Master's thesis
Computational Materials Physics

Image Recognition for Atomic Force Microscopy

Lauri Kurki

September 6, 2021

Supervisors: Dr. F. Urtev, Aalto University
M.Sc. N. Oinonen, Aalto University
Prof. A. Foster, Aalto University

Examiners: Prof. A. Foster, Aalto University
Prof. F. Djurabekova

UNIVERSITY OF HELSINKI
Master's Programme in Materials Research

P.O. Box 64 (Gustaf Hällströmin katu 2)
FI-00014 University of Helsinki

HELSINGIN YLIOPISTO — HELSINGFORS UNIVERSITET — UNIVERSITY OF HELSINKI

Tiedekunta — Fakultet — Faculty	Koulutusohjelma — Utbildningsprogram — Education programme
Faculty of Science	Master's Programme in Materials Research

Tekijä — Författare — Author
Lauri Kurki

Työn nimi — Arbetets titel — Title

Image Recognition for Atomic Force Microscopy

Opintosuunta — Studieinriktning — Study track
Computational Materials Physics

Työn laji — Arbetets art — Level	Aika — Datum — Month and year	Sivumäärä — Sidoantal — Number of pages
Master's thesis	September 6, 2021	71 pages

Tiivistelmä — Referat — Abstract

Atomic force microscopy (AFM) is a widely utilized characterization method capable of capturing atomic level detail in individual organic molecules. However, an AFM image contains relatively little information about the deeper atoms in a molecule and thus interpretation of AFM images of non-planar molecules offers significant challenges for human experts. An end-to-end solution starting from an AFM imaging system ending in an automated image interpreter would be a valuable asset for all research utilizing AFM.

Machine learning has become a ubiquitous tool in all areas of science. Artificial neural networks (ANNs), a specific machine learning tool, have also arisen as a popular method many fields including medical imaging, self-driving cars and facial recognition systems. In recent years, progress towards interpreting AFM images from more complicated samples has been made utilizing ANNs.

In this thesis, we aim to predict sample structures from AFM images by modeling the molecule as a graph and using a generative model to build the molecular structure atom-by-atom and bond-by-bond. The generative model uses two types of ANNs, a convolutional attention mechanism to process the AFM images and a graph neural network to process the generated molecule.

The model is trained and tested using simulated AFM images. The results of the thesis show that the model has the capability to learn even slight details from complicated AFM images, especially when the model only adds a single atom to the molecule. However, there are challenges to overcome in the generative model for it to become a part of a fully capable end-to-end AFM process.

Avainsanat — Nyckelord — Keywords

Säilytyspaikka — Förvaringsställe — Where deposited

Muita tietoja — övriga uppgifter — Additional information

Contents

1 Introduction **1**

2 Atomic Force Microscopy **4**

2.1 AFM imaging process 4

2.1.1 General principles 4

2.1.2 Frequency Modulation AFM 6

2.1.3 Tip functionalization 8

2.2 The inverse imaging problem 9

2.2.1 The need for an AFM simulator 9

2.2.2 Probe Particle Model 10

2.3 Database generation 12

2.3.1 Database of molecular geometries 12

2.3.2 Image augmentation 13

3 Machine Learning **16**

3.1 Overview of machine learning 16

3.2 Artificial neural networks 20

3.2.1 Multilayer perceptron 21

3.2.2 Activation functions 22

3.2.3 Optimizers 24

3.2.4 Backpropagation 25

3.2.5 Hyperparameter optimization 26
3.3 Convolutional neural networks 28
3.4 Graph neural networks 31
3.4.1 Motivation for graphs 31
3.4.2 Recurrent relational networks 32

4 Generative Graph Model **35**
4.1 Iterative scheme 35
4.2 Model architecture 38
4.2.1 Input branches 38
4.2.2 Output branches 42
4.3 Training and evaluation 43

5 Results and discussion **47**
5.1 Single atom predictions 47
5.2 Generative predictions 54

6 Conclusions **61**

Bibliography **64**
References 64

1. Introduction

The Nobel Prize winning invention of scanning tunneling microscopy [1] (STM) by Binnig and Rohrer in 1981 revolutionized surface science. It allowed imaging of individual atoms, a task that was naturally intriguing for surface scientists but up until then out of reach for the scientific community. Such high precision was achievable because STM and atomic force microscopy (AFM) alike were not optical microscopes and thus the optical diffraction limit was not a concern. STM and AFM belong to a wider class of scanning probe microscopy (SPM) methods for which the basic principle instead relies on "touching" the sample. However, for STM the reliance on tunneling current meant that there was one major constraint: the imaged sample had to be conductive. This ruled out imaging of all insulators, including many polymers, ceramics and organic materials.

While STM had its limitations, the method was further developed by the scientific community and later in 1985 Binnig et al. introduced the atomic force microscope [2]. Instead of measuring current as in STM, in AFM the close range interaction consisting of numerous different contributors, such as van der Waals forces, electrostatic forces and Pauli repulsion, between the probe and the sample directly affect the image. This made possible a wider range of samples. While subsequent research gave rise to many different operating modes of AFM for different purposes, in this thesis the imaging method is frequency modulation AFM [3] which is a specific variant of non-contact AFM.

Since the close range forces experienced by an AFM probe are strongest with the surface layer of the sample, images of two-dimensional samples are straightforward to interpret [4]. However, images of non-planar samples and molecular mixtures are often more obscure and their interpretation is time-consuming and difficult for human experts [5, 6]. In this thesis we aim to develop a tool to automate AFM image interpretation making the AFM imaging process more efficient.

The second major component in this thesis is machine learning which is proposed as an efficient framework for interpreting AFM images. The recent developments in computational methods, and more specifically in artificial neural networks, have changed approaches in computational materials science drastically, and using artificial neural networks in data processing has become a standard in almost all areas of science and technology.

Classical computational physics has the following approach: the model is given an input and the rules to follow to produce an output. For example in molecular dynamics (MD) simulations this would mean that we give to an MD code the starting positions and velocities of atoms and we consider the applicable laws of physics to update the system in discrete timesteps. The modern data-driven approach turns this mindset around: the model is given the input and the corresponding output and its task is to discover the mapping which produces an output object from a certain input condition. In a previous study, AFM images were analyzed using a machine learning model which learned to represent the sample structures as images containing information about the molecular geometry and chemical properties of the sample [7]. In this thesis we design a machine learning model which learns to translate the AFM image into a graph representation of the sample molecular geometry. The model is trained and evaluated on simulated AFM images [8, 9].

The thesis has the following structure and content. Chapter 2 explains the AFM imaging process, motivates this study and describes the AFM simulation

method. Chapter 3 introduces the general principles of machine learning and explains in detail the properties of artificial neural networks (ANNs) and two subclasses of ANNs, convolutional neural networks and graph neural networks. Chapter 4 describes the particular machine learning model designed in this thesis and elaborates the training and evaluation process. Results of the thesis and the discussion thereof is presented in chapter 5 while the conclusions and possible future prospects of development are discussed in chapter 6.

2. Atomic Force Microscopy

2.1 AFM imaging process

2.1.1 General principles

The basic principle behind atomic force microscopy is that an atomically sharp tip is constructed onto a cantilever and then brought close to the sample. The tip-sample interaction causes a deflection of the cantilever position which is measured by, for example, the beam deflection method [10]. A simplified schematic of the atomic force microscope is shown in figure 2.1. Based on whether the cantilever is excited to oscillate during the imaging process, the AFM methods can be categorized into static and dynamic AFM.

One example of a static AFM is the constant force mode in which the tip is very close to the sample and the tip-sample force F_{ts} causes a deflection Δz to the cantilever. The deflection is caused by a balance of forces and in equilibrium the total force acting upon the cantilever must vanish as $F_{tot} = F_{ts} + F_{cant} = 0$. Assuming the cantilever deflection is small, we can apply the Hooke's law $F_{cant} = -k\Delta z$ to say that a certain deflection Δz is caused by a tip-sample force F_{ts} of certain magnitude. By raster scanning the sample in the xy-plane while keeping the tip-height z constant using a feedback loop we are able to construct an isosurface of equal deflection Δz which is interpreted through Hooke's law as an isosurface of equal force $F_{ts} = k\Delta z$. The density of the raster scanning grid defines an upper limit for the resolution of

the AFM image.

The cantilever should be chosen so that its spring constant k_{cant} is smaller than the effective spring constant of the atomic bonds in the sample to avoid causing deformations to either. The reason for avoiding deformations in the sample is obvious but even in the tip any possible deformations might cause artifacts in the image, for example if the tip becomes asymmetric. However, if the cantilever is too flexible i.e. its spring constant is smaller than the maximum tip-sample force gradient $k_{cant} < \max\left(-\frac{\partial F_{ts}}{\partial z}\right)$, a sudden *snap-to-contact* might occur [11]. In dynamic AFM, this instability can be avoided by oscillating the cantilever at an amplitude A larger than $kA > \max(-F_{ts})$ even if the stiffness of the cantilever is small [11]. Nevertheless, in dynamic AFM the stiffness of the cantilever is typically much higher. Since static AFM suffers from these tip and sample deformations making it a difficult experimental realization, dynamic AFM is the more common choice for experiments and it has also been the target of more recent developments.

Dynamic AFM can be further classified into two categories, amplitude modulation (AM-AFM) and frequency modulation (FM-AFM). In AM-AFM, the cantilever is driven using an actuator at a frequency slightly deviated from its eigenfrequency f_0. As the tip approaches the sample, the tip-sample interaction causes a shift in the amplitude and phase of the oscillation while a feedback loop keeps the frequency constant. The amplitude shift is interpreted as the topographic image of the sample. AM-AFM however has a major drawback in that the method is very slow in vacuum where Q is large as the amplitude shift is not instantaneous but instead happens on a timescale of $\tau_{AM} \approx 2Q/f_0$ where the Q factor of the cantilever can be as high as in the magnitude of 10^5 [12]. The introduction of FM-AFM solved this obstacle as the change in oscillation frequency happens on a timescale of $\tau_{FM} \approx 1/f_0$ [12]. Of the two dynamic AFM operation modes, FM-AFM is the most widely used technique for atomic resolution AFM since it requires an ultra-high vacuum environment resulting

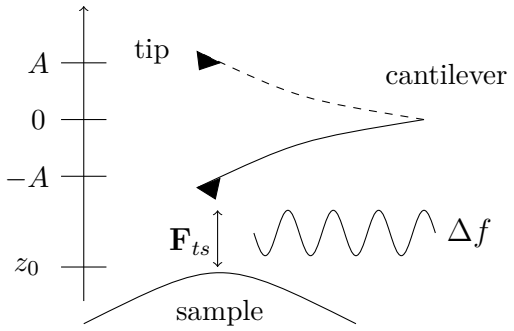

Figure 2.1: Simplified scheme of the FM-AFM system. Recreated from [12].

in a large Q factor [12]. In ambient conditions and biological samples, AM-AFM is the more commonly applied technique [13]. The FM mode is also the focus of this thesis and in the next section it will we described in detail.

2.1.2 Frequency Modulation AFM

The idea behind FM-AFM is illustrated above in figure 2.1. On the contrary to AM mode, in FM mode the cantilever is driven to oscillate exactly at its eigenfrequency f_0 and a feedback loop maintains a constant oscillation amplitude A. When the tip is close to the sample, the oscillation frequency changes to $f = f_0 + \Delta f$ caused by the tip-sample interaction gradient $k_{ts} = \frac{\partial F_{ts}}{\partial z}$. To solve for the frequency shift of the cantilever we consider the harmonic oscillator consisting of the cantilever and the tip-sample interaction. The eigenfrequency of a harmonic oscillator is

$$f = \frac{1}{2\pi}\sqrt{\frac{k^*}{m^*}} \tag{2.1}$$

where k^* and m^* are the effective spring constant and the effective mass. Additionally, if we assume the tip-sample force gradient k_{ts} to be constant in the oscillation range from $-A$ to A, we may write $k^* = k + k_{ts}$ to get

$$f = \frac{1}{2\pi}\sqrt{\frac{k + k_{ts}}{m^*}} \tag{2.2}$$

Further, if $k_{ts} \ll k$ and the effective mass m^* is the mass of the cantilever m then

$$f = \frac{1}{2\pi}\sqrt{\frac{k\left(1+\frac{k_{ts}}{k}\right)}{m}} \tag{2.3}$$

$$\approx \frac{1}{2\pi}\sqrt{\frac{k}{m}}\left(1+\frac{k_{ts}}{2k}\right) \tag{2.4}$$

$$= f_0\left(1+\frac{k_{ts}}{2k}\right) \tag{2.5}$$

Now we have an expression for Δf in a simplified system

$$\Delta f = \frac{k_{ts}}{2k} f_0 \tag{2.6}$$

However, in practise the force gradient k_{ts} varies by orders of magnitude inside one oscillation cycle and the approximation above cannot be directly utilized.

In [14], Giessibl et al. have taken into consideration the varying force gradient and utilized first-order perturbation theory and the Hamilton-Jacobi method to get a general relationship between the frequency shift and tip-sample interaction

$$\Delta f = -\frac{f_0}{kA^2}\left\langle F_{ts}(z_0 - q'(t))q'(t)\right\rangle \tag{2.7}$$

$$= -\frac{f_0^2}{kA^2}\int_0^{1/f_0} F_{ts}(z_0 - q'(t))q'(t)\mathrm{d}t \tag{2.8}$$

where $q'(t) = -A\cos(2\pi f_0 t)$ is the deflection of the cantilever and $\langle F_{ts}q'(t)\rangle$ is averaged over one oscillation cycle. A change of integration variable $t \to q'$ yields

$$\Delta f = -\frac{f_0}{2k}\frac{2}{\pi A^2}\int_{-A}^{A}\frac{F_{ts}(z_0 - q')q'}{\sqrt{A^2 - q'^2}}\mathrm{d}q' \tag{2.9}$$

and integrating this expression in parts gives us the frequency shift in terms of the force gradient k_{ts}

$$\Delta f = \frac{f_0}{2k}\frac{2}{\pi A^2}\int_{-A}^{A} k_{ts}(z_0 - q')\sqrt{A^2 - q'^2}\mathrm{d}q' \tag{2.10}$$

Comparing this to our earlier approximation Eq. 2.6 reveals a close connection as the tip-sample force gradient k_{ts} is merely replaced by its weighted average over

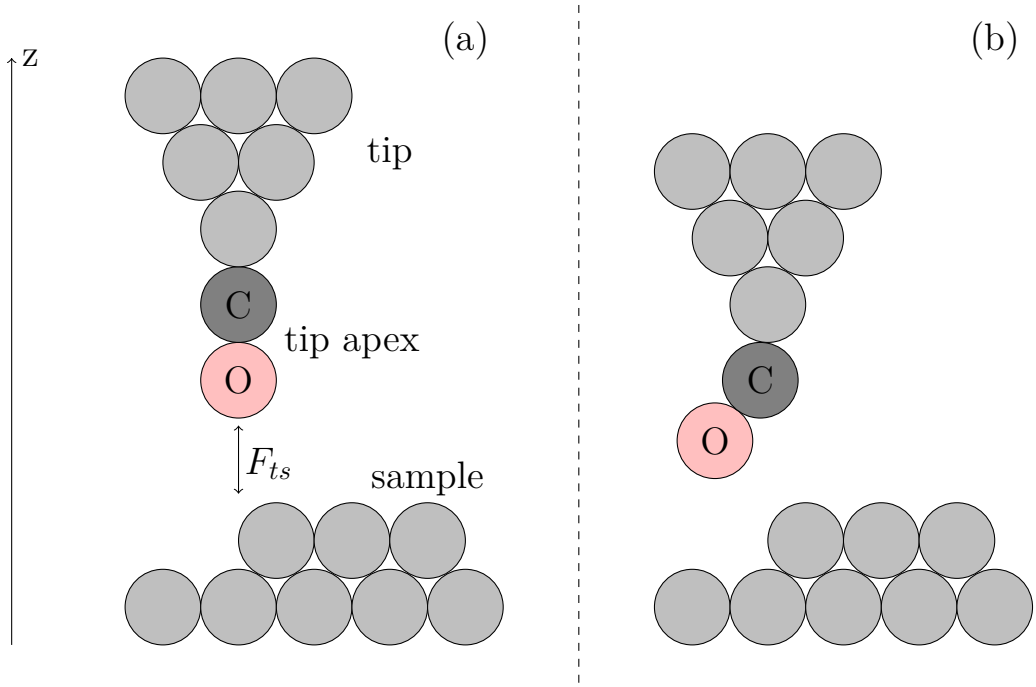

Figure 2.2: Illustration of the functionalized tip apex and the bending mechanism. (a) Tip apex is far away from the sample and aligned symmetrically. (b) Tip apex is close to the sample and bends away.

one cycle in the latter expression. As amplitude is a major contributor in the weight function, tuning the amplitude of the oscillation in an experiment allows AFM to be used in systems of varying force magnitudes [12].

2.1.3 Tip functionalization

High-resolution AFM achieves unmatched atomic resolution by utilizing an atomically well-defined tip termination in which the tip is functionalized by picking up a chemically inert, flexible tip apex [4]. This is illustrated below in figure 2.2. Compared to a bare metal tip, with a functionalized tip apex it is possible to oscillate the cantilever closer to the sample without deforming the sample which is a challenge in experimental AFM. This is because the inert CO molecule does not react with the sample and during the tip approach its flexibility allows for a lateral relaxation when the repulsive interaction F_{ts} is strong [15].

