Bachelor Project

Machine learning for interpreting HR-TEM images of two-dimensional materials

Johanne Birk Christensen

Supervisor
Jakob Schiøtz
Professor, Department of Physics

Department of Physics
Technical University of Denmark
October 15, 2024

Approval

Johanne Birk Christensen, s203788

Signature

31. maj 2024

Date

Abstract

Advancements in material science, particularly in semiconductor technology and catalysis, depend on the unique properties of two-dimensional materials like molybdenum disulphide (MoS_2).

This study explores the use of convolutional neural networks (CNN) to identify atomic columns containing a molybdenum atom (Mo), a single (1S) or two suplhur atoms (2S) in high-resolution transmission electron microscopy (HR-TEM) images of graphene-supported MoS_2. By incorporating the frozen phonon method to emulate thermal vibrations and Fourier filtering of the graphene support, HR-TEM images were simulated for training and testing purposes. Three CNN architectures; U-net, U-net++ and MSD-net were evaluated on simulated and real world HR-TEM focal series images.

This study found that U-net, U-net++ and MSD-net achieved a F1-score of above 0.99 with 10^{-11}m precision on simulated data. U-net and U-net++ achieved this using 10^7 parameters, and the MSD-net used 10^5 parameters, with at training set size of 10^4 simulated images.

Qualitative analyses on the prediction of real world HR-TEM images showed that the that three models reached a consensus on periodic features and the assignment of sub-lattices to molybdenum and sulfur. However, the models disagreed on non-periodic features such as the 1S atomic columns and at the edge of MoS_2 flakes.

Acknowledgements

I would like to thank my supervisor, Jakob Schiøtz, for his guidance and support throughout this project. His insightful feedback have been of great importance in shaping this work. I am very thankful for the time and dedication he has provided. Additionally, his seemingly boundless knowledge of all bash commands has saved me from countless formatting woes and has left me in awe of his command-line prowess. I would also like to thank Stig Helveg, for providing the necessary experimental data for this work.

Contents

1 Introduction

Material science is crucial for technological progress, supporting advancements in various industries and fields. Two key areas where material science plays a significant role are in the semiconductor technology and field of catalysis through the exploration of two-dimensional materials. Confining materials to a two-dimensional structure introduces novel properties and enables the fabrication of detailed structures. The semiconductor technology has led to solar cells, light emitting diodes and computer chips [1, 2]. Catalysts accelerate chemical reactions, enabling processes for industries ranging from pharmaceuticals to petrochemicals [3, 4]. Especially two-dimensional molybdenum disulphide (MoS_2) has attained researchers interest due to its superior structural and electronic properties [5]

To further advance in these technological fields, it is essential to have knowledge about the composition, structure and behavior of the two-dimensional materials. Material scientists use tools such as electron microscopy and image processing tools in order to characterize materials. Electron microscopy involves accelerating electrons in an electron beam to the relativistic regime. Upon striking the sample, these high-energy, high-velocity electrons facilitate imaging, enabling detailed analysis of the sample's structure and properties.

High-resolution transmission electron microscopy (HR-TEM) is of great importance for modern materials design and development, serving as the primary mean of directly observing the atomic-scale structure of materials. In HR-TEM the electron wave function progresses through three stages. First the incident wave being the accelerated electrons. Then the wave propagates through the sample, creating the exitwave. Finally, the exitwave traverses through lenses and is projected onto an image plane. Traditional methods struggle with the pixel intensities in HR-TEM images, making it challenging to apply techniques like thresholding for region separation and analysis.

Machine learning and especially deep learning has revolutionized fields such as healthcare [6], natural language processing [7] and image analysis [8]. Deep learning methods such as convolutional neural networks has shown to be promising for material analysis [9, 10]. The high-energy electrons in the microscope affect the sample by directly damaging it or inducing chemical reactions. To circumvent this, ensure reproducibility and a substantial dataset, the images used to train and test the model is simulated used the multislice method [11].

A previous study [12] investigated the use of convolutional neural networks for reconstruction of the exitwave. This work proved the use of deep learning neural networks as a successful tool for analyzing atomic-resolution image sequences obtained from transmission electron microscopes. It also underscores the limitations of neural networks and emphasizes the significance of modeling noise parameters to optimize results.

This project will incorporate the frozen phonon method and Fourier filtering in order to imitate the thermal vibrations of two-dimensional graphite supported MoS_2. The simulated data is generated using the multislice method. Three convolutional neural network architectures; U-net, U-net++ and MSD-net, will be trained on simulated data. Incorporating thermal vibrations and graphite support should result in a more robust convolutional neural network. The goal of the project is to develop a deep learning model that can identify atomic columns containing a Molybdenum atom (Mo), a single (1S) or two suplhur atoms (2S) in HR-TEM images of two-dimensional MoS_2.

2 Theoretical Background

2.1 Modelling of the High Resolution Transmission Electron Microscope

In HR-TEM the electrons traverse as plane waves until they hit the sample [12]. The electrons interact with the potential from the sample. The transmitted wave is denoted the exitwave. The exitwave encapsulates information about the sample's constituents [13]. The exitwave propagates through the objective lens. Subsequently, the transmitted wave, known as the image wave, passes through the objective aperture, functioning as a frequency cutoff. The image wave hits the image plane, where the intensity is measured. If the objective lens is perfect, the emerging wave will exhibit ideal spherical wave behavior, converging onto a single point within the image.

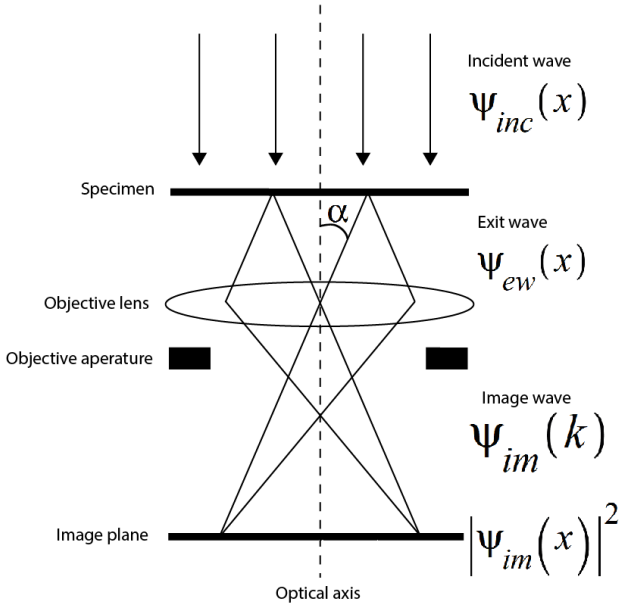


Figure 2.1: Simple model sketch of HR-TEM illustrating the process from incident wave to image wave. (Top), high energy, high velocity electrons passing through specimen. (Middle), The exitwave passing through the objective. (Bottom), Image wave passing through objective aperture down to the image plane into the CCD detector where intensity is measured. Sketch modified from [14]

However the lens has different types of aberrations. These aberrations is the cause of deviations from the ideal lens properties and can be used when characterizing the artifacts observed in the image. The presence of aberrations necessitates consideration during simulation.

For a comprehensive understanding of simulating HR-TEM images, the process is broken down chronologically. First, examining how incident waves interact with the sample. Then, we will explore how the microscope affects these waves, including aberrations. Finally, we will investigate the detection mechanism.

2.1.1 Modelling Specimen Electron Interaction: The Multislice Method

The multislice method is flexible for computer simulation of crystalline specimens with defects [14]. It is numerically efficient because it makes use of the Fast Fourier Transform (FFT).

