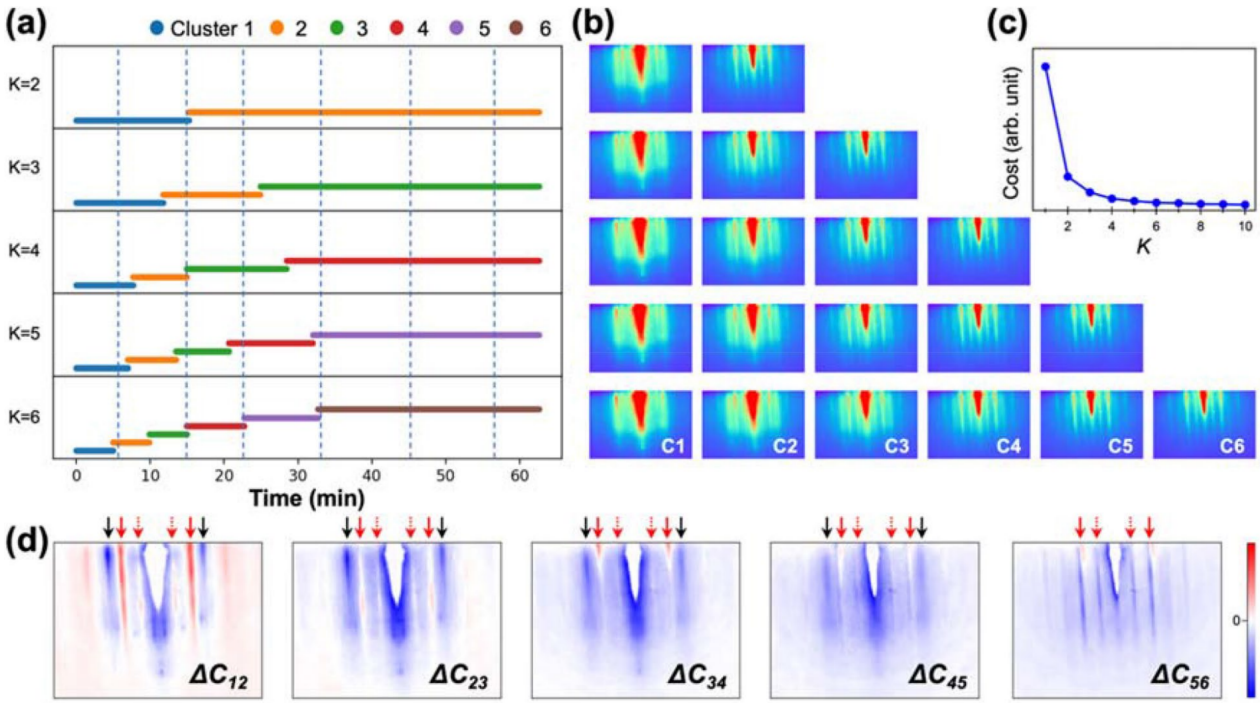

Figure 2.8: K-means clustering analysis of the RHEED video of the 3UC $ReSe_2$. a) Clusters with number of clusters (K=2–6) and b) their corresponding centroids. c) Cost function as a function of K. d) Difference between the adjacent centroids for K=6. Reprinted from [65].

a former centroid (C_i) from a latter one (C_{i+1}). Here, the positive (red) and negative (blue) regions represent the emerging and disappearing characteristics in the RHEED patterns, respectively. A distinct feature of ΔC_{12} is the emerging $ReSe_2$ streak signal (indicated with red arrows), which corresponds to the emerging $ReSe_2$ signal in the PCA. The graphene signal (black arrows) shows a gradually disappearing trend up to ΔC_{45} (23 min). This boundary corresponds to the third dashed line, at which the graphene signal nearly disappears as score 1 becomes negative in the PCA (Figure 2.7b). After the graphene signal disappears, ΔC_{56} mostly shows the intensity variations in the $ReSe_2$ streaks, implying a homoepitaxial growth regime. Therefore, the results obtained by K-means clustering with K>4 are consistent with those of the PCA.

2.2.2 Deep Learning

Classification of Reflection High-Energy Electron Diffraction Pattern Using Machine Learning [68]

The authors in this study propose a measurement method for identifying the RHEED pattern of GaAs substrates using a CNN model to classify (2×4) and $c(4\times4)$ surface reconstructions as illustrated in Figure 2.9.

(2×4) pattern images were taken at a substrate temperature of 580 $^{\circ}C$ and $c(4\times4)$ pattern images were taken at 480 $^{\circ}C$. RHEED patterns from [110], [1$\bar{1}$0], [$\bar{1}\bar{1}$0] and [$\bar{1}$10] directions were obtained with one rotation while rotating the substrate at 12 rpm. In case of no

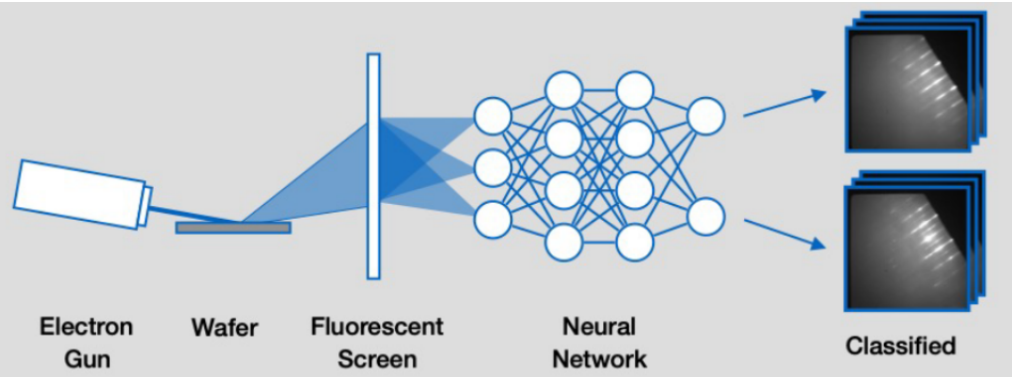

Figure 2.9: Binary classification of surface reconstructions. The RHEED system pattern images are fed to a neural network for binary classification. Reprinted with permission from [68]. Copyright 2020 American Chemical Society.

significant change in the incident angle and position of the electron gun due to rotation, the same RHEED image was obtained in [110] and $[\bar{1}\bar{1}0]$ directions, as well as $[1\bar{1}0]$ and $[\bar{1}10]$ directions. The collected RHEED images were stacked side-by-side as $[110]+[1\bar{1}0]$ and $[\bar{1}\bar{1}0]+[\bar{1}10]$ as shown in Figure 2.10, the stacked images are resized to 512×128 pixels. A total dataset of 340 (2×4) images and 111 $c(4 \times 4)$ images is used to train a CNN.

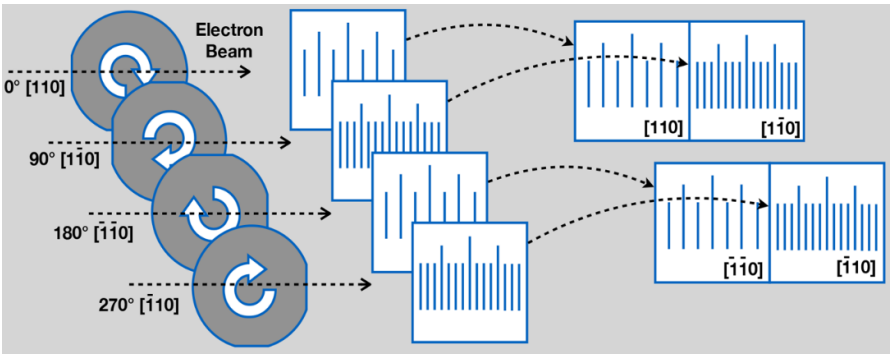

Figure 2.10: Data preprocessing for surface reconstruction classification. RHEED pattern images from [110], $[1\bar{1}0]$ are stacked side-by-side as well as the patterns from $[\bar{1}\bar{1}0]$ and $[\bar{1}10]$ directions in order to have different features in the same image. Reprinted with permission from [68]. Copyright 2020 American Chemical Society.

n = 451	predict: (2×4)	predict: $c(4 \times 4)$
Actual: (2×4)	100.0%	-
Actual:$c(4 \times 4)$	-	100.0%

Table 2.2: Confusion matrix of the binary classification model in [68].

The model is trained for 20 epochs. After learning, (2×4) and $c(4 \times 4)$ are distinguished by a 100.0 % probability for all test data (Table 2.2). As the authors said, this high accuracy at small epochs is attributed to the observations in a limited environment, only few classification categories and relatively little data. Relatively small data sets reduce the number of out-of-reference data sets, which increases the accuracy. However, this comes at the cost of the

generalizing capacity of the neural network. The model risks to break down if the input data changes, e.g. due to slight changes in the machine geometry or due to RHEED window deterioration.

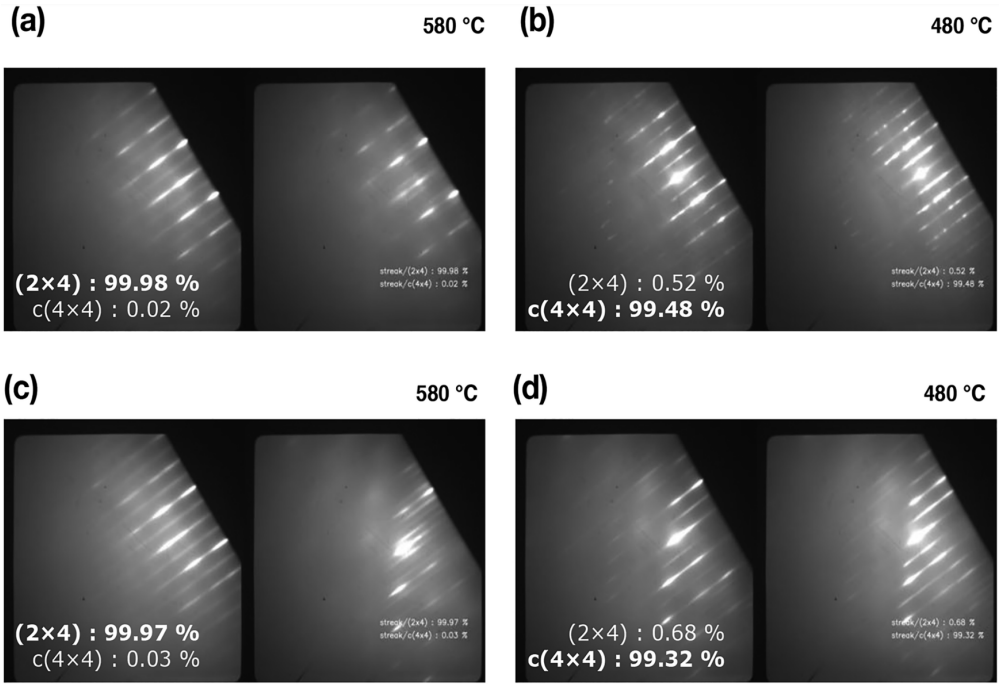

Figure 2.11: Results of the surface reconstruction classification model. Guessing probability of several test data sets. Reprinted with permission from [68]. Copyright 2020 American Chemical Society.

Figure 2.11 shows the test data as well as its guessing probability. Figures 2.11a,b represent typical (2×4) and $c(4 \times 4)$ image data, respectively, on which the predictions made by the neural network are displayed. On the other hand, (c) and (d) show images not taken from the exact $[110]/[1\bar{1}0]$ angles. Despite the images in this unusual measurement environment, the established model correctly classifies the two patterns.

Multiclass classification of reflection high-energy electron diffraction patterns using deep learning [69]

In this study, the authors (also of [68]) report a multiclass classification model for RHEED patterns using a CNN. The model was trained using images from three categories of patterns (GaAs (2×4), GaAs $c(4 \times 4)$ and InAs QD). The RHEED dataset is prepared as in the previous section; a difference is that the images are for this case resized to 256×128 pixels for fast calculation. 340 GaAs (2×4) images, 111 GaAs $c(4 \times 4)$ images and 143 InAs QD spot pattern images were prepared to train the multiclass CNN of which the architecture is depicted in Figure 2.12.

In classification problems, performance metrics such as accuracy, precision, recall and F1 score can be used to evaluate the predictive performance. Precision is the ratio of the correctly predicted positive images to the total number of predicted positive images. Recall is the ratio

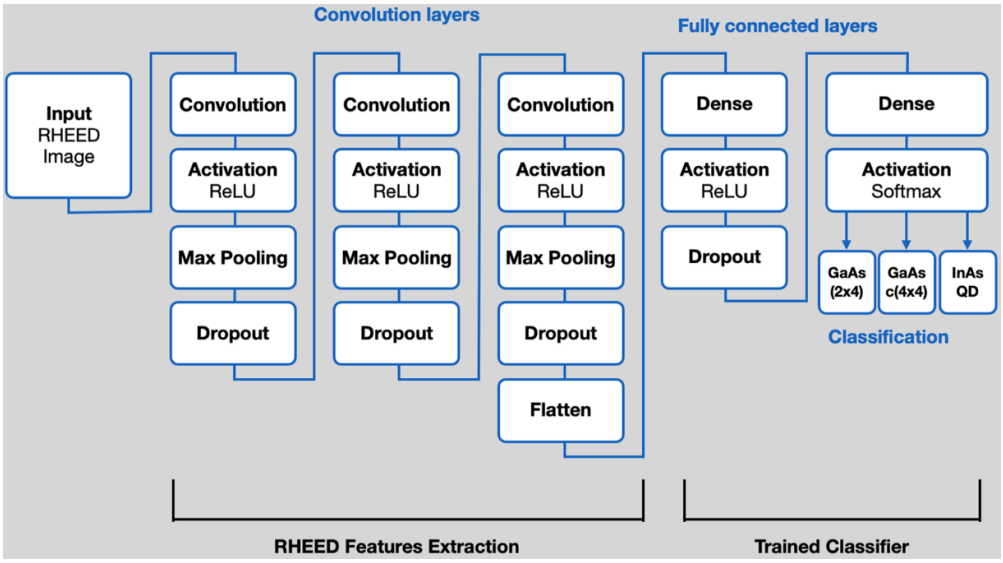

Figure 2.12: Multiclass classification model architecture. The classification model consists of three convolution blocks, two fully connected layers and three outputs corresponding to GaAs (2×4), GaAs $c(4 \times 4)$ and InAs QD with Softmax as final activation function. Reprinted from [69].

n = 119	predicted: (2×4)	predicted: $c(4 \times 4)$	predicted: InAs QD	Total	F1
Actual: GaAs(2×4)	100.0%	-	-	57.1% (68)	1.0
Actual: GaAs$c(4 \times 4)$	-	100.0% (23)	-	19.3% (23)	1.0
Actual: InAs QD	-	-	100.0% (28)	22.7% (28)	1.0
Total	57.1%(68)	19.3% (23)	22.7% (28)	100.0% (119)	1.0

Table 2.3: Confusion matrix of the multiclass classification model in [69].

of correctly predicted positive images to all the images in the actual class. The F1 score is the weighted average of precision and recall.
The neural network model is trained for 30 epochs where the accuracy is converged to 100%. Consequently, the established model can distinguish (2×4), $c(4 \times 4)$ and spot patterns with a 100.0% probability for 119 images of the test data. Since there were no false positive or negative cases, the F1 score for evaluating the model is 1.0 as shown in the confusion matrix in Table 2.3. As in the previous section, the authors consider that this high accuracy at small epochs was provided by observations in a controlled environment (fluorescent screen in a dark environment; same detector; same exposure time), grayscale images without color channels, only three classification categories and relatively short-term datasets.

Application of machine learning to reflection high-energy electron diffraction images for automated structural phase mapping [70]

This work proposes a deep learning-based analysis method for automating the identification of different RHEED pattern types that occur during the growth of a material.

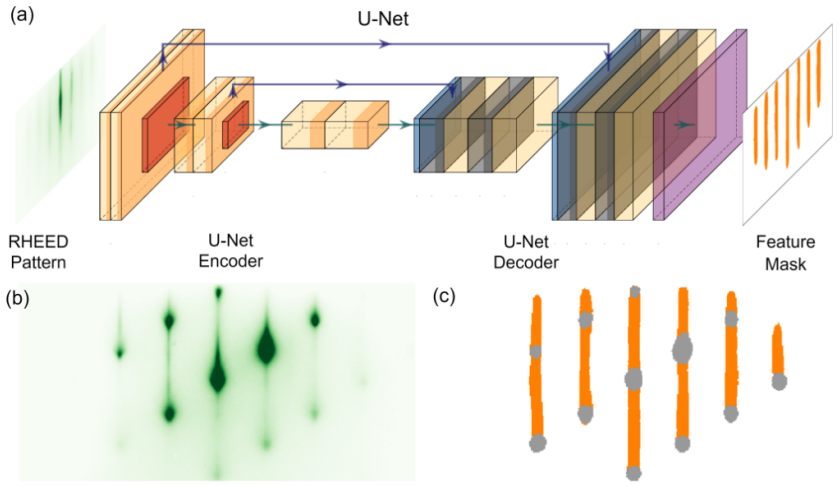

Figure 2.13: U-net architecture for RHEED pattern features extraction. Schematic diagram of U-Net and example results: (a) A simplified schematic of the U-Net model that takes in an RHEED pattern and outputs a binary mask of features of interest. (b) An example of input RHEED pattern containing both spot and streak features to the U-Net model. (c) The predicted masks for spot and streak features are shown in gray and orange colors, respectively. Reprinted from [70].

Identification of diffraction features. In terms of image analysis, the intensity variations, which can range from very low to exceeding the dynamic range of the camera, can be particularly problematic. Thus, as the first step in the RHEED image analysis pipeline, the authors of [70] created a system for extracting the diffraction pattern features of interest. This helps to reduce the dynamic range problem since it allows different pattern regions to be separated and analyzed individually. This step can be approached as a standard image segmentation task and a U-net is used which the architecture is depicted in Figure 2.13.

Separating diffraction regions. Since the spot and streak features in RHEED images are typically well separated spatially, a Connected Component labeling algorithm implemented by the Scikit-Image Python library is used to give each connected area a unique label. The algorithm works by finding and connecting neighboring pixels classified by the U-Net as either a spot or a streak. Pixels from disconnected areas are grouped into different regions.

Tracking the direct beam. The authors implemented an object tracking algorithm that matches regions from the current frame to regions from the previous frame. According to the authors, the Intersection over Union (IoU) tracking algorithm is sufficient for this application, as it selects the region that mostly overlaps with the previously matched direct beam region. The degree of overlap is measured by the IoU, which is given by

$$IoU = \frac{|A \cap B|}{|A \cup B|} \tag{2.5}$$

where A and B are the two masked areas in the two consecutive images. Tracking the shared region between image frames is done by finding the region pairs with the highest IoU value. The naïve approach to identify the direct beam spot as the top one is only used in the first image of the series or when the algorithm loses track of the direct beam.

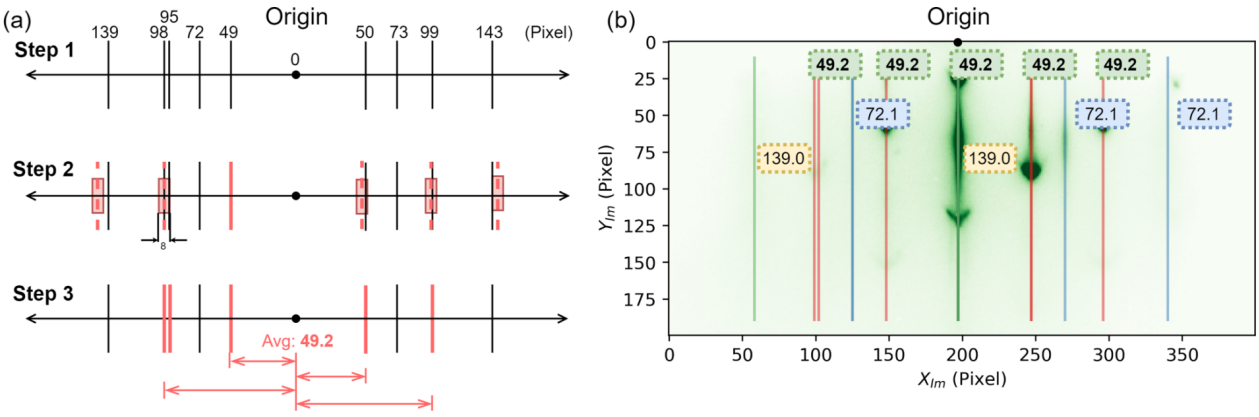

Figure 2.14: Extracted regions for RHEED features identification. Analysis of the peak location periodicities: (a) Diagram illustrating how peaks are grouped based on the distance from the origin. Solid lines show peak locations. Step 1: lists all the peaks discovered. Step 2: The peak closest to the origin is selected and labeled in red. Based on this peak, its multiplicity is constructed and shown in the red dash line with an error box. Step 3: Any peaks that are within the range of the error box are considered to be in the same base distance and are labeled in red. (b) All labeled peaks overlaid on the original diffraction pattern. The number shows the average peak distance from the origin normalized by the multiplicity. The red dash line highlights the position of peaks that are analyzed and grounded in (a). Reprinted from [70].

Identifying structural phases. The horizontal distances between diffraction spots or streaks represent different atomic spacings. Even if any internal structure of the streaks or the vertical positions of the spots are ignored, the horizontal spacings alone can be used to generate a fingerprint of a structural phase and thus distinguish different crystal phases. The trained U-Net model is used to create the spot and streak masks for each cropped image; these masks are then used to extract individual pattern features. For each spot and streak type feature, the signal is integrated vertically, compressing the image data in the analysis window to a one-dimensional feature. The central positions of the diffraction spots and streaks are found by the peak finding algorithm implemented in the SciPy python library. The distances between the extracted peak locations and the central specular peak can be used to discover periodicities in the atomic spacings. The analysis of the periodicity is done by finding sets of distances that are multiples of a base distance. This is achieved by determining all horizontal distances present in each image and picking the shortest distance of a peak from the specular position. The algorithm then iteratively extracts the base distances by choosing at each step the smallest unselected distance and adding it to the list of base distances. It then determines all peak distances that are multiples of the current base distance within a given tolerance and marks these distances as selected, assuming that all these peaks belong to the same base distance. The algorithm continues until all distances have been selected, producing a set of base distances characterizing each image. An illustration of the algorithm is depicted in Figure 2.14.

Phase mapping by region intensity. The horizontal spacing information is sufficient for detecting the presence of a given phase, but not for quantifying its abundance. The authors developed an algorithm to compute the relative intensity for each base peak distance, helping to quantify the phase composition of a multiphase sample. The algorithm computes the intensity relative to the total intensity for each identified base distance for each RHEED image. However, different RHEED images have different base distances; thus, a "unification" step is required to express them as a fixed-length vector for later pattern clustering. Here, the density-based spatial clustering of applications with noise (DBSCAN) algorithm is used to group similar base distances into clusters. The number of clusters, i.e., the overall number of distinct base distances, determines the total number of base distances, thus the size of the vector. Each vector provides a unique representation of the RHEED image of a sample, with components representing the relative intensity at a specific base distance, as shown in Figure 2.15. Sample identifier notation: sample made at 10^{-A} Torr at $B°C$ is denoted as A-B. For instance, the sample 5-600 was made at 10^{-5} Torr of P_{O2} at the substrate temperature of 600 $°C$. Figure 2.15b shows a growth parameter phase map with extracted feature vectors for the iron oxide growth mapping experiment which is compared to a human-labeled X-ray diffraction (XRD is a material analysis technique [71]) mapping result in Figure 2.15c. As can be seen there, regions containing similar vectors generally match well with the XRD measurement shown in Figure 2.15c.

Interpretation and analysis of RHEED feature mapping. The autors first present an analysis of just the RHEED features to show that one can achieve a semiquantitative analysis of phase mapping using RHEED data only. Phase compositions can be inferred from the extracted phase map from RHEED patterns when combined with knowledge of known materials phases and their structures. The phase map predominately consists of two distances: 13.83 and 20.38 nm^{-1}. $\alpha - Fe_2O_3$ is isostructural to sapphire thus epitaxial growth is expected. The first distance, 13.83 nm^{-1}, is in proximity to the $\alpha - Fe_2O_3$ $(10\bar{1}0)$ plane. Thus, the first distance shown as a green pie slice in the phase map represents the epitaxially grown hematite ($\alpha - Fe_2O_3$) phase. The second distance, 20.38 nm^{-1}, appears at both low and high temperatures and suggests a different phase. Comparing the distance with various lattice planes of high-symmetry Fe_xO_y phases, the authors identify the distance to be the Fe_3O_4 $(02\bar{2})$ plane. More detailed info about overall crystallinity can also be extacted.

2.2.3 Conclusion

Machine learning and Deep Learning techniques each have their advantages and disadvantages. The choice between the two depends on the resources available and the task in hand, whether the algorithms should be lighter or more precise. PCA and k-means are easy to use and efficient, but PCA is just a linear transformation and can fail to handle complex patterns; k-means needs to be re-done on new data. Deep Learning, on the other hand, is non-linear and thus can handle more complex characteristics with fast inference once trained, but requires a lot of high-quality data and training can be expensive. In the following section, we propose an approach of GaAs substrate deoxidation detection using Deep Learning.

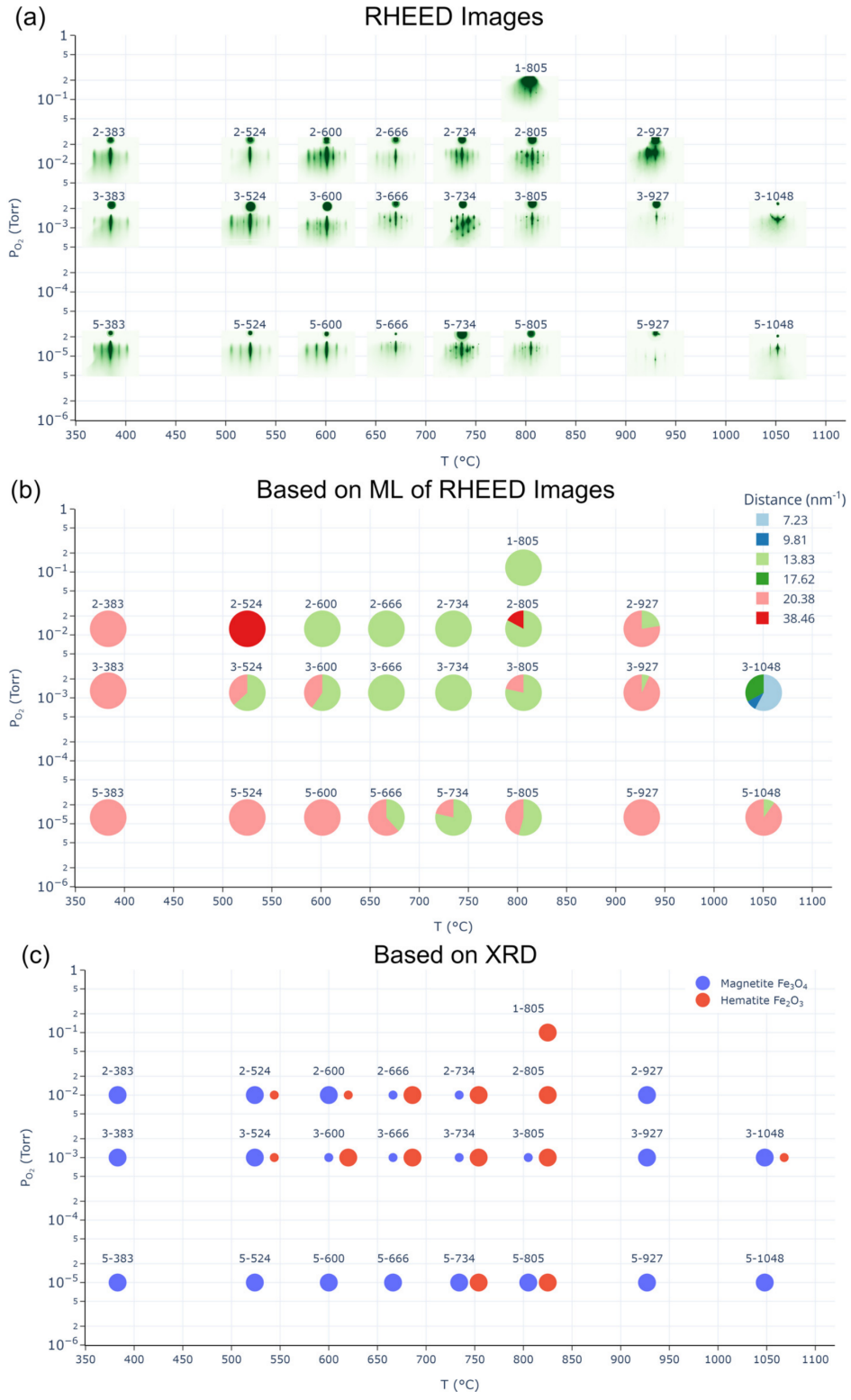

Figure 2.15: Phase mapping of iron oxide films: (a) Raw RHEED images recorded at the end of each deposition. (b) The extracted regions' intensities of each base distance from the RHEED pattern form a feature vector. Each feature vector is represented by a pie chart where each pie slice's area shows the relative intensity of each base distance. The color encodes the base distance of the region intensity as shown in the legend. The unit of the distance is nm^{-1}. (c) The red and blue scatter points show the XRD phase map of the hematite ($\alpha - Fe_2O_3$) and magnetite (Fe_3O_4) phases, respectively. The presence of a phase is plotted as a circle. The size of the circle indicates if the phase is major or trace material in a particular sample. Reprinted from [70].

2.3 Substrate deoxydation detection for GaAs growth

In this section, most of the content is from our paper at [35]. We present an approach for automated surveillance of GaAs substrate deoxidation in MBE reactors using deep-learning-based RHEED image-sequence classification. Our approach consists of an unsupervised autoencoder (AE) for feature extraction, combined with a supervised convolutional classifier network. We demonstrate that our lightweight network model can accurately identify the exact deoxidation moment. Furthermore, we show that the approach is very robust and allows accurate deoxidation detection for months without requiring retraining. The main advantage of the approach is that it can be applied to raw RHEED images without requiring further information such as the rotation angle, temperature, etc.

2.3.1 Context and motivation

A common application of RHEED is the monitoring of the native oxide removal from commercial substrates prior to crystal growth. Surface oxidation of a few nanometers due to exposure to oxygen is unavoidable during transport of epitaxial substrates, which renders their surface noncrystalline. In order to grow crystalline material on the substrate surface, this oxide needs to be removed before any epitaxial material deposition, which is usually done by heating. In the case of gallium arsenide (GaAs), the substrate is slowly heated to around 610 °C, while stabilizing the crystal with a constant arsenic flux of around $1.2 * 10^{-5}$ $Torr$, to avoid As evaporation [72]. Once the oxide is removed, in order to avoid damaging the crystal, further temperature ramping stops. Usually the temperature is in fact decreased. To detect the moment of deoxidation, the MBE operator supervises the RHEED image during temperature increase and once the diffraction pattern of a crystalline surface starts to form, the operator manually ends the heating procedure. Not only is the constant presence of the operator required, but also due to its manual character the deoxidation procedure is error-prone.