2.2 The inverse imaging problem

2.2.1 The need for an AFM simulator

An AFM imaging process can be formalized mathematically as $\Phi : (\vec{R}, Z) \rightarrow \Delta f(\vec{r})$ where $\vec{R}$ and Z are the positions and types of atoms in the sample and Δf is the imaging signal i.e. frequency shift. The object of this study is to recognize a sample $(\vec{R}, Z)$ from an image Δf without prior knowledge of the system which means the inverse of the imaging process $\Phi^{-1} : \Delta f(\vec{r}) \rightarrow (\vec{R}, Z)$ has to be found. This is a highly non-trivial inverse task since the imaging signal is a non-monotonic function of distance and the complicated tip-sample interaction arises from van der Waals forces, Pauli repulsion and others. Since no analytical solution is available, in this study we utilize machine learning to decipher the inverse function. Machine learning in general, and more specifically deep learning, will be discussed in detail in the next chapter but to highlight the need for a robust and fast AFM simulator the following has to be considered.

i. Experimental AFM is time-consuming

ii. Deep learning is very data-hungry

iii. Manual labeling is required to train a deep learning model with experimental AFM

This shows an evident obstacle in gathering a sufficient amount of training data for our model: using experimental AFM imaging to gather thousands of training samples is simply unfeasible and even if an adequately sized database was acquired, labeling the samples would be impossible since generally the imaged samples are unknown.

Many different AFM simulation methods have been developed to study the underlying mechanics of AFM and also to assist in overcoming the issues present

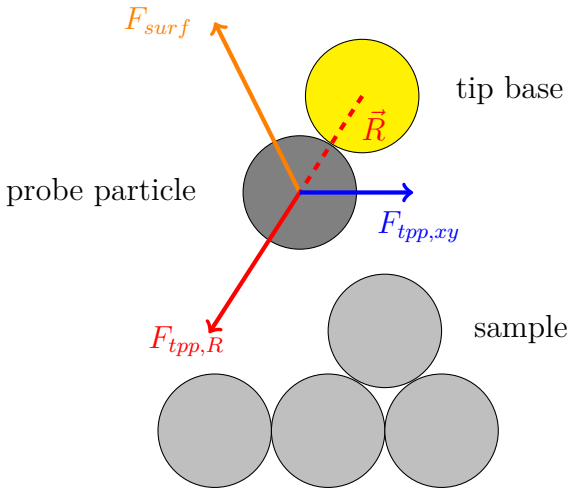

Figure 2.3: The force components in Probe Particle Model. F_{surf} consists of the sum of all pairwise LJ forces between the probe particle and the sample, and the electrostatic force. The radial force between the tip base and the probe particle is $F_{tpp,R}$ and the lateral component is $F_{tpp,xy}$ while $\vec{R}$ is the position of the probe particle with respect to the sample.

in experimental imaging [16, 17]. Two prominent approaches to AFM simulation are molecular dynamics based methods [18, 19] and *ab initio* quantum mechanical methods [20, 21], such as density functional theory (DFT). Of these methods, the quantum mechanical solutions offer more accurate representations of experimental images as they model the tip-sample interaction in high detail. The downside to high precision is the relatively high computational cost making it less suitable for large databases.

2.2.2 Probe Particle Model

In this study, we used the Probe Particle Model [8, 9, 22] to simulate AFM. It is a mechanical model based on describing the tip-sample interaction through three contributors that are demonstrated in figure 2.3.

(1.) The Lennard-Jones potential

$$V_{LJ}(r) = \varepsilon \left[\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^{6} \right] \tag{2.11}$$

is used to model the Pauli repulsion (r^{-12} term) and the attractive dispersion forces (r^{-6} term). The parameters ε and σ control the depth of the potential well and the distance at which the particle-particle potential vanishes and r is the distance between the particles. While the Lennard-Jones potential is relatively uncomplicated, it has been observed to describe many systems in good agreement with experimental results [9].

(2.) The electrostatic interaction

$$\vec{F}_{es}(\vec{R}) = \frac{\partial}{\partial \vec{R}} \int \rho_{probe}(\vec{r}) V_{surf}(\vec{r} - \vec{R}) \mathrm{d}\vec{r} \tag{2.12}$$

between the tip and the sample is the second contributor. In the expression above, $\rho_{probe}(\vec{r})$ is the charge density at the probe particle location, V_{surf} is the electrostatic potential obtained from point charges and $\vec{R}$ is the position of the probe particle with respect to the sample. The integration is computed over all space $\vec{r}$.

(3.) Thirdly, there are two separate forces acting between the probe particle and the tip base. One is the radial spring force between the probe and the tip base to keep the probe attached. Second is the lateral spring force stemming from the attractive potential of the tip base.

The simulation is done by scanning the tip over the sample in an xy-plane and at each lateral point, the tip is placed at an initial height z_0 and its z-position is lowered in small discrete steps. At each height, the probe particle is allowed to relax until $F_{surf} + F_{tpp,xy} + F_{tpp,R}$ is lower than a certain preset threshold value. The tip approach is computed separately for each lateral position which defines the image resolution. Once the tip approach has been computed, the vertical component of F_{surf} is obtained through projection onto the z-axis $F_z(z)$ and finally the formula of translating the force function to frequency shift Δf (Eq. 2.10) is used to integrate the imaging signal from $F_z(z)$. Using different slices of $F_z(z)$ corresponding to different oscillation heights allows us to acquire AFM images at multiple heights.

To speed up the simulation process, we use a GPU optimized version of the

Probe Particle Model. This is possible because the lateral scanning in an xy-grid is an embarrassingly parallel task [23], ideal for a GPU with thousands of cores. More specifically, we use a version developed using OpenCL [24] making it possible to obtain ~ 100 stacks of AFM images per second.

2.3 Database generation

2.3.1 Database of molecular geometries

The quality of data is of vital importance in machine learning tasks and a wide range of training samples is required for a well-generalizing model. In this study, we use a database of more than 6,000 free-standing small organic molecules in vacuum conditions optimized with DFT. For the Probe Particle Model, charge densities required for the electrostatic interaction (Eq. 2.12) are obtained from point charges.

The molecules consist of atoms lighter than bromine (H, C, N, O, F, Si, P, S, Cl, Br). The exact distributions of different atom types in the data set are shown in table 2.1. The most prevalent atom types are hydrogen and carbon, both of which are observed in over 94 % of the molecules. Carbon, oxygen and chlorine are also quite common in the data set. Silicon and phosphorus are the most rare elements with less than 1 % of the molecules containing either. Next to the table, figure 2.4 shows the distribution of sizes of molecules in the database. Most of the molecules contain between 15 and 30 atoms with few instances of very small molecules or molecules with $N_{atoms} > 40$.

AFM images are sensitive with respect to the scanning angle because the tip-sample interaction is very short ranged and thus an AFM image contains relatively little information about the deeper atoms in a molecule. This means that rotating the sample molecule even slightly can make the resulting image unrecognizable. We can utilize the rotational sensitivity to expand the database using rotational

Element	% of selected rotations
H	94.2
C	31.1
N	10.3
O	25.3
F	2.1
Si	< 0.1
P	0.2
S	8.5
Cl	20.0
Br	3.3

Table 2.1: Occurrences of different atom types in the selected rotations.

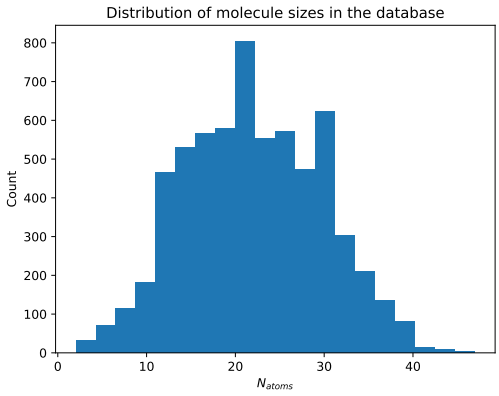

Figure 2.4: Distribution of molecule sizes within the database.

augmentation and doing the scanning for a single molecule from multiple scanning angles. The rotations are done so that the number of atoms visible in each rotation is maximized. That is, the following entropy function is maximized

$$S = \sum_{i=1}^{N_{atoms}} \exp(-\beta(z_i - z_{top})) \tag{2.13}$$

and 30 rotations with highest entropy are chosen. In the above expression, N_{atoms} is the number of atoms in the molecule, β is the decay parameter chosen to be 1Å^{-1}, z_{top} is the z-coordinate of the top atom in that specific rotation, and z_i are the z-coordinates of the atoms. After the rotations, our molecular database is effectively 30 times larger.

2.3.2 Image augmentation

Since the eventual target for our machine learning model is to generalize from simulated AFM data to predicting from experimental images, a few things have to be considered. Experimental AFM images are not always perfect; on the contrary, often there exists some impurities arising from e.g. electrical noise or experimental artifacts. To combat this, we slightly modify the simulated AFM images with the

following augmentations to make them more alike to their experimental counterparts.

The first two augmentations are parameter randomizations in the Probe Particle Model that are implemented in the AFM simulation phase. The rest are image augmentations applied in the data loading phase during machine learning model training. The following listing describes each augmentation trick.

i. **Probe particle rotation**. In experimental imaging, attaching the probe particle to the tip base in perfect symmetry is difficult and instead the tip termination is often slightly shifted with respect to the tip base. Asymmetric probe particle attachment means that there are "shadows" cast asymmetrically around the sample as the probe particle more easily bends in the direction of the probe particle tilt. To match this, in the simulations the probe particle equilibrium position is varied in a circle of radius 0.5 Å.

ii. **Tip-sample distance**. In experiments, the tip-sample distance is only known up until a certain precision but in simulations it is known exactly. In order to make the machine learning model not dependent on the exact imaging height, we slightly deviate the z-coordinate of the tip in the simulations.

iii. **Image normalization**. The simulated AFM images are normalized to zero mean and unit variance for each scanning height independently. Normalizing input data has been observed to be generally beneficial for training machine learning models [25].

iv. **Artificial noise**. We add uniform random noise to artificially make the simulated AFM images more blurry. This augmentation tries to imitate electrical and thermal noise present in experimental images.

v. **Scanning plane shift**. Each scanning plane is randomly shifted a slight amount in relation to the previous scanning plane so that any two adjacent

planes are shifted at most 2 % of the scanning plane width. The maximum shift in either x or y dimension along all scanning heights is 4 %. In experiments, when changing the scanning height, the images are often slightly misaligned which we try to incorporate here.

vi. **Cutouts**. We randomly add small rectangular patches of zeros to the simulated images to resemble artifacts which are sometimes observed in experimental images.

3. Machine Learning

3.1 Overview of machine learning

Machine learning is a broad subfield of an even larger field of artificial intelligence. For many materials scientists, machine learning has become an essential tool in solving complicated research problems, such as discovering new stable materials [26], predicting DFT functionals [27] and discovering structures in AFM images [7]. In this thesis, we build upon the research of Alldritt et al. [7] by predicting exact atomic coordinates from AFM images using a generative graph neural network. In this chapter, the necessary theoretical background for machine learning is described.

The principle behind machine learning is that a computer program should learn from data to make decisions without explicit instructions. In [28], Mitchell gives a succinct definition of a machine learning problem: a machine learning algorithm improves its performance P at executing a task T through experience E. A classic example is the problem of classifying hand-written digits into correct categories. Here, the task T would be the classification, the performance metric P might be the classification accuracy and the experience E would be examples of digits, each labeled with the correct answer. Determining the three components (T, P, E) unambiguously defines the machine learning problem.

The field of machine learning is traditionally separated into three categories: supervised learning, unsupervised learning and reinforcement learning. The differ-

ence between the categories is the information the model is able to infer from the data. Unsupervised learning deals with unlabeled data and is often used for clustering or segmenting the data set. Example algorithms of this class include k-means clustering, principle component analysis and singular value decomposition, among many others.

Reinforcement learning is a general class of learning algorithms in which an agent performs actions in some action space while getting feedback from the environment as it to maximize some reward by making the right decisions. These algorithm are useful when the feedback from some action is delayed, as is the case for example in many games, in which the feedback whether the agent has won or lost the game is only provided at the end of a game.

Supervised learning (SL) algorithms learn from data accompanied by the correct answers i.e. labels. In SL problems, the objective is to find a mapping which translates a certain input to a correct output (label). The model learns the mapping based on example input-output pairs. The class of supervised learning algorithms includes, for instance, regression methods, Naive Bayes methods, decision tree based algorithms and neural networks. The model used in this thesis is a supervised learning algorithm for which the data is the AFM images and the labels are the corresponding molecular geometries. Arguably the simplest machine learning algorithm is linear regression which learns a mapping from input vector $\mathbf{x} \in \mathbb{R}^n$ to a scalar value $\hat{y} \in \mathbb{R}$. The model in this case should linearly predict the output $\hat{y}$ when given a vector $\mathbf{x}$ as input i.e. the model is formulated

$$\hat{y} = \boldsymbol{w}^\top \mathbf{x} \tag{3.1}$$

where $\boldsymbol{w} \in \mathbb{R}^n$ are the parameters of the model. Often, an intercept or bias term b is added to the model for $\hat{y} = \boldsymbol{w}^\top \mathbf{x} + b$.

The behaviour of a machine learning model is controlled by its parameters and often the objective is to find the optimal set of parameters $\boldsymbol{w} \in \mathbb{R}^n$ that minimizes

a performance metric P, essentially turning the problem into an optimization task. Depending on the problem and the nature of the output, the parameter space $\mathbb{R}^n$ can vary both in length and rank.

In machine learning terms, the performance metric is called a loss function of which the mean squared error is a good example:

$$L = \frac{1}{N}\|\hat{\mathbf{y}} - \mathbf{y}\|_2^2 \tag{3.2}$$

The error represents the euclidean distances between the predictions $\hat{\mathbf{y}}$ and the targets $\mathbf{y}$ and it is minimized when $\hat{\mathbf{y}} = \mathbf{y}$. The minimum in this case is found by finding the root of the gradient with respect to $\boldsymbol{w}$

$$\nabla_{\boldsymbol{w}} \frac{1}{N}\|\hat{\mathbf{y}} - \mathbf{y}\|_2^2 = \nabla_{\boldsymbol{w}} \frac{1}{N}\|\mathbf{X}\boldsymbol{w} - \mathbf{y}\|_2^2 = 0 \tag{3.3}$$

However, this expression is only true for convex losses with a single minimum since non-convex loss functions can have multiple minima and maxima as well as saddle points which all would have zero gradient. With more complicated models and loss functions, the root of the gradient cannot be solved analytically and more sophisticated methods are required to minimize the loss function. Some of these methods are described in section 3.2.

The ultimate goal of a machine learning model is to make good predictions on unseen data. However, if the model was trained on a specific data set $(\mathbf{X}, \mathbf{y})$ containing the input vectors $\mathbf{X}$ and the corresponding output values $\mathbf{y}$ and the optimal parameters were found by minimizing the loss function for this exact data set, the model might have learned to make excellent predictions on $(\mathbf{X}, \mathbf{y})$ but it still could perform poorly on unseen data. To combat this, the data set is often split into training and testing sets. The training set $(\mathbf{X}^{(train)}, \mathbf{y}^{(train)})$ is used to train the model i.e. optimize the parameters and the testing set $(\mathbf{X}^{(test)}, \mathbf{y}^{(test)})$ is used to measure the performance. These sets are kept strictly apart to ensure that the testing set truly represents unseen data and the performance of the model can be

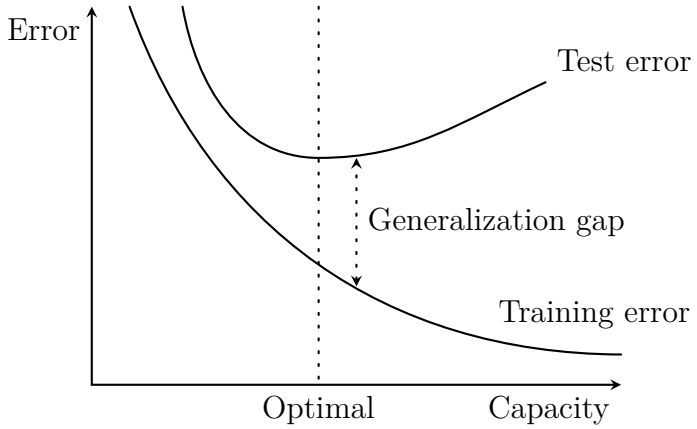

Figure 3.1: Illustration of training and testing errors as a function of model capacity. A larger capacity offers better training error with the cost of worse generalization. A model with the optimal capacity minimizes the test error.

reliably quantified. The parameter optimization task is updated

$$\frac{1}{N^{(train)}}\nabla_{\boldsymbol{w}}\left\|\mathbf{X}^{(train)}\boldsymbol{w}-\mathbf{y}^{(train)}\right\|_2^2=0 \tag{3.4}$$

and the test error is calculated

$$L^{(test)}=\frac{1}{N^{(test)}}\left\|\mathbf{X}^{(test)}\boldsymbol{w}-\mathbf{y}^{(test)}\right\|_2^2 \tag{3.5}$$

To achieve good performance, or generalization, on unseen data, the underlying assumption is that the training and testing sets must be identically distributed and the data items in both sets must be independent from each other.