The electron beam interacts with the specimen through the Coulomb potential of the electrons and the specimen. In the multislice method, the potential of the sample is sliced into smaller sections. The method transmits the electron wave through each slice of the potential, and propagates it to

the next slice. The potential of each slice induces a slight phase shift of the incident electron wave. The wave function of the electron accumulates a total phase shift from each of the slices of the specimens potential. This is just the integral of the potential of the specimen along z, which is an approximation made for thin specimens function [15]. The specimens potential is formed by a linear superposition of each atom's potential, treating each atom as if it were independent of the others.

.

$$v_z(\mathbf{x}) = \sum_{j=1}^{N} v_{zj}(\mathbf{x} - \mathbf{x}_j) \tag{2.1.1}$$

x_j is the j'th atom's position in the x-y-plane, the plane perpendicular to the optical axis z.

The projected atomic potential $v_z(\mathbf{x})$ represents the integral along the optical axis of the specimen:

$$V_{aj}(r) = 2\pi^2 a_0 e \sum_{i=1}^{3} \frac{a_i}{r} e^{-2\pi r\sqrt{b_i}} + 2\pi^{\frac{5}{2}} a_0 e \sum_{i=1}^{3} c_i d_i^{\frac{-3}{2}} e^{-\pi^2 \frac{r^2}{d_i}}, r = \sqrt{x^2 + y^2 + z^2} \tag{2.1.2}$$

$$v_{z_i j}(\mathbf{x}) = v_{z_i j}(x, y) = \int_{z_i}^{z_{i+1}} V_{aj}(x, y, z)\, dz \tag{2.1.3}$$

j is indicating the position of the j'th atom in the x-y-plane. $V_{aj}(r)$ is the potential of the j'th atom with bohr radius a_0, and at element specific constants a_i, b_i, c_i and d_i. $K_0(x)$ is the 0th order modified Bessel function. Modelling the problem with an infinite integral results in an analytical solution for the problem [14]. In this model, the entire atoms potential is assigned to the slice z. As we introduce atomic vibrations, with the frozen phonon method, this will cause discontinuity as the atom transitions between slices. In equation 2.1.3, the atomic potential is distributed across different slices. This approach requires a numerical approach to obtain a solution.

The effect of the specimens potential on the electron's wave function is to multiply the incident wave with the specimen transmission function $t(\mathbf{x})$.

$$\begin{aligned} \psi_{exit}(\mathbf{x}) &= t(\mathbf{x}) e^{2\pi i k_z z} \\ t(\mathbf{x}) &= e^{i\sigma_e v_z(\mathbf{x})} \end{aligned} \tag{2.1.4}$$

The potential of the specimen V_s is much smaller than the probing high velocity and high energy electrons $V \gg V_s$. This means that the electron motion will mostly be along the optic z axis, with a small perturbation from the specimen. It is useful to separate the complete wave equation into two components: a fast-varying part and a slow-varying part. The fast-varying plane wave traveling in the z direction, $e^{\frac{2\pi i z}{\lambda}}$ and remaining portion of the wave function that varies slowly with position z $\psi(x, y, z)$.

$$\psi_f(x, y, z) = \psi(x, y, z) e^{\frac{2\pi i z}{\lambda}} \tag{2.1.5}$$

The multislice algorithm performs the calculations:

$$\psi_{n+1}(x, y) = p_n(x, y, \Delta z_n) \otimes (t_n(x, y)\psi_n(x, y)) \tag{2.1.6}$$

$p(x, y, \Delta z)$ is the propagator function in real space for the distance Δz. $t_n(x, y)$ is the transmission function as in equation 2.1.16 between slice n and $n + 1$.

$\otimes$ is the two-dimensional convolution. It stems from a multiplication in Fourier space that has translated into a convolution in real space.

$$p(x, y, \Delta z) = \frac{1}{i\lambda\Delta z} e^{\frac{i\pi}{\lambda\Delta z}+(x^2+y^2)} \tag{2.1.7}$$

The wave function at slice $n+1$ is calculated by the convolution of the real space propagator function $p_n(x, y, \Delta z_n)$ and the product of the transmission and wave function $t_n(x, y)\psi_n(x, y)$. Given that the slices of the specimen are labeled $n = 0, 1, 2, 3...$, then the depth we have reached in the specimen, z_n corresponds to $n\Delta z$. The incident wave function $\psi_0(x, y)$ is a plane wave for HR-TEM $\psi_0(x, y) = 1$ [16]. Applying equation 2.1.6 repeatedly, results in the exitwave. Using the FFT on equation 2.1.6 results in:

$$\psi_{n+1}(x, y) = \mathcal{F}^{-1}(P_n(\mathbf{k}, \Delta z_n)\mathcal{F}(t_n(x, y)\psi_n(x, y)), \quad \mathbf{k} = (k_x, k_y) \tag{2.1.8}$$

Employing the FFT makes the computation time scale as $Nlog(N)$ instead of N^2 [14]. This saves a lot of computational cost as N grows. The FFT uses a discrete version of the Fourier transform: $\int g(x)dx \to \sum_x g(x_n)$.

2.1.2 Lens Electron Interaction: Aberrations

The emerging electron wave would exhibit perfect spherical wave behavior, if the objective lens was ideal. In practice the lens has imperfections called lens aberrations which cause the electron wavefront to deviate from the perfect spherical wavefront as seen in figure 2.2.

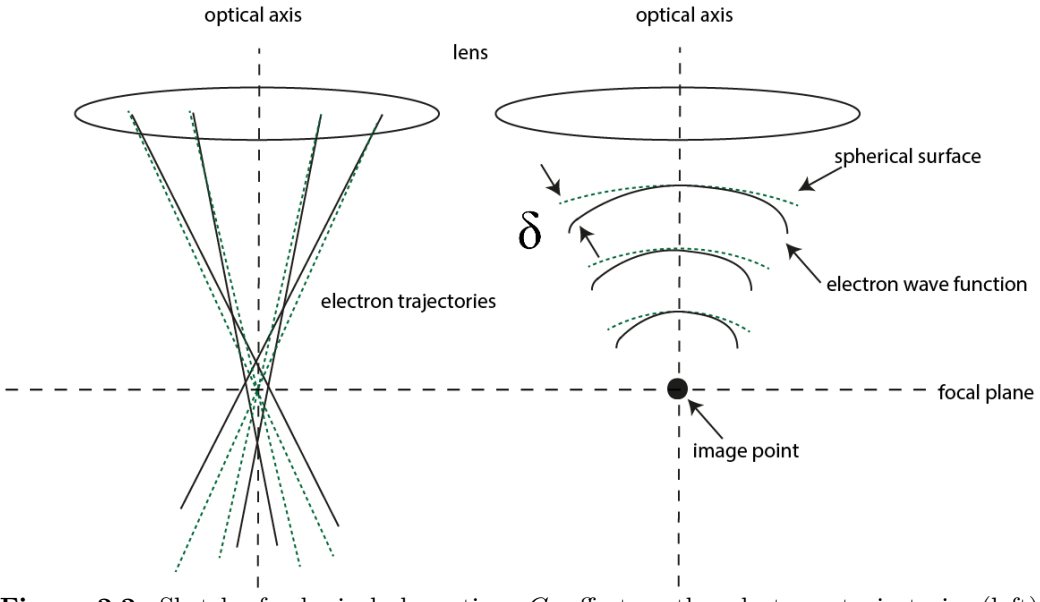


Figure 2.2: Sketch of spherical aberrations C_s effect on the, electrons trajectories (left) and electron wave function (right) . The dashed lines represents the ideal spherical wavefront, and the solid line shows the effect of the abberations. Sketch modified from [14]

These aberrations are influenced by two primary factors: the spatial distance from the optical axis, and the angle formed between the optical axis and the incoming electron rays [14].

The phase error caused by the aberrations can be presented as $\chi = \frac{2\pi}{\lambda}\delta$, where lambda is the de Broglie wavelength.