Automatic detection of the deoxidation is challenging, first because RHEED patterns are often weak since the raw substrate surfaces are not atomically flat and second because the RHEED image contrast is dependent on some parameters such as filament current or electron beam angle and hence is not exactly constant in each run. Finally, the substrate is usually lying on a rotating sample holder; hence, the RHEED pattern constantly changes.

Since the native oxide layer of a new substrate is not crystalline, the RHEED electron beam is scattered at the surface and does not create a clear diffraction pattern. On the other hand, without an oxide layer the RHEED electrons are diffracted by the atomic lattice of the now crystalline surface. However, the transition from oxidized to deoxidized is not instantaneous and during the deoxidation the classification is often difficult. Furthermore, due to the rotation of the sample, the diffraction pattern continuously changes and, especially during the deoxidation process, the pattern arises not similarly clearly for different rotation angles. Our operator classifies the surface as deoxidized when a clear diffraction pattern occurs repeatedly during at least one full rotation cycle of the substrate. Our goal is to automatically determine the moment of full oxide removal from a GaAs substrate with high precision by monitoring the RHEED pattern during the deoxidation process. However, as mentioned above, the image dynamics due to the constant rotation of the sample is a challenge for an algorithmic evaluation. Furthermore, disordered bright spots can occur also from oxidized

surfaces. Therefore, an algorithmic classification is not entirely trivial. By feeding short video sequences of several consecutive RHEED images to a classification neural network, we aim at determining the oxidation state of the substrate surface, in order to reduce the necessity of human supervision of the substrate cleaning process, which can take some tens of minutes.

2.3.2 Dataset preparation

To train a neural network on deoxidation reconnaissance, we generate a training dataset by capturing RHEED videos before and after the oxide removal procedure. The images are collected in real time at 24 frames per second, while the sample rotates with 12 rounds per minute. Hence, we capture 120 images per full rotation. The RHEED video is thereby captured image by image, using a CMOS camera (Allied Vision Manta G319B) with 44 pixel binning, resulting in raw images of (416×444) pixels at 12 bit grayscale intensity resolution. Those images are converted to 8 bit format and resized to (100×100) pixels.

In total we collected videos containing a total of 7644 RHEED images from five substrate oxide removal procedures within a period of a few days. 3110 of these images correspond to deoxidized surfaces, the rest are images from GaAs surfaces covered by a native oxide layer. GaAs surface oxides decompose at temperatures of around $580 - 630\ °C$ [73]. On our commercial substrates we typically observe deoxidation at around $610 - 630\ °C$. Substrate temperatures at which oxidized videos were taken are slightly lower, around $550 - 600\ °C$ (videos were recorded during ramping of the temperature). Deoxidized images were taken directly after oxide removal, at around $610\ °C$. During and after deoxidation, the As_4 pressure for surface stabilization is held at $1.2 \times 10^{-5}\ Torr$. We use 20% of the dataset for validation and the remaining 80% for training.

In addition to the oxidized and deoxidized image sets, we also captured images during the full deoxidation procedure. These are not used during training and serve for testing of the algorithm. RHEED images during a further deoxidation were captured around 6 months after generation of the initial dataset. These serve for an assessment of the long-term stability of the classification.

2.3.3 Neural network architecture

Our deoxidation monitor deep-learning model is composed of two stages, as depicted schematically in Figure 2.16. The first stage is a feature extractor network, compressing the full RHEED pattern images into compact latent vectors. This is done separately image by image. The second stage is the actual classification network. Its inputs are sequences of latent vectors, corresponding to short RHEED videos. We implemented the models in Python using Keras with TensorFlow as the backend [29, 74].

As a feature extractor we use a deep convolutional autoencoder (AE) neural network, which has been reported to offer slightly superior compression quality compared to other dimensionality reduction methods such as principal component analysis (PCA), especially at high compression rates. We note, however, that AEs require in general more computational resources. Thus, if computation speed is crucial, PCA could be used instead, for situations where a moderate reduction in encoding performance is expected [75]. The model details of

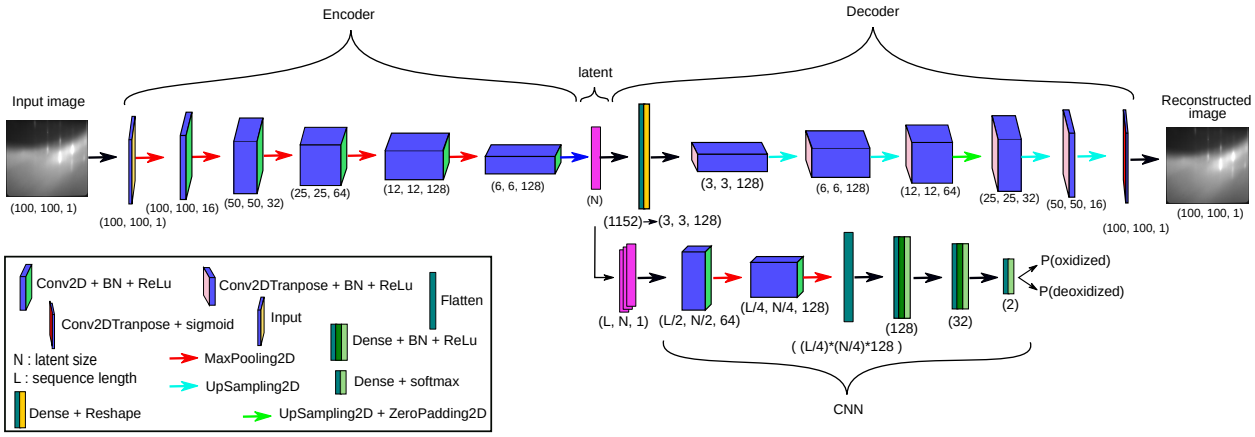

Figure 2.16: Deoxydation-detection neural network architecture. In a CNN autoencoder (AE,top), each RHEED image is first compressed through a convolutional encoder into a $1D$ latent vector of length N. Training target of the AE is reconstruction of the original image from the latent vector (through the decoder stage, only used during training). The second stage of the model is a classifier CNN model (bottom right), taking as input a sequence of L latent vectors, corresponding to a series of RHEED images. These L latent vectors are stacked and passed into a CNN for classification into two classes: oxidized and deoxidized. All convolutions are followed by batch normalization (BN) and ReLU activation. Reprinted with permission from [35]. Copyright 2023 American Chemical Society.

our AE are shown in the top row of Figure 2.16. A RHEED image goes through the encoder stage, being compressed into a latent vector of dimensionality N. For training, the latent vector is fed into a decoder stage, which is an exact mirror of the encoder, except for replacing convolution layers by transpose convolution layers, maxpooling by upsampling and applying zeropadding if required, to maintain correct image dimension. Through non-supervised training, the autoencoder learns to reconstruct the unlabeled input images from their learned latent vector representation. We optimized the network for low parameter number, in order to have a computationally efficient model. To this end we do not double the number of channels once a depth of 128 kernel filters is reached.

We also compared the architecture with a ResNet [76, 77], replacing the single convolutions by residual convolutional blocks each of which employing a sequence of three convolutions. The performance is similar and offers no advantage in deoxidation classification, at the cost of higher neural network parameter count. This applies to the specific problem discussed here, for other problems the slightly improved accuracy offered by a ResNet may very well be beneficial.

The second stage of our model is the actual classification network. Because the MBE sample is rotating, the RHEED images are constantly varying. Especially during deoxidation, the surface is not atomically flat and signatures of oxide removal often occur in the RHEED patterns only when the electron beam is aligned with the crystal lattice of the substrate. For a high accuracy, we therefore classify sequences of RHEED images which will be in detail explained below.

2.3.4 Results

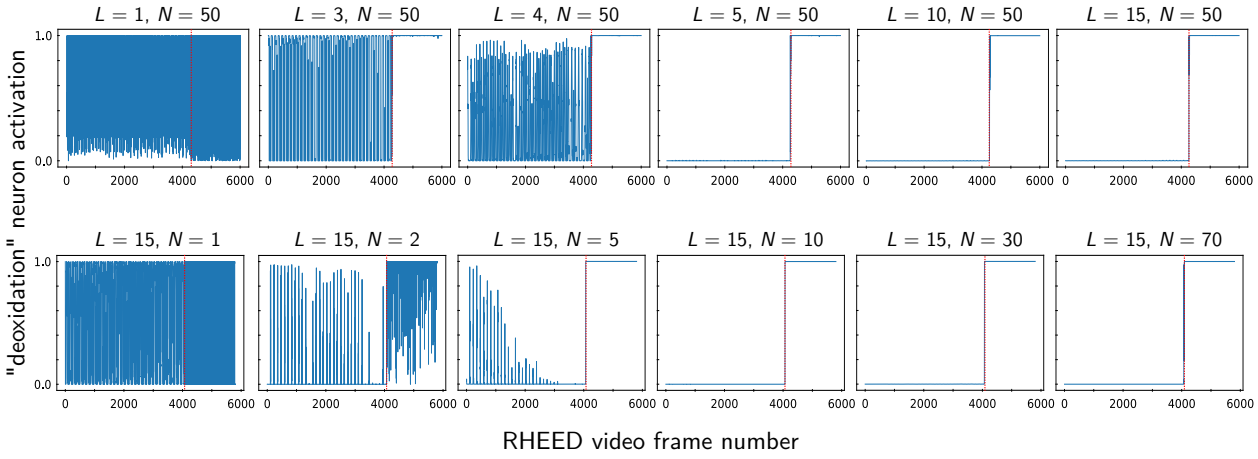

Figure 2.17: Detection accuracy of deoxidation moment. The impact of the sequence length as well as of latent vector dimension is tested on a video captured during a full deoxidation. The video consists of 31,819 RHEED images, the last 6,000 are shown. Deoxidation occurs around 1800 frames before the sequence end (indicated by a red dotted line).The RHEED video is captured with 24 frames per second. Top row: with an increase in sequence length L, the latent dimension of the autoencoder is fixed to N =50. Bottom row: with an increase in latent dimension N, the sequence length of the classifier is fixed to L =15.

We test whether the network is capable of determining the exact moment of deoxidation on a set of images captured during the entire deoxidation procedure. As illustrated in Figure 2.17, first we fix the latent size to $N = 50$ and increase the sequence length successively from $L = 1$ to $L = 15$. We find that starting from sequence length of $L = 5$ the network works accurately and is essentially error-free. It detects the precise moment of deoxidation with an agreement of a few seconds compared to the estimation of the human operator. We then fix the sequence length to $L = 15$ and vary the latent dimension between $N = 1$ and $N = 70$. For latent vectors of dimension $N = 10$ or larger, we find again quasi error-free classification and precise determination of the deoxidation moment.

In Figure 2.18, we show the classification results for a full video from a deoxidation run 6 months after the network training. While we observe a slightly reduced classification certainty regarding the exact moment of deoxidation, the non retrained neural network still performs sufficiently well on the deoxidation detection. We want to note that after training the pretrained network for a few additional epochs on a small set of new images, the fidelity of the network reaches the same confidence as observed in the right-hand panels of Figure 2.17. We demonstrated that the model accurately identifies the exact surface deoxidation moment and that the performance is robust during at least 6 months of MBE operation without requiring retraining.

2.3.5 Conclusion

In conclusion, we presented a deep-learning model based on a 2D convolutional autoencoder combined with a 2D CNN classifier to detect the surface oxidation state of GaAs substrates from raw RHEED image sequences, as typically available in molecular beam epitaxy. Our

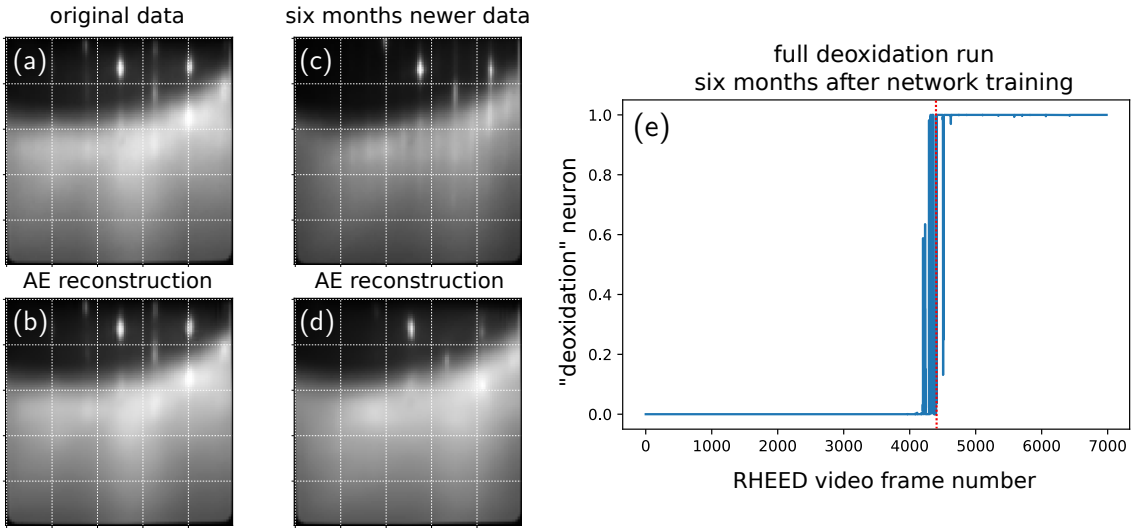

Figure 2.18: Test on six months newer data. (a) RHEED image of a deoxidized GaAs surface along [110] incidence, from training data, and(b) its reconstruction by the autoencoder with latent vector length $N = 50$. (c) RHEED image under the same incidence angle (along [110]), but 6 months later, after having performed more than 200 hours of crystal growth. In particular an increased metalization of the upper third of the RHEED screen is clearly visible, along with as light displacement of the diffraction pattern due to repeated alignment procedures. (d) Its reconstruction by the same autoencoder as used for (b), hence without retraining on new data. (e) Evaluation of the RHEED video from a full deoxidation run, recorded 6 months after the training data, without retraining the network model. Latent dimension $N = 50$, sequence length $L = 15$.

model consists of a first autoencoder neural network, which learns to compress individual RHEED images to a low dimensional latent space. Sequences of consecutive, compressed RHEED images are then classified for their surface oxidation state through a second convolutional network. We presented a systematic analysis of classification performance as a function of used compression ratio as well as RHEED video sequence length. We demonstrated that the model accurately identifies the exact surface deoxidation moment and that the performance is robust during at least 6 months of MBE operation without requiring retraining. While our specific, trained network will of course work only with the MBE setup and RHEED screen used for our training data generation, a generalization to other growth chambers can simply be done by training the same models on the relevant data. Video recording is straightforward; we demonstrated that the approach works well with data from only five deoxidation recordings and we proved it to function reliably during at least several months. In consequence, our approach is very appealing thanks to its simplicity and low computational cost. Without requiring additional hardware it can be easily set up in any RHEED-equipped MBE instrument.
As the RHEED provides a wealth of information about the growing sample surface, we present in the following section a Deep Learning model to monitor the surface reconstruction.

2.4 Surface reconstruction monitoring

Surface reconstruction monitoring using RHEED is a powerful technique in epitaxial thin film growth. RHEED offers real-time insights into surface morphology, crystalline structure and growth dynamics during thin film deposition processes. By analyzing the diffraction patterns produced when high-energy electrons interact with the growing surface, surface reconstructions can be monitored and provide valuable information for optimizing growth conditions and controlling material properties. In this aspect, this section covers a Deep Learning method to classify $c(4\times4)$ and (2×4) surface reconstructions. In contrast to [68,69] where pre-processed RHEED images at very specific crystal orientations where necessary, our goal is to develop a model capable to distinguish between surface reconstructions from raw data, in the same way as we did in the previous section with deoxidation detection.

2.4.1 Context and motivation

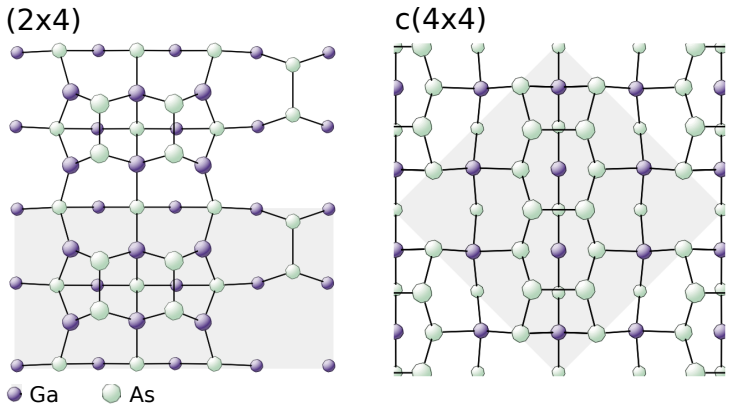

Figure 2.19: (2×4) and $c(4 \times 4)$ surface reconstruction illustrations. The atoms on the surface of the crystal are subject to rearrangements depending on the growth conditions, such as temperature. The scheme on the left illustrates a (2×4) surface reconstruction of GaAs, while the one on the right depicts a reconstruction of $c(4 \times 4)$. Readapted from [78].

Surface reconstruction refers to the rearrangement of surface atoms of the material under growth. The (001) GaAs surface can be terminated by either Ga or As atoms and has two dangling bonds for each surface atom that experiences inter-atomic forces from only the bulk side [79]. Due to this imbalance and in order to eliminate such dangling bonds, the surface atoms undergo complex reconstructions by assuming positions with different spacing and symmetry, as shown in Figure 2.19, from the bulk atoms.

Surface reconstruction in the case of Gallium Arsenide (GaAs) using Molecular Beam Epitaxy (MBE) and monitored by Reflection High-Energy Electron Diffraction (RHEED) is a crucial aspect of semiconductor devices fabrication. Many atomic structures are observed during GaAs growth, however we mainly focus on the As-rich (2×4) and $c(4 \times 4)$ as the surface during the MBE growth is stabilized under significant As flux and usually shows the (2×4) and $c(4 \times 4)$ reconstructions, RHEED patterns of such surfaces are depicted in Figure 2.20. These reconstructions are formed critically depending on the preparation conditions [79] and affect the surface morphology and electronic properties of the material. Detailed knowledge

of the surface reconstructions on GaAs(001) is essential for fabricating high speed electronic and optoelectronic devices, because the surface reconstruction plays an important role in the homo- and heteroepitaxy on GaAs(001) [79]. Larsen P et al. reported in [80] measurements on their experiment about substrate temperature T_s and As_2 fluxes J_{As_2} for different surface reconstructions; for $c(4 \times 4)$, $T_s = 475\ ^\circ C$ and $J_{As_2} = 5 \times 10^{14}\ mol.cm^{-2}.s^{-1}$, for (2×4), $T_s = 565\ ^\circ C$ and $J_{As_2} = 2 \times 10^{14}\ mol.cm^{-2}.s^{-1}$ and for (3×1), $T_s = 630\ ^\circ C$ and $J_{As_2} = 2 \times 10^{14}\ mol.cm^{-2}.s^{-1}$.

Following the first observation of the (2×4) reconstruction on surfaces prepared by ion bombardment [81], atomic structures of the (2×4) surface have been a subject of continuing interest [80, 82]. Cho first reported in [83] that the (2×4) and $c(8 \times 2)$ reconstructions were formed on the As- and Ga-stabilized surfaces, respectively, prepared by MBE. The $c(4 \times 4)$ reconstruction of the GaAs(001) surface is usually observed under extremely As-rich MBE conditions [79]. Chang et al. first reported that the $c(4 \times 4)$ reconstruction is observed during the MBE growth when the As/Ga flux ratio is increased or the substrate temperature is lowered [84, 85]. With a temperature range of $[645 - 775]\ ^\circ C$, the disordered $(2 \times 1) + (1 \times 2)$ changes to a predominantly $c(4 \times 4)$ surface via a (2×2) transitional surface [86]. Controlling material surface during MBE growth requires an effective monitoring of the changes taking place on the surface including reconstructions. The data preparation is explained below.

2.4.2 Dataset preparation

For training, we use a dataset of 3150 images containing 1610 of (2×4) images and 1540 of $c(4 \times 4)$ images. Examples from each category are shown in Figure 2.20, the first line represents (2×4) surface images and the second line $c(4 \times 4)$ surface images. The data is collected with same equipments as in 2.3.2. All the data is put together, then split into 85% for training and 15% for validation. The training set is aimed to be used by the neural network during training process in order to adjust its parameters (weights and biases). The validation set is also used during training, not to fit the model parameters but to get an idea of the precision of the neural network on new data. Two separate sets of 5433 and 5816 images are captured during the temperature increase and decrease and serve to detect the surface reconstruction transiting from $c(4 \times 4)$ to (2×4) and vice versa; these sets are not used for training the model and are valid to judge the model efficiency. The model predictions are shown in Figure 2.24 and Figure 2.25.
The raw images are (416×444) grayscale from 0 to 255 pixel values, in preprocessing we resized them into (138×148) by dividing spatial dimensions by 3 and taking the closest integer value; the pixel values are then divided by 255.0 to be in the range of $[0 - 1]$ for an optimum training process. We trained the model for continuous sample rotation, contrary to [68] that trains the model for specific azimuth angles. In our case, the prediction of a video sequence showed more precision than predicting single images, after comparison, the sequences of 6 images showed the most accurate prediction; Thus, the model is predicting surface reconstruction for sequences of size $(138 \times 148 \times 6)$, the images are stacked in the channel dimension. In the following section, we present the neural network architecture and explain why it was chosen.

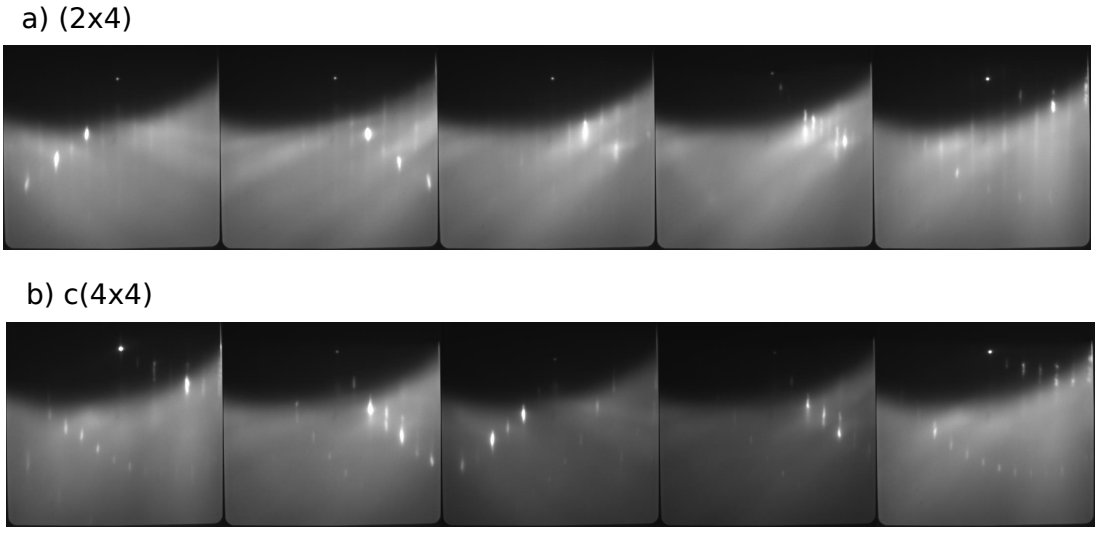

Figure 2.20: (2×4) and $c(4 \times 4)$ RHEED pattern examples: a) Five images representing (2×4) surfaces; b) Five images representing $c(4 \times 4)$ surfaces.

2.4.3 Neural network architecture

Convolutional neural networks have seen a gradual increase of the number of layers, starting from AlexNet [87], VGG [88], Inception [89] to Residual networks [76, 90] and have led to a series of breakthroughs for image classification [87, 91, 92]. Thanks to these architectures, a neural network can be configured by stacking many weight layers like VGGNet proposed by the Visual Geometry Group in [88]. The depth can be significantly further increased (up to more than 1000 layers) by using residual blocks with skip connections [93], instead of bare convolution layers. The problem that residual neural networks solve is the observation that the performance of convolutional neural networks deteriorates for very deep architectures, hence if too many layers are added. There is an ongoing debate about the origin of the problem, part of the problem are vanishing gradients in early layers for deep architectures [94]. To overcome this problem of accuracy degradation, Deep residual networks are known to allow the construction of deep architectures while maintaining a good performance [93]. Mascarenhas et al. carried out in [95] a comparison between VGG16 (13 convolution layers and 3 fully connected layers) [88], VGG19 that contains 16 convolution layers and 3 fully connected layers and ResNet50 [96] made up of 48 convolution layers, 1 Max Pooling layer and 1 average Pooling layer on image classification problem; with Categorical cross-entropy as loss function, the authors concluded that ResNet is the best architecture with an accuracy of 0.9733 while VGG16 and VGG19 achieved 0.9667 and 0.9707 respectively; the accuracy is based on the confusion matrix.

Taking into account these developments of neural network architectures, our choice is the residual networks that consists of Residual blocks also called "Residual units" that involves a main path of weight layers, batch normalization and activations alongside which a skip connection as depicted in Figure 2.21.
The original residual unit can be expressed as [90]

$$y_l = h(x_l) + F(x_l, W_l) \tag{2.6}$$

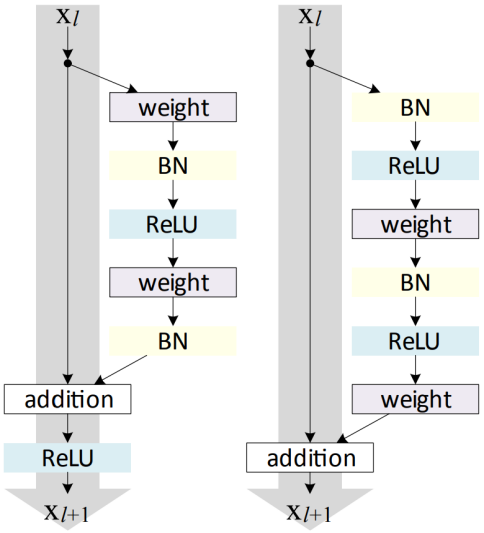

Figure 2.21: Original ResNet and identity mappings ResNet blocks. At the left is the original ResNet Unit where the input x_l goes through the main path that contains weight layers followed by batch normalization and ReLU activation except for the last layer which the output is added to an identity skip connection. The result is then fed to a ReLU activation function. At the right, for the Identity Mappings ResNet, the technique of "full pre-activation" is applied where the batch normalization and activation function come before the weight layer. The result of the main path is added to the skip connection without activation after the element-wise addition. Readapted from [90].

$$x_{l+1} = f(y_l) \tag{2.7}$$

where x_l and x_{l+1} are the input and the output of l-th residual unit, F is a residual function, h is the skip connection and f is a ReLU activation function after the element-wise addition between the skip connection and the output of the Residual function. The authors of [90] proposes to make f and h identities, with these conditions, we can put Eq.(2.7) into Eq.(2.6) and get

$$x_{l+1} = x_l + F(x_l, W_l) \tag{2.8}$$

Recursively, we have $x_{l+2} = x_{l+1} + F(x_{l+1}, W_{l+1}) = x_l + F(x_l, W_l) + F(x_{l+1}, W_{l+1})$ that leads to

$$x_L = x_l + \sum_{i=l}^{L-1} F(x_i, W_i) \tag{2.9}$$

The expression of Eq.(2.9) shows that any feature x_L of any deeper unit L can be represented as the feature x_l of any shallower unit l plus a residual function in a form of $\sum_{i=l}^{L-1} F$. Hence, it can be observed that the gradient of the linear component x_l is equal to one. In forward propagation, the feature $x_L = x_0 + \sum_{i=0}^{L-1} F(x_i, W_i)$, of any deep unit L, is the *summation* of the outputs of all preceding residual functions *plus* x_0 (except in a few rare cases in the architecture where the skip connection is a convolution to match the dimensions with the residual output). In the backward propagation and denoting the loss function ε, from the chain rule of backpropagation

$$\frac{\partial \varepsilon}{\partial x_l} = \frac{\partial \varepsilon}{\partial x_L}\frac{\partial x_L}{\partial x_l} = \frac{\partial \varepsilon}{\partial x_L}\left(1 + \frac{\partial \sum_{i=l}^{L-1} F(x_i, W_i)}{\partial x_l}\right) \tag{2.10}$$

Eq.(2.10) indicates that the gradient $\frac{\partial \varepsilon}{\partial x_l}$ can be decomposed into two additive terms: $\frac{\partial \varepsilon}{\partial x_L}$ that propagates information directly without concerning any weight layers and $\frac{\partial \varepsilon}{\partial x_L}\left(\frac{\partial \sum_{i=l}^{L-1} F(x_i, W_i)}{\partial x_l}\right)$ that propagates through the weight layers. Theses direct propagations in forward and backward are based on the conditions that skip connections h are identity functions and the use of full pre-activation for the residual function F which makes identity the operation after the element-wise addition between the skip connection and the residual output for each unit. To shed light on the importance of the identity shortcut, the authors of [90] analyze a modification of the identity mappings ResNet by taking $h(x_l) = \lambda_l x_l$ where λ_l is a modulating scalar and found

$$x_L = \left(\prod_{i=l}^{L-1} \lambda_i\right) x_l + \sum_{i=l}^{L-1}\left(\prod_{j=i+1}^{L-1} \lambda_i\right) F(x_i, W_i) \tag{2.11}$$

which results in propagation to:

$$\frac{\partial \varepsilon}{\partial x_l} = \frac{\partial \varepsilon}{\partial x_L}\left(\prod_{i=l}^{L-1} \lambda_i + \frac{\partial \sum_{i=l}^{L-1}(\prod_{j=i+1}^{L-1} \lambda_j) F(x_i, W_i)}{\partial x_l}\right) \tag{2.12}$$

and as the first additive term is modulated by the factor $\prod_{i=l}^{L-1} \lambda_i$, for a deep network, if $\lambda_i > 1$ for all i, this factor can be exponentially large and on the contrary if $\lambda_i < 1$, this factor can be exponentially small and vanish, which blocks the backpropagated signal from the shortcut and forces it to flow through the weight layers and results in optimization difficulties. Further experiments for prove are done in [90].

We adopt the Identity Mappings Residual Network reported by He Kaiming et al. in [90] for characteristics extraction using the ResNet block implemented in our previous work in [13], the detailed architecture is shown in Figure 2.22, the numbers on the right of the blocks indicate how many times the residual units are stacked before the *MaxPooling* operation that reduces the data size or the *flatten* layer. Our Residual neural network architecture shown in Figure 2.22 consists of residual units involving three convolution layers each preceded by batch normalization and LeakyReLU activation function respecting the two identity conditions analyzed in [90]. A convolution shortcut is used two times in the stack of 14 ResNet blocks for dimensionality matching, the consequence is minimal as explained in [90]. The following section reports the ResNet training process, hyperparameters and accuracy on the test datasets.