In machine learning problems, the goodness of the test performance fundamentally arises from two origins: the magnitude of the training error and the gap between training and testing errors. These factors are inseparable from the concepts of underfitting, overfitting, generalization and model complexity, or capacity, which are illustrated in figure 3.1. The capacity of a model refers to how complex the model is, for example, increasing the number of parameters increases the capacity. The model is said to underfit when the training error is large and to overfit when the gap between training and testing errors is large. The former occurs when the learning capacity of the model is inadequate and the latter when the capacity is

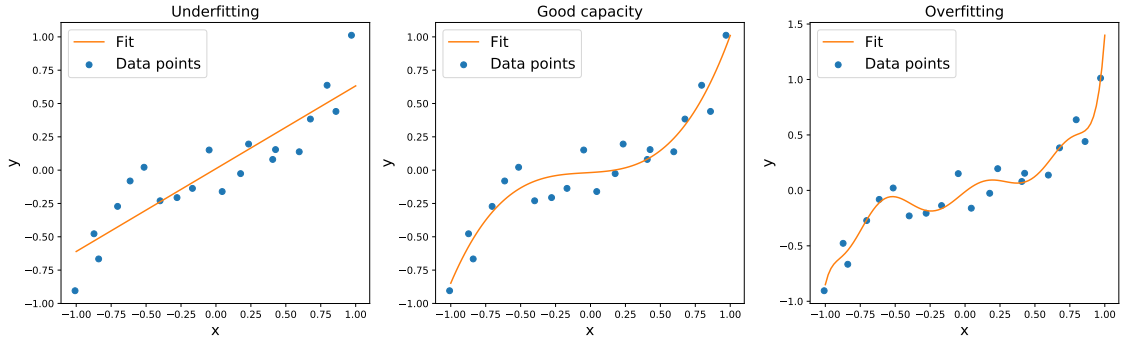

Figure 3.2: In all figures the data points are sampled from $y = x^3$ with added uniform noise. Left: Underfitting model with low capacity. Center: Appropriate capacity results in good fit. Right: Model has too high capacity and memorizes patterns from training data, bad generalization is expected.

too high and the model learns to remember specific features from the training set resulting in unreliable performance for unseen observations. Figure 3.2 illustrates underfitting and overfitting and illustrates the effect of capacity to fitting accuracy.

Traditional machine learning algorithms have proven themselves in simple classification and regression problems but more abstract problems such as image recognition or natural language processing require a more modern approach. One such technique, artificial neural networks, is discussed in the following section.

3.2 Artificial neural networks

Artificial neural networks (ANN) are a specific class of machine learning algorithms. The name and design of ANNs is inspired by biological neural networks in biological brains where the neurons are connected by synapses. As biological neural networks, ANNs also consist of neurons, or nodes, connected with edges that transmit a signal from one node to another. However, in practice ANNs and their biological counterparts work in completely different ways and the resemblance is only on the surface level.

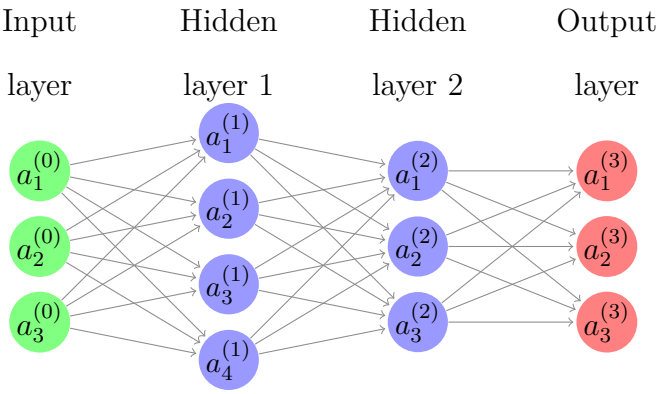

Figure 3.3: An example of a multilayer perceptron model with two hidden layers.

Since a typical artificial neural network is very complex due to the large number of nodes, ANNs require massive amounts of data to learn effectively. Availability of big data has made ANNs a ubiquitous tool in numerous fields, for example in medical diagnosis [29], computer vision for self-driving cars [30] and facial recognition systems [31,32]. Algorithms using ANNs have also reached superhuman capabilities in playing games [33].

3.2.1 Multilayer perceptron

A multilayer perceptron (MLP) is a simple example of an artificial neural network as illustrated by figure 3.3. They consist of an input layer, at least one hidden layer and an output layer and the layered structure of the networks is why the field of artificial neural networks is often referred to as deep learning. Each of the layers consists of nodes which are connected to the nodes of the previous and the next layer. In the case of fully connected layers each node is individually connected to all nodes of the neighboring layers. In an MLP, each connection has a weight corresponding to the relative importance of the connection and each node has an non-linear activation function and a bias term. Considering the weights, biases and activation functions

of each node by layer, the value of each neuron is

$$a^{(l)} = \sigma^{(l)}\left(W^{(l)}a^{(l-1)} + b^{(l)}\right) \tag{3.6}$$

Here, $a^{(l)}$ is the vector containing the values of neuron in the layer l, $W^{(l)}$ is the weight matrix between layer l and $l-1$ containing the entries w^l_{jk} for the weights between node j in layer l and node k in layer $l-1$. Also, $b^{(l)}$ is the bias vector of layer l and $\sigma^{(l)}$ is the activation function applied to layer l. Expanding the previous expression for the entire network of L layers leads to the output vector $g(x)$ for an input x

$$g(x) = \sigma^{(L)}\left(W^{(L)}\sigma^{(L-1)}\left(W^{(L-1)}\cdots\sigma^{(1)}\left(W^{(1)}x + b^{(1)}\right)\cdots\right) + b^{(L)}\right) \tag{3.7}$$

The activation functions are used since otherwise the above combination of functions would be reduced to only one matrix multiplication reducing the capacity of the model drastically. There are multiple different activation functions for different purposes and some of these functions are described next and also shown in figure 3.4.

3.2.2 Activation functions

In practice, a good default activation due to its simplicity is the rectified linear unit, or ReLU [34]:

$$\sigma_{ReLU}(z) = \max(0, z) \tag{3.8}$$

In the positive value regime it behaves as a linear unit and all negative values are truncated to zero. The ReLU activation can be used in networks with large depth as it does not suffer from the vanishing gradient problem as, for example, the sigmoid $\sigma_\sigma(z) = \frac{1}{1+e^{-z}}$ and hyperbolic tangent $\sigma_{tanh}(z) = \tanh(z)$ functions do [35]. Essentially, the vanishing gradient problem means that the early layers of a network learn very slowly since the gradient of the cost function is decreased

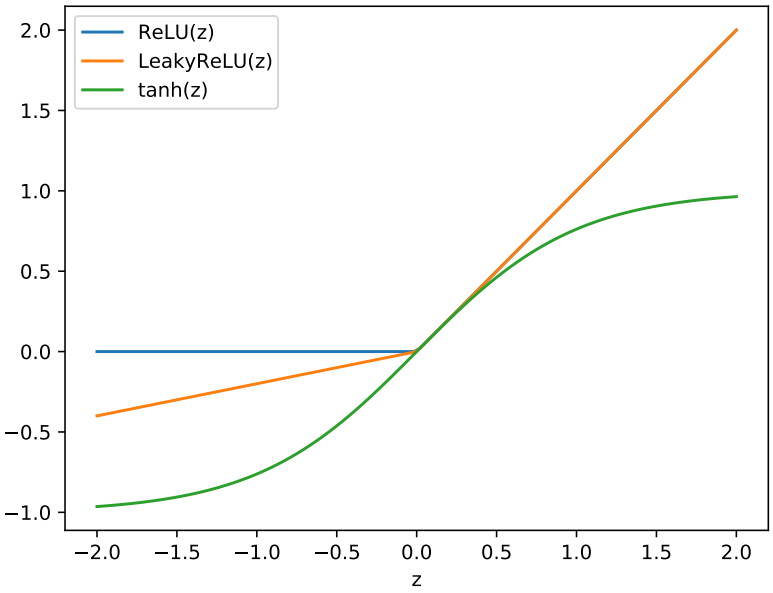

Figure 3.4: Visualization of the ReLU, Leaky ReLU and hyperbolic tangent activation functions

exponentially if the later gradients are small as the gradient is calculated backwards in the network. The backwards calculation, or backpropagation, of the gradient is explained in section 3.2.4.

However, in some cases ReLU causes neurons to die if all inputs to the activation function are negative. This causes the gradient to become zero and the entire network becomes a constant function. To avoid this, an adaptive learning rate should be used. Alternatively, there are variations of the ReLU function such as the Leaky ReLU

$$\sigma_{LeakyReLU}(z) = \max(\alpha x, x), \quad 0 < \alpha < 1 \tag{3.9}$$

which do not cancel the negative regime entirely but only suppress it with the hyperparameter α.

One more important activation function is the softmax

$$\sigma_{softmax}(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_i}} \quad \text{for } i = 1, \ldots, K \tag{3.10}$$

which is often used at the last layer in classification networks to normalize the output of a network to a probability distribution. It transforms each value z_i to range $(0, 1)$ and forces the transformed sum of components $\sigma_{softmax}(z_i)$ to equal 1.

3.2.3 Optimizers

In the previous example of linear regression, the loss function could be minimized analytically. However, even for a relatively simple neural network this is not possible and the parameters minimizing the loss function have to be found by different means. In the field of deep learning, the method of parameter optimization is called the optimizer.

Let $L(f(\boldsymbol{X}; \boldsymbol{\theta}), \boldsymbol{y})$ be a loss function, where f is the function representing the model, $(\boldsymbol{X}, \boldsymbol{y})$ is the data set and $\boldsymbol{\theta}$ is the model parameter vector to be optimized. An often used method is the gradient descent which iteratively improves the parameters by taking steps into the direction opposite to the gradient of the loss function. That is,

$$\boldsymbol{\theta}_{i+1} = \boldsymbol{\theta}_i - \gamma \nabla_{\boldsymbol{\theta}} L(f(\boldsymbol{X}; \boldsymbol{\theta}_i), \boldsymbol{y}) \tag{3.11}$$

where the learning rate γ controls the length of each step. The parameters $\boldsymbol{\theta}_0$ should be initialized to non-zero values to avoid the vanishing gradient problem but on the other hand the values should not be too large to avoid exploding gradients.

Especially with large networks and data sets, the computational load of calculating the gradient with the entire data set becomes unfeasible. A better alternative is stochastic gradient descent (SGD) which only uses a randomly selected part of the data set each time the gradient is calculated reducing the cost of the operation. A full pass over the training samples is called an epoch.

In this project, the Adam optimizer was used [36]. It is a variant of SGD which maintains an individual learning rate for each parameter in the model. Each learning

rate is also adjusted during training based on the gradient in each dimension. This has the benefit of faster convergence compared to SGD.

3.2.4 Backpropagation

Calculation of the gradient was mentioned multiple times in the previous chapter but considering the highly non-linear structure of neural networks, the computation of the gradient is an evident challenge. In practise, the gradient in neural networks is calculated using backpropagation which is an algorithm that computes the gradient of the loss function $\nabla_{\boldsymbol{\theta}} L(f(\boldsymbol{x}; \boldsymbol{\theta}), y)$ efficiently using the chain rule.

The goal of the backpropagation algorithm is to find the value of $\frac{\partial L}{\partial w_{ij}^l}$ for each weight w_{ij}^l from node i on layer $l-1$ to node j on layer l. The weighted input of node j in layer l before activation is

$$z_j^l = \sum_{k=1}^{n_{l-1}} w_{kj}^l a_j^{l-1} + b_j^{l-1} \tag{3.12}$$

For cleaner notation, the bias is implemented as an additional node to each layer $w_{0i}^l = b_i^l$ with its output fixed as a constant $a_0^{l-1} = 1$ and we use the chain rule to write

$$\frac{\partial L}{\partial w_{ij}^l} = \frac{\partial L}{\partial z_j^l} \frac{\partial z_j^l}{\partial w_{ij}^l} \tag{3.13}$$

Here, the first part is often labeled as the error term δ_k^l and the latter is calculated easily to get

$$\frac{\partial L}{\partial w_{ij}^l} = \delta_j^l a_i^{l-1} \tag{3.14}$$

where a_i^{l-1} is the output of node i in layer $l-1$.

The value for the error term is also calculated via chain rule, in this case with respect to the next layer $l+1$:

$$\delta_j^l = \frac{\partial L}{\partial z_j^l} = \sum_{i=1}^{n_{l+1}} \frac{\partial L}{\partial z_i^{l+1}} \frac{\partial z_i^{l+1}}{\partial z_j^l} \tag{3.15}$$

$$= \sum_{i=1}^{n_{l+1}} \delta_i^{l+1} \frac{\partial z_i^{l+1}}{\partial z_j^l} \tag{3.16}$$

where n_{l+1} is the number of nodes in the layer $l+1$. From this expression it is clear that the error term δ_j^l depends on error terms in the next layer δ^{l+1} and the gradient propagates backwards in the network. Inserting the expression for

$$z_i^{l+1} = \sum_{k=1}^{n_l} \sigma\left(w_{ki}^{l+1} z_k^l\right) \tag{3.17}$$

gives the following error term δ_j^l

$$\frac{\partial L}{\partial z_j^l} = \sum_{i=1}^{n_{l+1}} \frac{\partial L}{\partial z_i^{l+1}} \frac{\partial \sum_{k=1}^{n_l} \sigma\left(w_{ki}^{l+1} z_k^l\right)}{\partial z_j^l} \tag{3.18}$$

$$= \sum_{i=1}^{n_{l+1}} \frac{\partial L}{\partial z_i^{l+1}} w_{ji}^{l+1} \sigma'(z_j^l) \tag{3.19}$$

Combining the two results, the partial derivative $\frac{\partial L}{\partial w_{ij}^l}$ for each weight is obtained:

$$\frac{\partial L}{\partial w_{ij}^l} = \sigma'(z_j^l) a_i^{l-1} \sum_{k=1}^{n_{k+1}} w_{jk}^{l+1} \delta_k^{l+1} \tag{3.20}$$

Using this recursive logic, all parameters of the network can be computed while progressing backwards through the layers. Using dynamic programming, i.e. storing intermediate values for the parameters, each partial derivative has to be calculated only once which makes the algorithm very efficient memory-wise.

3.2.5 Hyperparameter optimization

When designing a machine learning model, there are numerous architectural choices the developer can make. For example, the number of layers, the number of neurons in each layer , the learning rate and the choice of activation functions can all be tuned. Practically, every component of a neural network has some parameter which may be adjusted for the problem at hand. These parameters are called hyperparameters and the choice of the hyperparameters can drastically affect the performance of the model. The difference between hyperparameters and model parameters is that the hyperparameters have to be set before the training process while model parameters, i.e. weights and biases, are learned from data during the training.

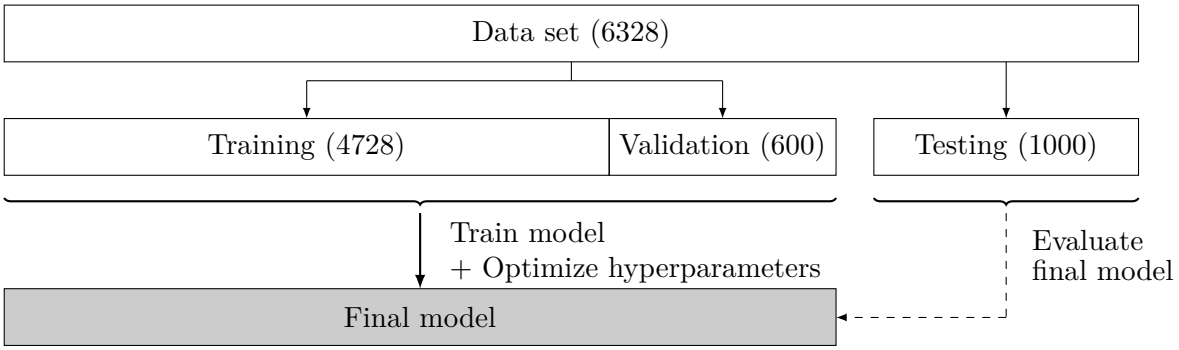

Figure 3.5: Flowchart showing how the data set splitting relates to the model training process. Figures in parentheses show the number of molecules of each data set.

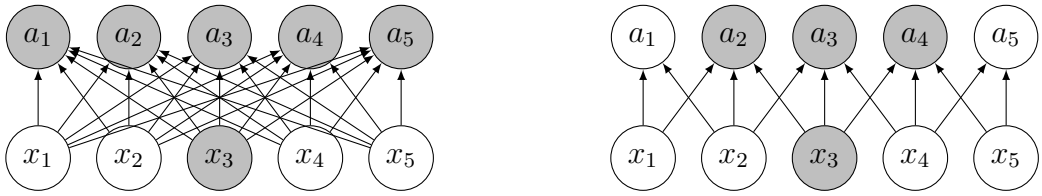

Figure 3.6: Left: Connections between two layers in a fully connected network. Right: Connections between layers in a convolutional neural network (Kernel width = 3). Recreated from [34].