The phase error can be expressed by the following power series.

$$\chi(\alpha, \phi) = \frac{2\pi}{\lambda} \sum_{mn} \frac{1}{n+1} C_{mn}\alpha^{n+1} cos(m(\phi - \phi_{mn})) \tag{2.1.9}$$

$$\chi(k,\phi) = \frac{2\pi}{\lambda}\sum_{mn}\frac{1}{n+1}C_{mn}(\lambda k)^{n+1}cos(m(\phi-\phi_{mn})) \tag{2.1.10}$$

α is the polar angle, ϕ is the azimuthal angle and k is the spatial frequency. ϕ_{mn} represents the real rotational angle of the aberration, and the constant C_{mn} stands for the type of aberration. Some of the aberrations are listed below:

- C_{10}: Defocus
- C_{12}: Two-fold astigmatism
- C_{21}: Axial coma
- C_{30}/C_s: Third order spherical aberration

These aberrations prevent the rays to converge to a point at the focal plane, see figure 2.2. For the majority of microscopes the net phase error stems from the spherical aberration and defocus as [14]:

$$\chi(\alpha) = \frac{2\pi}{\lambda}\delta(\alpha) \qquad \delta(\alpha) = \frac{1}{4}C_s\alpha^4 - \frac{1}{2}C_{10}\alpha^2 \tag{2.1.11}$$

The aberration function $\chi(\mathbf{K})$ is the deviation from the wavefront of the ideal spherical wavefront. The aberration function in Fourier space, labelled $H_0(\mathbf{k})$, is called the transfer function (CTF).

$$H_0(\mathbf{k}) = e^{-i\chi(\mathbf{k})} \tag{2.1.12}$$

The effect of the aberration on the electron wave function is computed as the product:

$$\psi_{image}(\mathbf{k}) = H_0(\mathbf{k})\psi_{exit}(\mathbf{k}) \tag{2.1.13}$$

The real space wave function is found by applying the inverse Fourier tranform:

$$\psi_{image}(\mathbf{x}) = \mathcal{F}^{-1}(H_0(\mathbf{k})\psi_{exit}(\mathbf{k})) = H_0(\mathbf{x}) \otimes \psi_{exit}(\mathbf{x}) \tag{2.1.14}$$

The imaging of the exitwave will depend on the CTF. The CTF oscillates with frequency $\alpha = k\lambda$. The defocus that maximizes the width of the frequency band to obtain a wide flat band region is called the scherzer defocus [17, 18]. The defocus depends on the spherical aberration C_s and wavelenght λ, see equation 2.1.11. the wavelength in turn depends on the beam energy [14].

The CTF works as a spatial frequency filter on the exitwave. The CTF of figure 2.3 is simulated with Scherzer defocus [19], spherical aberration of $C_s = 20\mu$m and energy E = 80keV of the plane wave. The defocus is chosen to optimize the band of frequencies with the same sign.

To further ensure coherence in the image, the objective aperture is placed after the objective lens, see figure 2.1. The objective aperture limits frequencies over a certain threshold. It works like a piece-wise function, setting all frequencies above the threshold $k \geq k_{cut}$ to 0. The aperture function $A(\mathbf{k})$ models this. This leads to rewriting equation 2.1.13.

$$H_0(\mathbf{k}) = e^{-i\chi(\mathbf{k})}A(\mathbf{k}) \tag{2.1.15}$$

$$A(k) = \begin{cases} 1 & \text{for } k < k_{\text{cut}} \\ 0 & \text{for } k \geq k_{\text{cut}} \end{cases} \tag{2.1.16}$$

Up until now, it has been assumed, that the the incident wave function is a plane wave parallel to the electron beam. If the angle between the optical axis and propagation of the incident beam is

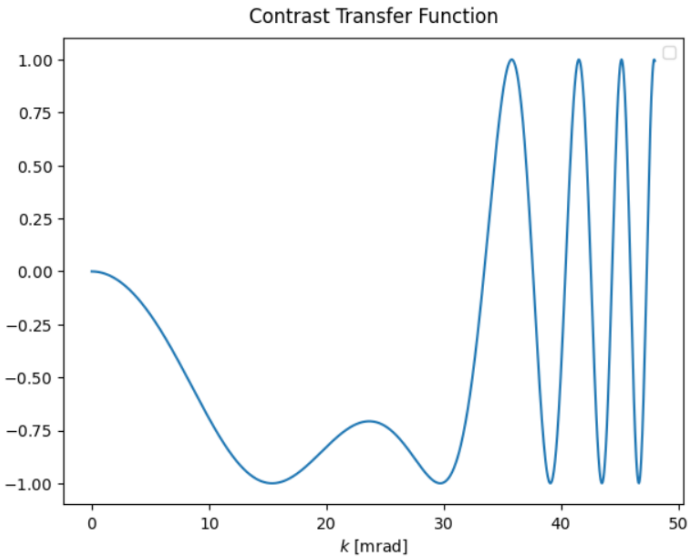


Figure 2.3: Simulated Contrast Transfer funciton (CTF). Simulated with Scherzer defocus, Spherical aberration of $C_s = 20\mu$m, and incident plane wave with energy $E = 80keV$.

sufficiently small we can use a quasi-coherent approximation if we include an envelope function $E(\mathbf{k})$ when calculating the intensity of the image wave function in equation 2.1.15.

The envelope function has three main components: the temporal coherence $E_t(k)$, the spatial coherence $E_s(k)$ and random deflections $E_d(k)$.

Temporal coherence refers to how well the phase of points on different wavefronts of the electron beam correlate in time in the direction of propagation. Lack of temporal coherence can result from slight energy spreads, causing a noticeable spread in the focal length of the objective lens. The temporal coherence is given by:

$$E_t(\mathbf{k}) = e^{-\frac{\Delta^2}{4}(\frac{\partial}{\partial C_{10}})\chi(\mathbf{k})^2} = e^{-\frac{\Delta^2}{4}(\pi\lambda)^2k^4}, \quad \alpha = k\lambda \tag{2.1.17}$$

Where Δ is the focal spread.

$$\Delta \approx C_c\sqrt{(\frac{\Delta E}{E})^2 + (2\frac{\Delta I}{I})^2 + (\frac{\Delta V}{V})^2} \tag{2.1.18}$$

E, I and V are the electron energy, lens currents and acceleration voltage respectively. ΔE, ΔI and ΔV are the $\frac{1}{e}$ width of their fluctuations. The coefficient C_c is the chromatic aberration.

The spatial coherence is given by:

$$E_s(\mathbf{k}) = e^{-\frac{\beta^2}{4\lambda^2}\left|\frac{\partial^2\chi(k)}{\partial k^2}\right|^2} \tag{2.1.19}$$

where β is the spread of angles of the incident electrons with respect to the optical axis. Lastly is the random deflection envelope stemming from specimen drift and magnetic noise. It is assumed to have a Gaussian distribution [11].

$$E_d(\mathbf{k}) = e^{-\frac{\sigma^2k^2}{2}} \tag{2.1.20}$$

σ being the Gaussian spread. We are now able to rewrite equation 2.1.15.

$$H_0(\mathbf{k}) = \psi_{exit}(\mathbf{k})A(\mathbf{k})E(\mathbf{k}), \quad E(\mathbf{k}) = E_t(\mathbf{k})E_s(\mathbf{k})E_d(\mathbf{k}) \tag{2.1.21}$$

2.1.3 Detection

Once the electrons has propagated through the sample and been focused through the microscope they are imaged by a detector. The detector used to detect the image influences the final image. A detector usually used is the charged-coupled device (CCD) detector. The CCD detector uses a scintillator screen, based on phosphor/fibre-optics that converts the electrons into light which can be detected by a camera. When an electron hits the CDD, not only that pixel(s) is activated, but also the surrounding pixels resulting in a blurring effect [11].