2.4.4 Results

Our neural network is trained with keras (TensorFlow as backend). We use some modules to optimize learning rate on the fly and save the best model during training process. *ModelCheckpoint* saves the model each time the validation loss improves during the training process to keep at the end the best version of the model, *EarlyStopping* stops the training

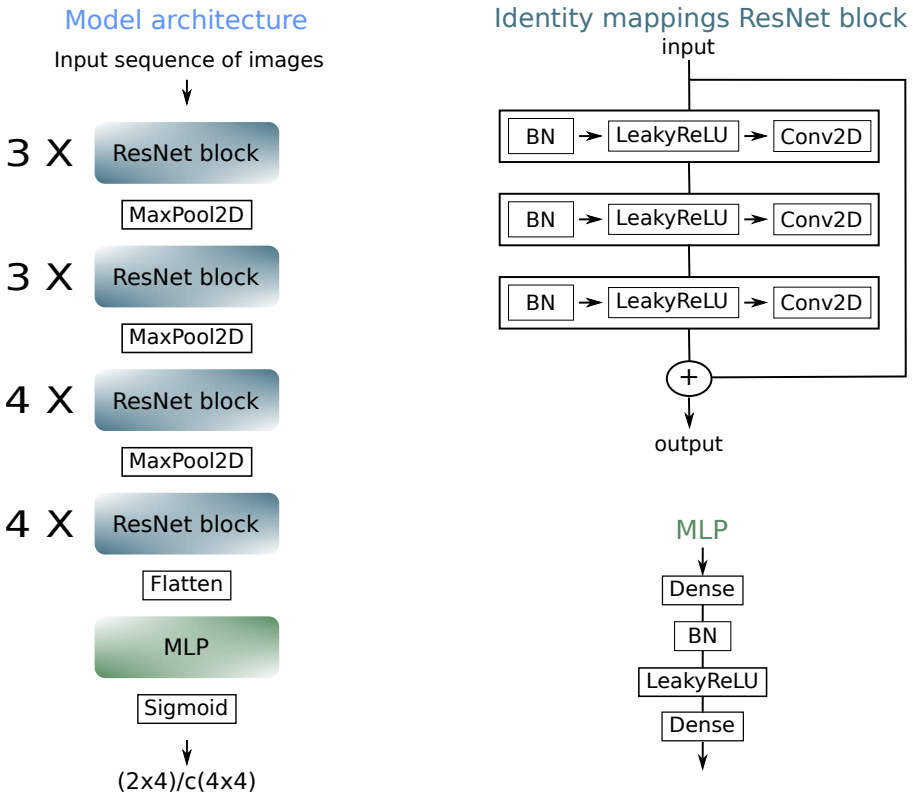

Figure 2.22: Residual neural network architecture for (2×4) and $c(4 \times 4)$ monitoring. The block of identity mappings ResNet also called "residual unit" at the top right is made up of three sub-blocks each containing a batch normalization, LeakyReLU activation and 2D convolution which the result is added to a shortcut of the block input. The architecture of the whole neural network at the left contains a stack of identity mappings ResNet blocks to extract informations from input images and 2D MaxPooling layers for dimensionality reduction. The result is then flattened and fed to a MLP with a dense layer followed by a batch normalization and LeakyReLU activation before the output Dense layer with sigmoid activation function.

process if no improvement of the validation loss for 15 epochs and *ReduceLROnPlateau* reduces the learning rate by a factor of 10 if no improvement of the validation loss is observed for 3 epochs. The curves of training loss, validation loss and learning rate are shown in Figure 2.23, curves are plotted in logarithmic for a better view. The training is launched with a batch size of 16, *Adam* optimizer with a learning rate of $1e-3$ and *binary_crossentropy* as cost function. For an initial number of epochs of 80, the *EarlyStopping* callback stopped the process at the 28^{th} epoch, the best model according to the validation loss is saved at the 13^{th} epoch by the *ModelCheckpoint* callback. The learning rate is decreased five times until the 28^{th} by factor 10 and goes from 10^{-3} to 10^{-8}.

In Figure 2.24, we have the ResNet model prediction on the test data captured during the temperature increase from $620°C$ to around $670°C$. As the temperature increases, the material surface passes from $c(4 \times 4)$ to (2×4). But this transition is not abrupt. Because the last activation function is *sigmoid*, the network output is in the range $[0-1]$, we set a threshold of 0.5 to classify as class 0 (i.e. $c(4 \times 4)$) when the prediction is smaller than the threshold and class 1 (i.e. (2×4)) when the prediction is bigger than 0.5. The raw

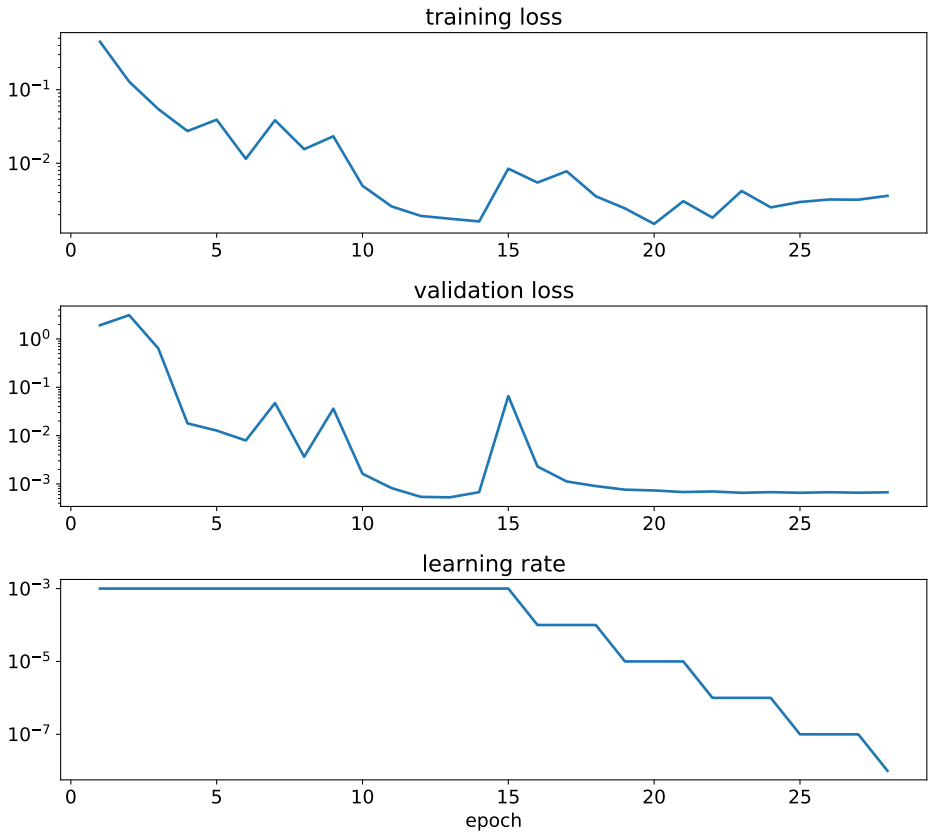

Figure 2.23: ResNet training history. The curve at the top is the model training loss falling sharply in the first 5 epochs. The curve in the middle shows the validation loss used by all the three callbacks (*ModelCheckpoint*, *EarlyStopping* and *ReduceLROnPlateau*) in order to monitor the training. The curve in the farthest down is the optimizer learning rate controlled by the *ReduceLROnPlateau* and decreases by a factor of 10 each time the validation loss does not improve for 3 epochs. It is plotted in logarithmic to allow a clear view as its value gets very small along the epochs.

prediction, the thresholded one and the corresponding thermocouple temperature are shown in Figure 2.24. The MBE chamber is then cooled down from $680°C$ back to $620°C$, Figure 2.25 shows the ResNet prediction during temperature decrease, the noise prediction in the last images is due to the fact that the thermocouple temperature is lower than the actual sample temperature as the temperature needs time to go down on the sample side indicating that there is still the presence of some (2×4) zones. Usually, $c(4 \times 4)$ starts to form at lower temperatures.

2.4.5 Conclusion

We presented a Deep Learning technique for surface reconstruction classification using residual network architecture. The best version of the model is saved during the training process thanks to the implemented callbacks. The surface identification is performed on data captured during MBE chamber heating (transition from $c(4 \times 4)$ to (2×4)) and cooling down (transition from (2×4) to $c(4 \times 4)$) to evaluate the efficiency of the model. The neural network detects quite well the surfaces during temperature changes. The following section reports our work on Azimuthal RHEED construction from raw RHEED patterns.

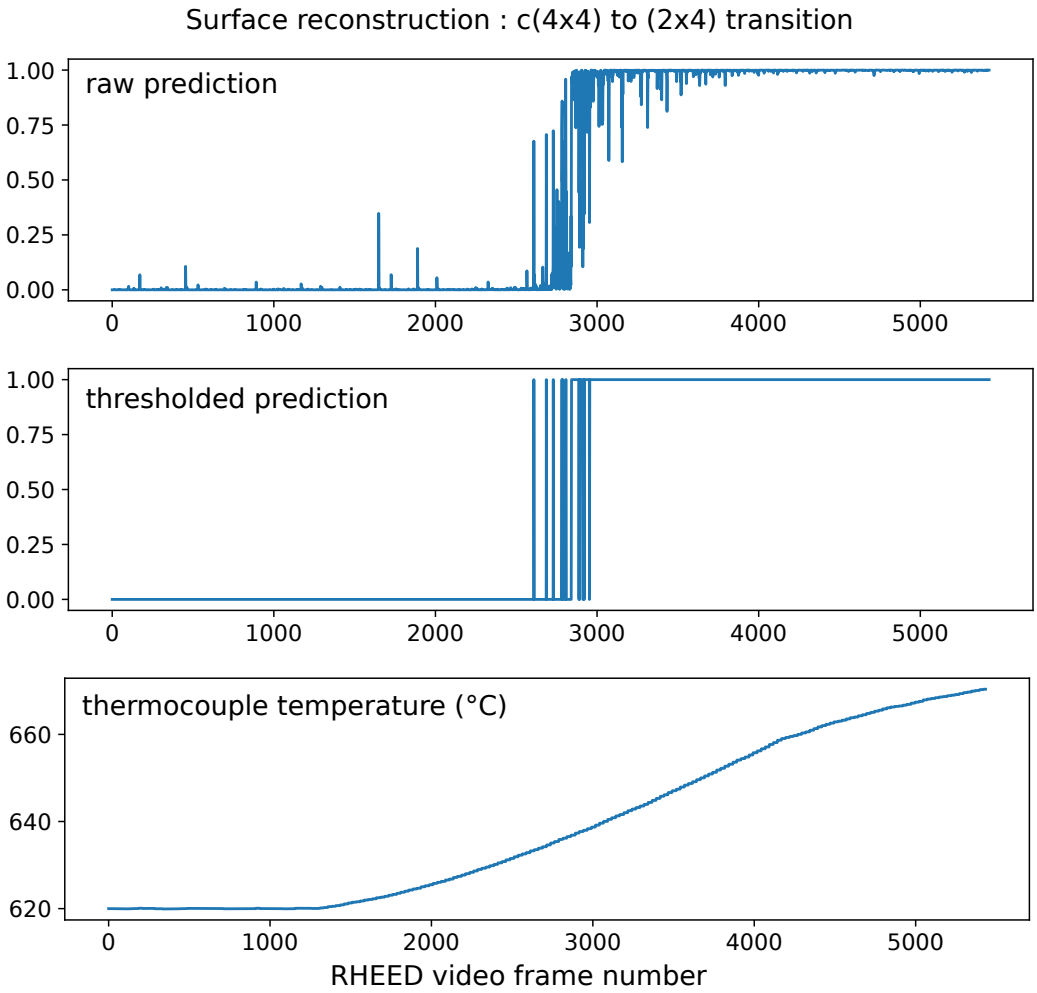

Figure 2.24: Surface reconstruction prediction during heating. The curve on the top shows the neural network raw prediction on the test samples of the data and indicates that the surface passes from $c(4 \times 4)$ to (2×4) just before the 3000^{th} sequence (of images). The middle curve shows the prediction with threshold, > 1 values are raised to 1 and < 1 values to 0. The plot has three parts, first part of class 0 from 0 to ~ 2600, a second part of uncertainty from ~ 2600 to ~ 3000 and the last part after ~ 3000 corresponding to the class 1. The curve at the bottom is the corresponding thermocouple temperature increasing during growth. This captured window goes from 620 to 670 $°C$. The surface change occurs at a temperature of around 640 $°C$.

2.5 Azimuthal RHEED construction

Raw RHEED patterns are difficult to interpret directly and therefore are typically pre-processed in some way. A common way to prepare RHEED patterns is the so-called Azimuthal reflection high-energy electron diffraction (ARHEED), in which slices through the specular spot of RHEED patterns are drawn in a polar plot as a function of the azimuthal angle (the rotation angle). ARHEED is the map of intensity obtained by arranging, as a function of the azimuthal angle, a single horizontal slice through the specular spot of each RHEED image [97]. ARHEED offers a relatively straightforward interpretability and allows to detect periodicities or epitaxial crystal orientations. ARHEED is employed to investigate the growth and epitaxial orientation of the crystal during MBE [98]. An Azimuthal RHEED

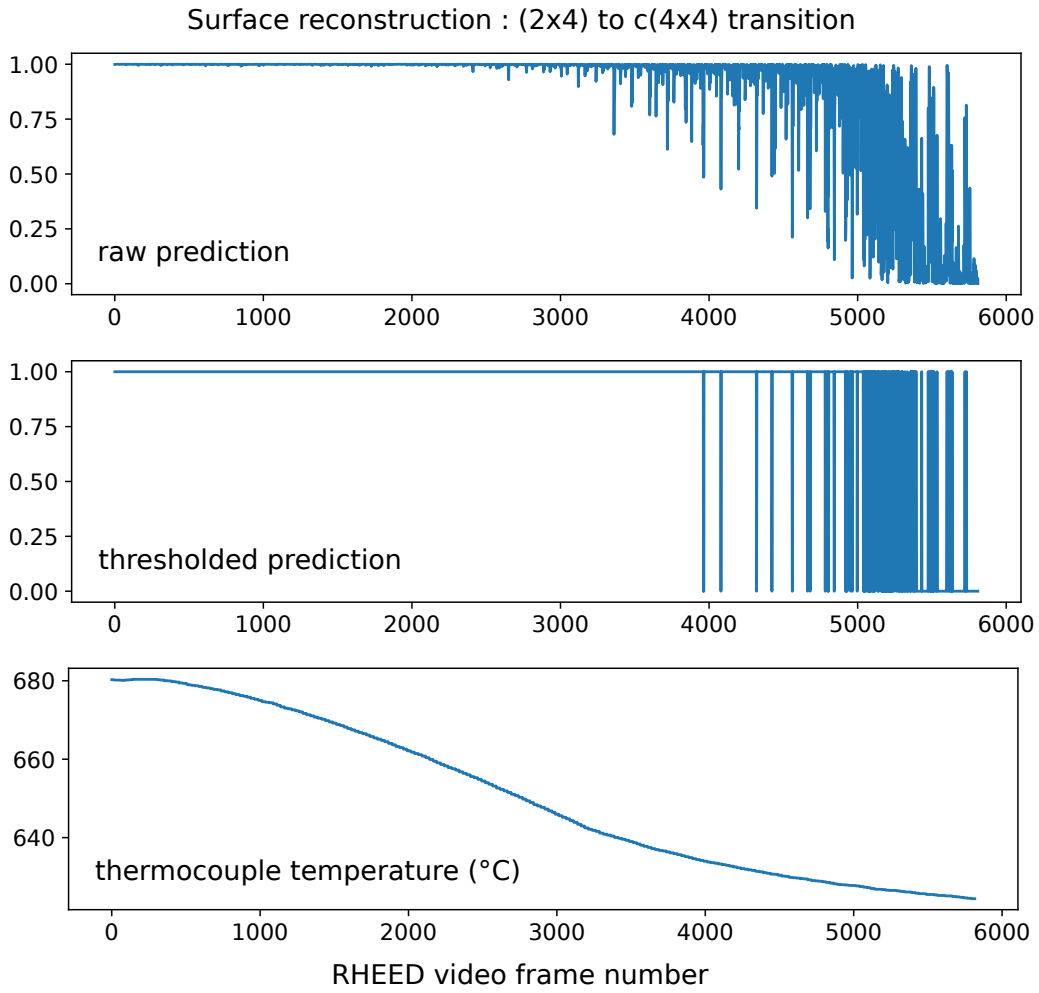

Figure 2.25: Surface reconstruction prediction during cooling down. The curve on the top shows the neural network raw prediction on the test data. The middle curve is the thresholded prediction, 0 to ~ 4000 is classified as class 0 but after ~ 4000, the model falls in uncertainty as the surface is not completely $c(4x4)$ instantaneously for the same temperature than before heating in Figure 2.24. The curve at the bottom is the corresponding thermocouple temperature during decreasing to get back to same initial temperature of 620 $°C$ as in the bottom curve of Figure 2.24.

of a $c(4 \times 4)$-reconstructed GaAs surface is illustrated on Figure 2.26. This section aims to draw the ARHEED from raw RHEED images by detecting the specular spot with semantic segmentation and determining the azimuthal angle via regression using ResNet.

2.5.1 Context and motivation

During MBE, it is essential to grow layers with substrate rotation to ensure uniformity [99]. Therefore, it is important to develop methods that allow access to the reciprocal lattice of rotating substrates [99]. Xiang et al. show in [100] that it is possible to obtain the entire reciprocal space structure of a 2D material by rotating the sample around the surface normal and measuring the RHEED patterns as a function of the azimuthal angle which makes ARHEED extremely interesting for surface characterization. A typical difficulty is associated with the fact that in the ultra-high vacuum MBE chambers, the rotational stage

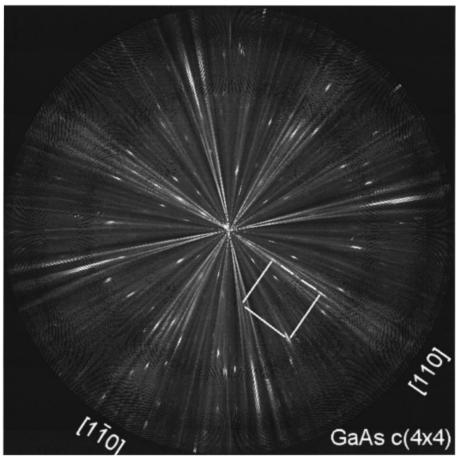

Figure 2.26: Azimuthal RHEED illustration. Azimuthal RHEED of a GaAs $c(4 \times 4)$-reconstructed surface. Reprinted from [99].

is coupled magnetically and the rotation is often not perfectly smooth. This makes it difficult to determine the crystal angle with high accuracy. Determining the crystal azimuthal angle from the RHEED pattern itself is therefore very interesting for high-accuracy ARHEED. Kikuchi lines, illustrated in Figure 2.31 with red lines, contain information on the sample orientation. They are (for samples with a cubic crystal lattice) 4-fold symmetric and thus strictly contain information only for 0-90 degrees. Using sequences Kikuchi could identify 0-180 degrees (because we "see" the rotation direction). Finally, if the RHEED screen is not perfectly centered (which is probably always the case), it should be even possible to distinguish between a full 0-360 degree rotation angle range. Kikuchi comes from the bulk, which is good because then it does not depend only on the currently growing atomic layer and angle determination could be possible in a quite robust way. But also the surface signal (diffraction spots and lines) contains of course information about the crystal angle. A neural network will probably use all of this information. In addition to the angle, it is necessary to know the specular spot position for the construction of the AHREED. It can be detected by semantic segmentation. Therefore, we propose in the following subsections Deep Learning models to detect specular spot and predict azimuthal angle from only raw RHEED images.

2.5.2 Specular spot tracking across RHEED patterns

In order to crop thin slices from RHEED images through the specular point, its position must be tracked across the RHEED patterns. Generally, the tracking process involves two algorithms for object detection and tracking. Object detection algorithms firstly identify and localize objects in images followed by object tracking algorithms that link the detected objects across video frames to maintain their identity and trajectory. In case of object disappearance, the process of object detection is relaunched.

In the context of the specular spot, it is lost from view at least four times per round as a consequence of the sample holder hooks, with occasional further losses due to fluctuations in light conditions and noise. This would require the detection algorithm to be restarted each time a loss occurred. To simplify the process, we instead implement a lightweight semantic segmentation model. Semantic segmentation enables to perform pixel-wise classification

partitioning the image into meaningful regions based on object categories. Unlike object detection, which identifies and localizes objects using bounding boxes, semantic segmentation provides a detailed pixel-level identification. In our case, the specular spot pixels are referred to as "class one" whereas the remaining is referred to as "class zero" leading to a binary mask from each RHEED image independently. The dataset preprocessing, the model architecture as well as the results are explained in the following.

Dataset preprocessing

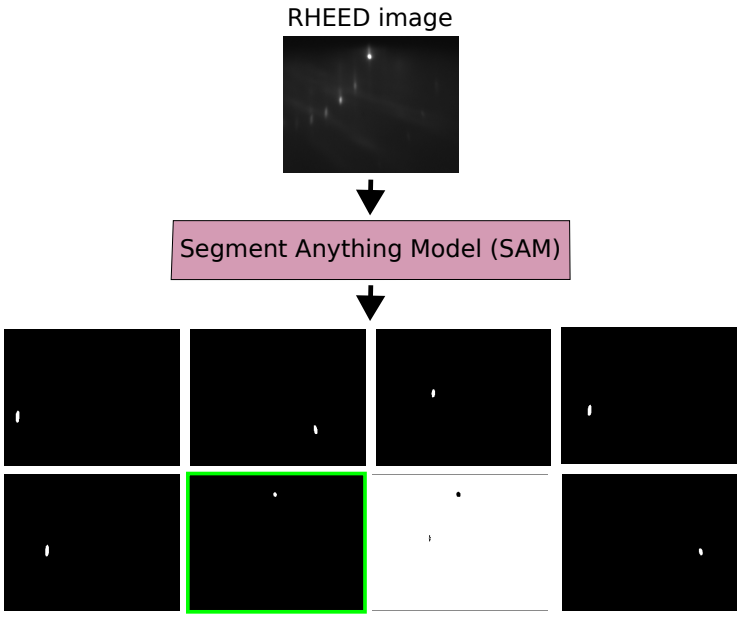

Figure 2.27: Generation of the specular spot mask. Segment Anything Model (SAM) [101] generates a separate mask for any object in the input image. The goal is to manually save a couple of an image and the corresponding specular spot mask (framed in green) in order to build a database for the training of a model that will generate only the specular spot mask.

A dataset of image-mask couples is set up to train a custom segmentation model. To this end, binary masks are generated for each spot and line on the RHEED images using the Segment Anything Model (SAM) [101] of Meta. SAM operates as a general-purpose segmentation framework, capable of identifying and segmenting objects in images. Its functioning revolves around three main components: a prompt encoder, an image encoder and a mask generator. The input prompt guides the segmentation process. It can be sparse (specific points on the image, bounding boxes around the target object to specify its approximate location or text descriptions) or dense (mask prompt). The image encoder is a pre-trained model that extracts high-level and multi-scale features from the input image. Based on the input prompt and the image features, the mask generator produces segmentation masks. SAM is designed to output multiple plausible masks when there is ambiguity, allowing to select the most appropriate one.

As the specular point is moving, we let the SAM generate masks from RHEED images without prompts leading to the generation of multiple masks for each image as shown on Figure 2.27. The mask corresponding to the specular spot, framed in green on Figure 2.27, is manually selected to build a database of 1026 couples (image-mask). Then data augmentation is

performed by shifting the images and masks to four sides increasing the data up to 5130 samples. 80 % of the data is used for training and 20 % for validation. The test is carried out with 1200 images which are not among those used for training and test.

Neural network architecture

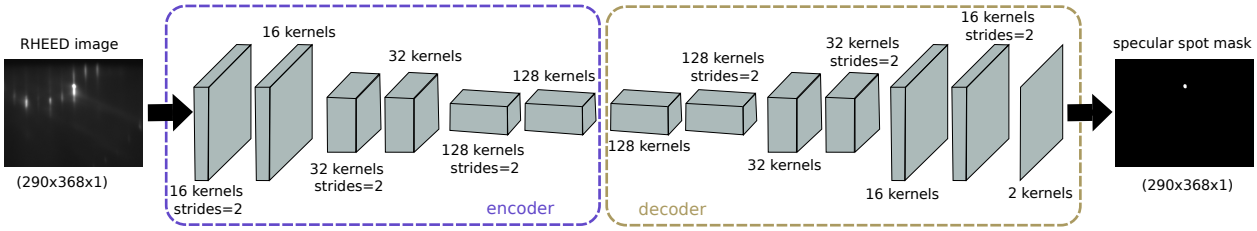

Figure 2.28: Semantic segmentation model architecture for the specular spot mask. The model reduces the spatial dimensions of the RHEED image through an encoder network while expanding the channel dimension. The decoder, mirror network of the encoder, constructs a matrix with the same spatial dimensions than the input and two channels, each representing the probability of a class (specular or not). The mask is determined by tacking the index (1 or 0) of the class with the highest probability value. This leads to a binary mask with ones at the specular point and zeros elsewhere.

The model is trained to generate a binary mask for the specular point from the RHEED image. The model architecture, depicted in Figure 2.28, comprises an encoder and a decoder and is fully convolutional. The encoder takes an image as input, extracts features through convolution layers with increasing number of kernels. As spatial information are important in the segmentation task, the downsamlping is performed with the convolution strides which enable to preserve the spatial features in the images unlike the *MaxPooling* layer that picks the maximum value from a window of usually four elements regardless of its position. A convolution layer with strides learns the downsampling, as it is part of the convolution. The output of the encoder is fed to a decoder, a mirror network of the encoder in which the convolution layers are replaced by transpose convolutions. The upsampling is also done using convolution strides. The decoder produces a matrix of two channels, each containing pixel-wise probability for the two classes (specular or not). The binary mask is determined taking the indices of the biggest values along the channel dimension. The specular spot coordinates correspond to the center of gravity of the area where the pixels are equal to one. The model is trained with *SparseCategoricalCrossentropy* loss function in keras with *Adam* optimizer. The obtained results are presented in the following.

Results

Once the segmentation model has demonstrated satisfactory performance on the validation data, it is tested on new data to examine its accuracy under real-world conditions. A total of 1 200 images are reserved for this purpose. Three illustrative examples are presented in Figure 2.29. These examples show three RHEED images, each accompanied by its mask generated by the segmentation model. A red cross denotes the position of the specular spot on each image, corresponding to the center of gravity of the specular point in the binary mask. The movement of the specular point in all 1 200 images is recorded in order to

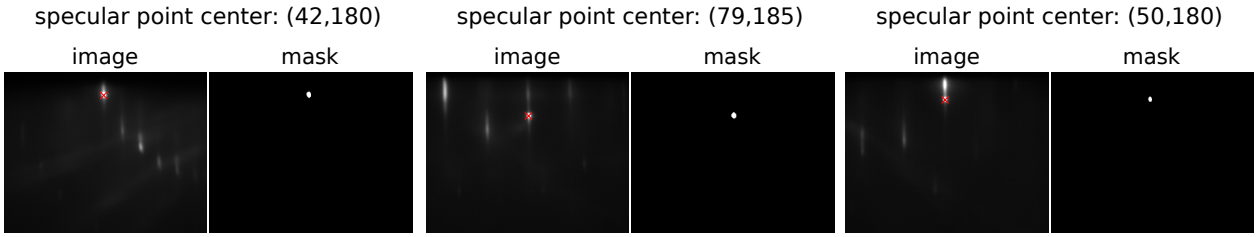

Figure 2.29: Examples of the segmentation model results. Three examples of RHEED images and their generated specular spot masks by segmentation model. Each red cross on an image indicates the center of gravity of the specular spot (bright zone) on the corresponding mask and the coordinates (in units of pixels) are on the top of each couple of image and mask.

plot the Azimuthal RHEED. Figure 2.30 shows the vertical and horizontal displacements. The elliptical movement of the specular point can be seen in these curves, with the vertical displacement having a greater amplitude than the horizontal one. The few overshoots of the curves are due to prediction errors. When the point is lost from view in the black images, its last known position is retained as showed by red dashes on Figure 2.30, although this will not allow any useful information to be extracted. Given that the position of the specular spot is now tracked, enabling the cutting of thin slices through it, the following subsection is about the determination of the azimuthal angle, which will allow the plotting of these slices as a function of the rotation and thus obtain the ARHEED.

2.5.3 Azimuthal angle determination

The azimuthal angle required to construct the Azimuthal RHEED is the angle between the rotating sample and the incident electron beam. The quality of the Azimuthal RHEED depends on the accuracy with which this angle is determined as well as the previously extracted positions. This subsection aims to retrieve that angle from raw RHEED images using Deep Learning techniques. The data preprocessing, neural network architecture and the results are presented in the following.

Dataset preprocessing

The sample holder has some weight, the faster we spin, the less perturbation of the rotation by clearance and friction will occur thanks to the inertia of the sample holder. Therefore, angle accuracy is higher, but the ARHEED resolution reduces for high rotation speed. Good angle resolution requires rotation speeds in the order of 3-4 rpm, where clearance has very noticeable impact on the rotation. For this reason, we trained a ResNet model for the azimuthal angle regression using a dataset of 21 600 images captured when the sample is rotating at 12 rpm for training. The dataset is split in such a way that 77 % is used for the model training, 15 % for validation and 8 % for test. The purpose of this split and the dataset collection equipment are detailed in 2.4.2. The test for ARHEED plotting is performed using images captured at 4 rpm. The raw images have (512×688) dimension in 16-bit (65536 grayscale). In preprocessing, we reduced the image dimensions to (137×229) and at the same time normalized pixel values between 0 and 1. After tests of different size scale factors, we chose the smallest dimensions without noticeable impact on the results. Sequences of 6 images bring better accuracy for the regression task thus the model input dimensions are

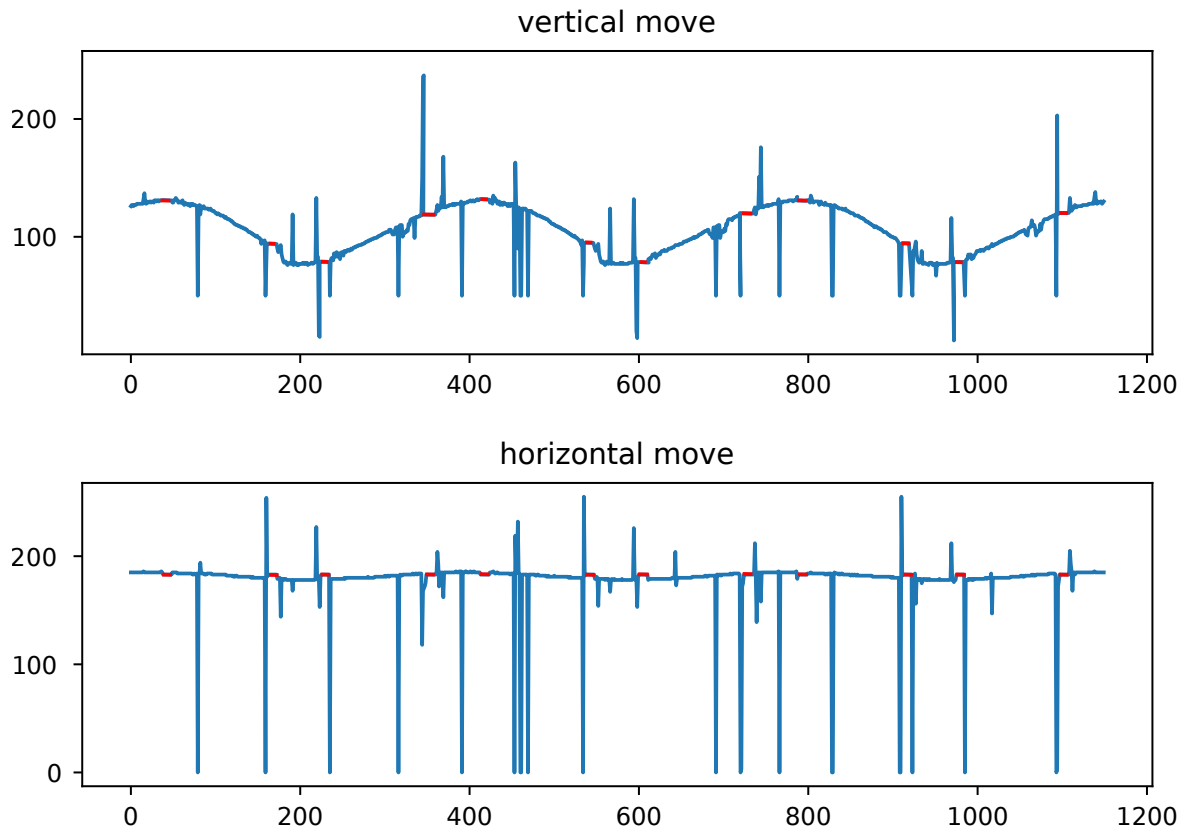

Figure 2.30: Specular spot movement tracking. The movement of the specular spot in a) vertical and b) horizontal in the test images is determined by the segmentation model via binary masks. The important amplitude of the vertical move than the horizontal one indicates the elliptical movement of the spot. The constant ranges of the vertical move curve indicated by the red dashes correspond to a blocked electron beam where the last known position is kept and the overshoots on both curves are due to prediction errors.