To find the optimal hyperparameters, the training set is further split into a training and a validation set. The model parameters are learned on the training set and the model performance is measured on the validation set. The hyperparameters are fine-tuned so that the performance on the validation set is maximized. The test set should be only used after the optimal model and its hyperparameters have been selected to ensure that the test set performance represents an unbiased estimate of performance on unseen data. Figure 3.5 highlights how the data set is split and how each data set contributes to the model training process. It also shows how the entire data set of 6328 molecules is split. In reality, the data set partitions are 30 times larger due to the rotation augmentation.

3.3 Convolutional neural networks

Convolutional neural networks (CNNs) are a useful tool when learning from data with strong local connections. Fully connected multilayer perceptrons typically perform badly in these tasks since all neurons in a layer are connected to all nodes of the next layers with individual weights meaning that even a slight spatial shift in the data example might completely change the output of an individual neuron. By contrast, in CNNs the neurons are only connected to spatially closely located neurons in the next layer which emphasises local features in the dat. The second important feature of CNNs is shift invariance which is achieved through weight-sharing meaning the weights of a filter are shared between locations in a layer. Figure 3.6 illustrates the difference in connections between layers for CNNs and MLPs. The figure also shows how a value in the latter layer is affected by all neurons in the previous layer in a fully connected network, but in a CNN the kernel size defines the number of connections. Kernels and other details of CNNs are described in this section. The reduced number of connections also help with the *curse of dimensionality*: increasing number of nodes exponentially increases the number of parameters to be optimized thus making fully connected networks computationally inefficient.

A typical example task for 2-dimensional CNNs is image classification. For example, one might want to classify images based on whether they contain a cat or a dog. In principle, it should not matter where the animal is located in the image, but the model should learn the local features that separate cats from dogs. The importance of shift invariance in CNNs is apparent here: if an edge is deemed significant in one part of the image, the edge is probably significant anywhere. Previously, CNNs have been successful in many applications, such as X-ray image analysis [37], character recognition [38] and in a related topic to this project, AFM image interpretation [7].

The term convolution refers to the convolution operation $(f * g)$ but in a

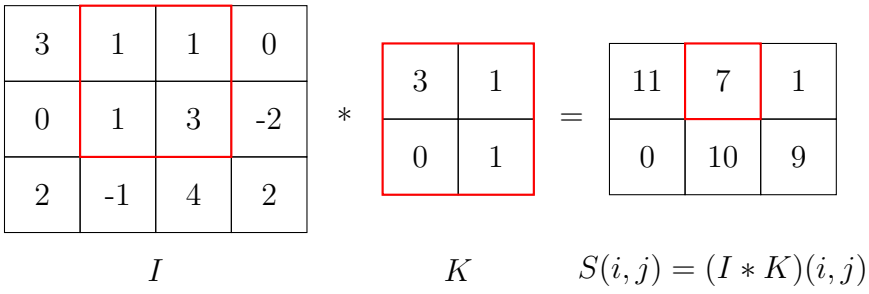

Figure 3.7: The kernel K slides across the input I creating the output S

mathematical sense, the operation in convolutional neural networks is actually cross-correlation which is a similar operation. In the field of deep learning, and also in this thesis, convolution is used to mean cross-correlation.

A convolution operation has two arguments: (1) the input I, for example an image which is a two dimensional grid with each value corresponding to the brightness of that pixel and (2) the kernel K which is also a two dimensional grid. In the operation, the kernel is "slid" across the input, and the output is formed as the sum of element-wise products of the input and the kernel:

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(i+m, j+n)K(m,n) \tag{3.21}$$

Here, m and n refer to the size of the kernel. Figure 3.7 shows how the operation is carried out. The image is often padded with zeros or using reflective boundary conditions before the convolution to keep the image dimensions unaltered. One convolutional layer typically contains multiple kernels, or filters, from which the output is passed to the next layer. The number of kernels in each layer, along with the size of the kernel and the stride used in convolution, are hyperparameters of the model. A convolutional layer can be used on data with any number of dimensions with the same logic.

Convolutional neural networks typically consist of convolutional blocks containing one or more convolutional layers, activation functions and pooling layers. The reasoning and details of using an activation function are discussed in section

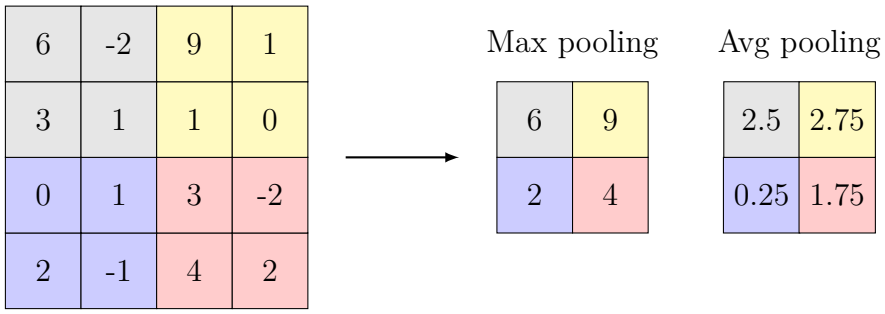

Figure 3.8: A 2-by-2 pooling operation on a 4-by-4 grid using max pooling and average pooling

3.2.2.

A pooling layer reduces the output dimension of a convolutional layer by replacing the output with a summary of the local environment around each pixel. Pooling is used to make the model approximately invariant to minor translations in the input which is often a desired quality. Pooling layers in consecutive blocks help to gradually increase the field of view of convolutional filters to collect more general properties from the input data. However, if the exact location of some feature is important, excessive pooling has to be avoided in the model to preserve the spatial information in a wider context. Another benefit of pooling is that since the spatial size of data deep in the network is reduced, a large number of filters can be applied allowing for more abstract features to be detected without increasing the computational cost.

Probably the most popular pooling method is the max pooling operation in which the maximum of the local filter space is used as the output. One other popular pooling operation is the average pooling in which the average of the local filter space is reported. These methods are shown in figure 3.8. The gradient of the loss function in convolutional networks is calculated using backpropagation and any of the previously mentioned parameter optimization methods can be used.

3.4 Graph neural networks

3.4.1 Motivation for graphs

Convolutional neural networks perform well at finding hidden patterns in data in Euclidean form. One example of this type of data is an image which is represented as a two dimensional grid. However, when the data is in non-Euclidean form and the interdependencies in the data are complex, a different machine learning model is required to capture the hidden patterns. For example, the connections between individuals in a social network and bonds between atoms in a molecule are non-Euclidean data structures. The input data used for this thesis is in euclidean form consisting of stacks of 2D AFM images but the underlying atoms and chemical bonds have natural graph representations.

Graphs are an abstract data type which models entities and the connections between them. Mathematically formulated, a graph is a 2-tuple $G = (V, E)$ consisting of vertices V and edges E where edge is a tuple of vertices defining its two endpoints, each of which might be associated with a weight a_i. In addition, $|V| = N$ is the number of vertices and $|E| = N^e$ is the number of edges. A graph can be also represented as an adjacency matrix $\mathbf{A} \in \mathbb{R}^{N \times N}$. For a simple graph, each entry $A_{ij} \in \{0, 1\}$ specifies whether there is an edge from vertex v_i to v_j. Graphs are either directed or undirected depending on whether the edges are oriented. Vertices are also interchangeably called nodes.

Another motivation for graph based learning in addition to being able to generalize to non-Euclidean data structures is graph representation learning which aims to capture graph features by representing the nodes in a graph as a low-dimensional vector. The node embedding vectors can contain information about the local graph structure and the node features. In the case of graph representation learning, there are two additional properties for a node v: $\mathbf{h}_v$ which is the hidden state vector and

$\mathbf{o}_v$ which is the output vector of the node.

Graph neural networks (GNNs) are an emerging field of research not as well-established as CNNs [39]. Still, there exists numerous different variants of GNNs with different model architectural choices. For example, the Graph Auto-Encoder [40] (GAE) is an auto-encoder model that reconstructs the adjacency matrix using an encoder-decoder type architecture. GAEs have been used for semi-supervised node classification tasks. Another usage are user recommendation systems: for example, a graph convolutional neural network is used at Pinterest to recommend images to users [41]. In chapter 4, the GNN used in this project is described in more detail.

3.4.2 Recurrent relational networks

Recurrent relational network is a general graph representation based method that learns about objects and their interactions in a graph. It learns to encode a node into an embedding vector which can be further processed. In a 2017 paper, Palm et al. [42] describe the model and use solving a Sudoku as an example application.

A Sudoku problem is represented as a graph consisting of nodes $i \in 1, 2, \cdots 81$ and edges from each node to nodes in the same row, column and box. Each node also has an input vector $\mathbf{x}_i$ which initialized based on whether the number in the cell is given or solved for. In principle, a node gathers messages from its connected cells in a message passing scheme which should inform the node which numbers are allowed, and the network should solve the puzzle based on the allowed choices.

The node encoding is an iterative process meaning that multiple message passing operations are required. The idea behind the message passing scheme is related to weight-sharing of CNNs but since the data here is in non-euclidean form the grid shaped filters need to be replaced by a more abstract data format and instead, the nodes share a part of the messages with their neighbors. At the start of the iteration,

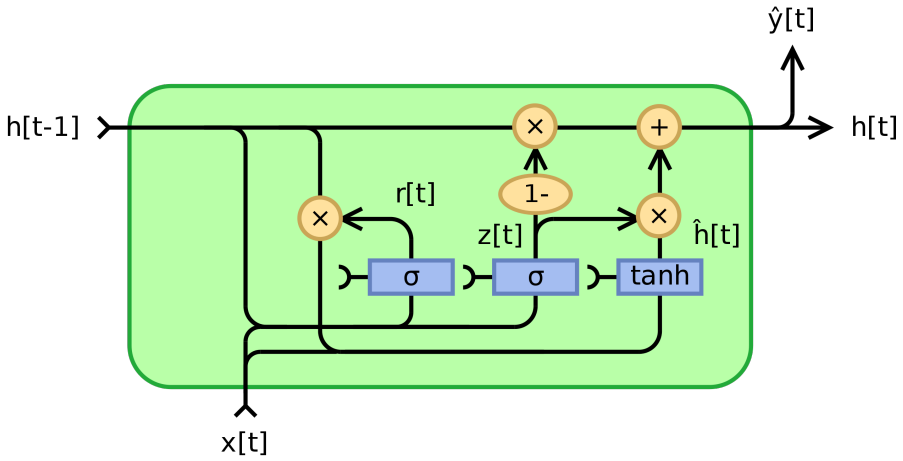

Figure 3.9: Structure of a GRU network. The gates of the GRU remember node states over iteration steps. [43]

the hidden state vector of each node is initialized as $\mathbf{h}_i^0 = \mathbf{x}_i$. At each iteration step t, the messages m_{ij} from each node i to neighboring nodes j are calculated

$$m_{ij}^t = f(h_i^{t-1}, h_j^{t-1}) \tag{3.22}$$

where f is a learned MLP message function. A learned message functions allows the model to decide on which features and connections in the graph to focus. In the message passing phase, the nodes aggregates all incoming messages

$$m_j^t = \sum_{i \in N(j)} m_{ij}^t \tag{3.23}$$

where $N(j)$ is the set of nodes connected to node j. At the end of each iteration, the hidden state of each node is updated. The states can be updated with a MLP as is done by Palm et al. [42] but other methods can be used as well. One such approach is by using a gated recurrent unit (GRU) [44]. A GRU is a neural network architecture similar to long short-term memory [45]. It is used to "remember" node states over iteration steps. Figure 3.9 shows the structure of a GRU unit. The

updated hidden states $\mathbf{h}^t$ are calculated via the following sequence:

$$\mathbf{r}^t = \sigma\left(\mathbf{W}_{ir}\mathbf{x}^t + \mathbf{b}_{ir} + \mathbf{W}_{hr}\mathbf{h}^{t-1} + \mathbf{b}_{hr}\right) \tag{3.24}$$

$$\mathbf{z}^t = \sigma\left(\mathbf{W}_{iz}\mathbf{x}^t + \mathbf{b}_{iz} + \mathbf{W}_{hz}\mathbf{h}^{t-1} + \mathbf{b}_{hz}\right) \tag{3.25}$$

$$\hat{\mathbf{h}}^t = \tanh\left(\mathbf{W}_{i\hat{h}}\mathbf{x}^t + \mathbf{b}_{i\hat{h}} + \mathbf{r}^t \odot (\mathbf{W}_{h\hat{h}}\mathbf{h}^{t-1} + \mathbf{b}_{h\hat{h}})\right) \tag{3.26}$$

$$\mathbf{h}^t = (1 - \mathbf{z}^t) \odot \hat{\mathbf{h}}^t + \mathbf{z}^t \odot \mathbf{h}^{t-1} \tag{3.27}$$

The reset gate $\mathbf{r}^t$ is calculated by first multiplying the current input vector $\mathbf{x}^t$ with the input-hidden weight matrix W_{ir} and the previous hidden state vector with the hidden-hidden weight matrix W_{hr}. The corresponding biases b_{iz} and b_{hz} are added and finally the sigmoid function translates the gate values to range between 0 and 1. The weight matrices and biases are learnable and separate parameters are kept for each gate. The update gate $\mathbf{z}^t$ is calculated similarly but using the trainable weight matrices $\mathbf{W}_{iz}$ and $\mathbf{W}_{hz}$, and biases $\mathbf{b}_{iz}$ and $\mathbf{b}_{hz}$. The new gate $\hat{\mathbf{h}}$ is otherwise calculated in the same way but the sum of the hidden terms is multiplied with the reset gate using Hadamard product $\odot$. Finally, the updated hidden state $\mathbf{h}^t$ is calculated by multiplying the element-wise inversion of the update gate with the new gate and adding the Hadamard product of the update gate and the previous hidden state.

4. Generative Graph Model

The previous chapters discuss the theoretical background for atomic force microscopy and machine learning as well as the data generation scheme via Probe Particle Model. This chapter ties together the two topics and describes the machine learning model used for interpreting the AFM images.

4.1 Iterative scheme

In a previous study, Alldritt et al. [7] have used machine learning to interpret AFM images as a set of image descriptors containing information about the geometry and chemical properties of the sample. This thesis aims to expand this study by predicting exact atomic coordinates of the molecules in a sample and also reveal the chemical bonds between the atoms.

Predicting the atomic coordinates as a list of xyz positions with a CNN model is a tempting idea but it is not a realistic approach since a list of coordinates is not a convenient descriptor for an ANN to process. Some of the challenges are that each molecule would have an output vector of different size, and permutating the output vector would result in a seemingly different molecule despite the same physical structure.

Graphs can consist of a different number of nodes and edges, and various GNN methods are capable at processing graphs of varying sizes. Also, graphs are inherently permutation symmetric. Another obvious reason for choosing a graph

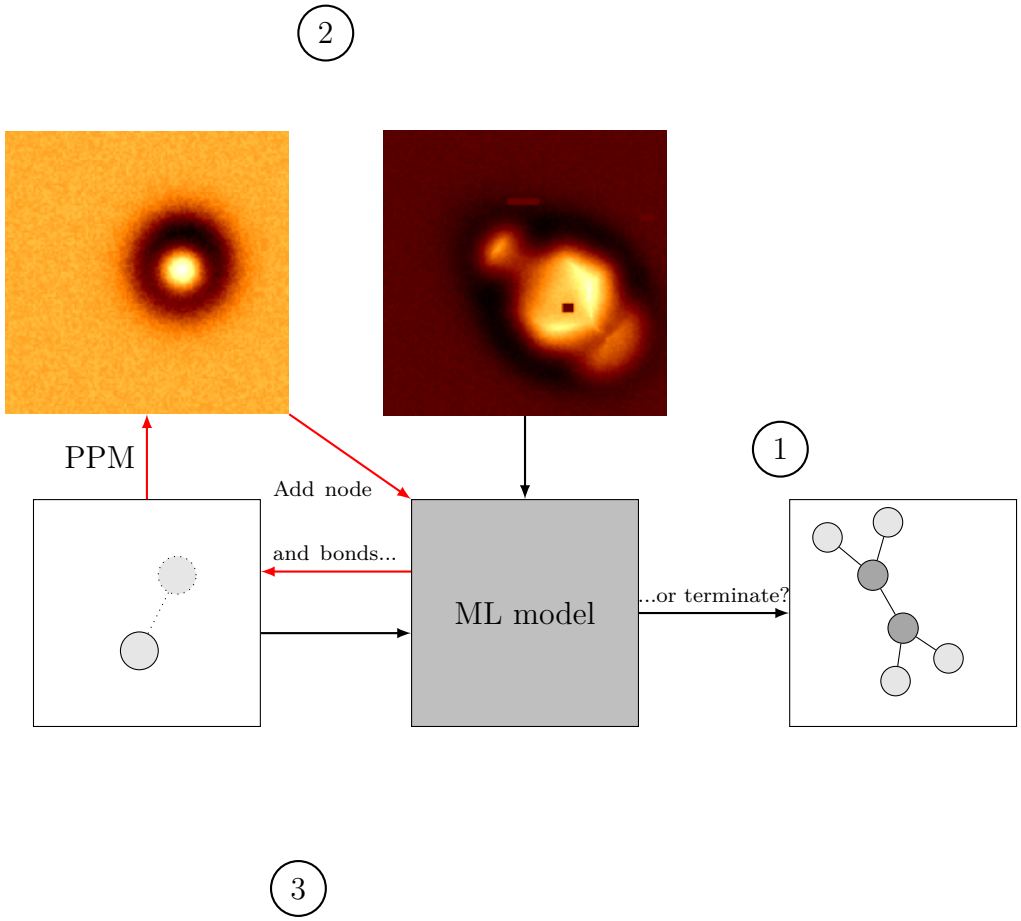

Figure 4.1: The iterative scheme used in the thesis. The red arrows highlight the iterative structure of the algorithm. The numbered arrows show the three inputs for the machine learning model.

based method is that a molecule can be naturally represented as a graph: the atoms are the nodes of a graph and the bonds are the connecting edges.