This effect can be modelled by the point-spread function (PSF). The PSF describes the response of a detector to a point source with its Fourier transform being the modulation transfer function (MTF).

$$MTF(k) = \frac{1 - c_2}{1 + (\frac{k}{2c_2 k_N})^2} + c_2 \tag{2.1.22}$$

Where $k_N = \frac{s}{2}$ is the Nyquist frequency with s denoting the detectors sampling rate. The MTF acts as a spatial filter and is applied, as for the CTF, by a product in Fourier space.

2.1.4 Frozen Phonon

So far, the atoms in the specimen have been treated as stationary. As most electron microscopy is done at room temperature $\approx$ 300K, the atoms in reality vibrate due to thermal energy.

Optical phonons typically exhibit frequencies not surpassing about 10^{12}Hz to 10^{13}Hz, while acoustic phonons are characterized by considerably lower frequencies [20]. The electrons only traverse through the specimen for about $0.7 \cdot 10^{-16}$s, which is on a much smaller timescale than the oscillations of the atoms in the specimen [14]. The imaging electrons therefore see the atoms as stationary with an offset form their original placement in the lattice.

Using the Debye-model, modelling the vibrations of the atoms as sound waves:

$$\langle x^2 \rangle = \frac{\int_0^{\omega_D} g(\omega) \langle x^2 \rangle d\omega}{\int_0^{\omega_D} g(\omega) d\omega} \tag{2.1.23}$$

Under the assumption that we can model the atoms as the quantum mechanical oscillator, where:

$$\langle x^2 \rangle = \frac{\hbar}{m\omega}(n_B + \frac{1}{2}), \qquad n_B = \frac{1}{e^{\frac{\hbar\omega}{k_B T}} - 1} \tag{2.1.24}$$

Then, using the 2-dimensional density of states $g(\omega)$, and under the assumption, that $T \ll \Theta_D$, we can use the following formula for modelling the atomic displacements [21].

$$\langle x^2 \rangle = \frac{3\hbar^2}{2mk_B\theta_D} \left(\frac{1}{4} + \frac{T}{\theta_D} \right) \tag{2.1.25}$$

m is the atomic mass, θ_D is the Debye temperature and $\hbar$ and k_B are the reduced Planck constant and Boltzmann constant, respectively. The term $\frac{1}{4}$ stems from the zero-point energy of vibration of the lattice, which is a consequence of the quantum theory of the harmonic oscillator.

Using a temperature at random from a proper given range in equation 2.1.25 allows randomly displacing atoms in order to emulate thermal vibrations.

2.1.5 Fourier filtering

In the analysis of certain materials, such as those with overlapping atomic columns, intensity variations lead to smearing along the mass circle of the Argand plot [22]. This challenge is often addressed by removing the substrate from the image using Fourier filtering.

One approach involves training the neural network on images with and without the substrate, treating the substrate as another form of noise for the network to learn to filter out automatically. Alternatively, the substrate can be Fourier filtered out of the image, aiding the network in disregarding it as noise.

The unwanted spatial frequency in the image is specified. Then a filter can be applied. The filters transfer function is often smoothed out to prevent diffraction artifacts, that would arise if the edges were a step function [22].

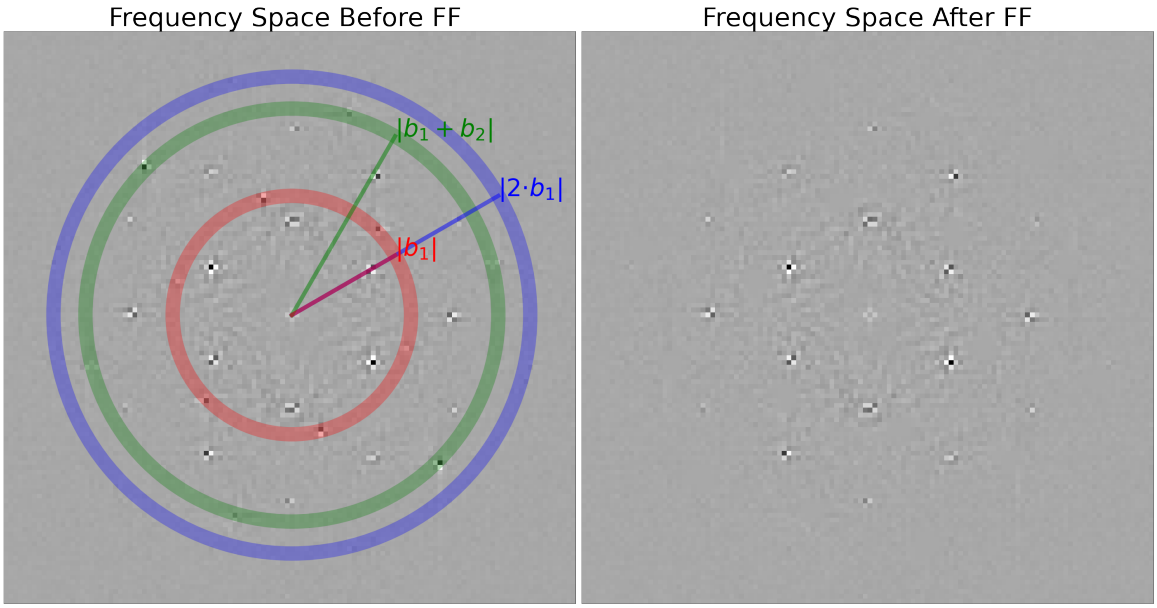


Figure 2.4: Right: First three rings of the hexagonal lattice of graphene in HR-TEM simulation in frequency space. Left: Fourier filtered HR-TEM simulation in frequency space.

Figure 2.4 shows a plot of a simulated HR-TEM image of graphene-supported MoS_2 in frequency space before and after Fourier filtering of the graphene-support. The radius of the first three rings of the hexagonal lattice in reciprocal space is defined by $|b_1|$, $|b_1 + b_2|$ and $|2 \cdot b_1|$, where b_1 and b_2 are the reciprocal lattice vectors of graphene, see appendix A. Given that the angle between the reciprocal lattice vectors is $\frac{\pi}{3}$, then by identifying a peak within the ring and subsequently remove all six peaks within the ring by rotating $\frac{\pi}{3}$, it is possible to find and remove the frequencies stemming from the graphene layer. An example of Fourier filtering is shown in figure 2.5.

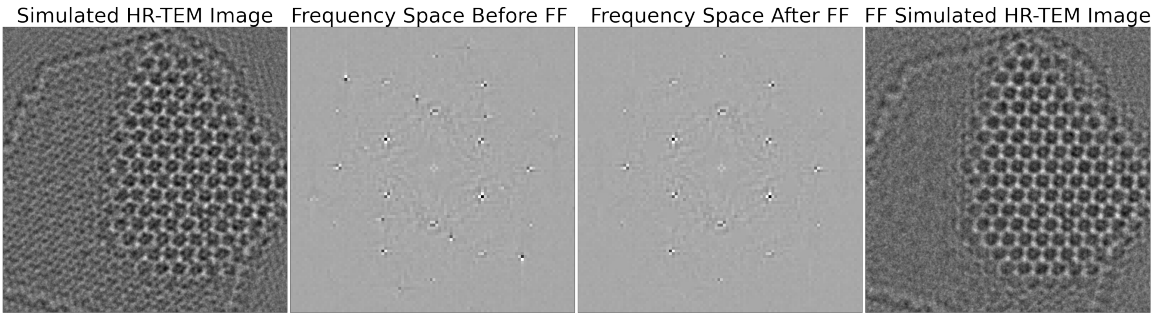


Figure 2.5: First: Simulated HR-TEM image before Fourier filtering, in real space. Second: Corresponding image presented in frequency space prior to Fourier filtering. Third: Resulting image after Fourier filtering, displayed in frequency space. Last: Reconstructed HR-TEM image after Fourier filtering, in real space

As the lattice constant of Molybdenum is larger than graphenes', the frequencies of Molybdenum in reciprocal space is closer to the centre than for graphene. Notice that the first patterns of graphene in reciprocal space is shifted from its classic position. This is because the graphene support is rotated in real space.