$(137 \times 229 \times 6)$ by stacking the images in the channel dimension. We define the prediction angle to be the angle of the first image of each sequence. As shown in the examples in Figure 2.31, the Kikuchipy [102] open-source software allows the Kikuchi lines to be highlighted, thereby enabling the model to focus on their movement and thus improving the prediction of the azimuthal angle. These lines depend not only on the surface of the sample but also on the bulk, which leads us to believe that this is a robust method for deducing the rotation angle from RHEED images. Two techniques are applied on the images using Kikuchipy: removing of dynamic background and adaptive histogram equalization. As illustrated on Figure 2.31 by red lines, the Kikuchi lines are significantly more prominent after pre-processing. The model architecture to conduct the regression task is detailed in the following.

Neural network architecture

We use again the Identity Mappings Residual Network reported by He Kaiming et al. in [90] for characteristics extraction. The details on our choice, the mathematics behind ResNet and the implementation of our ResNet block are reported in section 2.4.3. Figure 2.32 shows

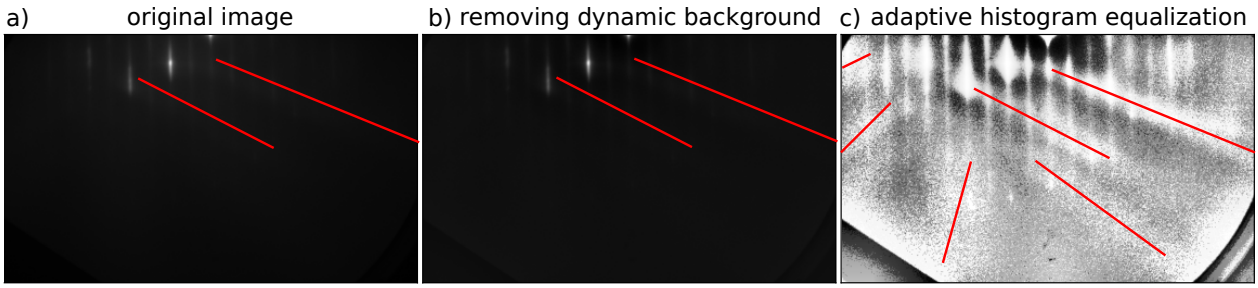

Figure 2.31: Image preprocessing using Kikuchipy [102]. Two techniques are applied on the a) original image: first b) removing of dynamic background and then c) adaptive histogram equalization to highlight the Kikuchi lines. The result is shown at each step and the red lines denote the visible Kikuchi lines.

the architecture of the specific RHEED-angle regression model, the numbers on the left of the blocks indicate how many times the residual units are stacked before the single *Conv2D* layer with strides equal to 2 for dimensionality reduction. Our residual neural network architecture consists of residual units involving three convolution layers each preceded by batch normalization and LeakyReLU activation function respecting the two identity conditions demonstrated in [90]. A convolution shortcut is used only for dimensionality matching, the consequence is minimal as explained in [90]. After ResNet blocks, a *flatten* layer combines the extracted features and passes them to a multilayer perceptron consisting of two dense layers each followed by a *Batch Normalization* and a *LeakyReLU* activation function ending with a single neuron with linear activation function for angle value output. The following reports the results of the ResNet model on 12 rpm and 4 rpm images.

Results

First we test the model on data from 12 rpm rotating sample. The histogram of the absolute error is shown in Figure 2.33a, the error is roughly distributed around zero with 2.25° standard deviation. In addition, Figure 2.34a shows the ResNet prediction plot versus the target values, where we can see superimposed curves demonstrating a good prediction accuracy. At this speed, the inertia of the sample holder is large enough to ensure a smooth rotation and we thus have angular values with high confidence.

In a second step, we test our model with data captured during 4 rpm rotating substrate. To simulate the 12 rpm speed on which the model is trained, the sequences of images are done taking every third image. The statistics of the absolute error of the 4 rpm test is shown in Figure 2.33b, showing a distribution around zero with 9.31° standard deviation. Figure 2.34b depicts the ResNet prediction and the target values plots. At this speed, the rotation is significantly less smooth due to the clearance and friction at the magnetic coupler. Despite these disruptions, the model predicts well the azimuthal angle. Now that we can detect the position of the specular point and determine the azimuthal angle from the RHEED images, the next section focuses on combining these results to construct the Azimuthal RHEED.

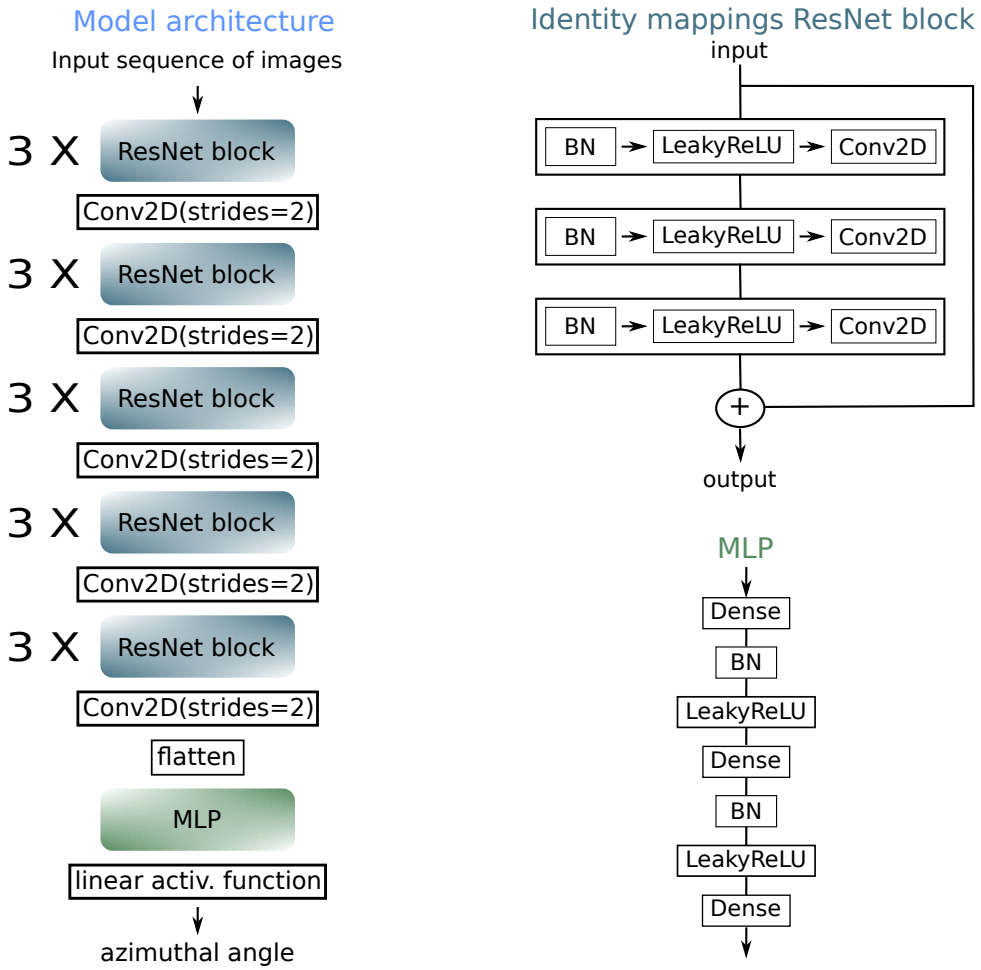

Figure 2.32: Residual neural network architecture for azimuthal angle. The block of identity mappings ResNet on the top right is called "residual unit" in [90] and contains three sub-blocks each consisting of a batch normalization, LeakyReLU activation function and 2D convolution. The result of the last convolution is added to a shortcut of the block input. The architecture of the whole neural network on the left is made of stacked identity mappings ResNet blocks each followed by a single 2D convolution layer for dimensionality reduction to extract informations from the input images. Afterwards, the features go through a *flatten* layer and are fed to an MLP to process the extracted features and output the final azimuthal angle value.

2.5.4 Azimuthal RHEED Plotting

The Azimuthal RHEED provides access to a wider spectrum of information than a single RHEED image. The ARHEED constructed in Figure 2.35 combines the essential features contained in 373 RHEED patters into a single image, making crystal growth monitoring more straightforward and rich in information. The ARHEED interpretation is illustrated in Figure 2.36, where the obtained ARHEED seems to indicate a $c(4\times4)$ surface reconstruction according to the brightness of the diffraction points in [010] and [100] directions. The large gray cross corresponds to the electron beam being blocked by the hooks on the sample holder.

Nevertheless, this Azimuthal RHEED can be further improved by perfecting the specular point detection and the number of rounds chosen in order to capture more details and conduct

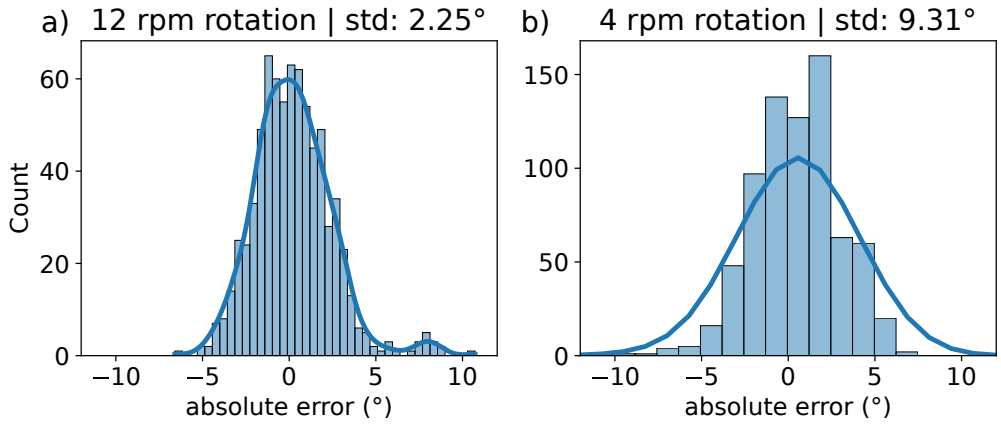

Figure 2.33: Histograms and density functions of the model test absolute error. Histogram and density of the absolute error between the prediction and target values for a) the test data from 12 rpm patterns, where the error is roughly distributed around zero with a standard deviation of 2.25° and b) the test data from 4 rpm patterns, where the error is also roughly distributed around zero with a standard deviation of 9.31°.

more advanced interpretations.

2.5.5 Conclusion

We proposed a semantic segmentation model to generate binary masks for the specular spot and thus detect its positions from RHEED images as well as a Residual Network for azimuthal angle prediction in order to construct the Azimuthal RHEED that requires a precise position detection and crystal rotation angle determination. Both models work well on the test data. The model for angle regression is trained on data from samples rotating at 12 rpm and the test is done on patterns from samples at 12 rpm and 4 rpm. The results are presented as histograms for absolute error distribution and plots comparing the model predictions and target values ending with the plotting of the Azimuthal RHEED. The following section concludes the chapter on the crystal growth characterization using Deep-Learning techniques on RHEED patterns.

2.6 Conclusion

We have reviewed the state-of-the-art where beyond Deep-Learning, the classical machine learning is used on RHEED images. Some Machine Learning algorithms can outperform Deep-Learning in some tasks and Deep Neural Network is not the solution to everything. According to the work done on DL and RHEED, the use of Convolutional Neural Network proved good potential [68,69]. Hence, we proposed CNN models for RHEED characterization automatization.

Firstly, we proposed a model for substrate deoxidation detection which typically is a tedious task done by the human MBE operator. The model is composed of two parts, an autoencoder to reduce image dimensionality and create compressed representations of entire

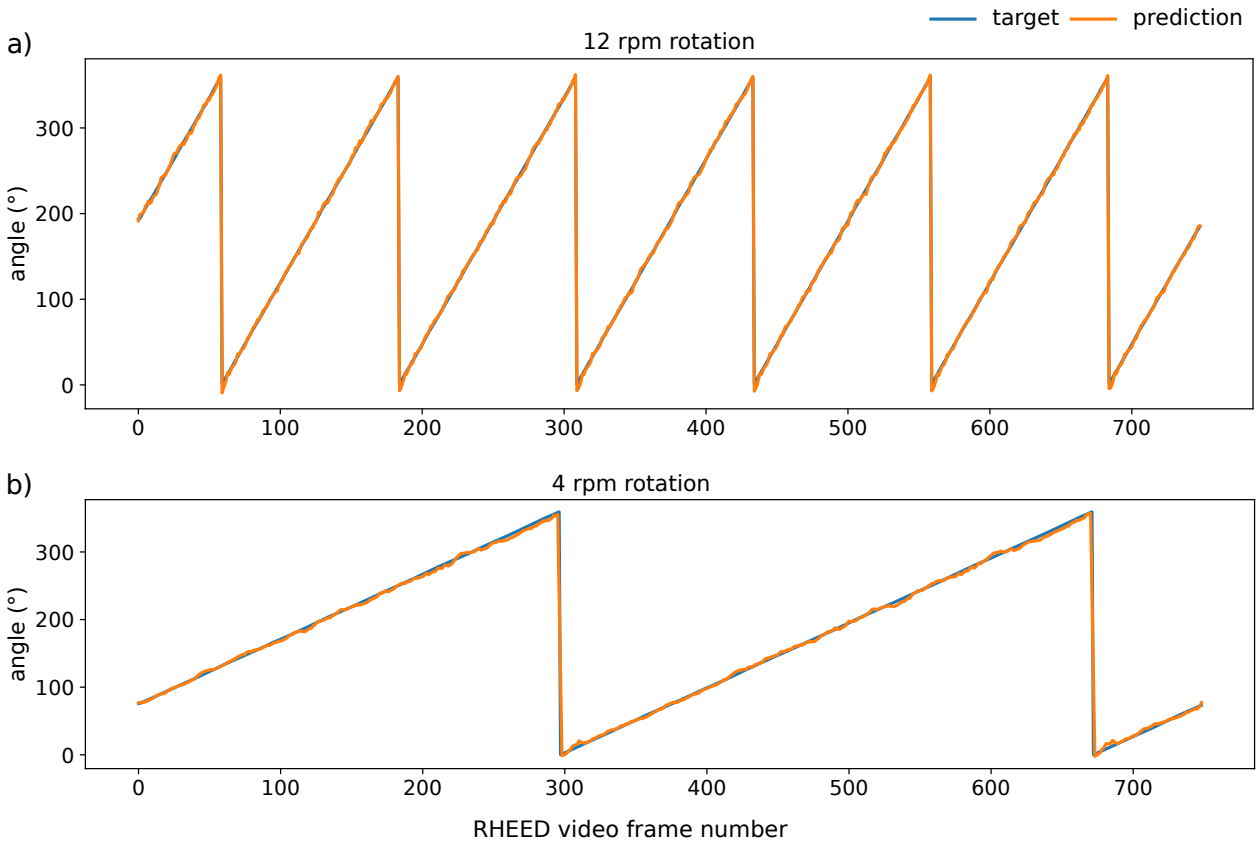

Figure 2.34: Azimuthal angle prediction. Azimuthal angle prediction (orange line) versus target values (blue line) for RHEED images captured during a) 12 rpm rotating substrate and b) 4 rpm rotating substrate for a random window of 749 images in each test dataset (12 rpm and 4 rpm).

RHEED video sequences, followed by a CNN classifier to distinguish data from oxidized and deoxidized patterns. We studied the neural network architecture by varying the length of the input image sequence and the size of the autoencoder latent space (hence the level of compression). We found that the use of image sequences instead of single images has significantly better accuracy. In addition, the model showed robustness over time periods as long as at least half a year, by working on data recorded six months after the training data without retraining despite the changes occurred on the MBE set up and the visible deterioration of the RHEED screen.

Secondly, we performed a classification task on surface reconstructions using Residual Network. We tested our model based on identity mappings ResNet on data captured during temperature increase and cooling down. In heating, the surface passes from $c(4 \times 4)$ to (2×4) and vice versa during temperature decrease.

Our final task in this chapter is the construction of the Azimuthal RHEED. This task needs the specular spot position to be detected and the azimuthal angle to be determined. The position is tracked using semantic segmentation model and the dataset is preprocessed using SAM, the segmentation model released by META. The determination of the azimuthal angle is done by regression using a ResNet model based on identity mappings like in the previous section. The training dataset is recorded while the sample is rotating at 12 rpm then we

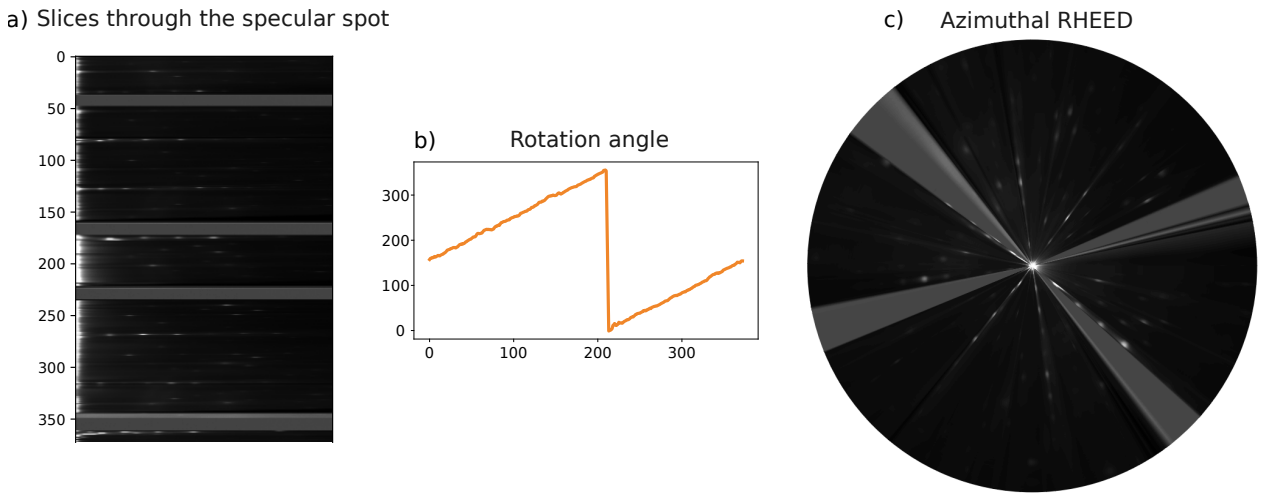

Figure 2.35: Azimuthal RHEED plotting. The images of one complete round captured during rotation of the sample at 4 rpm is taken to plot the Azimuthal RHEED, i.e. 373 images. This figure shows a) the thin slices through the specular point as cropped from the images stacked on top of each other, b) the azimuthal angle predicted for these images and c) the plotting of those slices in polar as a function of the angle providing the Azimuthal RHEED.

tested the model with data captured during 12 rpm rotation but also with 4 rpm data, when the rotation is less smooth. The ARHEED is plotted for a ser of 373 test images. A further challenge will be to generalize these Deep Learning models for materials other than GaAs for larger use on MBE-RHEED set up.

With this, we conclude the RHEED image analysis and move towards a different application of Deep Learning in nano-technology. In the following chapter we discuss the use of deep learning for inverse design and accelerated predictions in nanophotonics.

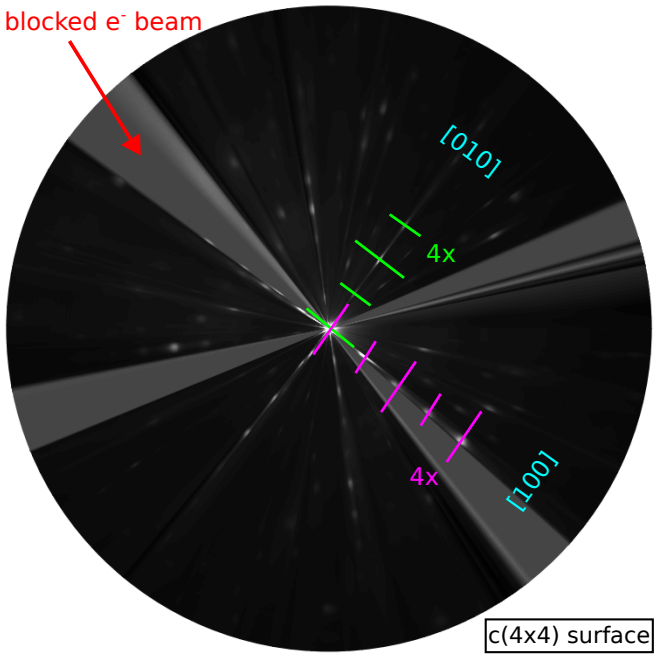

Figure 2.36: Azimuthal RHEED interpretation. The azimuthal RHEED contains information from all 373 images. The large gray cross corresponds to the electron beam being blocked by the hooks on the sample holder. The brightness of the diffraction points in the [010] and [100] directions increases and decreases successively, indicating that this is a $c(4 \times 4)$ reconstruction surface.

Chapter 3

Deep Learning-based inverse design for nano-photonic devices

3.1 Introduction

The majority of this introduction and the following two sections are derived from our previously published tutorial article [13] on nanophotonics inverse design with Deep Learning. The broad field of (nano-)photonics deals with the interaction of light with matter and with applications that arise from structuring materials at sub-wavelength scales in order to guide or concentrate light in a pre-defined manner [103–105]. Astonishing effects can be obtained in this way, such as unidirectional scattering, negative refraction, enhanced nonlinear optical effects, amplified quantum emitter yields or magnetic optical effects at visible frequencies. Tailoring of such effects via the rational design of nanodevices is typically termed "inverse design". Unfortunately, like most inverse problems, nanophotonics inverse design is in general an ill-posed problem and cannot be solved directly [106]. Usually, iterative approaches like global optimization algorithms or high-dimensional gradient based adjoint methods are used, which however are computationally expensive and slow, especially if applied to repetitive design tasks [107]. In the recent past it has been shown that deep learning models can be efficiently trained on predicting (nano-)optical phenomena [108–110]. This rapidly growing research interest stems from remarkable achievements that deep learning accomplished in computer science since around 2010, especially in the fields of computer vision and natural language processing. The main underlying assumption is that neural networks are universal function approximators [111]. It has been shown that deep learning is capable of solving various inverse design problems in nanophotonics. A non-exhaustive list of examples includes single nano-scatterers [112], gratings [113], Bragg mirrors [114], photonic crystals [115], waveguides [116], or sophisticated light routers [117].
As mentioned above, this work is from our previous article [13] in which we give a general Deep Learning introduction, address the question of when to use Deep Learning and when better not, explain the choice of model architectures for nanophotonics inverse design, all accompanied by a set of extensively commented Python notebook tutorials [118]. We demonstrated the full workflow from data generation and data-processing, over network architecture design and hyperparameter tuning, to an implementation of the different inverse design approaches. In the tutorial, we deal with two toy problems to implement the different inverse design methods. The first one is the reflectivity of a layer stack using *PyMoosh* [119] for the physics calculations and the second one is the scattering of dielectric nanostructures

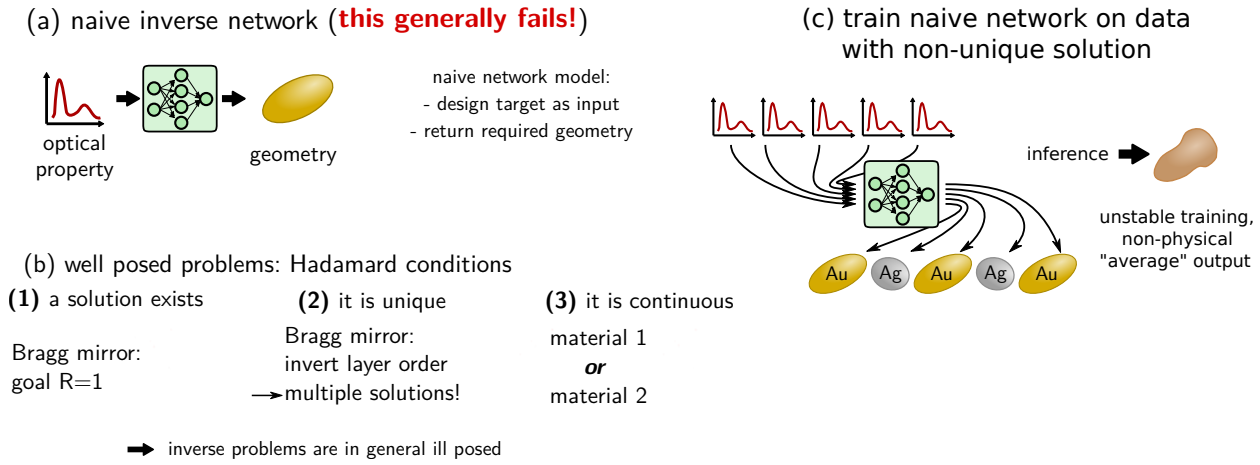

Figure 3.1: The crux of the ill posed problem. (a) A naive, not working implementation of a simple feed-forward inverse network would take as input the design target (e.g. an optical property) and returns the design that is required to obtain it. (b) Only well posed physical problems can be solved this way. Such problem obeys the three Hadamard conditions. However, neither of these conditions is in general fulfilled in photonics inverse design, as illustrated by a selected example under each condition. (c) In case of multiple solutions, the training process would iterate of these several times, every time adapting the network parameters to return a different design. Training is unstable and eventually the network will learn some nonphysical mix of the multiple solutions. If a non-continuous parameterization is used (here: two distinct materials), the naive network may also return non-allowed mixtures of those.

with *pyGDM2* [120, 121] for simulation. Brief descriptions of *PyMoosh* and *pyGDM2* are given in the appendix on page 112.

In the following section, we firstly introduce the inverse problem before focusing on inverse design methods of predicting the layer stack from the reflectivity with "one-shot" solvers consisting of end-to-end networks that can be implemented in different ways, we specifically discuss the Tandem network, as well as the conditional Variational Autoencoders. Secondly, we explain the Neural Adjoint method, a gradient-based iterative optimization to inverse design dielectric nanostructures from the target scattering, using deep learning models as ultra-fast and differentiable surrogates for slow numerical simulations. In this thesis an introduction to Deep Learning can be found in chapter 1, here we introduce possible approaches to tackle inverse problems with deep learning more conceptually, without going into detail about technical aspects of specific neural networks. This chapter will conclude with the proposal of a graph convolutional network (GCN) surrogate model of which one key advantage is handling structures with arbitrary size and form.

3.2 Inverse design: an ill-posed problem

Unlike the forward problem, where the optical response of a given structure is calculated, the inverse problem seeks to identify the structural parameters that yield a specific optical behavior. The primary goal is to design nanophotonic structures, such as photonic crystals and metamaterials that perform specific functions. These functions could range from achieving high-efficiency light trapping and guiding to tailoring the emission and absorption spectra.

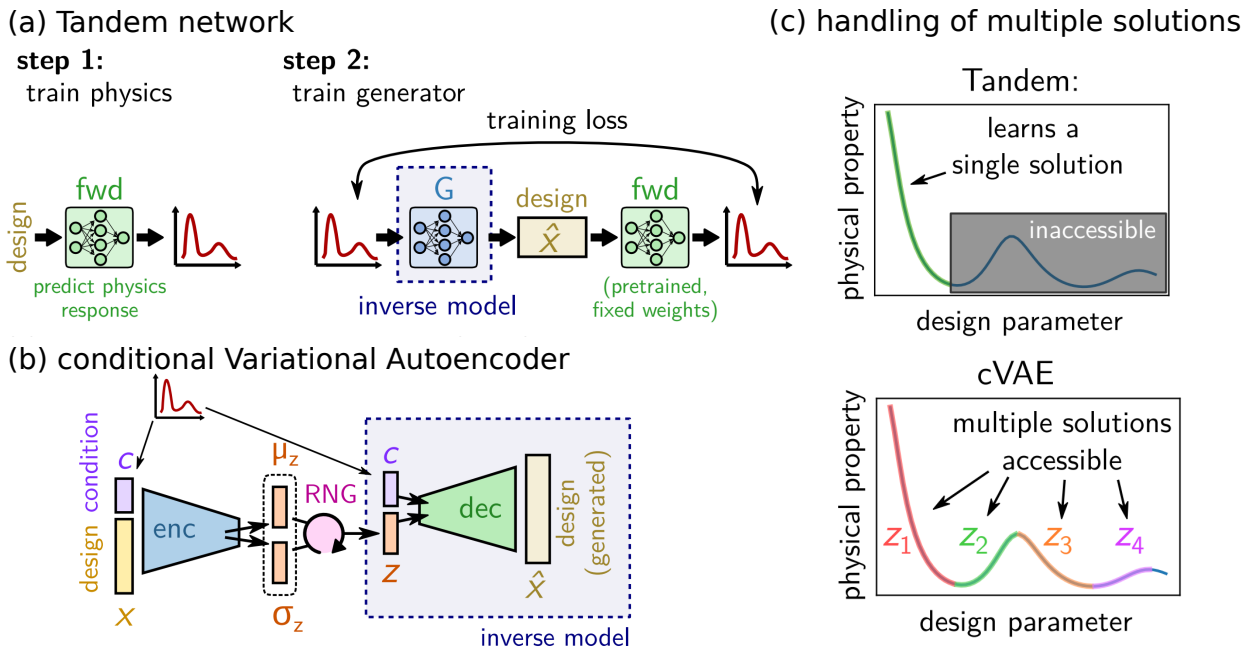

Figure 3.2: Tandem and cVAE architectures. (a) Tandem model. The training is divided in two steps. At first a forward predictor is trained on the direct problem. Subsequently, the forward network is fixed and used to train the generator, (b) The conditional Variational Autoencoder (cVAE) is trained end-to-end in a single run. A latent space z is used to provide additional degrees of freedom to handle ambiguities in the design problem. (c) inverse problems typically can be solved by multiple solutions. A Tandem model will learn only one of possibly multiple solutions, the other remain inaccessible. The cVAE on the other hand typically learns the set of possible solutions which can be retrieved via the latent vector z.