The machine learning model used in this thesis closely follows the generative graph model of Li et al [46]. Qualitatively, the scheme of the iterative model is the following:

i. The AFM image and an empty molecule graph are given as input to the machine learning model.

ii. The ML model decides whether to add a node and associated edges to the graph or to terminate the generative process based on the inputs.

iii. If a node is added at step (ii), Probe Particle Model is used to take a new AFM image of the incomplete molecule

iv. The reference AFM image, the newly simulated AFM image and the updated graph are given as input to the machine learning model.

v. Go to step (ii)

The idea at the core of the model is that the molecule prediction is generated atom-by-atom in an iterative scheme as shown in figure 4.1 and the model should learn the mapping between a small mutation in the molecule and the corresponding change in the AFM image. At a conceptual level, once the reference AFM image and the newly simulated AFM image are similar, the model has correctly built the reference molecule and the generative process is terminated at step (ii).

This scheme begs the question: how does the ML model decide what is the correct action to take in a given situation? As figure 4.1 shows, the ML model has three input branches: the reference AFM image (1), the newly simulated AFM image (2) and the updated graph (3). Each branch is processed to form an encoding vector which should contain the information of the specific branch in an abstract

representation. The vectors are used as inputs for the decision making. Next, the insides of the model will be explained by focusing on one input branch at a time, starting with the input graph.

4.2 Model architecture

4.2.1 Input branches

Input graph

At the start of the iteration the graph is empty and the graph is zero-encoded. At the second iteration and onwards the node features are computed using a single linear layer with the node positions as input and 20 output parameters

$$\mathbf{h}_v^0 = f_e(x_v) \tag{4.1}$$

where x_v is a containing the position and type of node v. Then, the recurrent relational network as described in the section 3.4.2 is used to propagate the messages and update the node features in the graph over 3 message passing iterations to aggregate messages from a larger neighborhood

$$\mathbf{h}_V^3 = \text{prop}(\text{prop}(\text{prop}(\mathbf{h}_V^0, G), G), G) \tag{4.2}$$

where each propagation $\text{prop}(\mathbf{h}_V^t, G)$ returns the updated node features $\mathbf{h}_V^{t+1}$, and $\mathbf{h}_V^t$ contains the node features of the entire graph G ($\mathbf{h}_V^t = \{\mathbf{h}_1^t, ..., \mathbf{h}_{|V|}^t\}$).

The node features are then expanded to a higher dimensionality

$$\mathbf{h}_v^G = f_m(\mathbf{h}_v) \tag{4.3}$$

where f_m is an MLP with a single hidden layer with 32 units and ReLU activation, and 128 output units. The individual node features are aggregated to form a single feature vector for the entire graph. For the aggregation, a separate gating network

is used

$$\mathbf{g}_v^G = \sigma(g_m(\mathbf{h}_v)) \tag{4.4}$$

for each node v, where σ is the logistic sigmoid activation and g_m is another MLP with 32 hidden units, ReLU activation and 128 output units. The dimensionality of the node feature vector is increased (Eq. 4.3) since the graph contains more information than single nodes.

The final graph encoding vector is obtained as a gated sum of the individual node features

$$\mathbf{h}^G = \sum_{v \in V} \mathbf{g}_v^G \odot \mathbf{h}_v^G \tag{4.5}$$

Additionally, the AFM image input branch requires a query vector for the attention mechanism as explained next. The query vector is computed from the graph encoding vector through an MLP

$$q = f_q(\mathbf{h}^G) \tag{4.6}$$

Input AFM images

The reference AFM image and the newly simulated AFM image are both encoded by an attention-gated CNN. An attention gate is another neural network which learns to selectively concentrate on the important parts of the input [47,48]. In this thesis attention gates are used to let the CNN focus on the most vital regions of an AFM image.

Figure 4.2 shows the structure of the CNN. The first 3D convolutional layer has a shape $128 \times 128 \times 10$ corresponding to the spatial dimension 128×128 of the AFM scan while 10 is the number of scanning heights. While the 3D convolutional layers are computationally expensive, they capture important features present in adjacent slices of the image stack which otherwise could not be utilized. The increasing

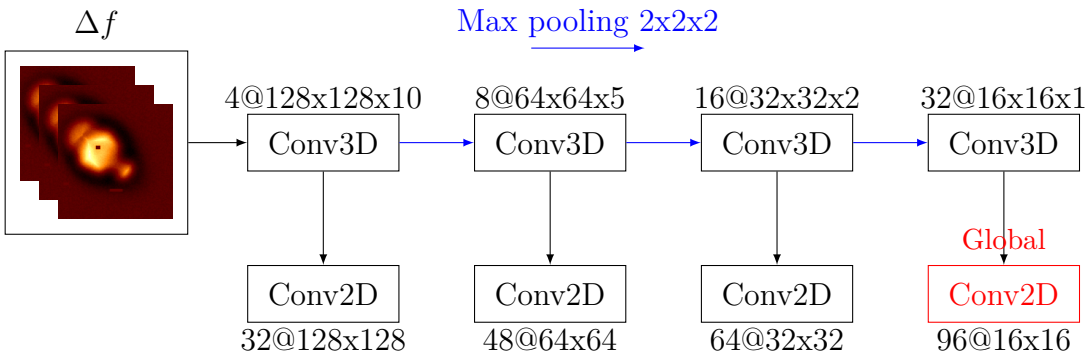

Figure 4.2: Structure of the CNN. The numbers next to the layers are in format ($N_{filters}$@XxYxZ). Max pooling operation with a 2x2x2 kernel is applied between the 3D convolutional layers. The output from the deepest 2D convolutional layer is referred to as the global feature map.

number of filters in the deeper layers of the network allows the model to capture more abstract features in the input image. Each convolutional layer is accompanied by a ReLU activation unit and after each 3D layer a max pooling operation is performed with a 2x2x2 kernel. Each output channel of the 3D layers is flattened and used as separate input channels to the corresponding 2D layers so that the number of input channels to the first 2D layer is $4 \times 10 = 40$, to the second $8 \times 5 = 40$ etc. ReLU activation function is also used for the 2D convolutional layers. Outputs of the 2D convolutional layers are called feature vectors. The output of the coarsest 2D convolutional layer (16×16), highlighted in red in figure 4.2, is referred to as the global feature map. The gating mechanism is illustrated in figure 4.3.

Each feature vector x^l is gated separately i.e. the attention mechanism is applied to each layer l of the CNN output including the global feature map g. As the dimensionality of the global feature map g is 16×16, to make it compatible with the larger feature vectors it is upsampled using bilinear interpolation. When applying the attention gate to the global feature map, upsampling is not necessary.

After the global feature map is upsampled to the correct dimensionality, it is processed through a 2D convolutional layer with ReLU activation after which

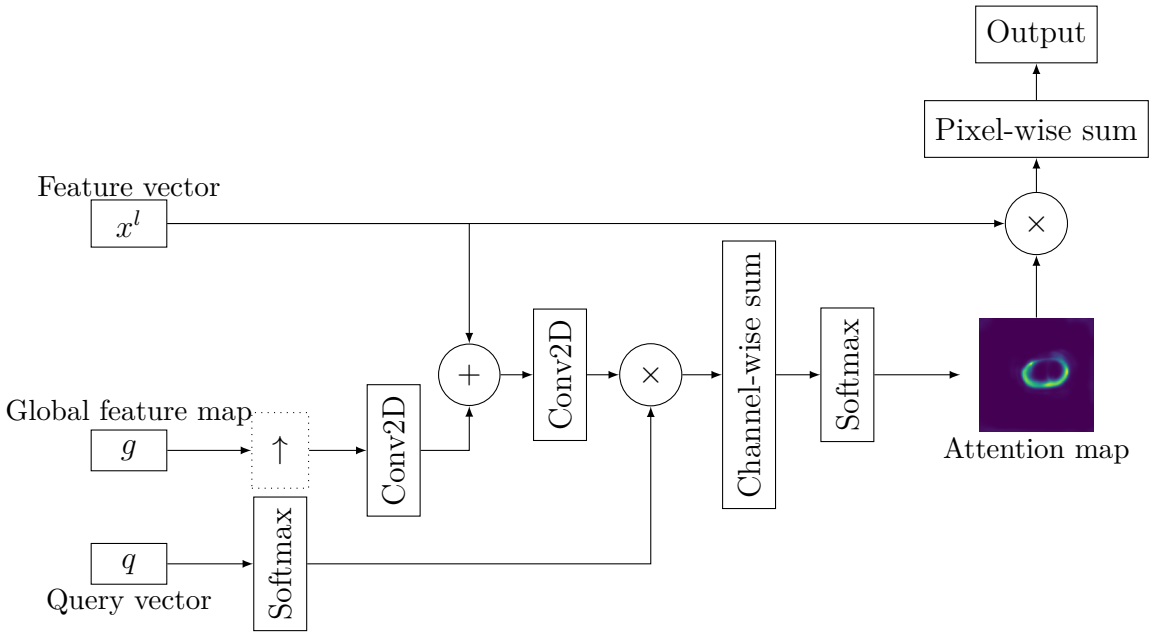

Figure 4.3: Flowchart of the attention gating mechanism.

the global features are added to the feature vector x^l. A 2D convolutional layer is applied to the sum $x^l + g$ and each channel of the sum is multiplied by the corresponding component of the query vector q to obtain the attention map. Before the multiplication operation, a softmax is applied to the query vector so in principle, the query vector functions as an attention gate on the channels of the convolutional feature maps. At this point, the attention map contains multiple channels but the pixels of the map are summed channel-wise to leave only a single channel before applying a softmax to the ultimate attention map. The attention map is multiplied with the feature vector x^l and a pixel-wise sum is applied to the result to get the output of the attention gate a.

Now there are attention-gated feature vectors for each layer in the CNN. The feature vectors are then concatenated and an MLP f_i is applied to the concatenated vector

$$x = \mathrm{ReLU}(f_i(a)) \tag{4.7}$$

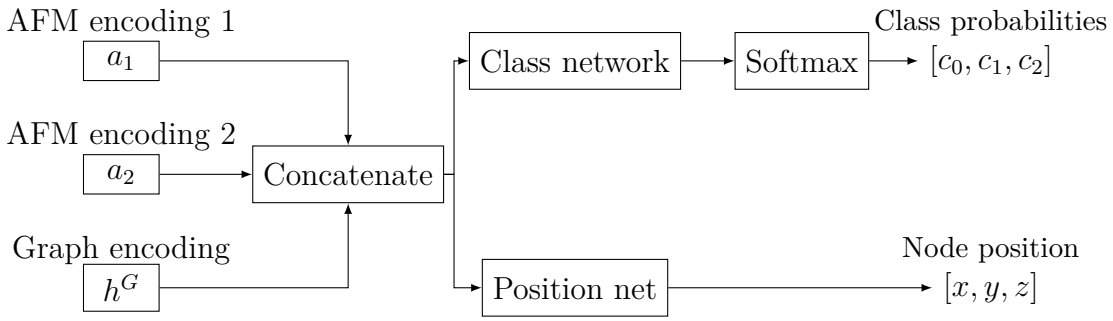

Figure 4.4: Structure of the output branches.

to get the image encoding vector x. The network f_i is a single linear layer with 128 output units.

The newly simulated AFM image, the reference AFM image and the graph are all encoded using their respective encoding networks and the resulting vectors are concatenated to form the complete encoding vector encompassing information about the AFM images and the current graph.

4.2.2 Output branches

While the input encoding networks are rather complicated, computing the output of the network is simple. As illustrated in figure 4.1, the model should either add one node and the required edges to the graph or terminate the generative process. The decision to add a node or to terminate is made via a classification network and the location of the added node via a regression network. The output processing network from encoding vector concatenation to the output layers is shown in figure 4.4.

The classification network is an MLP with 32 hidden units with ReLU activation. A softmax function is applied to the classification network so that its output can be interpreted as a probability. Predicting class c_0 means terminating the generative process. The atom type prediction is limited to distinguish between hydrogen atoms to class c_1 and all heavier atoms to class c_2. The position network has two hidden layers, both with 64 hidden units and ReLU activation. Readout of the po-

sition network is straightforward since the output layer consists of 3 units, each of which corresponds to one spatial coordinate of the added atom.

One component still missing from the graph are the edges between nodes, and more specifically to which nodes should the new node connect. To achieve this, the position and class vectors are concatenated and the node encoding network of Eq. 4.1 is used to obtain the node features of the new node. Then, the messages in the old graph are propagated once again and the new node features are concatenated to the features of each old node. The combined node features $\mathbf{x}_{\mathbf{c}V}$ are the input for an edge classification net

$$\mathbf{e}_V = \sigma(f_{\text{edge}}(\mathbf{x}_{\mathbf{c}V})) \tag{4.8}$$

where the sigmoid activation function σ normalizes the output to range $[0, 1]$ such that there is a connection from the new node u to an old node $v \in V$ if $\mathbf{e}_v > 0.5$. Once the edges are added to the graph, we have passed one iteration in the generative scheme and a new iteration starts.

The architecture of the machine learning model used in this thesis has now been explained starting from encoding the three input branches and ending in the readout of the output branches. The model was implemented with PyTorch [49] and the Adam optimizer was used to compute the gradients with PyTorch default parameters (learning rate $= 1 \times 10^{-3}$, $\beta_1 = 0.9$, $\beta_2 = 0.999$). Total number of parameters in the model was 750,335. The model was trained on a NVIDIA Tesla v100 GPU on Triton at Aalto University [50].

4.3 Training and evaluation

The task of building a molecule from scratch atom-by-atom is a challenge for a machine learning model since the reward of correct geometry prediction only comes after the generative process, leading to a very delayed feedback: a badly positioned

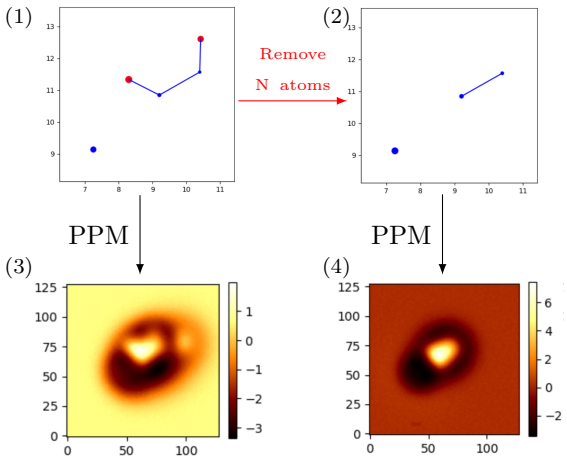

Figure 4.5: Example of a training data object.

atom at the start of the generation will lead to a bad prediction in the end, but from the perspective of the model, it does not have sufficient information to discover the source of the error. Another issue is the comparison of different sized graphs as a loss function i.e. how to evaluate predictions with varying numbers of atoms? Is predicting too few atoms in exactly the right position better than identifying the correct number of atoms but predicting their positions badly?

The aforementioned ambiguities led us to make the design choice of training the model in single-atom steps. This means that in one data example in the training data set, there are (1) the original molecule generated by a coupled-cluster calculation, (2) a "mutant" which is generated by randomly removing atoms from the original, and (3, 4) the AFM images from both molecules generated by PPM. In terms of the iterative scheme (Fig. 4.1), the graph representation of the mutant is the updated graph, the AFM image from the mutant is the newly simulated image and the original AFM image is the reference, while the original molecule is the reference graph the model tries to predict. Figure 4.5 shows one such data example. The number of atoms to remove from the reference is chosen randomly so that the model learns to make consistent predictions at any iteration.

Since the tip in AFM scanning is only sensitive to the top atoms of the sam-

ple and the images do not contain information from deeper atoms, one additional challenge in interpreting AFM images is that identifying lower atoms is difficult. Thus, we have made the decision of excluding atoms that are below $z_{min} = -0.7$ Å when the topmost atom is fixed at $z = 0$ Å. While some information is lost, a good prediction of the surface layer should be obtained.