2.2 Deep learning

Deep learning is a branch of machine learning. A deep artificial neural network, is an artificial neural network with multiple hidden layers.

Neural networks and deep learning are recognized as the best approaches for a wide array of issues in image recognition, speech recognition and natural language processing. A variant of the deep neural network that is especially good at classifying images is the convolutional neural network (CNN) [23]. It uses a special network architecture well-adapted for image classification.

Prior to delving into the theory behind the CNN, it is imperative to understand the basic concepts behind the deep neural network.

2.2.1 Deep Neural Network

Deep feedforward neural networks are of great importance to machine learning practitioners [24].

These models are described as feedforward due to the flow of information in the network. The input **x** flows through intermediate computations to then finally create the output **y** with no feedback connection to previous layers. The term deep is referring to the fact that there are multiple layers in the neural network. The "Neural" refers to the structure, trying to mimic the way the brain works with connected neurons. Similar to what goes on in the brain, the activated neuron activates the other neurons in a chain effect leading to decision making. In neural networks, these neurons are often modelled as a perceptron. The perceptron is activated by the following rule:

$$output = \begin{cases} 0, & \text{if } \sum w_j x_j \leq threshold \\ 1, & \text{if } \sum w_j x_j > threshold \end{cases} \tag{2.2.1}$$

The perceptron maps the input x with its weight w to an output. The weights are real numbers stating the importance of the input to the output. The neurons output, 0 or 1, is determined by whether the sum exceeds a given threshold. Let me present an example. Imagine your friend asks you to go to the beach. You make your decision based on the following three factors:

1. Are there jellyfish
2. Is the weather good
3. Is there ice-cream

Let the three binary variables x_1, x_2 and x_3 represent if there are jellyfish, if the weather is good and if there is ice-cream respectively. Then the weights corresponds to the importance of that variable. If you really enjoy ice-cream at the beach, do not care about jellyfish and somewhat enjoy sunlight the weights could look like this: $w_1 = 1$, $w_2 = 3$, $w_3 = 6$. If the threshold is set to 5, then you would only go to the beach if there is ice-cream. This is a very simple example, but it goes to show the key concepts. Varying the weights and thresholds changes the model's decision-making.

A feed forward neural network is made of layers, the input layer, the hidden layers and the output layer. Figure 2.6 illustrates a feed forward neural network containing two hidden layers.

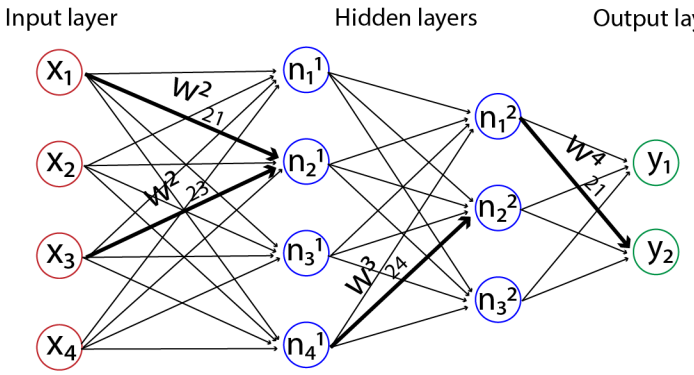


Figure 2.6: Illustration of a feed forward deep neural network containing input layer, two hidden layers and an output layer. The layers are connected through weights. all connections have weigths, only 4 weights presented in this illustration due to simplicity.

This neural network is fully connected as every neuron in layer n is connected to every neuron in layer $l + 1$. The output of each neuron affects each of the neurons in the subsequent layer. The connections are only in the forward direction hence the name "feed forward".

The weight w^l_{jk} denoting the connection between the output of the k^{th} neuron in layer $(l-1)^{th}$ layer to the j^{th} neuron in the l^{th} layer.

The output activation $\mathbf{a}^l$ of the l^{th} layer given the input activation of the $(l-1)^{th}$ layer $\mathbf{a}^{l-1}$, weights matrix $\mathbf{w}^l$ and bias vector $\mathbf{b}^l$ is given by equation 2.2.2.

$$\mathbf{a}^l = f(\sum_{j=1}^{N} a_k^{l-1} w_{jk}^l + b_j^l) \tag{2.2.2}$$

Equation 2.2.3 expresses the flow of information through the neural network, where N is the number of neurons in layer l and f is the activation function. The activation function introduces non-linearity to the activation of the neuron. The step function introduced with the perceptron in equation 2.2.1 is a type of activation function that maps the input to a binary output. These models are referred to as networks as there are multiple layers making the final mapping function a composition of each layers function [24]:

$$\tilde{f}(\mathbf{x}) = f^{(H)}(f^{H-1}(f^{H-2}...f^1(\mathbf{x}))) \tag{2.2.3}$$

The number of layers H determines the depth of the neural network.

2.2.2 Activation Functions

Activation functions are crucial in order to learn non-linear trends as they introduce non-linearity to the activation of the neuron. There are many different activation functions, and choosing a fitting one is an experimental art. It is often the case that different layers have different activation functions. Some well known activation functions include the rectified linear unit (ReLU), parametric rectified linear unit (PReLU), tanh and sigmoid.

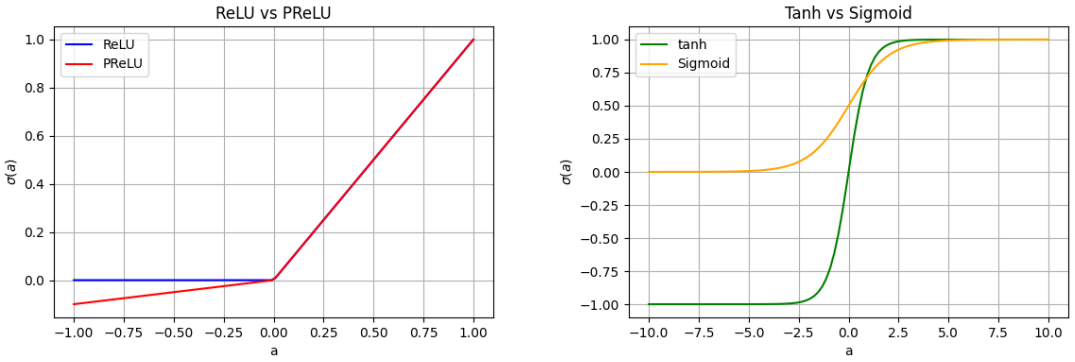
$$f_{\text{ReLU}}(x) = \max(0, x) \tag{2.2.4}$$

$$f_{\text{PReLU}}(x, \alpha) = \begin{cases} x & \text{if } x > 0 \\ \alpha x & \text{otherwise} \end{cases} \tag{2.2.5}$$

$$f_{\tanh}(x) = \tanh(x) \tag{2.2.6}$$

$$f_{\text{sigmoid}}(x) = \frac{1}{1 + e^{-x}} \tag{2.2.7}$$

Below in figure 2.7, is a plot of the above stated activation functions.



(a) The ReLU and PReLU activation functions plotted for x = [-1;1] with $\alpha = 0.1$

(b) The tanh and sigmoid activation function for x = [-10;10]

Figure 2.7: Plots of some of the most well known activation functions for applying non-linearity to the output of a neuron, ReLU, PReLU, tanh and sigmoid.

Though discovering the right activation function can be challenging, many activation functions are associated with specific types of problems.