The main challenge in nanophotonics inverse design is that the problem is in general ill posed, in consequence it is impossible to solve the problem directly. J. Hadamard described a so-called "well posed problem" as one for which a solution does exist, this solution is unique and continuously dependent on the parameterization [106]. As depicted in Figure 3.1, the typical inverse design problem however has in general non-unique solutions (multiple geometries yield the same or very similar property). Often design targets exist that cannot be optimally implemented, hence no exact solution exists (e.g. a mirror with unitary reflectivity). And finally, in many cases the physical property of a device is not continuously dependent on the geometry, but the parameter space is at least partially discrete (e.g. if a choice from a finite number of materials has to be made). Training of a naive network on a problem with multiple possible solutions will oscillate between the different possible outputs and finally learn some non-physical average between those different solutions [122]. Fortunately, methods exist to solve ill posed inverse problems with deep learning. We discuss in the following two popular groups of approaches, namely the "one-shot" inverse design methods and the iterative methods that use optimization algorithms to discover the best possible solution(s).

3.3 Inverse design methods

The naive approach to solve inverse design with deep learning would be to use a feed-forward neural network that takes the optical property as input and returns the geometry parameters

as output. This would be trained on a large dataset. Unfortunately such approach does not work. In the following, we discuss Tandem network, cVAE and Neural Adjoint method.

3.3.1 Tandem network

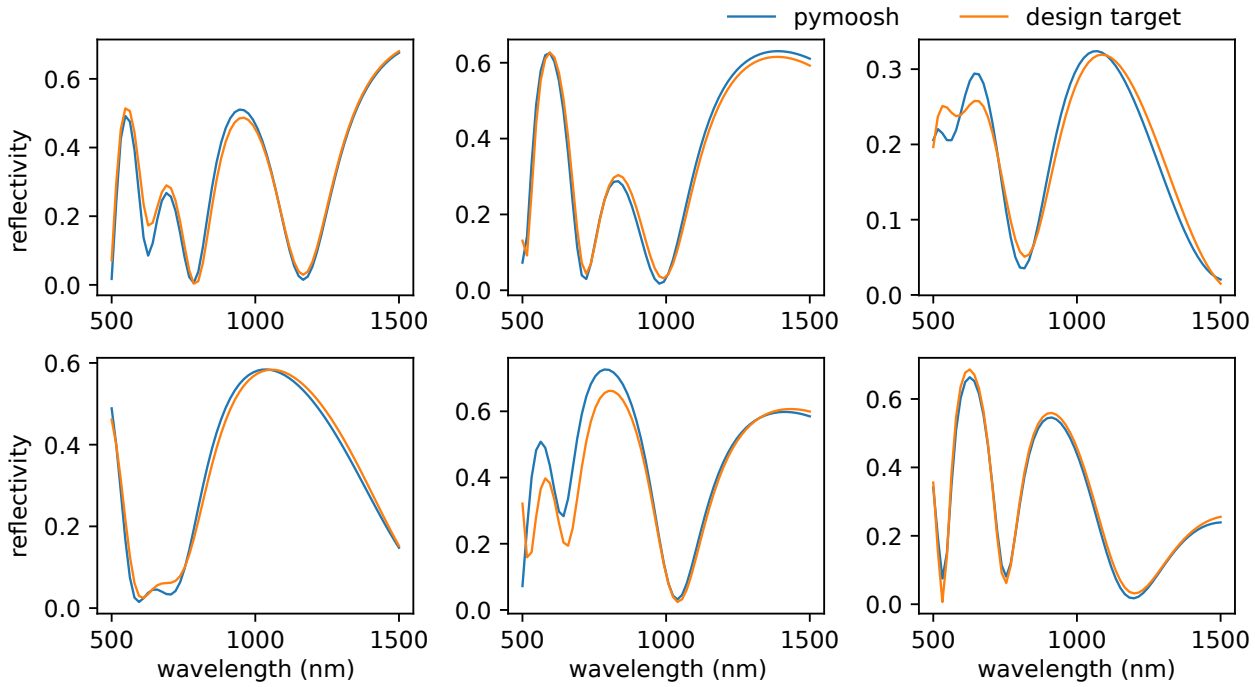

Figure 3.3: Tandem network inverse design. Target reflectivity spectrums (orange curve) are fed to the inverse model which predicts the structures (five layers characterized by refractive indices and thicknesses for each structure); and then reflectivity spectrums of the predicted designs are recalculated with PyMoosh (blue curve).

One of the most simple configurations for a one-shot inverse design network is the so-called "tandem network" [123]. The tandem network is a variation of an autoencoder acting on the physical domain. It takes as input the desired physical property and returns a reconstruction of the physics (for instance a target reflectivity spectrum and its reconstruction). The difference to a conventional autoencoder is that the decoder is trained in a first step on predicting the physical properties using the design parameters as input. This means, the decoder is simply a "forward" physics predictor, solving the direct problem ("fwd" in Figure 3.2a). Subsequently, a second training step is performed, in which the forward model weights are fixed and the encoder, which is actually trained on generating the designs, is added to the model (generator "G" in Figure 3.2a). In this second step, the full model is trained, but now only the physical responses from the training set are used. The physical property (e.g. reflectivity spectrum, etc...) is fed into the encoder, which predicts a design. However, instead of comparing this design to the known one from the dataset, the generated design is fed into the forward model, that predicts the physical property of the suggestion. This predicted response is finally compared with the input response, the error between both being minimized as training loss. This means, that even if multiple possible design solutions exist, the training remains unambiguous since only the physical response of the design is evaluated, regardless of how it is achieved. The full model is then essentially an autoencoder of which

the latent space is being forced to correspond to the design parameters by using the fixed, pre-trained forward network as decoder.

A practical advantage of the Tandem is that the inverse problem is split in two sub-problems, that are individually easier to fit, compared to end-to-end training of the full inverse problem. In a first step the forward problem is learned, which is usually a relatively straightforward task. This physics knowledge is then used in the second step to guide training of the generator network. In this example, we trained the neural network to predict multilayer architectures of five layers with refractive index n in the range [1.3, 2.5] and material thickness d in the range [20, 180] nm. To visually test the model accuracy, as shown in Figure 3.3, we feed target reflectivity spectrums (orange curve in Figure 3.3) from the test dataset to the inverse model to predict geometry designs; and then we compute the reflectivity spectrums (blue curve in Figure 3.3) of the predicted structures using *PyMoosh* and finally compare them with the target ones. As ill-posed problems don't necessarily have solutions for all inputs, target spectrums are pre-calculated, so we know that layer-stacks exist as solution. The inverse model demonstrates good accuracy on such inverse problem.

3.3.2 Conditional variational autoencoder - cVAE

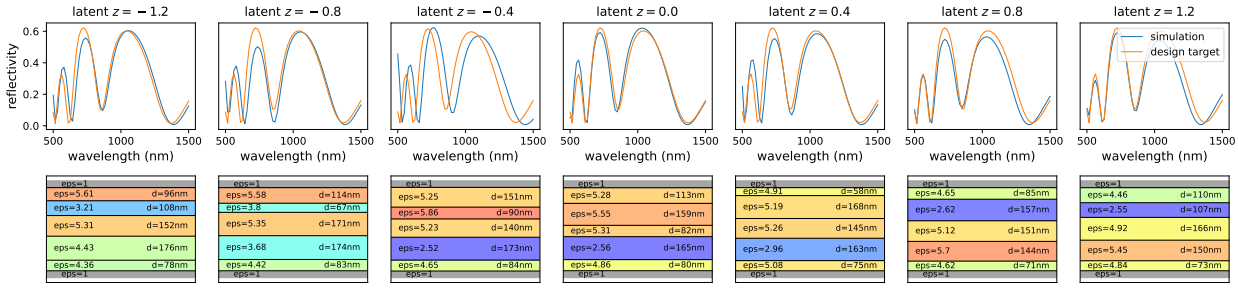

Figure 3.4: cVAE inverse design. Dielectric layer stacks implementing an arbitrary reflectivity design spectrum. Inverse designed by a cVAE. By sweeping through the latent space of the cVAE generator with fixed target spectrum, multiple possible design solutions can be identified. Note that the cVAE discovered that mirrored structures yield the same reflectivity spectra (c.f. for example latents z = 0 and z = 0.8). A systematic latent inspection can also be done for further optimizing the solution, for example by a search for the best possible spectral match, or by identification of the most robust design, etc..

A drawback of the Tandem network is that only a single solution is learned, even if multiple designs are possible to reach the design target. Several network architectures have been developed to learn mappings to the set of multiple solutions in ambiguous inverse problems. We discuss in the following a very efficient and robust model, the conditional Variational Autoencoder (cVAE).

The reason why a "vanilla" variational autoencoder (VAE) can not be directly trained on an inverse design task is the correlation between geometry and physics domain. The latent space of the autoencoder forms during training and represents the most efficient and compact, reduced representation of the inputs. This is of course generally not the design parametrization. In consequence, it is necessary to force the latent space to correspond to the design

space, which is achieved in the "Tandem" architecture via a two-step training procedure. The tandem is hence an autoencoder with design-regularized latent space.
By conditioning the designs on their physical properties, a modified variant of a VAE, a so-called conditional variational autoencoder (cVAE), can however be trained as an inverse design network. To this end, the classical VAE, that reproduces a design through an encoder-decoder architecture, is extended by an additional input, the design condition. Here this is a physical property (e.g. a reflectivity spectrum), the design target. As depicted in Figure 3.2b, this additional condition is added as input to both, the encoder (blue) as well as the decoder (green). During training, multiple possible solutions are associated with different values of the latent vector z, i.e. can be treated without training ambiguities, as illustrated in Figure 3.2c.

As mentioned before, cVAEs require a latent regularization scheme in their training. The goal is for the latent space to become continuous and smooth (to allow meaningful interpolation). This is achieved using perturbative random latent sampling in the forward path, so the network learns that similar latent values correspond to similar solutions. In order to additionally achieve compactness (no blank regions in latent space), a weighted ("β"-coefficient) KL-loss is added to the training, which pushes the latent variables to a normal distribution around zero with unitary variance. If the KL loss weight is too large, the latent space will be normally distributed around zero, but reconstruction will fail. If the KL loss weight is too small, it has no effect. Then reconstruction will be good, but blank spaces in latent space may occur that do not carry useful information and impede to perform meaningful interpolation between solutions. In consequence, the weight of the KL loss with respect to the reconstruction loss needs to be carefully chosen ("β-VAE" [124]).

Unfortunately this value needs to be adapted for each problem / network model, so some trial and error is required to find the adequate value. Good starting values are typically $\beta = 0.01$ or $\beta = 0.001$. Blank latent spaces may be difficult to spot in training, so the easier approach to find a good weight is to increase the β value until the reconstruction loss starts suffering notably. The condition is the design target and may be a high-dimensional construct such as a reflectivity spectrum. It is possible to process the condition with a sub-network (e.g. a 1D-CNN), that can be a common, shared network, before the two input branches (encoder and decoder).

After successful network training, only the decoder is used for the inverse design. The design target is fed to the decoder (the actual inverse model) accompanied by a random normal latent z. The reflectivity spectrum of the predicted design is calculated with *PyMoosh* for comparison with the target design. Figure 3.4 illustrates the possibility to identify multiple possible solutions and how to go through those solutions in a smooth way by the example of multi-layer designs for a fixed reflectivity target, this also demonstrates the strength of the regularized latent space in cVAE. An advantage of the cVAE is its generally robust training. It often also works well with low-dimensional latent spaces, so that the latent space can be explored systematically, to identify different possible solutions [114]. A recent comparison indicates, that cVAEs are among the most effective methods for direct inverse design tasks [125].

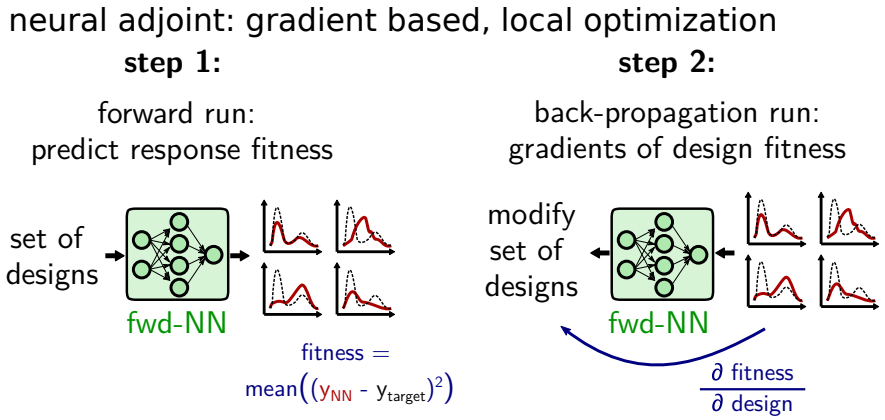

Figure 3.5: Inverse design via gradient-based iterative optimization using a forward model as fast physics solver surrogate. Neural adjoint method reduces the risk of local minima by operating on a large set of random initial designs. The optimization tries to minimize the error between the predicted optical property (solid red line) and the design target (dashed black line).

3.3.3 Neural Adjoint method

While global optimization, numerical analysis that attempt to find the global solution for inverse problems, is robust and generally converges well towards the overall optimum, such methods are also inherently slow since they do not take advantage of gradients. This is unfortunate because gradients are available "for free" in deep learning surrogate models. On the other hand, gradient based approaches tend to get stuck in local minima. This can be accounted for to a certain extent, but it usually depends strongly on the individual design problem if a gradient based method will work. The idea of gradient based optimization is the same as in the Newton-Raphson method. A fitness function is defined such that it is a measure of the error of a solution compared to the design target. Then, the derivatives of the fitness function with respect to the design parameters of a test solution are calculated and used to modify the test-design towards the negative gradient. By minimizing the fitness function in this way, the solution iteratively gets closer and closer to the ideal design target until a minimum is reached.

Typical numerical simulation methods are not differentiable and hence gradient based methods cannot be applied directly. While gradients can be calculated using adjont methods [126], these still require multiple calls of the, generally, slow simulation and hence are usually computationally expensive. Both problems can be solved to some extent by forward neural network models. A key advantage is, besides the evaluation speed, that gradients can be calculated "for free", because the network is an analytical mathematical function. For the same reason, the gradients of the surrogate model are also continuous, since this is a key requirement for the network training alogrithms. As stated in the beginning, the training procedure of a neural network is in fact a gradient based optimization by itself, therefore the main functionality of all deep learning toolkits is automatic differentiation. A forward neural network model can thus always be used for gradient based inverse design, which consists of two steps that are illustrated in Figure 3.5. In a first step, a set of test-designs (typically random initial values) is evaluated with the forward model. Their predicted physical behavior is compared to the design target, for which a fitness function evaluates the error between

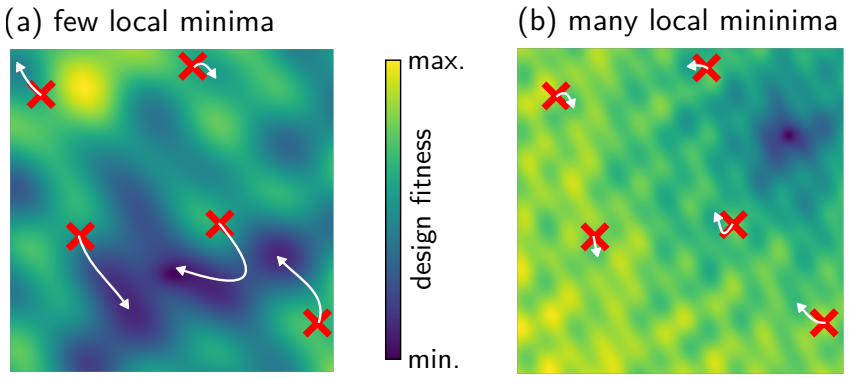

Figure 3.6: Schematic fitness landscapes of (a) a friendly problem with relatively few local extrema. (b) a complicated problem with many local fitness minima. Using gradient based methods with a large number of initial test-sets, problem (a) will likely converge to the global optimum. In problem (b) on the contrary, the chance is high that none of the initial designs is close enough to the global optimum and optimization will converge to a local solution. The paths taken by a gradient-based method path are indicated by white arrows.

target and prediction. Now the deep learning toolkit is used to calculate the gradients of this fitness with respect to the input design parameters via backpropagation and the chain rule. Finally, the designs are modified by a small step towards the negative gradients. Repeating this procedure minimizes the fitness [127–129].

As mentioned before and depicted in Figure 3.6, the main difficulty in this approach is to avoid getting stuck in local minima of the fitness function. To a certain extent this can be accounted for by optimizing a large number of random initial guesses for the designs in parallel. While such strategy would be prohibitively expensive using numerical simulations, with a machine learning surrogate model it is typically possible to optimize several hundreds or even thousands of designs in parallel. However, depending on the problem, the number of local extrema may be too large for successful convergence. This can be tested by running the optimization several times. If multiple runs do not converge to a similar solution, the parameter landscape of the problem is probably too "bumpy" for gradient based inverse design, Figure 3.6 shows such a landscape with many local minima.

As explained above, it is crucial also in gradient based optimization to remain in the forward model's interpolation regime since extrapolation bears a high risk of converging towards non-physical minima of the deep learning model [127]. Also, if the dimensionality of a problem is high, the risk of strongly varying gradients further increases and optimization may always converge to unsatisfying local minima. As discussed above, in such cases it is helpful to train a separate generator network that maps the design parameters onto a regularized latent space (e.g. VAEs or GANs see Figure 3.7a,b). Instead of optimizing the physical design parameters, the optimizer then acts on this design latent space. Because the latent space is regularized, it is possible to constrain the designs to the neural network's interpolation regime e.g. by using a KL loss term in the fitness function.

To practically show the effect of constraining design parameters in the interpolation regime of the surrogate model, we trained a ResNet model on the direct problem to predict the

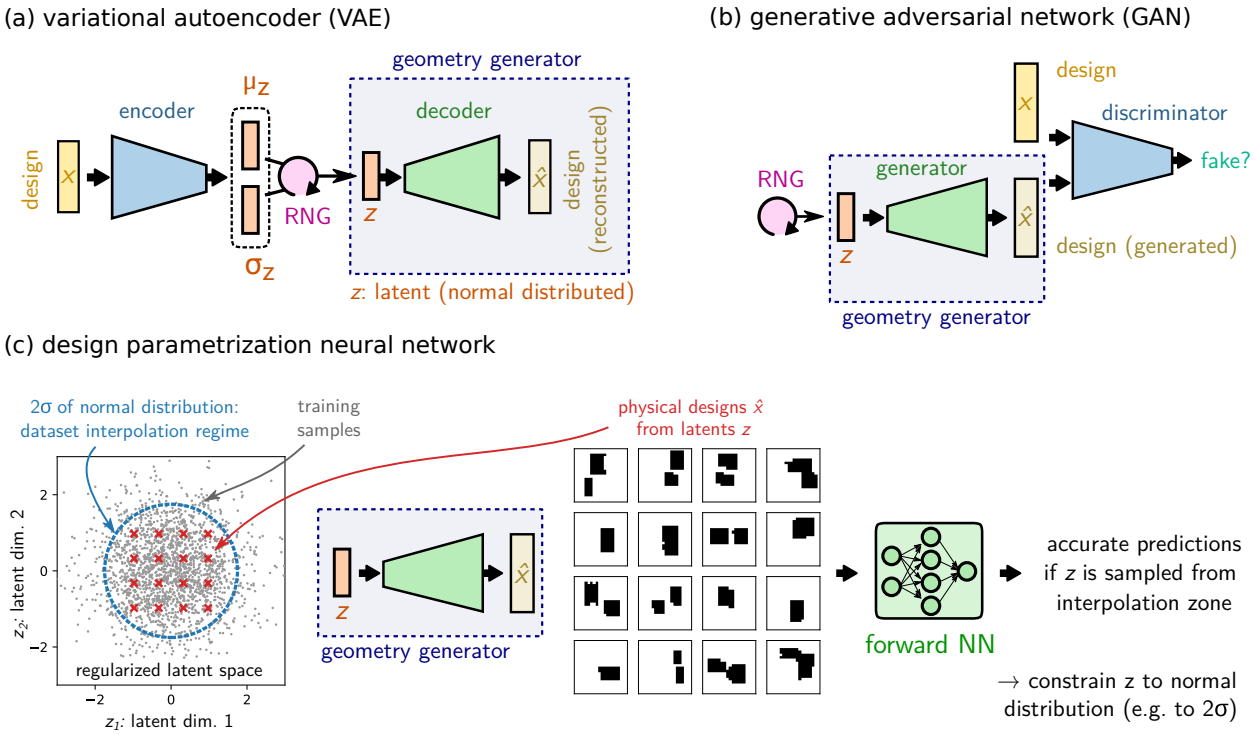

Figure 3.7: Constraint of the forward model into the interpolation regime via generator. (a) Sketch of a variational autoencoder (VAE) trained to reconstruct the design. In a VAE, the encoder is trained to return the mean value (μ_z) and standard deviation (σ_z) of the latent variable z. A randomized, normal distributed latent vector is passed to the decoder for the reconstruction task (via random number generator "RNG"). By further constraining σ_z with a KL loss, one obtains a compact and smooth latent space that is normally distributed. (b) Sketch of a generative adversarial network (GAN). As in the VAE, by using a normal distributed random number generator during training for the latent space input, the generator develops a smooth and compact latent space, essentially representing the interpolation regime of the dataset. (c) Re-parameterized the forward model using a learned latent representation as design input: The trained geometry generator (e.g. from VAE or GAN) is simply plugged before the input of the forward network. It converts a latent vector z to a physical design $\hat{x}$. If the latent space was properly regularized, sampling from within the range of a normal distribution with unitary variance will generate geometries in the interpolation regime of the training data, where the forward network works accurately. Instead of optimizing the physical design parameters, we can now run the optimization on the latent variable of the geometry generator. Constraining the numerical range of the optimization parameters accordingly, renders iterative optimization robust.

scattering of dielectric nano-structures which are represented by the geometry top view. The dataset is generated using *PyGDM* [120, 121]. The full model architecture is illustrated in Figure 3.7c where the latent space variable of the geometry generator is constrained to 2σ of normal distribtution. To test the inverse design accuracy, we take a geometry and the corresponding scattering from the test dataset, we optimize a design via neural adjoint method using only the physical property; and then, we calculate the physical property of the obtained geometry via both *PyGDM* simulation and forward model prediction; and finally we compare the results from the inverse designed geometry, forward model prediction and the test set (the reference). In such case, the neural adjoint-optimized geometry is successfully verified with *PyGDM* as shown in Figure 3.8a. In a second step we apply the neural adjoint method without latent space constraint and then without any generator (running the NA on image pixels), the results of both cases are reported in Figure 3.8b,c respectively and show that the method fails when pushed into extrapolation regime.

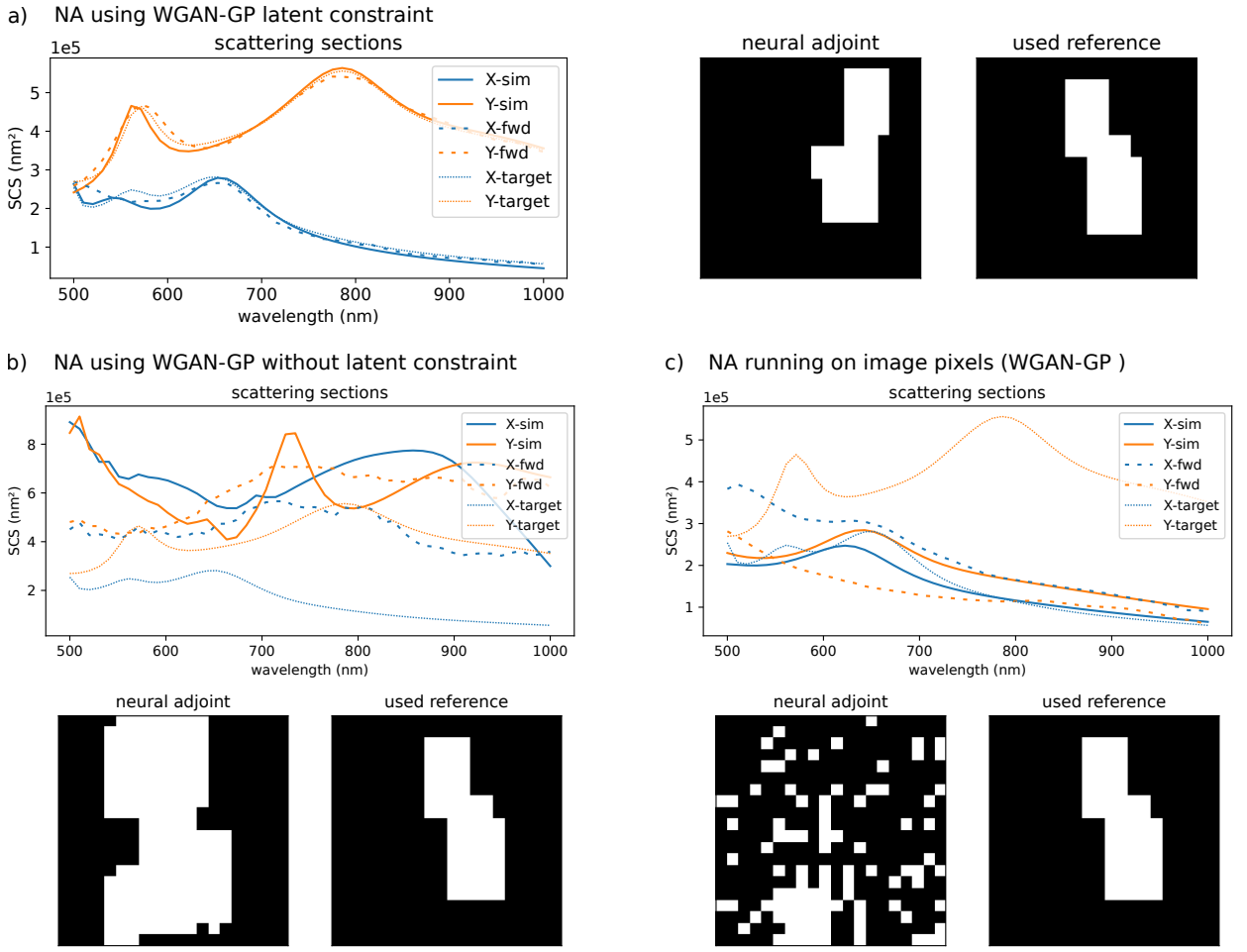

Figure 3.8: NA and WGAN-GP results. At the left of the top row a) the physical properties of the used reference, neural adjoint-optimized design (simulated with *PyGDM*) and the prediction of the forward model matched. At the right, the geometry generated via NA method optimization and the used reference from the test dataset. Neural adjoint method failing cases: The NA method is launched b) without geometry generator latent space constraint and c) without any geometry generator where the process is directly optimizing the image pixels. The method fails in both cases.

3.3.4 Conclusion

The ill-posed inverse design problem can be solved with some Deep Learning architectures like Tandem network and cVAE which are known as "one-shot" methods; but also with the iterative Neural Adjoint method. The advantage of generator-based networks (e.g. cVAE) is the access of the multiple possible solutions by varying the latent space variable where a Tandem network can enable only getting one of those possible solutions. In the case of NA method, it is required to constrain the surrogate model in the interpolation regime because otherwise a high risk exists the gradient-based optimization diverges to the failing extrapolation regime.
In case of nano-structures inverse design, we used the geometry top view images as design parameters but the nanophotonics devices are not always representable via images especially in case of complex forms that are not suitable for 2D representation. It is important to find ways to handle arbitrary forms and sizes. To this end, we propose in the following a Graph Neural Network surrogate model.

3.4 Graph Neural Network surrogate model

Graph Neural Networks (GNNs) are a class of deep learning methods designed to perform inference on data described by graphs. Due to the different possibilities offered by graph machine learning and the large number of applications where graphs are naturally found, GNNs have been successfully applied to a diverse spectrum of fields to solve a variety of tasks. In physics, GNNs have been used to learn physical models of complex systems of interacting particles [130–132] and in chemistry, for the prediction of quantum molecular properties as well as the generation of novel compounds and drugs [133]. GNNs have also been largely applied to the biological sciences, with applications like the recommendation of medications [134].

3.4.1 Geometric Deep Learning

The deep learning technologies, for instance, the convolutional neural networks that we used in our models, have achieved remarkable outcomes in certain machine learning applications, including object detection, image classification, speech recognition and machine translation. Despite the considerable success of deep learning in processing traditional signals, such as images, sounds, videos, or text, the current research on deep learning is primarily focused on these data, which are defined in the Euclidean domain, namely grid-like data. With the advent of larger data sets and more powerful GPU computing capabilities, there is a growing interest in processing data in non-Euclidean domains, such as graphs. This type of data is pervasive in real-world scenarios, making it crucial to investigate deep learning techniques in non-Euclidean domains. This is referred to as geometric deep learning.

As mentioned in the introduction chapter, a graph is composed of nodes and edges of the network structure data. For instance, in social networks, each node represents a person's information and the edge represents the relationship between people. The edges can be directed or undirected depending on the relationship of the connecting vertices (nodes). These geometric data are irregularly arranged (i.e. there is no the relative positions such as left, right, up and bottom like between image pixels), which makes it difficult to find out

the underlying pattern using convolution operations like those on images [135]. On the other hand, data like images in the Euclidean domain can be regarded as a special graph data, with nodes arranged in a regular way, for instance in an image-graph, each pixel can be represented by a node connected to its neighbor pixels and each node contains the information about brightness and color of a pixel. Non-Euclidean data can have extraordinarily large scale. For example, molecular graphs can have hundreds of millions of nodes, for this case, it is unlikely to use the traditional deep learning technology to carry out analysis and prediction tasks [135]. In general, geometric deep learning can be mainly divided into two major research directions, one is for processing graph data (i.e. graph networks or grid-like data); the other is for processing manifold data (i.e. generally for processing 3D point cloud data). Among them, graph data processing is more popular. The adaptation of the convolution operation to graphs is achieved through two distinct techniques, namely the spatial and spectral methods.

3.4.2 Spatial Graph Convolutional Networks

Spatial Graph Convolutional Networks (GCNs) are a class of neural networks designed to handle graph-structured data by leveraging the inherent spatial relationships among nodes, this approach implements the message passing method. The primary principle behind spatial-based GCN is the iterative updating of node representations by aggregating and combining information from a node's local neighborhood (connected nodes) [136]. This approach directly exploits the graph topology, ensuring that the node embeddings reflect both the node features and the structural context provided by neighboring nodes. In a multi-layer GCN, each graph convolutional layer updates graph node informations simultaneously (one layer corresponds to one graph update). The iterative nature of the updates allows the network to capture increasingly complex patterns as information propagates through the GNN layers. Spatial-based GCNs operate in two main phases: aggregation and combination.