Since the model predicts three quantities, the loss function is composed of three parts, one for node regression, node classification and edge classification each. For the positional regression task, mean squared error is used

$$L_{pos} = (x_{true} - x_{pred})^2 + (y_{true} - y_{pred})^2 + (z_{true} - z_{pred})^2 \tag{4.9}$$

Predicting the z-coordinate is inherently more difficult than predicting coordinates in the scanning plane. Thus, a discount $d = -\frac{z_{true}}{2z_{min}} + 1$ is applied to the loss function. The discount is larger when predicting deeper atoms and smaller when predicting surface level atoms.

The node classification task is evaluated using cross-entropy

$$L_{node} = -\sum_{i=0}^{2} c_{i,true} \log c_{i,pred} \tag{4.10}$$

where c_i corresponds to the relevant classes. The edge classification task is evaluated with binary cross-entropy

$$L_{edge} = -\frac{1}{m} \sum_{i=1}^{m} b_{i,true} \log b_{i,pred} + (1 - b_{i,true}) \log(1 - b_{i,pred}) \tag{4.11}$$

where $b_i = 1$ if there exists a bond between the predicted node and node i, and $b_i = 0$ otherwise. In the regression evaluation, the loss is calculated from the nearest atom in the reference to the prediction. For the classification tasks, the type and bonds of the nearest atom are used as the reference. Combining the individual loss functions we get the loss function for the model

$$L = dL_{pos} + 3L_{node} + 2L_{edge} \tag{4.12}$$

where the factors for the node and edge losses ensure a balanced evaluation for each contributor.

The model should learn to predict accurately one of the missing atoms in the mutant in any situation. The training set also includes instances where none of the atoms were removed and the model should learn to terminate to process when necessary. From these single-atom additions, the model should generalize to being able to predict a whole sequence of additions from scratch to the complete molecule. In the next chapter results from the single-atoms additions, in addition to the results from the generative processes, are shown.

5. Results and discussion

5.1 Single atom predictions

While a single-atom prediction is only the training procedure of the model, it is important to know how the model performs since the training performance sets an upper limit for the generative prediction performance. The single-atom prediction accuracy was also used for model selection, since ranking the models based on sequence predictions was difficult due to the ambiguous comparison of graphs as mentioned previously.

The model was trained on the molecule data set described by table 2.1 and figure 3.5. The training data set consisted of 4728 data examples, the validation set of 600 and the testing set of 1000 examples, each augmented with 30 rotations using Eq. 2.13. Figure 5.1 shows the total loss function and its contributors as a function

	Training	Validation	Testing
Position	2.067×10^{-1}	2.007×10^{-1}	1.936×10^{-1}
Node class	0.979×10^{-1}	0.934×10^{-1}	0.881×10^{-1}
Edge class	0.347×10^{-1}	0.330×10^{-1}	0.350×10^{-1}
Total	5.698×10^{-1}	5.469×10^{-1}	5.279×10^{-1}

Table 5.1: Training, validation and testing losses after 50 training epochs before applying the loss factors.

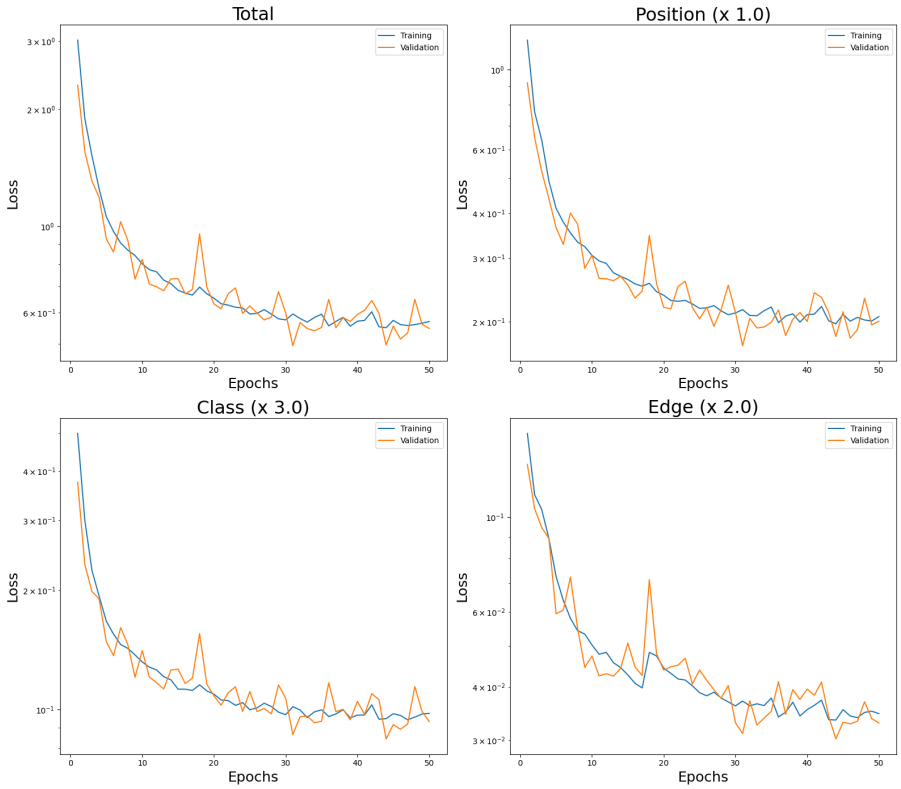

Figure 5.1: Training and validation losses as a function of training epoch. Top left: Total loss. Top right: Node position mean-squared loss. Bottom left: Node classification cross-entropy loss. Bottom right: Edge classification binary cross-entropy loss. The numbers in parentheses are the corresponding loss factors.

Position errors	|r| Å	|x| Å	|y| Å	|z| Å
Mean	0.50	0.29	0.29	0.14
Median	0.40	0.20	0.21	0.11
Std	0.38	0.31	0.30	0.12
90^{th} %	0.94	0.66	0.63	0.32
95^{th} %	1.19	0.87	0.84	0.39
99^{th} %	1.94	1.43	1.43	0.52

Table 5.2: Statistics of the node position predictions. Here, $|r|^2 = |x|^2 + |y|^2 + |z|^2$.

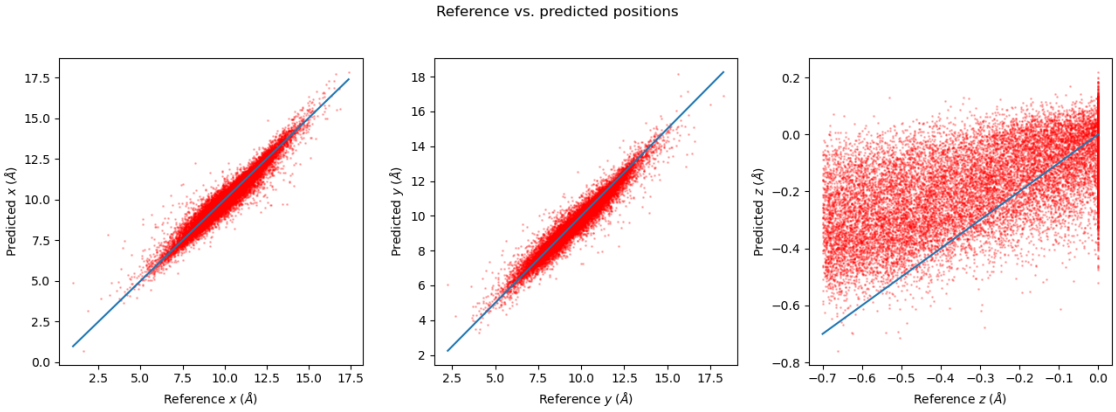

Figure 5.2: Parity plot of the reference coordinate versus predicted coordinate. Left: x-coordinate. Center: y-coordinate. Right: z-coordinate.

of the training epoch and table 5.1 shows the training, validation and testing losses after 50 training epochs. The learning curves show that the model does not improve significantly anymore after 50 epochs at which point the training was stopped.

Since the model should iteratively generate the molecule atom-by-atom, the node placement accuracy for single-atom additions is important. Table 5.2 shows statistics of the node placement predictions for the testing set. It shows that the mean position error is 0.50 Å. The error in the scanning plane direction is larger than in the z-direction. This is explained by the sizes of these dimensions, the scanning place spans over 15 Å in both directions whereas the surface layer atoms we consider in the predictions lie in the region -0.7 Å $< z < 0$ Å. Thus, the error relative to

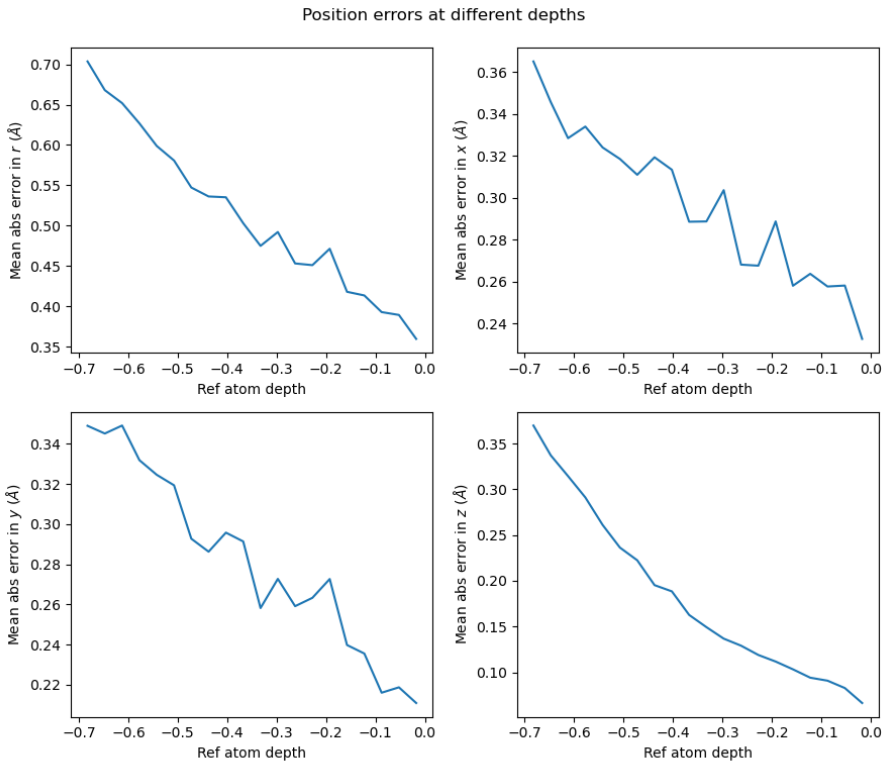

Figure 5.3: The mean error in each direction as a function of atom depth. Top left: Error in r. Top right: Error in x. Bottom left: Error in y. Bottom right: Error in z.

the prediction range is larger in the z-direction. The difficulty of the z-coordinate prediction is also highlighted in the parity plot of figure 5.2 where the true position of a node is on the x-axis and the predicted position of that node is on the y-axis. The predictions of both horizontal coordinates are generally quite accurate but the model does not learn to predict the vertical coordinate. A worse performance in the vertical direction was expected since an AFM scan is sensitive in vertical direction to the top atoms but the probe cannot penetrate the sample and information about the deeper atoms is lost.

One more interesting result from the position predictions is how the depth of a predicted atom affects the prediction accuracy. This is shown in figure 5.3 which presents the increasing difficulty of position prediction the deeper we go vertically. In all directions, predicting atoms close to the surface is considerably easier than

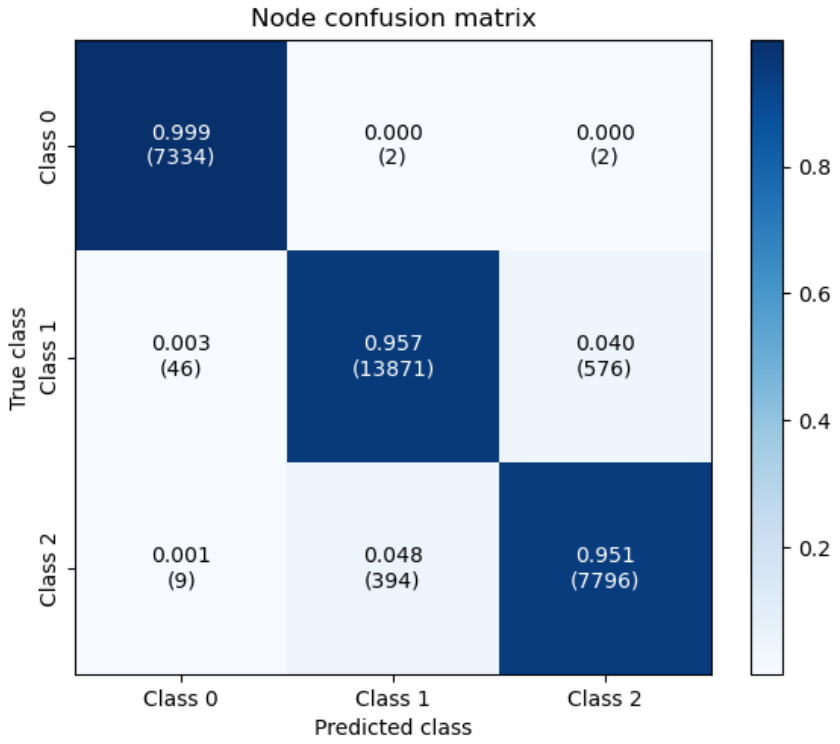

Figure 5.4: Confusion matrix of the node classification task. In each box, the top number shows the fraction of predictions that fall into each class while the number in parentheses show the total number of such cases.

predicting deep atoms. The total error r is twice as large at $z = -0.7$ Å than at $z = 0$ Å. The tip-sample interaction is the strongest for the surface layer atoms and degrades rapidly which explains the lack of information about the deeper atoms in a molecule and the resulting bad performance in predicting deep atoms.

In addition to the node position prediction, the model also classifies the nodes into classes of hydrogen atoms (class 1) and all heavier atoms (class 2). Class 0 represents the termination class for the generative process. The node classification performance is shown in figure 5.4 in a confusion matrix format. A confusion matrix shows the performance of a classification algorithm in a matrix where each column represents a predicted class while each row represents the true class.

For all classes, the true positive rate is higher than 95 % which is very good. Especially the termination class is predicted very reliably and there are few false

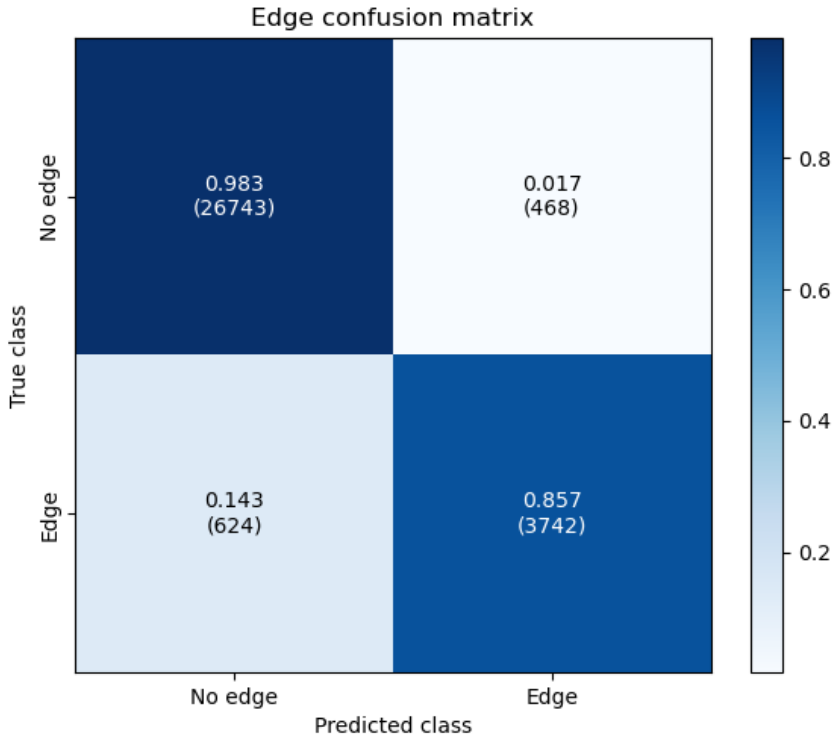

Figure 5.5: Confusion matrix of the edge classification task. In each box, the top number shows the fraction of predictions that fall into each class while the number in parentheses show the total number of such cases.

positives or negatives. In the training phase if no atoms were removed during the mutation, the two input AFM images would seem similar to the model and the difference would be the random pre-processing applied to the images, and in that case the model should be able to terminate the process with high confidence.

The excellent distinguish-rate between hydrogen atoms and heavier atoms is however surprising since the difference between a missing hydrogen atom and a missing heavier atom is not obvious in an AFM image.