ReLU is a piece-wise activation function often used in neural networks. It maps the input x to 0 if the input x is negative or else x itself. ReLU is a computationally efficient activation function. It helps mitigate the vanishing gradient problem, meaning the gradient does not saturate as the input grows large [25]. Because ReLU maps every negative value to 0, some neurons are never activated, and their weights are never updated. PReLU aims to circumvent this issue. the difference between PReLU and ReLU is that PReLU maps every negative value by multiplying with a tuneable parameter. This allows for the optimal slope to be trained, diminishing the dead neuron problem.

The sigmoid and the tanh functions are a great choice for binary problems as they always map between 1 and respectively 0 and -1, and are fully differentiable. The main difference is the gradient of the functions around 0, where the gradient of the tanh is larger. They both can cause the vanishing gradient problem. Looking at figure 2.7b it is clear, that the gradient goes towards 0 as the input grows larger. This stands an issue when the neural network learns. How the Neural network learns will now be investigated.

2.2.3 Loss Functions and Back Propagation

Consider the dataset $\mathcal{D} = \{(x_i, y_i)|i = 1, ..., N\}$, where x are the input data, y is the label, and N is the size of the dataset. After the input x_i has propagated through the layers of the neural network, the neural network makes a prediction in the output layer, $\tilde{y}_i$. This prediction is then compared to the label ground truth, y. The error is quantified using a loss function. The loss function, just like the activation function in the previous section, is problem specific. One popular choice of loss function for regression problems is the averaged mean squared error:

$$\mathcal{L}_{MSE}(\mathbf{y}, \tilde{\mathbf{y}}) = \frac{1}{N} \sum_{i=1}^{N} ||\mathbf{y}_i - \tilde{\mathbf{y}}_i||^2 \tag{2.2.8}$$

This loss function does not work for classification purposes. In classification the $\mathbf{y}$ and $\tilde{\mathbf{y}}$ represents the probability of belonging to a specific class or category. The difference between the models prediction, and the ground truth is measured using cross-entropy.

The categorical cross-entropy loss function quantifies the discrepancy between the predicted probability distribution, derived from the neural network's output, and the actual distribution.

$$\mathcal{L}_{CCE}(\mathbf{y}, \tilde{\mathbf{y}}) = -\frac{1}{N} \sum_{i=1}^{N} \sum_{j=1}^{C} y_{i,j} \; log(\tilde{y}_{i,j}) \tag{2.2.9}$$

Where C is the category. The loss function is 0 for a specific category, if the guess is correct ie. $\tilde{\mathbf{y}} = 1$. As $\tilde{\mathbf{y}}$ only takes values between 0 and 1, the logarithm can never be positive, and the loss function increases with decreasing $\tilde{\mathbf{y}}$.

After the loss has been measured, it is imperative to optimize it. Similar to other optimization problems, one could believe that such a minimum of loss could be computed using derivatives. However, as the loss function for neural networks depends on many variables, using calculus is not feasible [23]. To solve this problem the method gradient descent is often used. The gradient descent algorithm works by repeatedly computing partial derivatives of the loss function, with respect to all tuneable parameters $\boldsymbol{\theta}$ by:

$$\boldsymbol{\theta}_{i+1} = \boldsymbol{\theta}_i - \eta \frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}} \tag{2.2.10}$$

Where $\boldsymbol{\theta} = (\boldsymbol{W}^{(1)}, \boldsymbol{W}^{(2)}, ...)$, with $\boldsymbol{W}^{(l)}$ being the weight matrix for the l^{th} layer.

θ_{i+1} are the updated parameters. They are updated by taking the sum of the previous parameters θ_i and the partial derivative of the loss function times η. η is a hyper-parameter known as the *learning rate*. It determines the magnitude of the change. If the learning rate is small, then the neural network learns slowly, but converges more accurately. Conversely, a larger learning rate accelerates the learning process but increases the risk of overshooting optimal parameter values.

The iterative process of computing gradients and adjusting weights is called back-propagation. Training neural networks, entails a sequence of feedforward and back-propagation iterations until the loss function converges or the predefined maximal cycles are obtained. These cycles are referred to as training epochs.

2.2.4 The Convolutional Neural Network

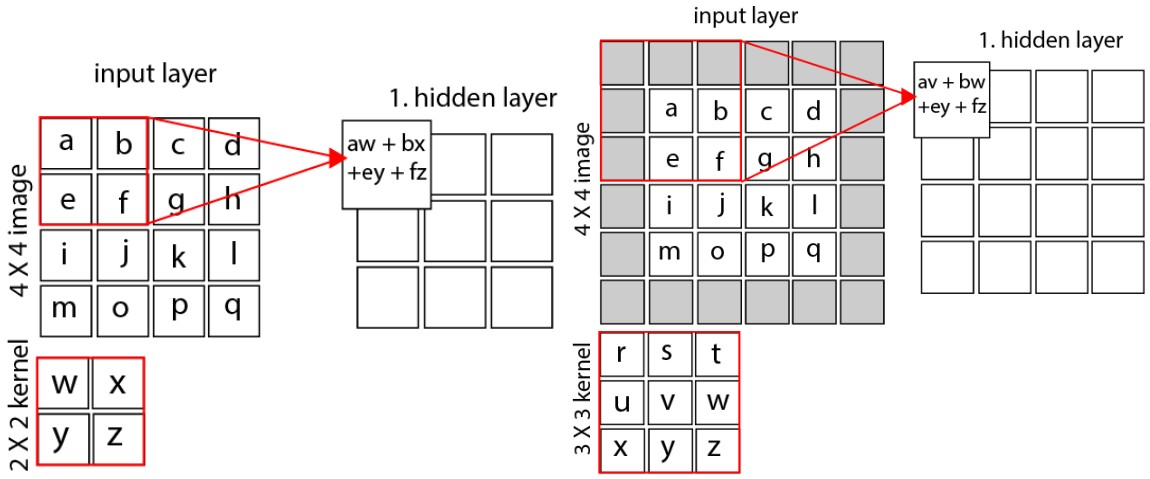
Consider the feedforward neural network illustrated in figure 2.6, where input images are transformed into 1-dimensional vectors instead of retaining their original 2-dimensional structure. This simplification creates challenges for the network, particularly when dealing with larger images, as it requires a substantial number of connections between layers. Besides creating a computational

burden, it also is doubtful that there is enough data to tune the amount of parameters. As a result, fully-connected feedforward networks become inefficient for handling images or large arrays.

The convolutional neural network (CNN), is a specialized type of neural network designed for analyzing data structured in a grid-like topology [24]. An example of such a data-structure is image data, which has a 2-D grid of pixels. As implied by its name, CNNs utilize the mathematical operation known as convolution. This involves applying a two-dimensional kernel, denoted as K, to an image I, with pixels represented as (i, j). The convolution operation between the kernel and the image is defined as follows:

$$S(i,j) = (I \otimes K)(i,j) = \sum_m \sum_n I(m,n)K(i-m, j-n) \tag{2.2.11}$$

$S(i, j)$ is denoted the feature map. The convolution between the kernel and input image is illustrated in figure 3.6. Convolution leverages three key ideas: sparse interactions, parameter sharing and equivariant representations [24].



(a) An example of a 2D convolution. The local receptive field (red square) sweeps over the input, creating the feature map.

(b) An example of a 2D convolution with padding. The padding allows for the subsequent layer to maintain the dimensions of the previous one.

Figure 2.8

Sparse interactions stems from having a kernel that is smaller than the input image leading to a reduction in data size during convolution. Consequently, although the input image may contain numerous pixels, the output dimension is reduced while retaining important features such as edges. This results in fewer parameters which entails a more computationally efficient model.