During the aggregation phase, a node gathers information from its immediate neighbors. This local aggregation is crucial as it allows each node to integrate information from its surroundings, forming a context-aware representation. Various aggregation functions, such as sum, mean and max are used to pool the neighborhood information. Each function has its unique benefits; for instance, sum aggregation captures the total contribution from neighbors, while mean aggregation normalizes the aggregated information, providing a balanced view regardless of the number of neighbors.

The combination phase then fuses the aggregated neighborhood information with the node's own features, typically through a learnable transformation followed by a non-linear activation function, enhancing the network's ability to capture complex relationships and patterns within the graph. Spatial-based GNNs are often constructed with multiple layers, each performing a round of aggregation and combination. Stacking layers enables the network to capture information from increasingly larger neighborhoods. However, care must be taken to avoid over-smoothing, where too many layers lead to indistinguishable node features. Some spatial-based GNNs, such as Graph Attention Networks (GATs), incorporate attention mechanisms to assign different importance weights to the neighbors of a node. This approach allows the network to focus on more relevant neighbors, enhancing the quality of the aggregated information and improving the overall performance.

The main advantages of such a method are the intuitive design by directly operating on the graph structure as well as the flexibility, since these models are highly flexible and can be tailored to various types of graphs and domains by selecting appropriate aggregation and combination strategies. However, while spatial-based GNNs excel at capturing local neighborhood information, they might struggle with capturing global graph properties, which can be essential for some tasks.

3.4.3 Spectral Graph Convolutional Networks

Spectral GCNs utilize the mathematical foundations of spectral graph theory to perform convolution operations on graphs. Unlike spatial-based GCNs, which directly aggregate node features from neighbors in the graph domain, spectral-based GNNs apply graph convolutions in the frequency domain [136] by transforming the graph signals using the eigenvalues and eigenvectors of the graph Laplacian matrix. This involves representing the graph in terms of its Laplacian matrix L which captures the connectivity and structure of the graph. It is computed from the adjacency matrix A (which represents connections between nodes) and the degree matrix D (which represents the number of edges incident to each node) by $L = D - A$. Spectral convolutions capture graph characteristics by transforming the node features into the spectral domain, applying a filter and then transforming back to the spatial domain.

This spectral-based method presents advantages like effectively capturing the global information of the graph making them suitable for tasks that require an understanding of the entire graph. However, they may not be as effective in capturing local structures compared to spatial-based GCNs. One of the main drawbacks of spectral GCNs is the computational complexity as calculating the eigenvalues and eigenvectors of the graph Laplacian is computationally intensive, especially for large graphs. This limits the scalability of spectral-based methods.

Graph convolutional networks are applied to a variety of domains such as analyzing molecular structures, where nodes represent atoms and edges represent bond as well as on social networks for user behavior understanding, community detection and influence propagation by analyzing the social interactions among users. They are also used in recommendation systems in order to enhance the recommendations by capturing user-item interactions and their underlying graph structure. In the following, we discuss the graph convolutional layer used for our applications.

3.4.4 Employed graph convolutional layer

In order to combine the advantages offered by both methods (spatial and spectral), some methods have been proposed recently to reduce the computational burden of graph Fourier and inverse graph Fourier transforms, while still utilising their foundations in the spectral domain. Such methods are different from both pure spatial and pure spectral convolutions. These methods are not designed using eigenvalues, but are instead implicitly designed as a function of structural information (adjacency, degree, Laplacian matrices) and perform convolution in the spatial domain like all spatial convolutions [137].

We use such a method for our problems in this section by building a graph convolutional network using Spektral [138], a Python library based on Keras and Tensorflow that provides tools to create and manipulate graphs, build graph neural networks and perform automatic differentiation to train geometric Deep Learning models. We created the GCN on the base of *GCSConv*, a convolutional graph layer that computes

$$X_{k+1} = D^{-1/2}AD^{-1/2}X_kW_1 + X_kW_2 + b \tag{3.1}$$

where D is the degree matrix, A is the adjacency matrix symmetrically normalized with $D^{-1/2} = \frac{1}{\sqrt{D}}$ so that the nodes with high number of connections do not be in very different range and shade the nodes with fewer connections as the network goes deeper, X_k represents the node features at the k-th layer, W_1 and W_2 are weight trainable matrices and b is a biases trainable vector. As mentioned before, this layer is function of graph structural informations (D and A) like the spectral method which uses the Laplacian matrix $L = D - A$ while the computation is performed in the graph spatial domain, which makes it a bridge between these two methods. To avoid over-smoothing of the node features, we only stack four or five graph convolutional layers in the architectures.

Graph neural network tasks can be broadly categorized into two types: graph-level tasks and node-level tasks. Graph-level tasks focus on predicting properties or labels for entire graphs rather than individual nodes. These tasks are crucial in scenarios where the holistic structure and features of the graph are important. Graph classification involves assigning a label to an entire graph. Graph regression involves predicting continuous values for entire graphs. Graph generation involves creating new graphs that possess certain desired properties. Node-level tasks focus on predicting properties or labels of individual nodes within a graph. These tasks leverage the local and sometimes global structure of the graph to enhance node-specific predictions. Common node-level tasks include node classification, node regression and node clustering. Node classification involves assigning a label to each node in a graph. Node regression involves predicting continuous values for nodes. Node clustering aims to group nodes into clusters based on their structural and feature similarities.

In the following, we apply GCNs on nano-cylinders multiscattering and multilayer stacks reflectivity which consist of node-level regression and graph-level regression tasks, respectively. For the first task, we aim to predict characteristics to individual nodes. Graph convolutional layers update the node representations in five rounds (five consecutive layers) and the final layer node features represent the model prediction, an illustration is depicted in Figure 3.9. In the second task, we predict the reflectivity spectrum of the hole graph (stack of multilayers), four graph convolutional layers are followed by a pooling layer in order to coarse the graph into a 1D vector and a final *Dense* layer of 64 neurons outputs the reflectivity spectrum.

3.4.5 Electric polarization of nano-cylinders

As mentioned before, we apply GCN on the problem of light multiscattering from nano-cylinders. The simulations to generate data are performed using *pyGDM2* [120, 121]. The cylinders have 15 nm radius with a refractive index $n = 3$ and are distributed in a grid of positions with 50 nm spacing, they are assumed to be infinitely high which leads to a 2D problem. Figure 3.10 shows an illustration of a structure containing eight nano-cylinders, Figure

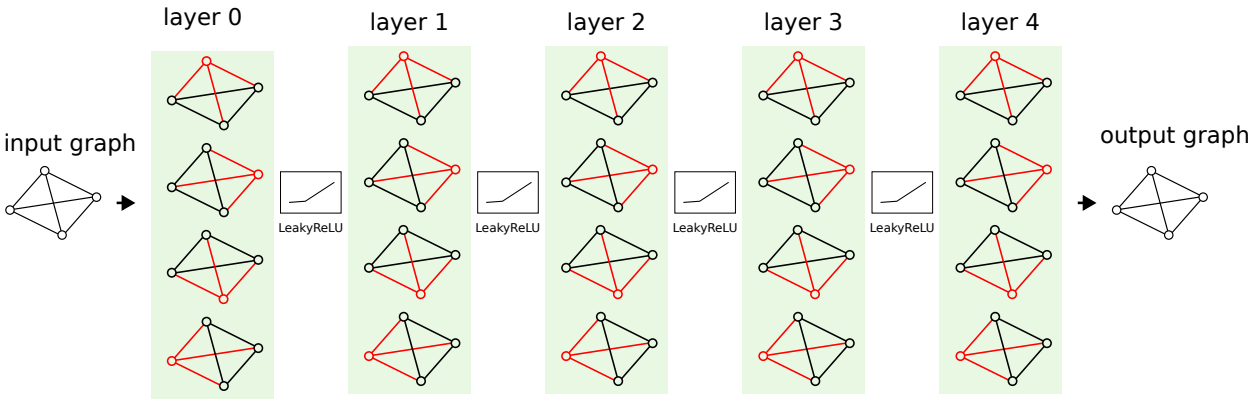

Figure 3.9: Graph neural network architecture. The model takes a graph of four nodes as input, transforming it with four hidden layers. The updating nodes are represented in red, showing that every layer updates all the nodes in the graph simultaneously. Graph convolutional layers are followed by LeakyReLU activation functions. An output layer with linear activation produces the model prediction. The characteristics of the nodes in the output graph represent the predicted polarizations.

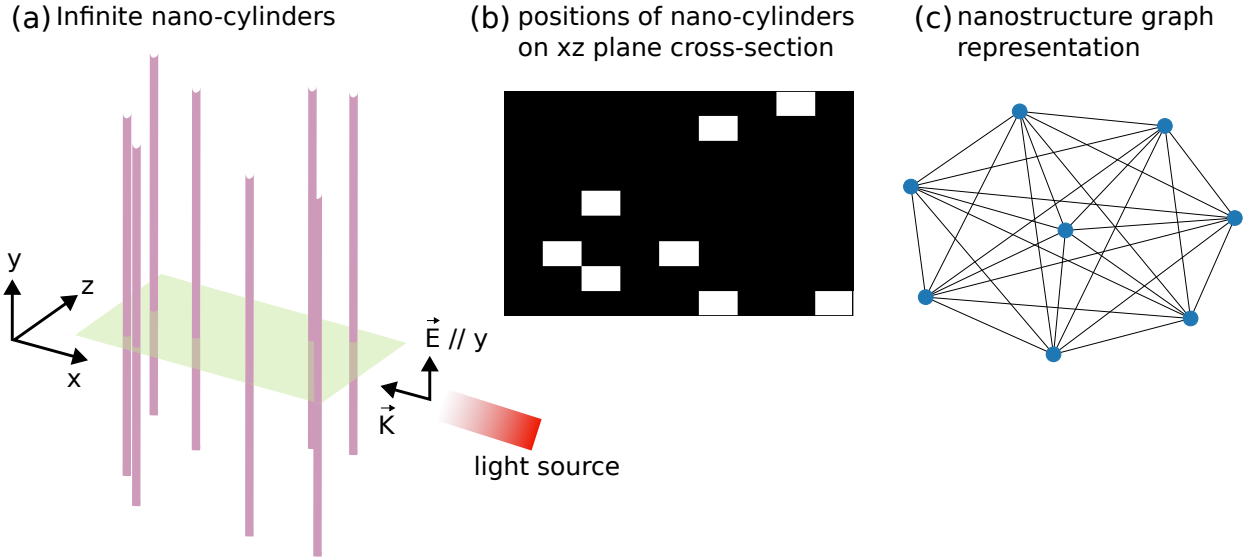

Figure 3.10: Nano-cylinders, horizontal section image and corresponding graph illustration. Illustration of (a) infinitely high nano-cylinders to which the light is directed from the side and the electric field $\vec{E}$ parallel to the cylinders on the y axis; their (b) horizontal section top view image showing the relative positions through white pixels and (c) the graph representation of the nanostructure where each nano-cylinder is encoded as a node.

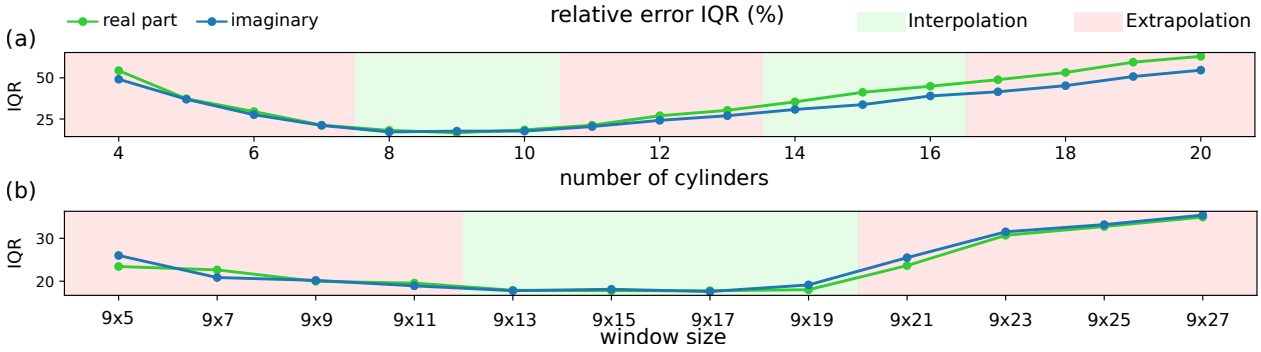

Figure 3.11: Multiscattering: relative error IQR. The GCN test is conducted by varying (a) the number of cylinders in a first step and then (b) the size of the window (range of particle positions). The GCN model is trained with sizes in the interpolation regime (green zones), it's also tested with graphs sizes out of the training range to test the extrapolation (red zones) ability. The green curve corresponds to the real parts of the polarization complex numbers and the blue one to the imaginary parts.

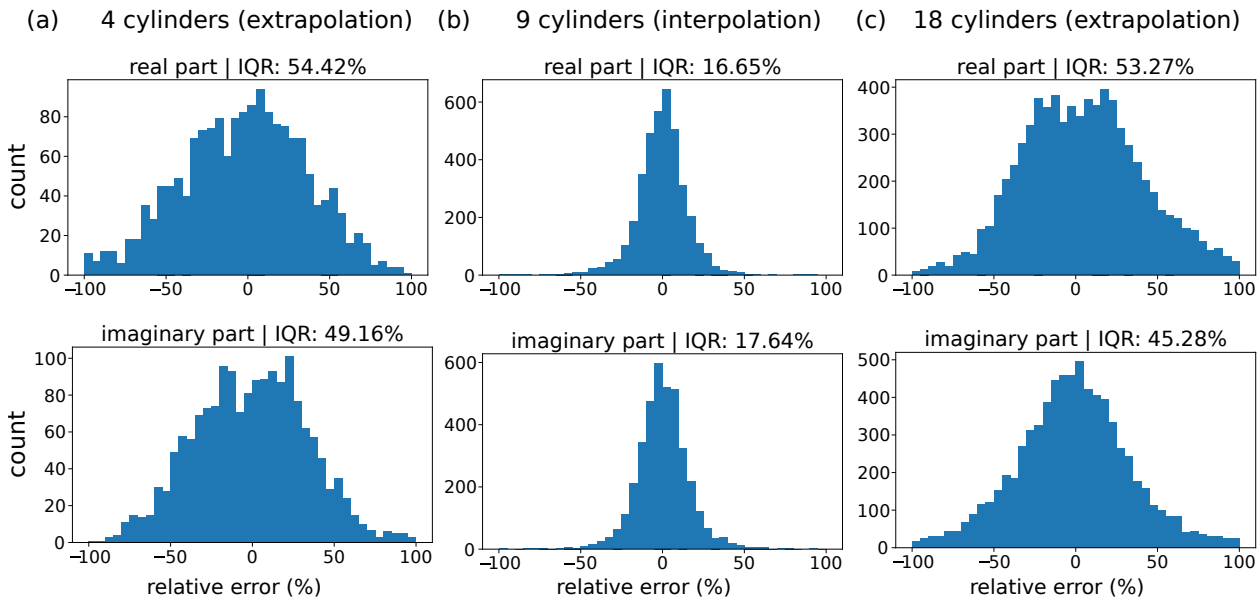

Figure 3.12: Histograms of the model test relative error in % on the number of cylinders change with a constant window size of 11×11 pixels, the window dimensions (in pixels) represent the space in which the cylinders are randomly distributed. Three examples of the relative errors between the GCN predictions and targets for (a) 4 cylinders, (b) 9 cylinders and (c) 18 cylinders, where 9 cylinders are interpolation whereas 4 and 18 cylinders are extrapolation cases. For each example, the top histogram represents the real part error and the bottom one is the imaginary part error. The interquartile range (IQR) is set as title on top of each histogram.

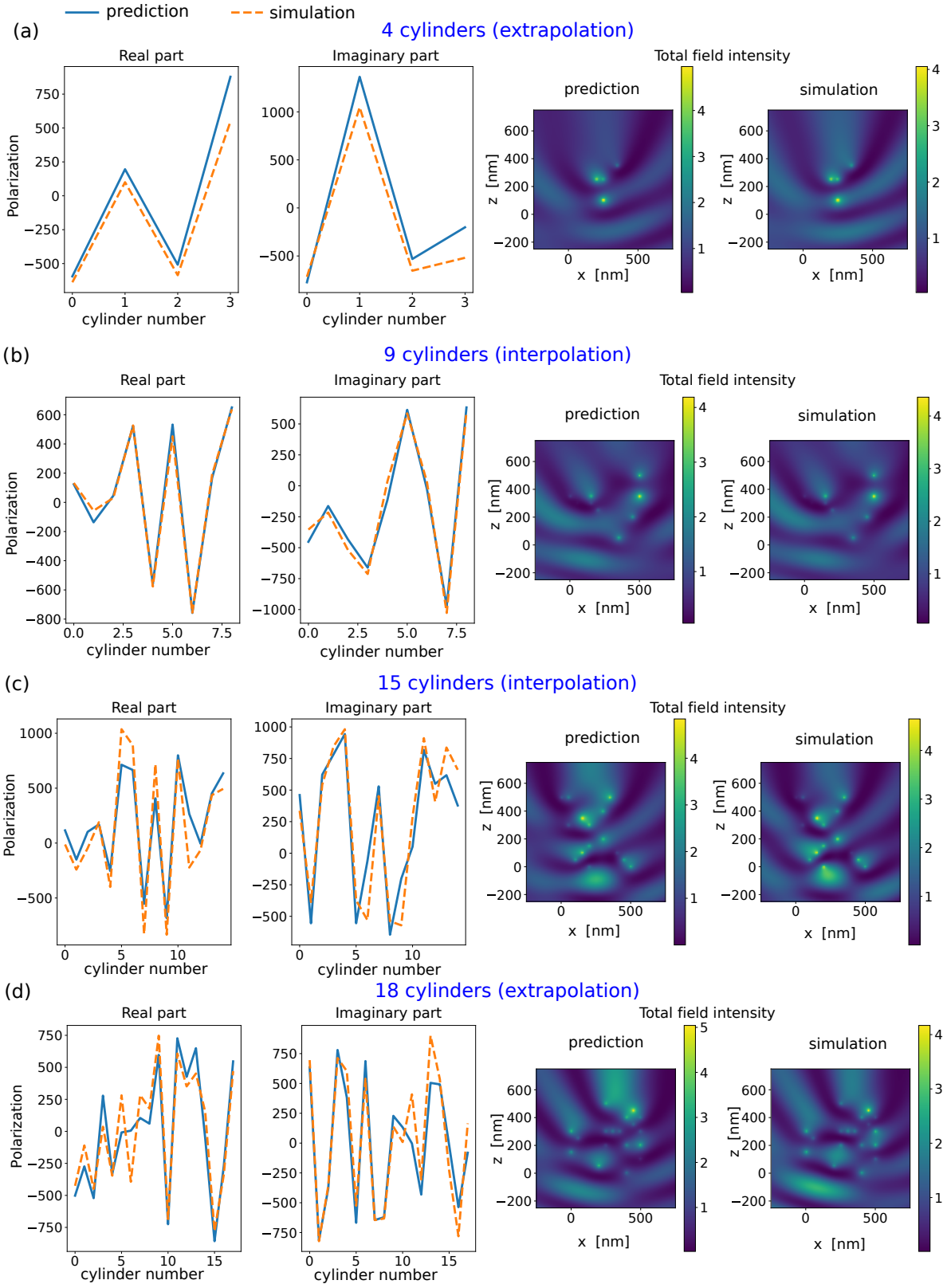

Figure 3.13: Randomly selected examples in the model predictions on the number of particles change problem. These examples are from (a) 4, (b) 9, (c) 15 and (d) 18 cylinders. For each example, real and imaginary parts are plotted for the prediction (blue curve) and simulation (orange dashed curve) as well as the scattering electric field maps using the predictions and simulations for re-propagation.

3.10a, the top view image of their representations, Figure 3.10b, where each nano-cylinder is modeled by a pixel (suitable for a regular CNN) and the corresponding graph, Figure 3.10c (suitable for a GNN). We describe the optical response in frequency domain, where the time dependence is harmonic and fields can be described with complex numbers as $E = \hat{E} \cdot e^{-i\omega t}$. The polarization is generally a vector $p = \alpha E_0$, where E_0 is the illumination's electric field vector and α is the polarizability tensor. We assume electric field along the direction of the infinite cylinder axis. This makes the problem scalar and p, α and E_0 are then just complex numbers. The goal is to predict the electric polarization of a random and complicated arrangement of nano-cylinders illuminated from the side. Between the cylinders, coupling effects as well as optical multi-scattering will occur, rendering the electric polarization very complex. Such problems are often used using coupled dipole methods [139], which become time consuming for large problems as their numerical complexity scales with N^3 (N is the number of particles). We want to predict the fields instead with a neural network. Since the number of scatterers and their positions are typically highly variable, a model that can work on flexible problem sizes with continuous input values is required, which matches perfectly with GCNs. The model takes as input the nano-cylinders' positions and the incident electric field evaluated at those positions to predict the electric polarization. Each node in the graph stands for a nano-cylinder and contains the values of the incident electric field in addition to the particle position (in the 2D top view image). It is important to note that the positions of the cylinders are processed like features by the Graph model and the GNN has initially no knowledge about the concept of an Euclidean coordinate space, therefore it needs to "understand" the meaning of these values implicitly from the correlations in the dataset. Every node is labeled by its electric polarization, which makes this task a node-level regression. We carry out two applications, in a first step we vary the number of cylinders distributed in a fixed 2D space of 11×11 pixels; in a second step, we vary the window size for a fixed number of cylinders (10). For each application we test the capacity of the model to predict in the field of its training but also its capacity to extrapolate. In the extrapolation tests, we predict the optical responses for structures with less/more cylinders than the training data structures for the first application and for smaller/larger windows in the second application of window size varying.

For the application of number of cylinders varying, the training and validation datasets contain graphs of 8, 9, 10, 14, 15 and 16 nodes, the number of graph nodes correspond to the number of cylinders. The test dataset includes graphs with different size compared to those used for training and validation (test with sizes from 4 to 20 cylinders). The objective of this experiment is to verify the capacity of the model to extrapolate on graphs with "unseen" sizes. As depicted in Figure 3.11a by the interquartile range (IQR) of the relative error between the predicted polarizations and the target ones, the model performs better, as might be expected, in the interpolation cases. However, we notice that the extrapolation close to interpolation zones is better handled by the model compared to the extrapolation at the outer border of the dataset range like the optical response predictions for 4 or 20 cylinders. The relative errors are normalized with the modulus of the target optical response for each particle. The statistics of 3 numbers of cylinders are shown in Figure 3.12 as examples, two extrapolation cases (4 and 18 cylinders) and one interpolation case (9 cylinders) showing, as anticipated, less error compared to extrapolation cases. The optical responses predicted by GCN on some interpolation (9 and 15 cylinders) and extrapolation (4 and 18 cylinders) cases are reported in Figure 3.13 for visual comparison with the target ones. Each example

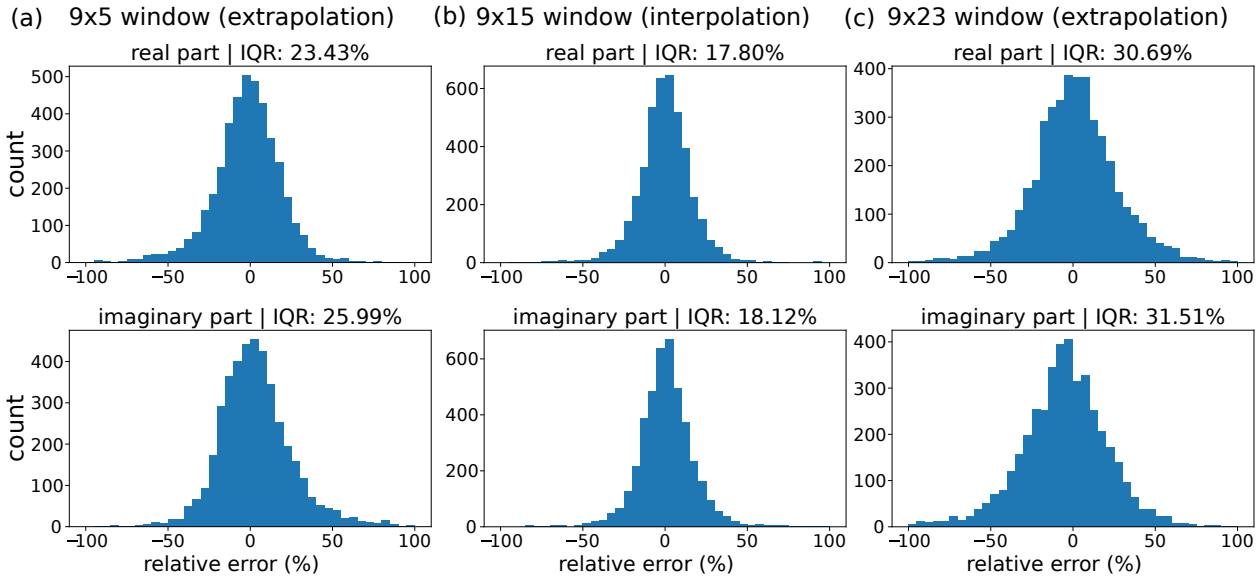

Figure 3.14: Histograms of the model test relative error in % on window size change with a constant cylinders number of 10. The window dimensions (in pixels) represent the space in which the cylinders are randomly distributed. Three examples of the relative errors between the GCN predictions and targets for (a) 9×5, (b) 9×15 and (c) 9×23, where 9×15 are interpolation whereas 9×5 and 9×23 are extrapolation cases. For each example, the top histogram represents the real part error and the bottom one is the imaginary part error. The interquartile range (IQR) is set as title on top of each histogram.

shows also the scattering electric field maps when using the predictions and simulations for re-propagation, the derived optical fields are quite accurate which shows that the quite large relative errors in the order of 10s of % are actually not so bad after all. We attribute this to the largest relative errors resulting from very small values of the polarizations (where the predictions are divided by small values). However, these particles with small polarization values do not contribute much to the overall scattering.

We repeated the same procedure by changing the window size, the training and validation datasets contain window sizes of 9×13, 9×15, 9×17 and 9×19 for a fixed number of 10 cylinders. The test dataset includes window sizes different from those used in training and validation going from 9×5 to 9×27 (changing the window size on the direction of light propagation). As depicted in Figure 3.11b by the interquartile range (IQR) of the relative error, the model shows better performance, without surprise, in the interpolation cases. The statistics of 3 window sizes are shown in Figure 3.14 as examples, two extrapolation cases (9×5 and 9×23) and one interpolation case (9×15) showing, as we would expect, less error compared to extrapolation cases. Randomly selected optical response predictions by the GCN on some interpolation (9×13 and 9×15) and extrapolation (9×5 and 9×27) zones are reported in Figure 3.15 for visual comparison with the target ones. Each example shows also the scattering electric field maps when using the predictions and simulations for re-propagation, the derived optical fields are again very accurate.

In these two applications, the model performs quite well in interpolation but less in extrapolation. We attribute this again to the fact that neural networks are typically strong in

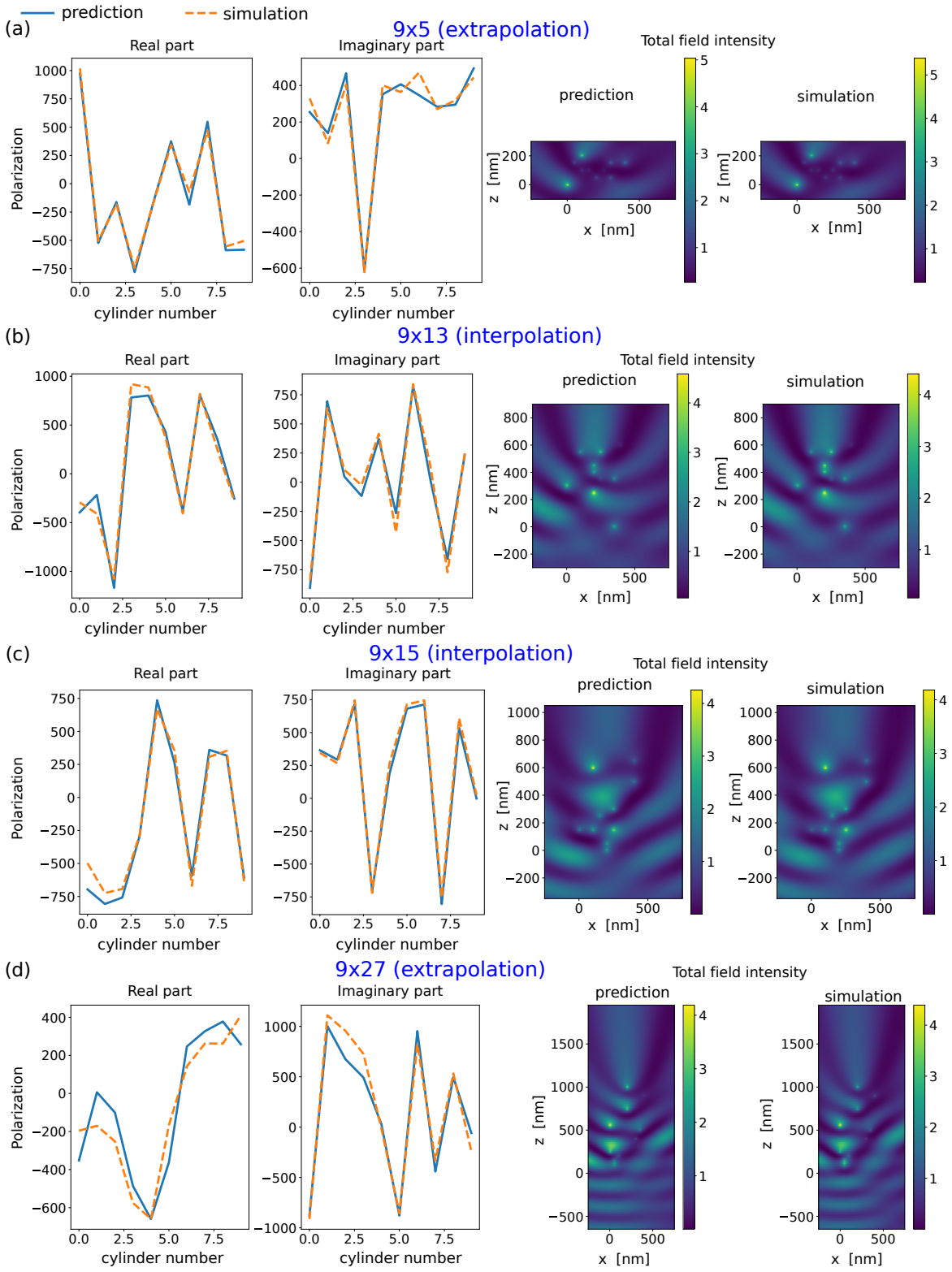

Figure 3.15: Randomly selected examples in the model predictions on the window sizes change problem. These examples are from (a) 9×5, (b) 9×13, (c) 9×15 and (d) 9×27. The window dimensions (in pixel) represent the space in which the cylinders are randomly distributed. For each example, real and imaginary parts are plotted for the prediction (blue curve) and simulation (orange dashed curve) as well as the scattering electric field maps using the predictions and simulations for re-propagation.