Edge classification is the third task for the model and the accuracy of correctly identifying bonds from the added atom is presented in figure 5.5 as a confusion matrix. The true positive rate for "no edge" is 98.3 % and for edge class it is 85.7 %, 14.3 % of the edge classes are misclassified as "no edge". It seems that the model is biased towards predicting no edges. From the case numbers in parentheses we see

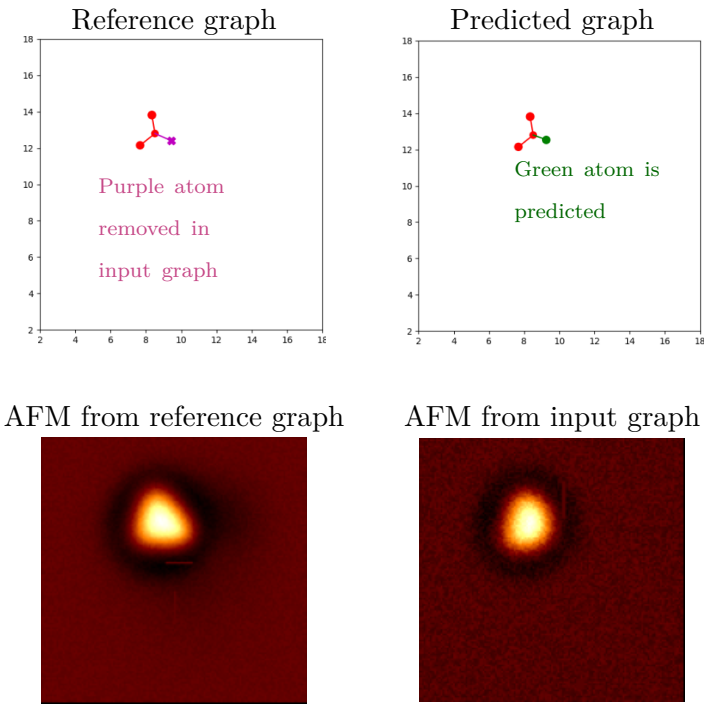

Figure 5.6: Single-atom prediction. Top left is the reference graph from which the reference AFM image (bottom left) is taken. To get the input graph the purple node is removed of which the second input AFM image (bottom right) is taken. The top right image shows the predicted graph where the green node is the newly added node.

that "no edge" is the predominant class in the data set also which might steer the model to overemphasize those predictions.

To give an example of a single-atom prediction, figure 5.6 shows the inputs and the resulting prediction of the model. The reference graph is shown in the top left image while the input graph consist of the red nodes i.e. the node in purple was removed from the reference. This is also the node that the model should predict. In this example only one node was removed, but the number of removed nodes is chosen randomly. The figure also shows the AFM images both from the reference graph and the input graph. The prediction is shown in the top right image where the predicted node is highlighted in green. The z-coordinate of a node is represented by its relative size: a larger node means that the atom has a higher z-coordinate.

Comparing the two AFM images by eye, one can observe that the shape of the bright region changes and might assume that an atom has been removed in that area. In this example, the model has come to the same conclusion. The exact location of the prediction is not completely accurate and the atom has been predicted too close to the other atoms.

5.2 Generative predictions

Using these single-atom additions, the machine learning model should generalize to make intelligent predictions in a generative process. The eventual goal of the model is to predict the molecule geometry from a truly unknown experimental image but in this thesis we are limited to predictions from simulated data where the reference geometry is known. Due to the challenges in comparing the predicted graph to the reference graph quantitatively, in this section three example generative predictions are presented, one best-case, one approximately average and one worst-case scenario. Each example will be accompanied by discussion on the details of the prediction.

The first example in figure 5.7 is the best-case scenario prediction. The graph prediction sequence by iteration is shown on the top row and below each graph is the corresponding AFM image. The bottom right image is the reference AFM image which we try interpret and next to it is the reference geometry. The final prediction is shown in the bottom left corner.

The predicted molecule has generally the same geometry as the reference which is a success for the model. Each of the predicted atoms is correctly classified but one atom from the reference is clearly missed, but looking at the reference AFM, the contrast from the missed atom is insignificant. This probably happens because the closest neighboring atom is located closer to the tip and the missed atom is left in its shadow. Looking into the details of the prediction we see that each individual atom is also slightly off from its reference position and there is no bond between the

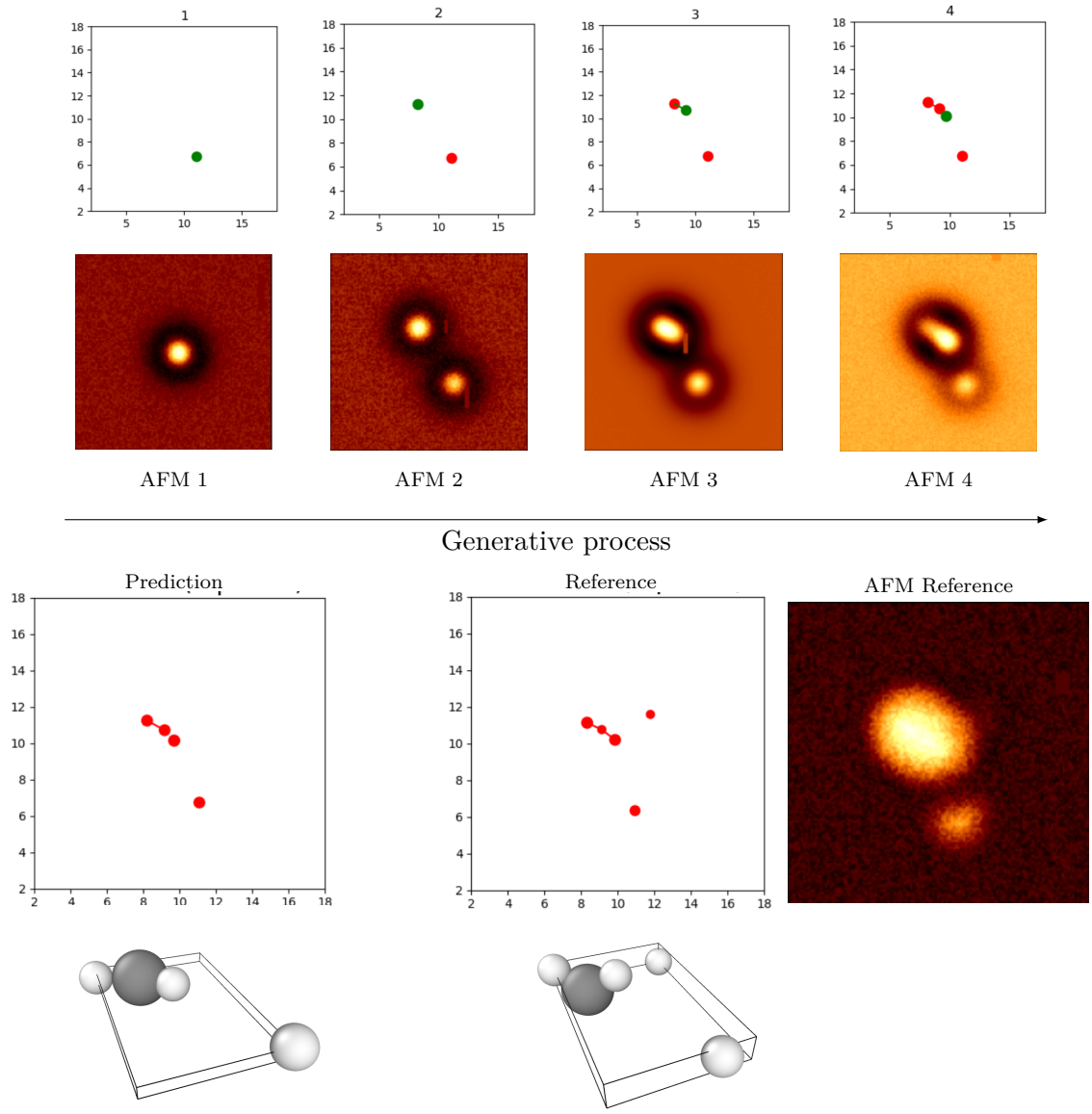

Figure 5.7: Best-case scenario. The topmost rows show the graph generation sequence and the corresponding AFM images. The bottom row shows the predicted graph, reference graph, and the input AFM image. The graphs are accompanied by 3D rendered images of the atoms.

carbon atom and one of the hydrogen atoms where there should be. The single-atom predictions showed that there is variance in the node placement and bond classification accuracies which explain the deviation from the reference. Additionally, the reference molecule shows vertical deviation between the atoms which is missed in the prediction. Here, the difficulty of predicting the vertical coordinates becomes apparent and the model essentially disregards the vertical component and predicts all nodes at approximately the same height. This is best seen in the 3D rendered image of the prediction. Generally, when the reference molecule is small i.e. the part above the vertical threshold $z_{min} = -0.7\,\text{Å}$ consists of 1-3 nodes, the predictions are quite robust and the placement of the nodes is consistently accurate.

The next example is from an approximately average prediction. The prediction sequence in figure 5.8 is presented in the same format as the previous example. Again the two first rows show the generative process and the last row shows the reference geometry and AFM image in addition to the predicted geometry.

The triangular shape of the reference geometry is retained in the prediction but the actual positions and the number of predicted nodes is not accurate. This prediction also highlights the challenge of AFM image comparison as the reference AFM image and the last AFM image from the sequence have very similar properties: both images have a general round shape and there is a region of lighter contrast at the left edge.

Probably the most significant flaw in the model is that it does not know physics. The model has no inherent information of how a molecule is constructed or what are allowed positions for hydrogen atoms connected to a carbon atom, but instead the model should learn this from the thousands of example geometries. Partially, this is an advantage as we allow the model to come to its own conclusions and we let the model decide independently what are the important features in the data. On the other hand, this leaves room for the model to make unphysical decisions. For

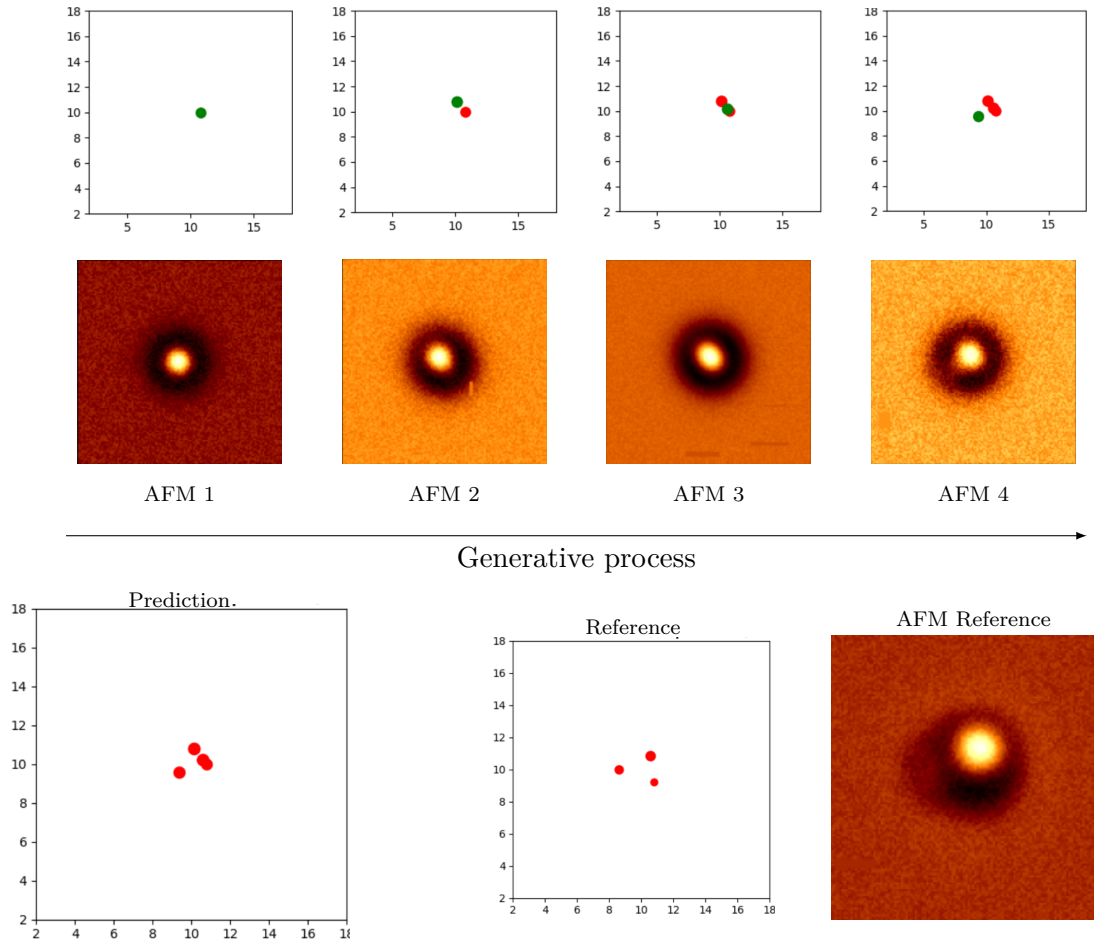

Figure 5.8: An approximately average case prediction. The topmost rows show the graph generation sequence and the corresponding AFM images. The bottom row shows the predicted graph, reference graph, and the input AFM images.

example in this prediction, where in the third iteration the model has placed an atom on top of an existing atom. This is obviously not an allowed position. The vertical coordinates of the predicted atoms are again approximately the same.

The third example shows a worst-case scenario prediction. The AFM images from the sequence are left out for this example and figure 5.9 shows only the graph generation sequence, the reference molecule and the reference AFM image. Immediately one can observe that the generative process has not produced an intelligent prediction of the reference and many atoms are placed in the same position. A closer inspection into the predicted coordinates reveals that the atoms are placed always slightly above the previous atom in z-direction. Again, this comes back to the lack of physical intuition in the model.

To solve this issue, we tried to implement a computationally inexpensive pseudo-physical potential as another input for the model, essentially giving the model information about physically viable positions for new atoms. However this did not erase the erratic behaviour of the model and since it increased the training time by 20 %, the model presented in this thesis does not use this additional input.

Definitive statements about predictions of a machine learning model are difficult to make. The massive amount of parameters and layers make it next to impossible to establish why a certain prediction has been made. Still, some intuitive guesses for this chaotic behaviour can be offered. The model in this thesis is deterministic, meaning that a certain set of inputs will always result in the same prediction.

For example, consider the inputs to the model after iterations four and five from which the newly simulated AFM images are shown in figure 5.10. At iteration 4 based on the current inputs, the model decides to place an atom at approximately $(x, y) = (10.5, 10)$ which would seemingly be translated to the brightest region of the reference AFM from figure 5.9. After this addition, AFM 5 is obtained from a

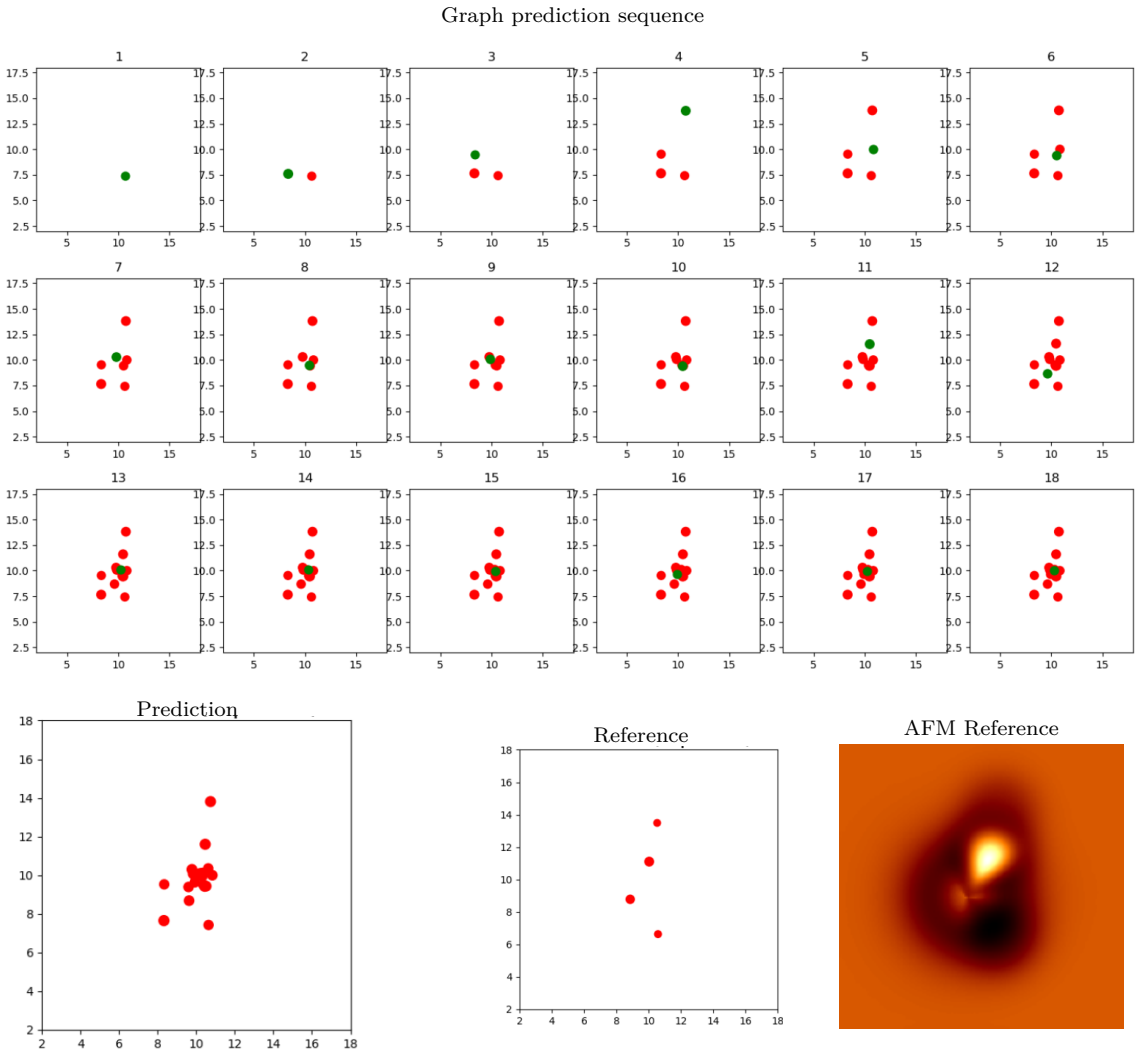

Figure 5.9: A worst-case scenario prediction. The top images show the graph generation sequence. The images at the bottom are the predicted graph, reference graph and the reference AFM image.