Each feature map within a CNN detects a distinct feature or pattern. This implies that the weights remain the same across the entire feature map. Parameter-sharing refers to multiple functions in the CNN sharing the same weight. This helps reducing the number of parameters of the model, which again, is beneficial for the computational efficiency.

The output is equivariant under translation, implying that a shift in the input will lead to a corresponding shift in the output. For instance, regardless of whether Batman is portrayed on the ground or in the air, his image remains recognizable [23]. Another key feature is that the CNN can learn spatial hierarchy, see figure 2.9 from [26].

Spatial hierarchy in CNNs refers to the organized progression from low-level to high-level features based on the spatial arrangement of pixels in the input data. For instance, in an image of a cat, low-level features like edges and textures are detected in the early layers, while higher-level features such as the cat's eyes, nose and ears are abstracted in the later layers.

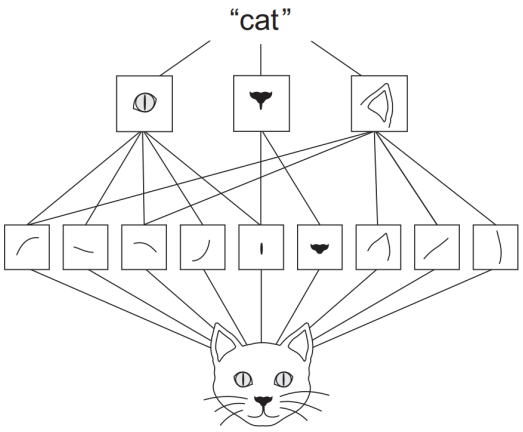


Figure 2.9: Illustration of CNN spatial hierarchy. Edges of cat in the first layer is translated into objects in the second which makes the cat. Image from [26]

This hierarchy is facilitated by sparse interactions, parameter-sharing, and translational equivariance. Each feature map specializes in detecting a distinct pattern, with shared weights across the entire map. CNNs exhibit equivariance under translation, ensuring consistent recognition regardless of spatial shifts in the input.

The CNN consists of three stages. In the first layer the convolutions are performed creating a set of linear activations. The second stage consists of running each linear activation through a nonlinear activation function, like the ones seen on figure 2.7. In the last stage a pooling function is used to further manipulate the output of the layer. An example of such an operation is the max pooling operation. The max pooling operation outputs the maximum activation of the rectangular neighborhood. Pooling aids in making the output invariant to small translations and down-sampling feature maps [26]. In the process there are three key parameters that the reader should be familiar with:

Size: This parameter refers to the dimensions of the local receptive field. For example, in figure 3.6a, the size is set to 2 x 2.

Padding: Padding is essential for maintaining the dimensions of the input image during convolution. It ensures that the output size matches the input size, see figure 3.6b.

Stride: The stride parameter determines the step size of the local receptive field as it moves across the input image during convolution. It controls how much the receptive field shifts for each step. In figure 3.6, the stride is set to $(1, 1)$.

2.2.5 U-net and MSD-net

Commonly used network structures often incorporate extra operations and connections into the conventional architecture to enable the training of deeper networks. This section will look in to two key architectures, the U-net and MSD-net.

CNNs are frequently used for classification tasks that require class labels. However, there are instances where the desired output should include not only class labels but also positional information ie. each pixel should have a class assigned.

The convolutional layers of the U-net architecture apply a set of filters to the input image, generating feature maps. Subsequently, the max pooling layers down-sample these feature maps by selecting the maximum value within a window of pixels. This process occurs within the architecture shown in figure 2.10, where the U-shape arises from the contraction on the left side and expansion on the right side. The left side take the input image and applies convolutions (without padding) followed by the rectified linear unit (ReLU). Then a max pooling is applied with a set stride, to down-sample. At

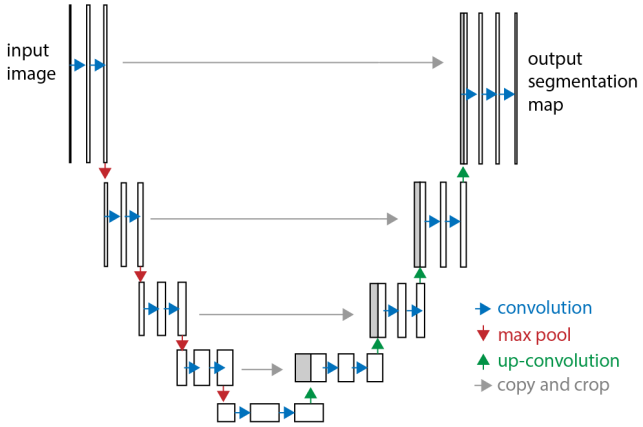


Figure 2.10: Illustration of the U-net architecture. Each arrow represents an operation. each white box represents a multi-channel feature map and each grey represents copied feature maps. Image modified from [27]

each down-sample step, the the number of feature channels are doubled [27]. Correspondingly, for each up-sampling on the right side, the number of feature channels are halved. The large amount of feature channels in the up-sampling part, allows the network to propagate context information to higher resolution layers [27]. The U-net has many parameters due to the skip connections (copy and crop) and additional layers in the expansive part resulting in a higher computational cost.

One other CNN architecture used for image analysis is the mixed-scale dense convolutional neural network (MSD-net). The MSD-net is a less complicated architecture using fewer parameters that **mixes scales** within each layer and **densely** connects all feature maps [28].

Mixing scales: The MSD-net uses dilated convolutions instead of up-sampling and down-sampling for capturing specific features of the image. The dilated convolution introduces a new parameter, the dilation rate. The dilation rate determines the size of the gaps. A dilation rate of 1, reduces the dilated convolution to a regular convolution. On figure 2.11 a dilation rate of 2 is used.

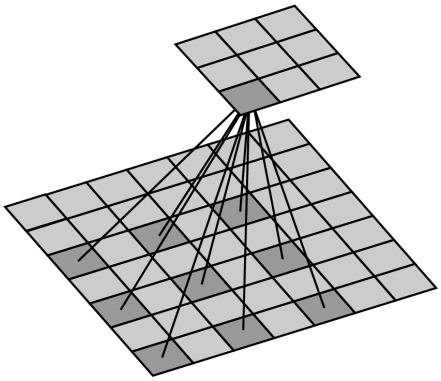


Figure 2.11: Illustration of dilated convolution. Two-dimensional convolution applied using a 3x3 kernel, with a dilation rate of 2 and no padding.

It has been shown that dilated convolutions are able to capture additional features. In addition the mixed-scale approach allows each individual channel of a feature map within a single layer to operate at different scales, rather than having every layer operate at a fixed scale, as seen in traditional CNNs [28].

The mixed-scale approach helps in reducing some disadvantages of the up-scaling and down-scaling approaches, such as the U-net.

Larger scale information of the image is quickly done accessible in the early layers of the network. This is an effect due to the large dilations, which improves the results of the deeper layers. Information from one scale can directly influence decisions at other scales without traversing intermediate layers.

No additional parameters have to be learned during training, since the mixed-scale approach does not include learned up-scaling operations. This results in smaller networks that are easier to train.

Dense connections: All network feature maps have the same dimension as the input image and output image for all layers. This enables the utilization of not only the preceding layer but also all previous layers, including the input layer, to compute the feature map for a specific layer. This also implies that all previous feature maps can be used to compute the output image y. This practice of using all previously computed feature maps is referred to as densely connecting a network.

Combining the mixed-scaled dilated convolutions and dense connections results in the MSD-network architecture.

Figure 2.12 shows a schematic representation of the MSD-net.