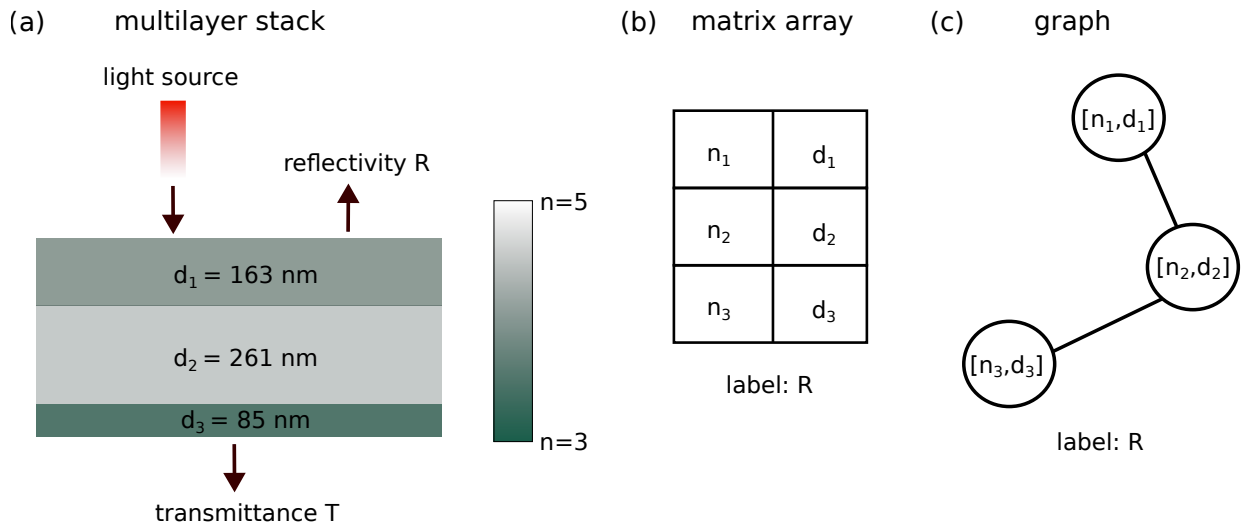

Figure 3.16: Multi-layer thin film stack, matrix array and the corresponding graph. (a) Illustration of the geometry showing a randomly selected layer stack. The light is directed towards a multi-layer stack constituted of layers of different thicknesses and refractive indices, the transmitted light is indicated by T whereas the reflected light is indicated by R. (b) Array representation of the structure with three rows and two columns, each row represents a layer from the stack in (a) and contains its thickness d and refractive index n. The whole matrix is labeled by the reflectivity R. (c) The representative graph of the matrix in (b). Each node stands for a row (layer) and holds a vector containing the thickness d as well as the refractive index n. The entire graph is labeled by its reflectivity R.

interpolation but weak in extrapolation. These tasks show the capacity of GNNs to work on graph-structured data with arbitrary size and connectivity schemes and thus suggest GNNs as a suitable tool for working on real-life data (like particles, social network, etc) without breaking their forms by making assumptions to fit them into regular neural networks. In the following, we switch the problem and test GNNs on a graph-level task by predicting the reflectivity spectrum of multi-layer stacks.

3.4.6 Reflectivity of multi-layer thin film stacks

We conduct an experiment using the same type of graph neural network with essentially identical configuration on a second problem, namely light reflection from dielectric thin-film multi-layer stacks. The multilayer structures are composed of dielectric materials and are illuminated at normal incidence. The first step is to set up the graph dataset, for the multi-layer stacks and differently from the nano-cylinders in the section above, the graphs look like chains. An illustration of a thin-film multilayer stack, as well as a representative matrix array and the corresponding graph are shown in Figure 3.16; Each node of the graph represents a layer and initially contains as features the values of the thickness d and the refractive index n. In contrast to the node-level prediction that we performed in the preceding subsection, here the whole layer stack (hence the full graph) is characterized by a reflectivity spectrum, therefore we now perform the so-called "graph-level regression". After four graph convolutional layers, the node representations are transformed into a 1D vector by *GlobalAvgPool*, an average pooling layer that pools a graph by computing the average of its node charac-

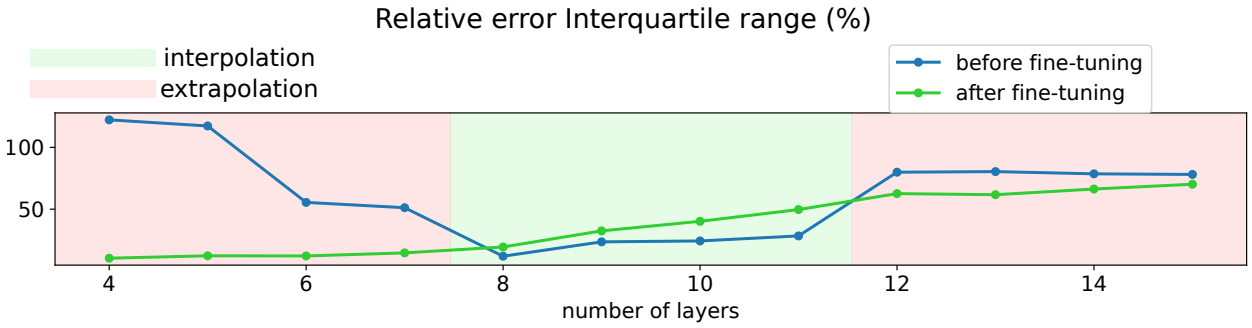

Figure 3.17: Relative error (%) interquartile range of the GCN. The graph convolutional model is trained on the data in the interpolation regime (green background) and then tested with data of multilayers from 4 to 20 layers. According to the IQR (blue curve) of the relative error, the model performs better in the interpolation regime. In a second step, the model is fine-tuned on data from interpolation and extrapolation regimes and the IQR of the relative error (green curve) shows that the model performs now better on data with generally smaller sizes.

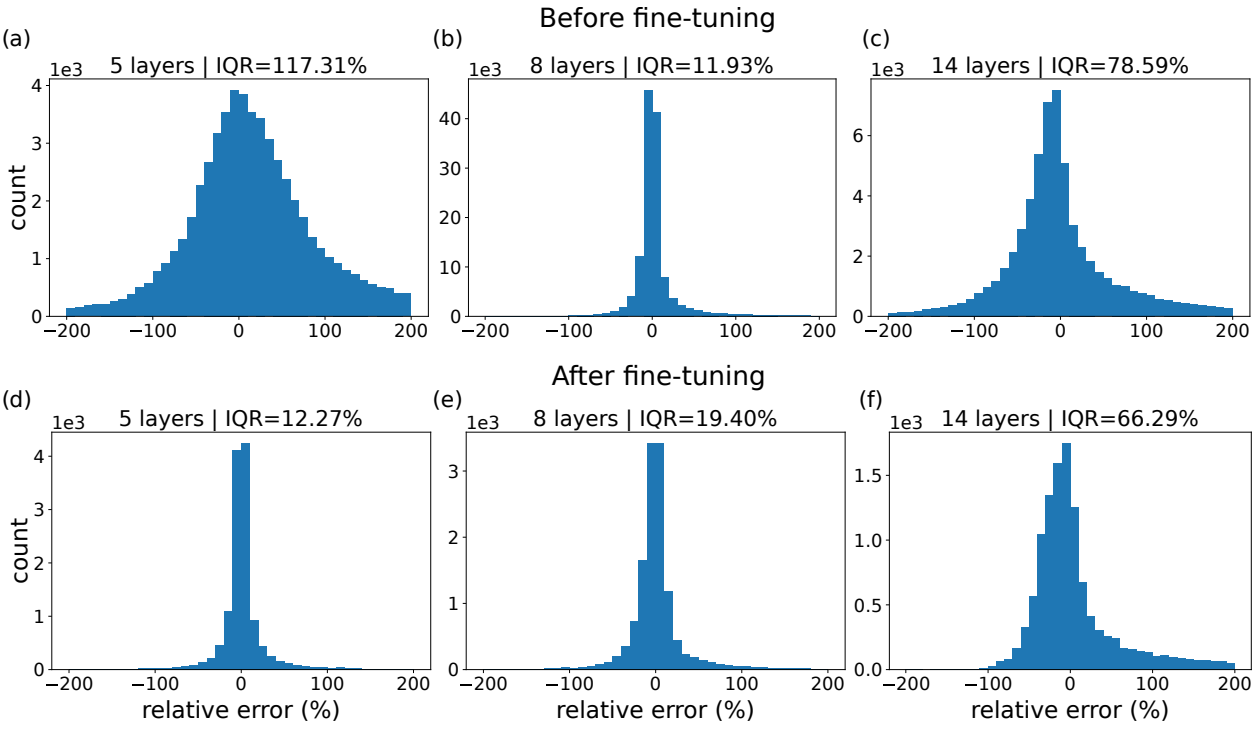

Figure 3.18: GCN statistics: before and after fine-tuning. Before fine-tuning, the model is tested on interpolation cases like multilayers with (b) 8 layers, where the results are better with IQR = 11.93, % and also tested with data from extrapolation, examples are (a) and (c) with 117.31% and 78.59%, respectively. In a second step, the model is fine-tuned on data from interpolation and extrapolation regimes, selected statistics show that the model has now better accuracy on structures with smaller number of layers as seen by comparing (d) 12.27%, (e) 19.4% and (f) 66.29%.

teristics. The pooled node representations are processed by a fully connected *Dense* keras layer of 64 neurons to output the reflectivity spectrum. The spektral module graph layers are fully compatible with keras layers. For this experiment, the data simulation is realised with *pyMoosh* [119].

The graph convolutional network is trained with stacks of 8 to 11 layers. That is presented as the interpolation zone in the model test. On top of that, the model is tested with multi-layers of sizes from 4 to 7 and from 12 to 15, in order to test the extrapolation capacity of the graph network. Figure 3.17 shows the trend of the relative error represented by the interquartile range in blue curve. The interpolation and extrapolation statistics are shown in Figure 3.18, expressed as histograms with the interquartile range at the top of each one to compare the relative error distribution while ignoring the outliers. As expected, the model shows better accuracy on interpolation regime. In order to visually examine the model accuracy, randomly selected cases of interpolation and extrapolation zones are shown in Figure 3.19. The GCN model demonstrates satisfying accuracy in interpolation regime even though the graph sizes varies. Concerning the extrapolation regime, closer to the interpolation regime, better is the accuracy. These results show the potential of graph neural networks to handle graph-structured data of arbitrary sizes and forms. Furthermore, to expand the model's field and make it work on the extrapolation data with more favourable outcomes, the so-called fine-tuning process is used by relaunching a training of few iterations using relatively small resources including parameter ranges not covered by the original dataset.

In general, fine-tuning involves taking a pre-trained neural network and making slight adjustments to its weights to adapt it to a specific task. Fine-tuning leverages the general features learned by a pre-trained network on a large dataset and refines them for more specialized applications, enhancing performance and reducing training time [140]. Mainly two approaches, namely the full fine-tuning and the partial fine-tuning, enable to fine-tune a pre-trained model on new data with the same physics than the initially used data. Full fine-tuning consists of adjusting all layers while partial fine-tuning involves freezing some of the early layers and only retraining the later layers [140]. Freezing layers means their weights remain unchanged during the fine-tuning training process. In large neural network models, retraining all layers is more flexible and can adapt to the nuances of the target task but requires more computational resources and a larger amount of labeled data; hence retraining only the last few layers is computationally efficient and requires little data when the initial and target tasks are quite similar.

In order to fit the GCN model on the extrapolation data, we have opted for the full fine-tuning approach because we are dealing with a lightweight model comprising only five trainable layers. The initial training has been performed on 25 000 samples per structure size whereas for the fine-tuning we used 2400 samples per structure size which is more than ten times less. In the same way than the initial training, the relative error trend is plotted in Figure 3.17 (green curve), the fine-tuning reduced the errors of all the extrapolation cases; however, the interpolation zones get slightly more error. Furthermore, we notice that the fine-tuning process benefited the most to the structure of smaller size (i.e. 4 to 7 layers), as also shown on the statistics in Figure 3.18d-f, but also on the randomly selectd predictions in Figure 3.20. It is the same looking at the interpolation regime of the initial training in Figure 3.17 (blue curve), the IQR has increasing trend. It is evident that the model performs well on

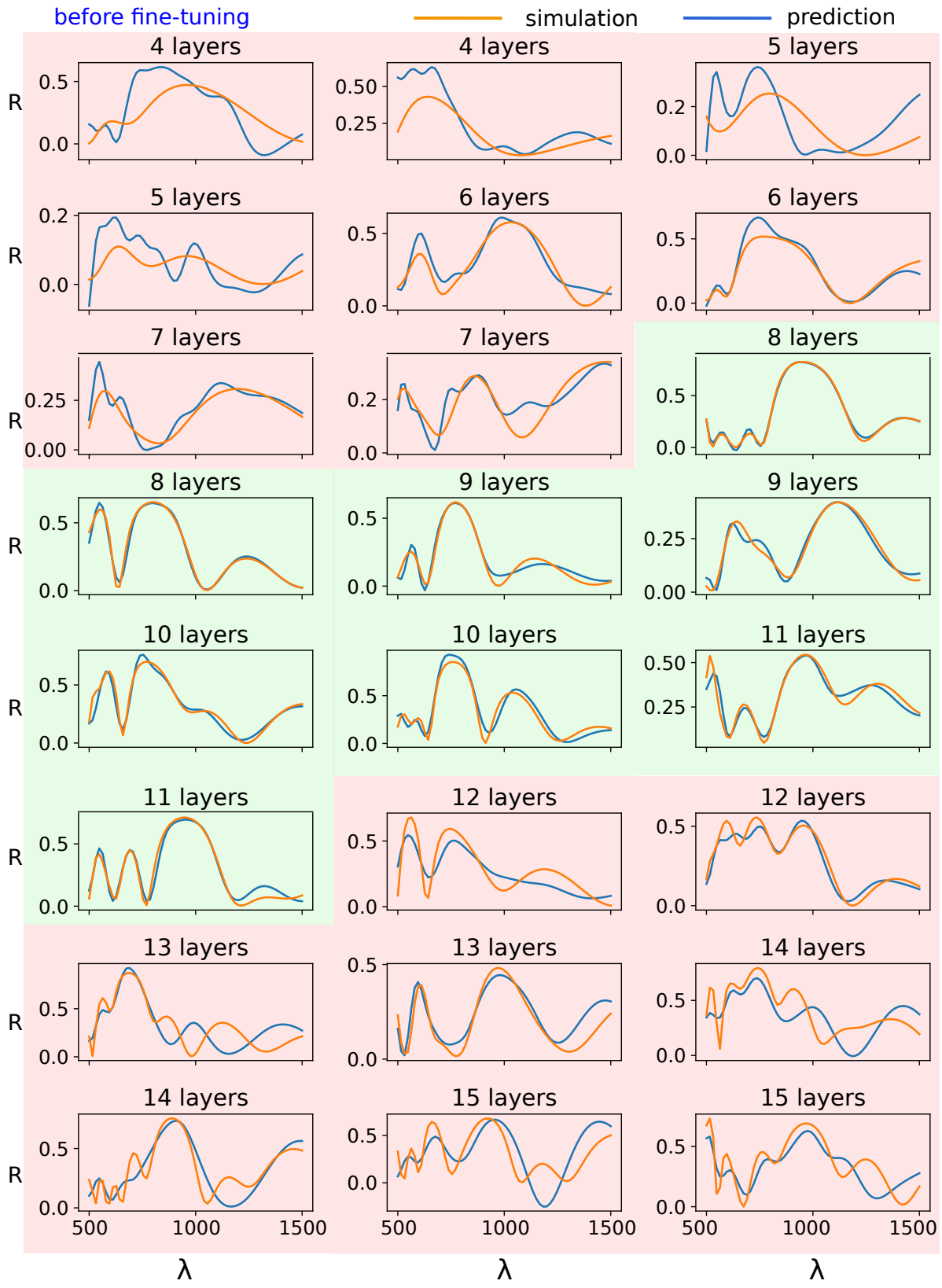

Figure 3.19: Randomly selected GCN predictions on multilayer reflectivity before fine-tuning in order to have an overview of the model accuracy. The model predicts quite well the sizes (8, 9, 10 and 11) on which it was trained. For data in extrapolation, closer to the interpolation regime, better is the accuracy.

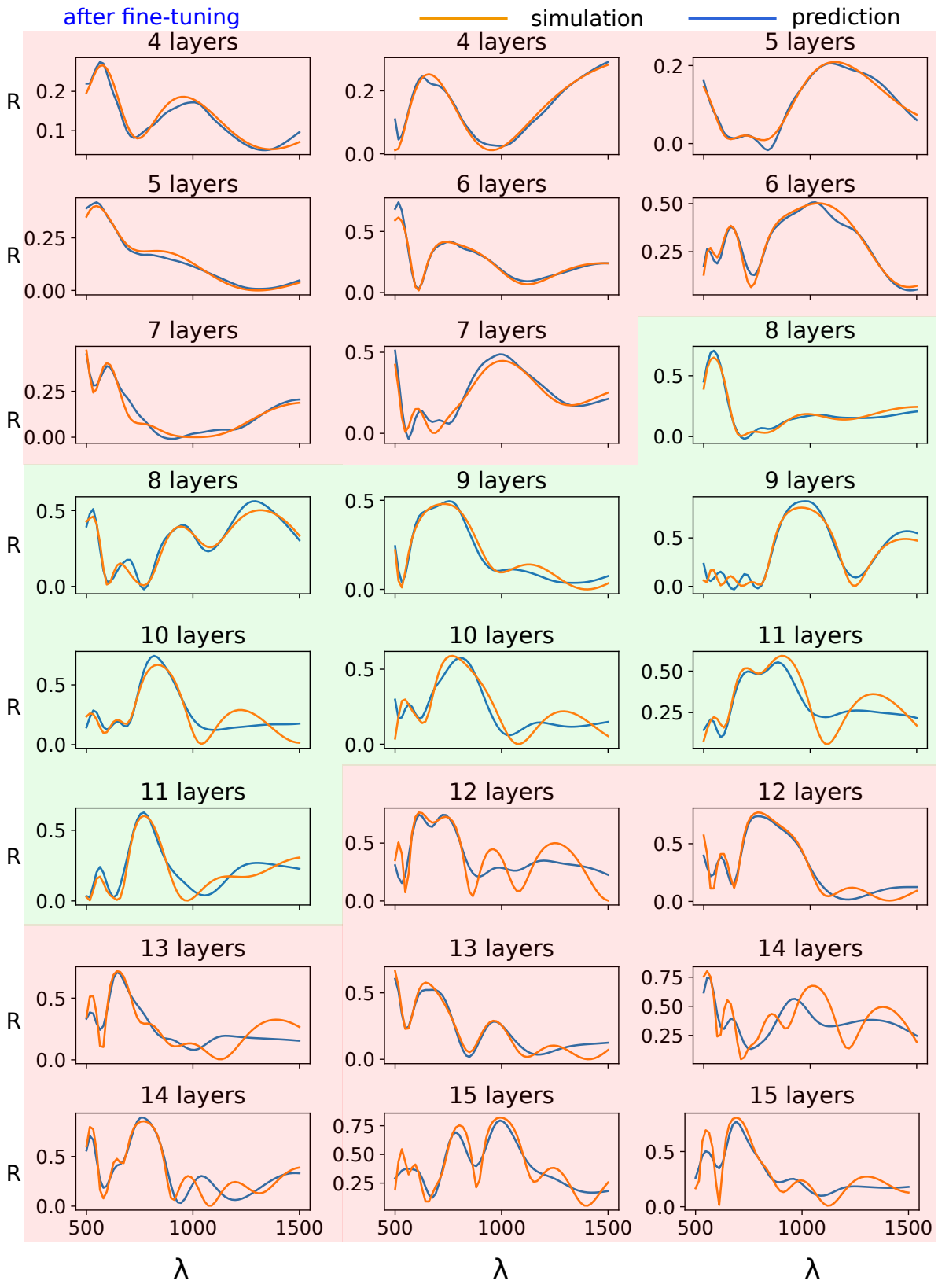

Figure 3.20: Randomly selected GCN predictions on multilayer reflectivity after fine-tuning in order to have an overview of the model accuracy. The predictions trend shows that the model has now higher performance on data with smaller sizes.

structures with relatively fewer layers in the field of training as their physics are simpler and responses less complex. This fine-tuning process demonstrates that it is possible to readapt a graph model (like every Deep Learning model in general) on new data with same physics than the training data without beginning the model training from scratch.

3.4.7 Conclusion

We proposed a graph convolutional network (GCN) as surrogate model for physics problems (and potentially other problems), that can then be used for further applications, e.g. inverse design. The performed proof of concept concerns the forward problem on the multiscattering of nanocylinders and the reflectivity of multilayer stacks. The models show good qualitative performance in both experiments. The extrapolation tests revealed that graph models can still work on graphs with sizes close to those in the training data. Nevertheless, it's possible to finetune the model on "unseen" data to extend its working field. The aim of these experiments is to show the potential of the graph neural networks to deal with structures of arbitrary sizes and complex forms. Finally, we fine-tuned the GCN model and shown that it reduces the errors, particularly in the extrapolation zones. The fact that it generally requires less data and computational power makes it a practical approach for applications with limited labeled data or computational resources. Further work is to improve the overall model accuracy and test more sophisticated applications.

3.5 Conclusion

In conclusion, we explained why ill-posed inverse design problem in nanophotonics cannot be solved with naive deep learning models that take the physical property as input and output the design parameters. Fortunately, some deep learning architectures are able to deal with inverse design. We presented Tandem networks and conditional variational autoencoder to solve the inverse design problem by the example of light reflectivity multilayer stacks. The Tandem network reaches one of the possible solutions while the conditional variational autoencoder learns to identify and address possible multiple solutions thanks to its latent space. These two architectures are both so called "one-shot". We also demonstrated an experiment on gradient-based iterative method, namely the Neural Adjoint method using a CNN as surrogate model. Because the NA iteratively uses gradients to optimize a solution, there is a elevated risk of diverging outside the interpolation regime to a failed extrapolated solution. Hence it is crucial to constrain the designs to the interpolation regime of the forward model. To this end, we demonstrated how a generative model like a WGAN can be used by learning a regularized re-parametrization of the designs, optimizing this new latent description instead and constraining this to the center region of the latent space, which corresponds to the interpolation regime.

However, it is possible that one has to deal with nanophotonic devices of arbitrary sizes and forms, for such scenario we proposed a graph convolutional surrogate model in order to handle such non-Euclidean data. We applied GCN on optical multiscattering in complex ensembles of nano-particles and optical reflection of multilayer thin film stacks. Finally, we demonstrated the extrapolation ability of such models as well as the possibility to fine-tune GCNs to improve their performance on regions outside of the parameter range of the original

training data, using only little data. Further work is to improve the model accuracy and extend the study into more sophisticated applications to benefit from the full potential of graph convolutional network.

General conclusion

In conclusion, in this thesis we demonstrated how neural networks can be used to characterize MBE crystal growth by processing images of RHEED patterns. Furthermore, we treated the inverse design problem of nanophotonics with generative models and proposed a surrogate graph neural network by representing nanophotonic structures as graphs.

In Chapter 1, we introduced the concept of deep learning, starting with the main building block, the artificial neuron. The latter takes inputs, then calculates the weighted sum, adds a bias and passes the result to an activation function that adds non-linearity to the neuron. An artificial neural network is a set of artificial neurons organized in layers. In the case of a multilayer perceptron, each neuron in a given layer is connected to the set of neurons preceding it. This type of network is designed to manage high-level features. There are also convolution neural networks, which have been particularly successful in image processing and generally in computer vision. Recurrent neural networks specialize in sequential data, while graph neural networks can handle graph-structure data. These models are trained with a stochastic gradient descent algorithm that improves the model step by step by updating the network parameters (weights and biases). Automatic differentiation tools provided by modern Deep Learning libraries such as keras and PyTorch facilitate this process. These neural networks are used to create complex architectures for specific tasks, for instance the autoencoder, which involves two networks: an encoder that compresses the data into a latent space and a decoder that attempts to reconstruct the original data from the latent space. For instance, after training, the encoder can be used as a data compressor. The variational autoencoder works with an encoder and a decoder like the regular autoencoder except that the latent space is a probability distribution characterized by a mean and a standard deviation provided by the encoder, the decoder now reconstructs from this probability distribution, enabling well-organized latent space and thus more sophisticated sample generation. Generative Adversarial Network (GAN) is known for its high quality image generation. This model is designed by training a generator and a discriminator in a competitive process where the generator attempts to produce images similar to those in the database while the discriminator's role is to distinguish the true images from the synthetic generated images. Finally, U-Net, an image segmentation network also involves an encoder-decoder architecture but with connections between these paths at corresponding level (same spatial dimensions).

In Chapter 2, we characterize RHEED patterns using deep learning models. Given that this application is not very widespread to date, we have carried out a state of the art review of a few studies that use machine learning methods such as PCA and deep learning. This is followed by the section taken from our published paper [35] on substrate deoxidation detection with an architecture involving two networks: an autoencoder that compresses images

and a CNN that classifies sequences of these compressed images. We studied the impact of sequence length by keeping the size of the latent space of the autoencoder constant (N =50) and vice versa with a constant sequence length (L =15). Classification is accurate from L =5 and N =10. We then classified the surface reconstructions (2×4) and $c(4 \times 4)$. Surface reconstruction consists of the rearrangement of surface atoms due to growth conditions such as temperature. This classification task is performed using a residual neural network. The last section of this chapter reports our work on Azimuthal RHEED construction using Deep Learning. Azimuthal RHEED consists of slices through the specular spot plotted as a function of the azimuthal angle and allows more advanced characterization with good resolution at low crystal rotation speeds (around 3-4 rpm). This task involves two steps, the detection of the specular spot and the regression of the crystal rotation angle. The specular spot detection is performed using a semantic segmentation model that takes a RHEED image as input and produces a binary mask with the same dimensions as the input image containing ones in the specular spot area and zeros elsewhere. The specular spot center of gravity coordinates are chosen as its position. The regression task of the azimuthal angle is performed using a residual neural network in order to determine the orientation of the crystal with respect to the incident electron beam. The challenge is that at these low speeds, the angle measurement is not precise. The sample holder is magnetically linked to the motor located outside the ultra-high vacuum growth chamber. This link is subject to clearance and friction at low speeds but is effective at high speeds (e.g. 12 rpm) thanks to the inertia. Our idea is therefore to train a model on data at 12 rpm and use it on data captured at 4 rpm. In tests, the model performs well on both 4 and 12 rpm data. The Azimuthal RHEED is then plotted using 4 rpm data.

In chapter 3, we focus on nanophotonics and in particular on the inverse design problem, which consists of determining a nanostructure from a target optical response. This is an ill-posed problem according to Hadamard's three conditions [106]: a problem is well-posed when a solution exists, that solution is unique and continuous. In order to circumvent this issue, numerical global optimization methods are used, which are generally expensive and slow. Deep learning techniques have been proposed, in particular generative models and these methods stand out mainly because of their execution speed. We have presented three methods, taken from our published article [13]. The first is the so-called Tandem architecture, which involves two networks. The first network takes the optical response as input to build a nanostructure and the second, a pre-trained forward model, predicts the optical response from this nanostructure, the training loss function compares then the input and the predicted optical responses, thus avoiding the problem of multisolutions. The second method is the conditional variational autoencoder, which generates nanostructures conditioned on the desired optical response. The final presented technique is the neural adjoint method, which optimizes a set of initial candidates and then selects the candidate that is closest to the desired optical response. This method uses a forward propagation neural network model to calculate the candidate responses and optimizes candidate architectures in a gradient-based optimization. It is important to avoid the extrapolation zone from the forward model, to this end, we trained a WGAN-GP with regularized latent space. At the end of the chapter, we propose a graph neural network surrogate model to structure design parameters in the form of graphs, which allows the processing of nanostructures with arbitrary sizes and forms. Two tasks are studied, light multiscattering from nanocylinders and the reflectivity of multilayer stacks. For each application, we test the interpolation and extrapolation capabilities of the

graph convolutional network. Finally, we finish with a fine-tuning for the application of the multilayer stack reflectivity in order to include extrapolation data in the graph neural network training field.

Essentially, we developed Deep Learning techniques for monitoring MBE crystal growth using only raw RHEED images, in particular for detecting the deoxidation moment of the substrate, the classification of the surface reconstructions and the azimuthal RHEED plotting involving the tracking of the specular spot position and the determination of the crystal rotation angle with respect to the electron beam in the case of GaAs. These techniques pave the way for the use of AI and in particular deep learning in the characterization of crystal growth. They could also be used for other tasks such as determining the proportions of different materials on the surface of a crystal. Deep Learning-based crystal growth monitoring could be pushed further by mixing RHEED patterns with data from ex-situ characterization techniques such as X-Ray Diffraction (XRD) and thus make it possible the extraction of information that are not yet accessible in-situ. In a second application, we have demonstrated the capacity of these models to address the inverse problem in nanophotonics using generative models such as Tandem, cVAE and neural adjoint method. Furthermore, we proposed a graph neural network that handles the direct problem, allowing nanostructures to be represented in the form of graphs in non-Euclidean space. This approach paves the way for the development of neural network models for very large nano-optical devices such as metasurfaces.

In addition, this work demonstrates the significant potential of deep learning in addressing scientific challenges such as crystal growth monitoring and nanophotonics inverse design. However, it also highlights the complexity involved in adapting this approach to different problems. For each new problem, a tailored deep learning strategy is required.

Appendix

PyMoosh [141]

PyMoosh (Python-based Multilayer optics optimization and simulation hub) is a simulation library designed to provide a comprehensive set of numerical tools allowing the computation of essentially all optical characteristics of multilayer structures such as reflectivity and transmittance. It is the python version of Moosh, which was implemented in Octave/MATLAB [142]. More details on the theoretical basis at the core of PyMoosh can be found in [141].

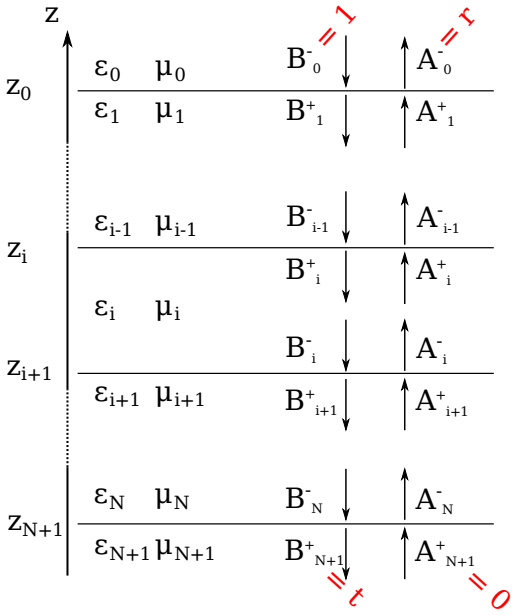

Figure 3.21: Multilayer structure and definition of important variables. The light is considered coming from the top of the image. ϵ_i and μ_i are, respectively, the permittivity and the permeability of layer i. A_i (resp. B_i) are defined as the amplitude of the upward propagating (resp. downward propagating) field in layer i and the + (resp. -) superscript indicates that the field value is taken at the top (resp. bottom) of layer i. Reprinted from [141].