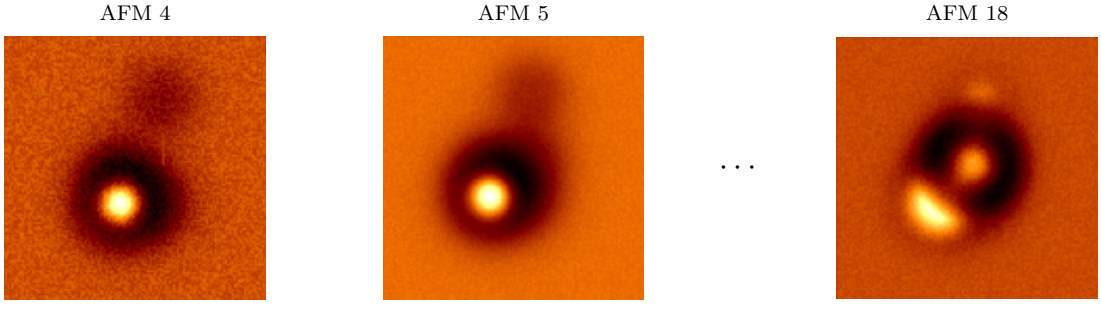

Figure 5.10: Three AFM images from the worst-case scenario prediction. Left from iteration 4, center from iteration 5. The right image shows the AFM image from the final 18th prediction.

PPM simulation and used as an input for iteration five. However, comparing AFM 4 and AFM 5 by eye, the difference in contrast in that region is not significant at least when compared to the reference AFM image. This means that the set of inputs for iteration 5 is nearly identical as the previous iteration and the model makes another prediction in the same region, eventually resulting in the model adding multiple atoms in the same position. The iteration is eventually stopped after 18 iterations, and the AFM image corresponding to the final prediction is shown rightmost in figure 5.10.

One more observation from comparing the reference AFM image to either AFM 4 or AFM 5 from the sequence is that the brightest region is at a different location. Generally, the brightest region corresponds to the atom closest to the tip and thus it seems that correctly identifying the top atom of a molecule is vital for a good prediction. Unfortunately, the model often predicts the first atom close to $z = 0\,\text{Å}$ which is the z-coordinate of the top atom regardless of which atom it is trying to predict. This means that if the model does not predict the top atom first, later it will most likely have trouble trying to replicate the bright features of the top atom in the AFM image when there are already atoms near $z = 0\,\text{Å}$.

6. Conclusions

In this thesis we present an image-to-graph translation method for atomic force microscopy image interpretation. In general, for human experts AFM images are difficult and time-consuming to interpret manually. This motivated the thesis for which the objective was to develop a tool to automate the process of identifying sample geometries. The eventual goal for the developed tool is to assist experimentalists and make the AFM image interpretation process more efficient and robust. However, in this thesis we were limited to consider only simulated AFM data.

The results in this thesis are separated into two parts. First of these is the results for single-atom addition which acts as the training phase for the generative model which learns to construct a molecule atom-by-atom. The single-atom predictions are important as they set an upper limit for the prediction performance in the generative phase. The accuracy of atom placement in the scanning plane direction was significantly more accurate than in the vertical direction. This was expected as AFM images inherently contain less information in the tip oscillation direction. In addition to the atom placement, all atoms were classified as either hydrogen or something heavier. The classification was done with an error rate of less than 5 % suggesting that the model learned with very high confidence what distinguishes light atoms from heavier ones. In further studies, the classification task can be expanded to account for more atom types. To conclude the single-atom prediction results, the obtained predictions were accurate and showed that the model has the capability to

learn features from AFM images and translate those features into a geometry.

While the single-atom predictions are interesting, the main results in the thesis are from the generative predictions. In this thesis, we presented three different predictions corresponding to roughly the best possible prediction, average prediction and the worst-case prediction.

The best-case prediction was qualitatively very accurate, missing only one atom that was hardly observable from the AFM image even with knowledge of the reference geometry. The most accurate predictions came generally from geometries where the part above the threshold $z = -0.7\,\text{Å}$ consists of three or fewer atoms. The average and worst-case predictions highlight the issues of the model and give possible clues for which parts of the model are most crucial for good performance. This also paves the way for how the model may be improved in further studies. The glaring problem was the bad prediction in the worst-case scenario. Some possible explanations for the chaotic behaviour included the deterministic nature of the model and the difficulty in predicting the z-coordinate of an added atom. The z-coordinate prediction accuracy could be improved by varying the hyperparameters in the model. Another approach could be to bias the training data set differently. Currently, the data set is augmented using such rotations that the number of visible atoms in each image is maximized. However, this is not necessarily the optimal augmentation strategy and it could be studied how a different emphasis in the rotations would affect the performance of the model.

Another possible improvement would be to introduce domain knowledge into the model. More specifically, we know how a molecule should be constructed and what are the physically allowed bond lengths and angles for specific atoms. This information could be introduced to the model as another input as was mentioned previously, or alternatively one might use a molecular dynamics simulation in between the iterations to relax the generated molecule and to force the model to stay

within the boundaries of known physics.

To conclude this thesis, we presented a generative graph neural network based model for predicting molecular geometries from AFM images. The model learns to detect and translate features from simulated AFM images to a graph representation. Still, more work has to be done before the tool can be realized in an experimental framework.

References

1 G. Binnig, H. Rohrer, Ch Gerber, and E. Weibel. Tunneling through a controllable vacuum gap. *Applied Physics Letters*, 40(2):178–180, jun 1982.

2 G. Binnig, C. F. Quate, and Ch Gerber. Atomic force microscope. *Physical Review Letters*, 56(9):930–933, mar 1986.

3 T. R. Albrecht, P. Grütter, D. Horne, and D. Rugar. Frequency modulation detection using high-Q cantilevers for enhanced force microscope sensitivity. *Journal of Applied Physics*, 69(2):668–673, jun 1991.

4 Leo Gross, Fabian Mohn, Nikolaj Moll, Peter Liljeroth, and Gerhard Meyer. The chemical structure of a molecule resolved by atomic force microscopy. *Science*, 325(5944):1110–1114, aug 2009.

5 Bruno Schuler, Gerhard Meyer, Diego Peña, Oliver C. Mullins, and Leo Gross. Unraveling the Molecular Structures of Asphaltenes by Atomic Force Microscopy. *Journal of the American Chemical Society*, 137(31):9870–9876, aug 2015.

6 Bruno Schuler, Shadi Fatayer, Gerhard Meyer, Estrella Rogel, Michael Moir, Yunlong Zhang, Michael R. Harper, Andrew E. Pomerantz, Kyle D. Bake, Matthias Witt, Diego Peña, J. Douglas Kushnerick, Oliver C. Mullins, Cesar Ovalles, Frans G. A. van den Berg, and Leo Gross. Heavy Oil Based Mixtures of Different Origins and Treatments Studied by Atomic Force Microscopy. *Energy and Fuels*, 31(7):6856–6861, jul 2017.

7 Benjamin Alldritt, Prokop Hapala, Niko Oinonen, Fedor Urtev, Ondrej Krejci, Filippo Federici Canova, Juho Kannala, Fabian Schulz, Peter

Liljeroth, and Adam S. Foster. Automated structure discovery in atomic force microscopy. *Science Advances*, 6(9), 2020.

8 Prokop Hapala, Ruslan Temirov, F Stefan Tautz, and Pavel Jelínek. Origin of High-Resolution IETS-STM Images of Organic Molecules with Functionalized Tips. 2014.

9 Prokop Hapala, Georgy Kichin, Christian Wagner, F. Stefan Tautz, Ruslan Temirov, and Pavel Jelínek. Mechanism of high-resolution STM/AFM imaging with functionalized tips. *Physical Review B - Condensed Matter and Materials Physics*, 90(8):1–9, 2014.

10 Constant A. J. Putman, Bart G. De Grooth, Niek F. Van Hulst, and Jan Greve. A detailed analysis of the optical beam deflection technique for use in atomic force microscopy. *Journal of Applied Physics*, 72(1):6, jun 1998.

11 F J Giessibl, S Hembacher, M Herz, Ch Schiller, and J Mannhart. Stability considerations and implementation of cantilevers allowing dynamic force microscopy with optimal resolution: the qPlus sensor. *Nanotechnology*, 15(2):S79, jan 2004.

12 Franz J. Giessibl. Advances in atomic force microscopy. *Reviews of Modern Physics*, 75(3):949–983, 2003.

13 Hendrik Hölscher and Udo D Schwarz. Theory of amplitude modulation atomic force microscopy with and without Q-Control. *International Journal of Non-Linear Mechanics*, 42:608–625, 2007.

14 Franz J. Giessibl. Forces and frequency shifts in atomic-resolution dynamic-force microscopy. *Physical Review B - Condensed Matter and Materials Physics*, 56(24):16010–16015, dec 1997.

15 Prokop Hapala, Georgy Kichin, Christian Wagner, F. Stefan Tautz, Ruslan Temirov, and Pavel Jelínek. Mechanism of high-resolution STM/AFM imaging with functionalized tips. *Physical Review B - Condensed Matter and Materials Physics*, 90(8):1–9, 2014.

16 Horacio V. Guzman, Pablo D. Garcia, and Ricardo Garcia. Dynamic force microscopy simulator (dForce): A tool for planning and understanding tapping and bimodal AFM experiments. *Beilstein Journal of Nanotechnology*, 6(1):369–379, feb 2015.

17 Yanyan Wang and Sen Wu. Modeling and Simulation of an Atomic Force Microscopy System in the Z Direction. In *IOP Conference Series: Materials Science and Engineering*, volume 381, page 012089. Institute of Physics Publishing, aug 2018.

18 Kazuya Kobayashi, Yunfeng Liang, Ken Ichi Amano, Sumihiko Murata, Toshifumi Matsuoka, Satoru Takahashi, Naoya Nishi, and Tetsuo Sakka. Molecular Dynamics Simulation of Atomic Force Microscopy at the Water-Muscovite Interface: Hydration Layer Structure and Force Analysis. *Langmuir*, 32(15):3608–3616, may 2016.

19 Bernhard Reischl, Paolo Raiteri, Julian D. Gale, and Andrew L. Rohl. Atomistic Simulation of Atomic Force Microscopy Imaging of Hydration Layers on Calcite, Dolomite, and Magnesite Surfaces. *Journal of Physical Chemistry C*, 123(24):14985–14992, jun 2019.

20 Minjung Kim and James R. Chelikowsky. Simulated non-contact atomic force microscopy for GaAs surfaces based on real-space pseudopotentials. *Applied Surface Science*, 303:163–167, jun 2014.

21 Philipp Schaffhauser and Stephan Kümmel. Simulating atomic force

microscope images with density functional theory: The role of nonclassical contributions to the force. *PHYSICAL REVIEW B*, 94:35426, 2016.

22 Probe particle model code available at. https://github.com/ProkopHapala/ProbeParticleModel/, 2014.

23 Maurice Herlihy and Nir Shavit. *The Art of Multiprocessor Programming*. 2012.

24 John E. Stone, David Gohara, and Guochun Shi. OpenCL: A parallel programming standard for heterogeneous computing systems. *Computing in Science and Engineering*, 12(3):66–72, may 2010.

25 Yann A. LeCun, Léon Bottou, Genevieve B. Orr, and Klaus-Robert Müller. Efficient BackProp. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 7700 LECTURE NO:9–48, 2012.

26 Prasanna V. Balachandran, Benjamin Kowalski, Alp Sehirlioglu, and Turab Lookman. Experimental search for high-temperature ferroelectric perovskites guided by two-step machine learning. *Nature Communications*, 9(1):1–9, dec 2018.

27 Ryo Nagai, Ryosuke Akashi, and Osamu Sugino. Completing density functional theory by machine learning hidden messages from molecules. *npj Computational Materials*, 6(1):1–8, dec 2020.

28 Thomas M Mitchell. *Machine Learning*. McGraw-Hill, Inc., USA, 1 edition, 1997.

29 Ravi Aggarwal, Viknesh Sounderajah, Guy Martin, Daniel S. W. Ting, Alan Karthikesalingam, Dominic King, Hutan Ashrafian, and Ara

Darzi. Diagnostic accuracy of deep learning in medical imaging: a systematic review and meta-analysis. *npj Digital Medicine*, 4(1):1–23, dec 2021.

30 Shun Yang, Wenshuo Wang, Chang Liu, Weiwen Deng, and J. Karl Hedrick. Feature analysis and selection for training an end-To-end autonomous vehicle controller using deep learning approach. In *IEEE Intelligent Vehicles Symposium, Proceedings*, pages 1033–1038. Institute of Electrical and Electronics Engineers Inc., jul 2017.

31 Yi Sun, Xiaogang Wang, and Xiaoou Tang. Deep Learning Face Representation by Joint Identification-Verification. *Advances in Neural Information Processing Systems*, 3(January):1988–1996, jun 2014.

32 Yi Sun, Ding Liang, Xiaogang Wang, and Xiaoou Tang. DeepID3: Face Recognition with Very Deep Neural Networks. feb 2015.

33 David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529, 2016.

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

35 Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In Geoffrey Gordon, David Dunson, and Miroslav Dudík, editors, *Proceedings of the Fourteenth International Conference*

on Artificial Intelligence and Statistics, volume 15 of *Proceedings of Machine Learning Research*, pages 315–323, Fort Lauderdale, FL, USA, 11–13 Apr 2011. PMLR.

36 Diederik P. Kingma and Jimmy Lei Ba. Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*. International Conference on Learning Representations, ICLR, dec 2015.

37 Diego Ardila, Atilla P. Kiraly, Sujeeth Bharadwaj, Bokyung Choi, Joshua J. Reicher, Lily Peng, Daniel Tse, Mozziyar Etemadi, Wenxing Ye, Greg Corrado, David P. Naidich, and Shravya Shetty. End-to-end lung cancer screening with three-dimensional deep learning on low-dose chest computed tomography. *Nature Medicine*, 25(6):954–961, jun 2019.

38 Patrice Y Simard, Dave Steinkraus, and John C Platt. Best Practices for Convolutional Neural Networks Applied to Visual Document Analysis. Technical report, 2003.

39 Michael M. Bronstein, Joan Bruna, Yann Lecun, Arthur Szlam, and Pierre Vandergheynst. Geometric Deep Learning: Going beyond Euclidean data, jul 2017.

40 Thomas N. Kipf and Max Welling. Semi-Supervised Classification with Graph Convolutional Networks. *5th International Conference on Learning Representations, ICLR 2017 - Conference Track Proceedings*, sep 2016.

41 Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L. Hamilton, and Jure Leskovec. Graph Convolutional Neural Networks for Web-Scale Recommender Systems. *Proceedings of the ACM*

SIGKDD International Conference on Knowledge Discovery and Data Mining, 10:974–983, jun 2018.

42 Rasmus Berg Palm, Ulrich Paquet, and Ole Winther. Recurrent Relational Networks. *Advances in Neural Information Processing Systems*, 2018-December:3368–3378, nov 2017.

43 Jeblad. Gated recurrent unit. https://commons.wikimedia.org/wiki/File:Gated_Recurrent_Unit,_base_type.svg, 2018. License: Creative Commons BY-SA 4.0.

44 Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using RNN encoder-decoder for statistical machine translation. In *EMNLP 2014 - 2014 Conference on Empirical Methods in Natural Language Processing, Proceedings of the Conference*, pages 1724–1734. Association for Computational Linguistics (ACL), jun 2014.

45 Sepp Hochreiter and Jürgen Schmidhuber. Long Short-Term Memory. *Neural Computation*, 9(8):1735–1780, nov 1997.

46 Yujia Li, Oriol Vinyals, Chris Dyer, Razvan Pascanu, and Peter Battaglia. Learning deep generative models of graphs, 2018.

47 Ashish Vaswani, Google Brain, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention Is All You Need. Technical report.

48 Jo Schlemper, Ozan Oktay, Michiel Schaap, Mattias Heinrich, Bernhard Kainz, Ben Glocker, and Daniel Rueckert. Attention Gated Networks:

Learning to Leverage Salient Regions in Medical Images. Technical report, 2019.

49 Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. PyTorch: An Imperative Style, High-Performance Deep Learning Library. *Advances in Neural Information Processing Systems*, 32, dec 2019.

50 The calculations presented above were performed using computer resources within the aalto university school of science "science-it" project.