The optical response of the multilayer structures considered in this thesis is the reflectivity R. In PyMoosh, each layer is characterized by its relative permittivity ϵ_r and permeability μ_r as illustrated in Figure 3.21. There are different ways to eliminate all the unknowns to obtain the reflection or the transmission coefficients. To this end, the library implements five different approaches: transfer and scattering matrices, as well as Abelès, Dirichlet-to-Neumann (DtN) and Admittance Formalisms. The scattering matrix formalism is the most accurate,

but also the computationally most expensive one. For this reason, the computation of the reflectivity for the multilayer structures in this thesis is performed based on the scattering matrices method.

The S-matrices method links the incoming fields to the outgoing fields for each interface and layer through a matrix. Iteratively applying this on the layer stack leads to the scattering matrix S of the whole structure:

$$\begin{pmatrix} A_0^- \\ B_{N+1}^+ \end{pmatrix} = S \begin{pmatrix} B_0^- \\ A_{N+1}^+ \end{pmatrix}$$

where S is a (2×2) matrix typically noted as $S = \begin{bmatrix} S_{00} & S_{01} \\ S_{10} & S_{11} \end{bmatrix}$

The complex reflection and transmission coefficient can be read directly, knowing $B_0^- = 1$, $A_{N+1}^+ = 0$: $r = S_{00}$, $t = S_{10}$. The reflectivity R and the transmittance T are finally given by the absolute squares: $R = |r|^2$ and $T = |t|^2$.

PyGDM [120, 121]

PyGDM is a Python-based computational toolkit designed for conducting electro-dynamical simulations in nano-optics. It leverages the Green Dyadic Method (GDM), a numerical volume integral approach for solving Maxwell's equations in systems with arbitrary shapes and material properties making PyGDM suitable for analyzing the interaction of light with individual nanostructures. A key feature of PyGDM is its ability to compute light scattering by nanostructures with a wide range of geometries. The toolkit enables to model nanostructures placed in a homogeneous environment and to calculate their scattering cross-sections under different illumination conditions.

To simulate nanostructures, PyGDM discretizes arbitrary three-dimensional geometries into smaller subunits, small enough such that a dipolar approximation for the optical response of each subunit is justified. It computes the scattered electromagnetic fields by solving a system of linear equations that describe the dipolar interactions between all subunits within the structure. This discretization approach ensures flexibility in handling diverse shapes and material configurations.

PyGDM is designed for computational efficiency, enabling the rapid simulation of complex structures and geometries. Moreover, the toolkit integrates well with Python scientific libraries, allowing for customized data analysis, post-processing and visualization. More details on the theoretical basis as well as a list of the functionalities available in PyGDM can be found in [121].

Bibliography

[1] Oswald Campesato. *Artificial intelligence, machine learning, and deep learning*. Mercury Learning and Information, 2020.

[2] Athanasios Voulodimos, Nikolaos Doulamis, Anastasios Doulamis, and Eftychios Protopapadakis. Deep learning for computer vision: A brief review. *Computational intelligence and neuroscience*, 2018(1):7068349, 2018.

[3] Daniel W Otter, Julian R Medina, and Jugal K Kalita. A survey of the usages of deep learning for natural language processing. *IEEE transactions on neural networks and learning systems*, 32(2):604–624, 2020.

[4] Li Deng, Geoffrey Hinton, and Brian Kingsbury. New types of deep neural network learning for speech recognition and related applications: An overview. In *2013 IEEE international conference on acoustics, speech and signal processing*, pages 8599–8603. IEEE, 2013.

[5] Meghavi Rana and Megha Bhushan. Machine learning and deep learning approach for medical image analysis: diagnosis to detection. *Multimedia Tools and Applications*, 82(17):26731–26769, 2023.

[6] Sampo Kuutti, Richard Bowden, Yaochu Jin, Phil Barber, and Saber Fallah. A survey of deep learning applications to autonomous vehicle control. *IEEE Transactions on Intelligent Transportation Systems*, 22(2):712–733, 2020.

[7] Erfan Khoram, Zhicheng Wu, Yurui Qu, Ming Zhou, and Zongfu Yu. Graph neural networks for metasurface modeling. *ACS Photonics*, 10(4):892–899, 2022.

[8] Tharindu Kaluarachchi, Andrew Reis, and Suranga Nanayakkara. A review of recent deep learning approaches in human-centered machine learning. *Sensors*, 21(7):2514, 2021.

[9] Leonardo Noriega. Multilayer perceptron tutorial. *School of Computing. Staffordshire University*, 4(5):444, 2005.

[10] Fionn Murtagh. Multilayer perceptrons for classification and regression. *Neurocomputing*, 2(5-6):183–197, 1991.

[11] Andrea Apicella, Francesco Donnarumma, Francesco Isgrò, and Roberto Prevete. A survey on modern trainable activation functions. *Neural Networks*, 138:14–32, 2021.

[12] Sagar Sharma, Simone Sharma, and Anidhya Athaiya. Activation functions in neural networks. *Towards Data Sci*, 6(12):310–316, 2017.

[13] Abdourahman Khaireh-Walieh, Denis Langevin, Pauline Bennet, Olivier Teytaud, Antoine Moreau, and Peter R Wiecha. A newcomer's guide to deep learning for inverse design in nano-photonics. *arXiv preprint arXiv:2307.08618*, 2023.

[14] Hong Hui Tan and King Hann Lim. Vanishing gradient mitigation with deep learning neural network optimization. In *2019 7th international conference on smart computing & communications (ICSCC)*, pages 1–4. IEEE, 2019.

[15] Andrinandrasana David Rasamoelina, Fouzia Adjailia, and Peter Sinčák. A review of activation function for artificial neural network. In *2020 IEEE 18th World Symposium on Applied Machine Intelligence and Informatics (SAMI)*, pages 281–286. IEEE, 2020.

[16] Stamatis Mastromichalakis. Alrelu: A different approach on leaky relu activation function to improve neural networks performance. *arXiv preprint arXiv:2012.07564*, 2020.

[17] Matt W Gardner and SR Dorling. Artificial neural networks (the multilayer perceptron)—a review of applications in the atmospheric sciences. *Atmospheric environment*, 32(14-15):2627–2636, 1998.

[18] Jiuxiang Gu, Zhenhua Wang, Jason Kuen, Lianyang Ma, Amir Shahroudy, Bing Shuai, Ting Liu, Xingxing Wang, Gang Wang, Jianfei Cai, et al. Recent advances in convolutional neural networks. *Pattern recognition*, 77:354–377, 2018.

[19] Zewen Li, Fan Liu, Wenjie Yang, Shouheng Peng, and Jun Zhou. A survey of convolutional neural networks: analysis, applications, and prospects. *IEEE transactions on neural networks and learning systems*, 2021.

[20] Hojjat Salehinejad, Sharan Sankar, Joseph Barfett, Errol Colak, and Shahrokh Valaee. Recent advances in recurrent neural networks. *arXiv preprint arXiv:1801.01078*, 2017.

[21] Larry R Medsker and LC Jain. Recurrent neural networks. *Design and Applications*, 5(64-67):2, 2001.

[22] Timothy P Lillicrap and Adam Santoro. Backpropagation through time and the brain. *Current opinion in neurobiology*, 55:82–89, 2019.

[23] Jie Zhou, Ganqu Cui, Shengding Hu, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, Lifeng Wang, Changcheng Li, and Maosong Sun. Graph neural networks: A review of methods and applications. *AI open*, 1:57–81, 2020.

[24] Alice Moallemy-Oureh, Silvia Beddar-Wiesing, Rüdiger Nather, and Josephine M Thomas. Fdgnn: Fully dynamic graph neural network. *arXiv preprint arXiv:2206.03469*, 2022.

[25] Jianan Zhao, Xiao Wang, Chuan Shi, Binbin Hu, Guojie Song, and Yanfang Ye. Heterogeneous graph structure learning for graph neural networks. In *Proceedings of the AAAI conference on artificial intelligence*, volume 35, pages 4697–4705, 2021.

[26] Yunchong Song, Chenghu Zhou, Xinbing Wang, and Zhouhan Lin. Ordered gnn: Ordering message passing to deal with heterophily and over-smoothing. *arXiv preprint arXiv:2302.01524*, 2023.

[27] Zijun Zhang. Improved adam optimizer for deep neural networks. In *2018 IEEE/ACM 26th international symposium on quality of service (IWQoS)*, pages 1–2. Ieee, 2018.

[28] Meenal V Narkhede, Prashant P Bartakke, and Mukul S Sutaone. A review on weight initialization strategies for neural networks. *Artificial intelligence review*, 55(1):291–322, 2022.

[29] François Chollet et al. keras, 2015.

[30] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. 2017.

[31] Francois Chollet. *Deep learning with Python*. Simon and Schuster, 2021.

[32] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[33] Johnson Kolluri, Vinay Kumar Kotte, MSB Phridviraj, and Shaik Razia. Reducing overfitting problem in machine learning using novel l1/4 regularization method. In *2020 4th International Conference on Trends in Electronics and Informatics (ICOEI)(48184)*, pages 934–938. IEEE, 2020.

[34] Chathurdara Sri Nadith Pathirage, Jun Li, Ling Li, Hong Hao, Wanquan Liu, and Pinghe Ni. Structural damage identification based on autoencoder neural networks and deep learning. *Engineering structures*, 172:13–28, 2018.

[35] Abdourahman Khaireh-Walieh, Alexandre Arnoult, Sébastien Plissard, and Peter R Wiecha. Monitoring mbe substrate deoxidation via rheed image-sequence analysis by deep learning. *Crystal Growth & Design*, 23(2):892–898, 2023.

[36] Carl Doersch. Tutorial on variational autoencoders. *arXiv preprint arXiv:1606.05908*, 2016.

[37] Huajie Shao, Shuochao Yao, Dachun Sun, Aston Zhang, Shengzhong Liu, Dongxin Liu, Jun Wang, and Tarek Abdelzaher. Controlvae: Controllable variational autoencoder. In *International conference on machine learning*, pages 8655–8664. PMLR, 2020.

[38] Mikhail Figurnov, Shakir Mohamed, and Andriy Mnih. Implicit reparameterization gradients. *Advances in neural information processing systems*, 31, 2018.

[39] Chen Zhang, Riccardo Barbano, and Bangti Jin. Conditional variational autoencoder for learned image reconstruction. *Computation*, 9(11):114, 2021.

[40] Antonia Creswell, Tom White, Vincent Dumoulin, Kai Arulkumaran, Biswa Sengupta, and Anil A Bharath. Generative adversarial networks: An overview. *IEEE signal processing magazine*, 35(1):53–65, 2018.

[41] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks. *Communications of the ACM*, 63(11):139–144, 2020.

[42] Nahian Siddique, Sidike Paheding, Colin P Elkin, and Vijay Devabhaktuni. U-net and its variants for medical image segmentation: A review of theory and applications. *Ieee Access*, 9:82031–82057, 2021.

[43] Zongwei Zhou, Md Mahfuzur Rahman Siddiquee, Nima Tajbakhsh, and Jianming Liang. Unet++: Redesigning skip connections to exploit multiscale features in image segmentation. *IEEE transactions on medical imaging*, 39(6):1856–1867, 2019.

[44] Matthew Brahlek, Jason Lapano, and Joon Sue Lee. Topological materials by molecular beam epitaxy. *Journal of Applied Physics*, 128(21), 2020.

[45] BA Joyce. Molecular beam epitaxy. *Reports on Progress in Physics*, 48(12):1637, 1985.

[46] William Nunn, Tristan K Truttmann, and Bharat Jalan. A review of molecular-beam epitaxy of wide bandgap complex oxide semiconductors. *Journal of materials research*, pages 1–19, 2021.

[47] Alfred Y Cho. How molecular beam epitaxy (mbe) began and its projection into the future. *Journal of Crystal Growth*, 201:1–7, 1999.

[48] Shozo Ino. Some new techniques in reflection high energy electron diffraction (rheed) application to surface structure studies. *Japanese Journal of Applied Physics*, 16(6):891, 1977.

[49] Yoshimi Horio, Yasuyuki Hashimoto, and Ayahiko Ichimiya. A new type of rheed apparatus equipped with an energy filter. *Applied surface science*, 100:292–296, 1996.

[50] Wolfgang Braun. *Applied RHEED: reflection high-energy electron diffraction during crystal growth*, volume 154. Springer Science & Business Media, 1999.

[51] Ayahiko Ichimiya and Philip I Cohen. *Reflection high-energy electron diffraction*. Cambridge University Press, 2004.

[52] Ugo Valdrè. *Surface and Interface Characterization by Electron Optical Methods*, volume 16. Springer Science & Business Media, 2013.

[53] Janghyun Jo, Youngbin Tchoe, Gyu-Chul Yi, and Miyoung Kim. Real-time characterization using in situ rheed transmission mode and tem for investigation of the growth behaviour of nanomaterials. *Scientific reports*, 8(1):1694, 2018.

[54] Claes Thelander, Philippe Caroff, Sébastien Plissard, and Kimberly A Dick. Electrical properties of inas1- xsbx and insb nanowires grown by molecular beam epitaxy. *Applied Physics Letters*, 100(23), 2012.

[55] B Daudin, G Feuillet, J Hübner, Y Samson, F Widmann, A Philippe, C Bru-Chevallier, G Guillot, E Bustarret, G Bentoumi, et al. How to grow cubic gan with low hexagonal phase content on (001) sic by molecular beam epitaxy. *Journal of Applied Physics*, 84(4):2295–2300, 1998.

[56] Akihiro Ohtake, Takaaki Mano, and Yoshiki Sakuma. Strain relaxation in inas heteroepitaxy on lattice-mismatched substrates. *Scientific Reports*, 10(1):4606, 2020.

[57] Markus Ringnér. What is principal component analysis? *Nature biotechnology*, 26(3):303–304, 2008.

[58] Sidharth Prasad Mishra, Uttam Sarkar, Subhash Taraphder, Sanjay Datta, D Swain, Reshma Saikhom, Sasmita Panda, and Menalsh Laishram. Multivariate statistical data analysis-principal component analysis (pca). *International Journal of Livestock Research*, 7(5):60–78, 2017.

[59] Lindsay I Smith. A tutorial on principal components analysis. 2002.

[60] Hervé Abdi. Singular value decomposition (svd) and generalized singular value decomposition. *Encyclopedia of measurement and statistics*, 907:912, 2007.

[61] Kenneth Lange and Kenneth Lange. Singular value decomposition. *Numerical analysis for statisticians*, pages 129–142, 2010.

[62] Carlos Oscar Sánchez Sorzano, Javier Vargas, and A Pascual Montano. A survey of dimensionality reduction techniques. *arXiv preprint arXiv:1403.2877*, 2014.

[63] Jinkwan Kwoen and Yasuhiko Arakawa. Classification of in situ reflection high energy electron diffraction images by principal component analysis. *Japanese Journal of Applied Physics*, 60(SB):SBBK03, 2021.

[64] Rama K Vasudevan, Alexander Tselev, Arthur P Baddorf, and Sergei V Kalinin. Big-data reflection high energy electron diffraction analysis for understanding epitaxial film growth processes. *ACS nano*, 8(10):10899–10908, 2014.

[65] Hyuk Jin Kim, Minsu Chong, Tae Gyu Rhee, Yeong Gwang Khim, Min-Hyoung Jung, Young-Min Kim, Hu Young Jeong, Byoung Ki Choi, and Young Jun Chang. Machine-learning-assisted analysis of transition metal dichalcogenide thin-film growth. *Nano Convergence*, 10(1):10, 2023.

[66] JB Phipps. Dendrogram topology. *Systematic zoology*, 20(3):306–308, 1971.

[67] Masaki Nakano, Yue Wang, Yuta Kashiwabara, Hideki Matsuoka, and Yoshihiro Iwasa. Layer-by-layer epitaxial growth of scalable wse2 on sapphire by molecular beam epitaxy. *Nano letters*, 17(9):5595–5599, 2017.

[68] Jinkwan Kwoen and Yasuhiko Arakawa. Classification of reflection high-energy electron diffraction pattern using machine learning. *Crystal Growth & Design*, 20(8):5289–5293, 2020.

[69] Jinkwan Kwoen and Yasuhiko Arakawa. Multiclass classification of reflection high-energy electron diffraction patterns using deep learning. *Journal of Crystal Growth*, 593:126780, 2022.

[70] Haotong Liang, Valentin Stanev, Aaron Gilad Kusne, Yuto Tsukahara, Kaito Ito, Ryota Takahashi, Mikk Lippmaa, and Ichiro Takeuchi. Application of machine learning to reflection high-energy electron diffraction images for automated structural phase mapping. *Physical Review Materials*, 6(6):063805, 2022.

[71] Hayat Khan, Aditya S Yerramilli, Adrien D'Oliveira, Terry L Alford, Daria C Boffito, and Gregory S Patience. Experimental methods in chemical engineering: X-ray diffraction spectroscopy—xrd. *The Canadian journal of chemical engineering*, 98(6):1255–1266, 2020.

[72] F Bastiman and AG Cullis. Gaas (0 0 1) planarization after conventional oxide removal utilising self-governed inas qd site selection. *Applied surface science*, 256(13):4269–4271, 2010.

[73] AJ SpringThorpe, SJ Ingrey, B Emmerstorfer, P Mandeville, and WT Moore. Measurement of gaas surface oxide desorption temperatures. *Applied physics letters*, 50(2):77–79, 1987.

[74] M Abadi, A Agarwal, P Barham, et al. Tensorflow: Large-scale machine learning on heterogeneous systems, v1. 14, 2015.

[75] Quentin Fournier and Daniel Aloise. Empirical comparison between autoencoders and traditional dimensionality reduction methods. In *2019 IEEE Second International Conference on Artificial Intelligence and Knowledge Engineering (AIKE)*, pages 211–214. IEEE, 2019.

[76] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[77] Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, and Alexander Alemi. Inception-v4, inception-resnet and the impact of residual connections on learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 31, 2017.

[78] Evgeni Penev, Peter Kratzer, and Matthias Scheffler. Atomic structure of the g a a s (001)-c (4× 4) surface: First-principles evidence<? format?> for diversity of heterodimer motifs. *Physical review letters*, 93(14):146102, 2004.

[79] Akihiro Ohtake. Surface reconstructions on gaas (001). *Surface Science Reports*, 63(7):295–327, 2008.

[80] PK Larsen, PJ Dobson, JH Neave, BA Joyce, B Bölger, and J Zhang. Dynamic effects in rheed from mbe grown gaas (001) surfaces. *Surface science*, 169(1):176–196, 1986.

[81] F Jona. Observations of "clean" surfaces of si, ge, and gaas by low-energy electron diffraction. *IBM Journal of research and development*, 9(5):375–387, 1965.

[82] BA Joyce, JH Neave, Pl J Dobson, and PK Larsen. Analysis of reflection high-energy electron-diffraction data from reconstructed semiconductor surfaces. *Physical Review B*, 29(2):814, 1984.

[83] AY Cho. Gaas epitaxy by a molecular beam method: observations of surface structure on the (001) face. *Journal of Applied Physics*, 42(5):2074–2081, 1971.

[84] LL Chang, L Esaki, WE Howard, and R Ludeke. The growth of a gaas–gaalas superlattice. *Journal of Vacuum science and Technology*, 10(1):11–16, 1973.

[85] LL Chang, L Esaki, WE Howard, R Ludeke, and G Schul. Structures grown by molecular beam epitaxy. *Journal of Vacuum Science and Technology*, 10(5):655–662, 1973.

[86] WK Liu, SM Mokler, N Ohtani, C Roberts, and BA Joyce. A rheed study of the surface reconstructions of si (001) during gas source mbe using disilane. *Surface science*, 264(3):301–311, 1992.

[87] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25, 2012.

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

[89] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.

[90] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. In *Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11–14, 2016, Proceedings, Part IV 14*, pages 630–645. Springer, 2016.

[91] Wessam M Salama and Moustafa H Aly. Deep learning in mammography images segmentation and classification: Automated cnn approach. *Alexandria Engineering Journal*, 60(5):4701–4709, 2021.

[92] Wenmei Li, Ziteng Wang, Yu Wang, Jiaqi Wu, Juan Wang, Yan Jia, and Guan Gui. Classification of high-spatial-resolution remote sensing scenes method using transfer learning and deep convolutional neural network. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 13:1986–1995, 2020.

[93] Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. *arXiv preprint arXiv:1605.07146*, 2016.

[94] Andreas Veit, Michael J Wilber, and Serge Belongie. Residual networks behave like ensembles of relatively shallow networks. *Advances in neural information processing systems*, 29, 2016.

[95] Sheldon Mascarenhas and Mukul Agarwal. A comparison between vgg16, vgg19 and resnet50 architecture frameworks for image classification. In *2021 International conference on disruptive technologies for multi-disciplinary research and applications (CENTCON)*, volume 1, pages 96–99. IEEE, 2021.

[96] Yang Liu, Zelin Zhang, Xiang Liu, Lei Wang, and Xuhui Xia. Deep learning-based image classification for online multi-coal and multi-class sorting. *Computers & Geosciences*, 157:104922, 2021.

[97] NJC Ingle, A Yuskauskas, R Wicks, M Paul, and S Leung. The structural analysis possibilities of reflection high energy electron diffraction. *Journal of Physics D: Applied Physics*, 43(13):133001, 2010.

[98] Dillip K Satapathy, Bernd Jenichen, Klaus H Ploog, and Wolfgang Braun. Azimuthal reflection high-energy electron diffraction study of mnas growth on gaas (001) by molecular beam epitaxy. *Journal of applied physics*, 110(2), 2011.

[99] W Braun, H Möller, and Y-H Zhang. Reflection high-energy electron diffraction during substrate rotation: A new dimension for in situ characterization. *Journal of Vacuum Science & Technology B: Microelectronics and Nanometer Structures Processing, Measurement, and Phenomena*, 16(3):1507–1510, 1998.

[100] Y Xiang, FW Guo, TM Lu, and GC Wang. Reflection high-energy electron diffraction measurements of reciprocal space structure of 2d materials. *Nanotechnology*, 27(48):485703, 2016.

[101] Alexander Kirillov, Eric Mintun, Nikhila Ravi, Hanzi Mao, Chloe Rolland, Laura Gustafson, Tete Xiao, Spencer Whitehead, Alexander C Berg, Wan-Yen Lo, et al. Segment anything. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 4015–4026, 2023.

[102] Håkon Wiik Ånes, Jarle Hjelen, Bjørn Eske Sørensen, ATJ van Helvoort, and Knut Marthinsen. Processing and indexing of electron backscatter patterns using open-source software. In *IOP Conference Series: Materials Science and Engineering*, volume 891, page 012002. IOP Publishing, 2020.

[103] Peter Muhlschlegel, H-J Eisler, Olivier JF Martin, Bert Hecht, and DW Pohl. Resonant optical antennas. *science*, 308(5728):1607–1609, 2005.

[104] Christian Girard. Near fields in nanostructures. *Reports on progress in physics*, 68(8):1883, 2005.

[105] Arseniy I Kuznetsov, Andrey E Miroshnichenko, Mark L Brongersma, Yuri S Kivshar, and Boris Luk'yanchuk. Optically resonant dielectric nanostructures. *Science*, 354(6314):aag2472, 2016.

[106] J Hadamard. Princeton university bulletin. *1902*, 13:49–52, 1902.

[107] Mahmoud MR Elsawy, Stéphane Lanteri, Régis Duvigneau, Jonathan A Fan, and Patrice Genevet. Numerical optimization methods for metasurfaces. *Laser & Photonics Reviews*, 14(10):1900445, 2020.

[108] Itzik Malkiel, Michael Mrejen, Achiya Nagler, Uri Arieli, Lior Wolf, and Haim Suchowski. Plasmonic nanostructure design and characterization via deep learning. *Light: Science & Applications*, 7(1):60, 2018.

[109] Peter R Wiecha and Otto L Muskens. Deep learning meets nanophotonics: a generalized accurate predictor for near fields and far fields of arbitrary 3d nanostructures. *Nano letters*, 20(1):329–338, 2019.

[110] Taigao Ma, Haozhu Wang, and L Jay Guo. Optogpt: a foundation model for inverse design in optical multilayer thin film structures. *arXiv preprint arXiv:2304.10294*, 2023.

[111] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989.

[112] Ana Estrada-Real, Abdourahman Khaireh-Walieh, Bernhard Urbaszek, and Peter R Wiecha. Inverse design with flexible design targets via deep learning: Tailoring of electric and magnetic multipole scattering from nano-spheres. *Photonics and Nanostructures-Fundamentals and Applications*, 52:101066, 2022.

[113] Jiaqi Jiang and Jonathan A Fan. Global optimization of dielectric metasurfaces using a physics-driven neural network. *Nano letters*, 19(8):5366–5372, 2019.

[114] Peng Dai, Kai Sun, Xingzhao Yan, Otto L Muskens, CH de Groot, Xupeng Zhu, Yueqiang Hu, Huigao Duan, and Ruomeng Huang. Inverse design of structural color: finding multiple solutions via conditional generative adversarial networks. *Nanophotonics*, 11(13):3057–3069, 2022.

[115] Takashi Asano and Susumu Noda. Iterative optimization of photonic crystal nanocavity designs by using deep neural networks. *Nanophotonics*, 8(12):2243–2256, 2019.

[116] Tian Zhang, Jia Wang, Qi Liu, Jinzan Zhou, Jian Dai, Xu Han, Yue Zhou, and Kun Xu. Efficient spectrum prediction and inverse design for plasmonic waveguide systems based on artificial neural networks. *Photonics Research*, 7(3):368–380, 2019.

[117] Nicholas J Dinsdale, Peter R Wiecha, Matthew Delaney, Jamie Reynolds, Martin Ebert, Ioannis Zeimpekis, David J Thomson, Graham T Reed, Philippe Lalanne, Kevin Vynck, et al. Deep learning enabled design of complex transmission matrices for universal optical components. *ACS photonics*, 8(1):283–295, 2021.

[118] Peter R. Wiecha. A newcomer's guide to deep learning for inverse design in nano-photonics. https://gitlab.com/wiechapeter/newcomer_guide_dl_inversedesign, July 2023.

[119] Antoine Moreau. PyMoosh. https://github.com/AnMoreau/PyMoosh, July 2023.

[120] Peter R Wiecha. pygdm—a python toolkit for full-field electro-dynamical simulations and evolutionary optimization of nanostructures. *Computer Physics Communications*, 233:167–192, 2018.

[121] Peter R Wiecha, Clément Majorel, Arnaud Arbouet, Adelin Patoux, Yoann Brûlé, Gérard Colas Des Francs, and Christian Girard. "pygdm"-new functionalities and major improvements to the python toolkit for nano-optics full-field simulations. *Computer Physics Communications*, 270:108142, 2022.

[122] Peter R Wiecha, Arnaud Arbouet, Christian Girard, and Otto L Muskens. Deep learning in nano-photonics: inverse design and beyond. *Photonics Research*, 9(5):B182–B200, 2021.

[123] Dianjing Liu, Yixuan Tan, Erfan Khoram, and Zongfu Yu. Training deep neural networks for the inverse design of nanophotonic structures. *Acs Photonics*, 5(4):1365–1369, 2018.

[124] Irina Higgins, Loic Matthey, Arka Pal, Christopher P Burgess, Xavier Glorot, Matthew M Botvinick, Shakir Mohamed, and Alexander Lerchner. beta-vae: Learning basic visual concepts with a constrained variational framework. *ICLR (Poster)*, 3, 2017.

[125] Yuxin Wu and Justin Johnson. Rethinking" batch" in batchnorm. *arXiv preprint arXiv:2105.07576*, 2021.

[126] Jakob Søndergaard Jensen and Ole Sigmund. Topology optimization for nano-photonics. *Laser & Photonics Reviews*, 5(2):308–321, 2011.

[127] Yang Deng, Simiao Ren, Kebin Fan, Jordan M Malof, and Willie J Padilla. Neural-adjoint method for the inverse design of all-dielectric metasurfaces. *Optics Express*, 29(5):7526–7534, 2021.

[128] Yongxin Jing, Hongchen Chu, Bo Huang, Jie Luo, Wei Wang, and Yun Lai. A deep neural network for general scattering matrix. *Nanophotonics*, 12(13):2583–2591, 2023.

[129] Yannick Augenstein, Taavi Repan, and Carsten Rockstuhl. Neural operator-based surrogate solver for free-form electromagnetic inverse design. *ACS Photonics*, 10(5):1547–1557, 2023.

[130] Peter Battaglia, Razvan Pascanu, Matthew Lai, Danilo Jimenez Rezende, et al. Interaction networks for learning about objects, relations and physics. *Advances in neural information processing systems*, 29, 2016.

[131] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

[132] Steven Farrell, Paolo Calafiura, Mayur Mudigonda, Dustin Anderson, Jean-Roch Vlimant, Stephan Zheng, Josh Bendavid, Maria Spiropulu, Giuseppe Cerati, Lindsey Gray, et al. Novel deep learning methods for track reconstruction. *arXiv preprint arXiv:1810.06111*, 2018.

[133] Kien Do, Truyen Tran, and Svetha Venkatesh. Graph transformation policy network for chemical reaction prediction. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 750–760, 2019.

[134] Junyuan Shang, Cao Xiao, Tengfei Ma, Hongyan Li, and Jimeng Sun. Gamenet: Graph augmented memory networks for recommending medication combination. In *proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 1126–1133, 2019.

[135] Wenming Cao, Zhiyue Yan, Zhiquan He, and Zhihai He. A comprehensive survey on geometric deep learning. *IEEE Access*, 8:35929–35949, 2020.

[136] Si Zhang, Hanghang Tong, Jiejun Xu, and Ross Maciejewski. Graph convolutional networks: a comprehensive review. *Computational Social Networks*, 6(1):1–23, 2019.

[137] Muhammet Balcilar, Guillaume Renton, Pierre Héroux, Benoit Gauzere, Sebastien Adam, and Paul Honeine. Bridging the gap between spectral and spatial domains in graph neural networks. *arXiv preprint arXiv:2003.11702*, 2020.

[138] Daniele Grattarola and Cesare Alippi. Graph neural networks in tensorflow and keras with spektral [application notes]. *IEEE Computational Intelligence Magazine*, 16(1):99–106, 2021.

[139] O Leseur, R Pierrat, JJ Sáenz, and R Carminati. Probing two-dimensional anderson localization without statistics. *Physical Review A*, 90(5):053827, 2014.

[140] Nima Tajbakhsh, Jae Y Shin, Suryakanth R Gurudu, R Todd Hurst, Christopher B Kendall, Michael B Gotway, and Jianming Liang. Convolutional neural networks for medical image analysis: Full training or fine tuning? *IEEE transactions on medical imaging*, 35(5):1299–1312, 2016.

[141] Denis Langevin, Pauline Bennet, Abdourahman Khaireh-Walieh, Peter Wiecha, Olivier Teytaud, and Antoine Moreau. Pymoosh: a comprehensive numerical toolkit for computing the optical properties of multilayered structures. *JOSA B*, 41(2):A67–A78, 2024.

[142] Josselin Defrance, Caroline Lemaître, Rabih Ajib, Jessica Benedicto, Emilien Mallet, Rémi Pollès, Jean-Pierre Plumey, Martine Mihailovic, Emmanuel Centeno, Cristian Ciracì, et al. Moosh: A numerical swiss army knife for the optics of multilayers in octave/matlab. *Journal of Open Research Software*, 4(1):e13–e13, 2016.