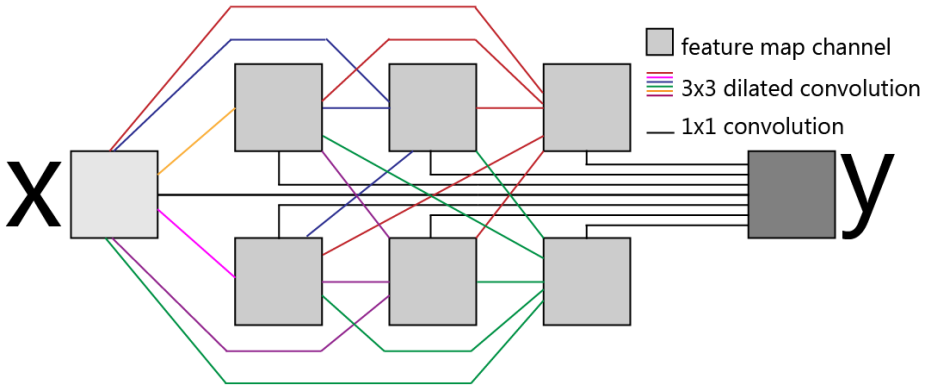


Figure 2.12: Illustration of the MSD network. Each of the colored ines represents 3x3 dilated convolutions corresponding to a different dilation. This MSD-net has width of 2 and depth of 3. Illustration modified from [28].

Because of the mixing scales and dense connections, the MSD-net are able to produce accurate results using few feature maps and trainable parameters. Additionally, its consistent layer connectivity and use of standard operations make it easy to implement and use in real-world applications [28].

3 Models

3.1 Simulating Structures

To train the CNN, a substantial dataset of HR-TEM images is required. To ensure a comprehensive dataset with an accurate ground truth, the HR-TEM images is simulated.[1].

The structures for the simulation is generated using the Atomic Simulation Environment (ASE) [29]. The Python library ASE facilitated generating well-defined structures such as MoS_2 and determining their atomic potentials. The experimental data consists of a MoS_2 flake laying on top of a graphite support. However, since the atomic library of ASE does not incorporate graphite structures, they must be defined explicitly within the code.

Graphite is composed of stacked parallel layers of graphene [30]. It has a hexagonal crystal lattice type, for which the most common stacking sequence is the -ABABAB- stacking order [30]. The points drawn on the lattice represents the carbon atoms.

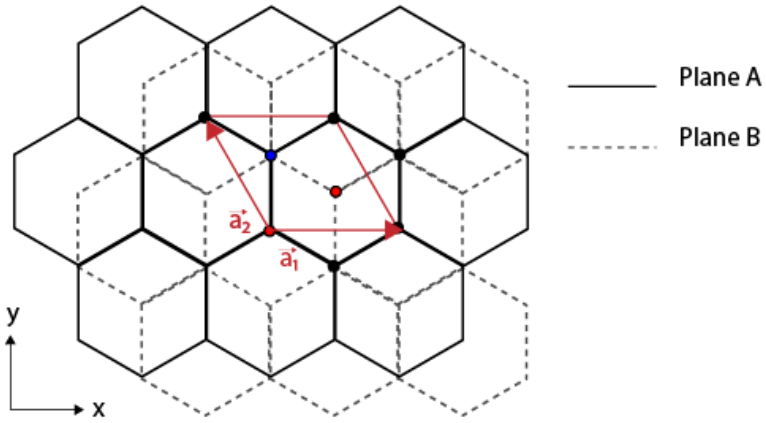


Figure 3.1: Sketch of crystal structure of graphite. It shows the x-y-plane view of the hexagonal lattice described by the two primitive lattice vectors $\vec{a}_1$ and $\vec{a}_2$ which spans the unit cell. The Solid and dashed lines denote structures originating from the A and B planes respectively. The Carbon atoms within the lattice are depicted as dots, while red lines signifies the conventional unit cell.The red dots shows the positions of the basis, with the blue dot signifying two atoms.

The solid line represents the A-plane layer of graphene and the B-plane layer is represented by the dashed line. The sketch represents the x-y plane, where the red rhombus signify the conventional unit cell, see figure 3.1. The unit cell is spanned by the three primitive lattice vectors, one of them not visible in the x-y-plane. $\vec{a}_1$, $\vec{a}_2$ and $\vec{a}_3$. We can write the primitive lattice vectors that defines the lattice as:

$$\vec{a}_1 = a\hat{\mathbf{x}} \tag{3.1.1}$$

$$\vec{a}_2 = \frac{-1}{2}a\hat{\mathbf{x}} + \frac{\sqrt{3}}{2}a\hat{\mathbf{y}} \tag{3.1.2}$$

$$\vec{a}_3 = c\hat{\mathbf{z}} \tag{3.1.3}$$

$\vec{a}_1$ and $\vec{a}_2$ both have the length of the lattice constant $a = 2.46$ Å and $\vec{a}_2$ has the length of the lattice constant $c = 6.7$ Å [31]. The red dots on figure 3.1 resembles the carbon atoms creating the basis. The blue dot signifies two carbon atoms, one situated in the A plane, and one in the B plane.

We can describe the basis of this crystal structure, containing four carbon atoms per unit cell, with respect to the lower most lattice point as:

[1]Scripts used for simulating structures, generating exitwaves and image and the 3 types of CNN's stems from the work of [12]

$$
\begin{aligned}
C_1 &= 0\vec{a}_1 + 0\vec{a}_2 + 0\vec{a}_3 = 0\hat{\mathbf{x}} + 0\hat{\mathbf{y}} + 0\hat{\mathbf{z}} \\
C_2 &= \frac{2}{3}\vec{a}_1 + \frac{1}{3}\vec{a}_2 + 0\vec{a}_3 = \frac{a}{2}\hat{\mathbf{x}} + \frac{a}{2\sqrt{3}}\hat{\mathbf{y}} + 0\hat{\mathbf{z}} \\
C_3 &= \frac{1}{3}\vec{a}_1 + \frac{2}{3}\vec{a}_2 + 0\vec{a}_3 = 0\hat{\mathbf{x}} + \frac{a}{\sqrt{3}}\hat{\mathbf{y}} + 0\hat{\mathbf{z}} \\
C_4 &= \frac{1}{3}\vec{a}_1 + \frac{2}{3}\vec{a}_2 - \frac{1}{2}\vec{a}_3 = 0\hat{\mathbf{x}} + \frac{a}{\sqrt{3}}\hat{\mathbf{y}} - \frac{c}{2}\hat{\mathbf{z}}
\end{aligned}
\tag{3.1.4}
$$

It is now possible to construct the graphite structure using the primitive lattice vectors 3.1.1, 3.1.2 and 3.1.3, and the basis 3.1.4.

```python
from ase import Atoms
from ase.build import mx2
from abtem import show_atoms
import ase.build
import matplotlib.pyplot as plt
import numpy as np

#Generate graphite support
a, c = 2.46, 6.708 # Lattice constants in Å
cell = [[a, 0, 0],[-0.5*a, np.sqrt(3)/2 * a, 0],[0, 0, c]]
positions = [[0,0,0],[a/2, a/(2*np.sqrt(3)), 0],[0, a/(np.sqrt(3)), 0],[0,  a/(np.sqrt(3)), c/2]]
graphite = Atoms(symbols='CCCC',positions=positions ,cell=cell,pbc=[True,True,False])
graphitelayers = 2
graphite *= (1, 1, (graphitelayers + 1) // 2 )
support = graphite.repeat((10, 10, 1))
# Generate MoS2
prototype = ase.build.mx2(formula='MoS2', size=(6, 6, 1))
prototype.rotate(30, 'z') #rotate MoS2 slab around z-axis

#stacking structures
stacked_structure = ase.build.stack(prototype,support,maxstrain=100)

# Show the stacked structure
abtem.show_atoms(stacked_structure)
plt.show()
```

Figure 3.2: Code showing example of generating structure using the ase library. Structure consists of graphite support with a rotated flake of MoS2 on top.

The plotted structure is shown in figure 3.3.

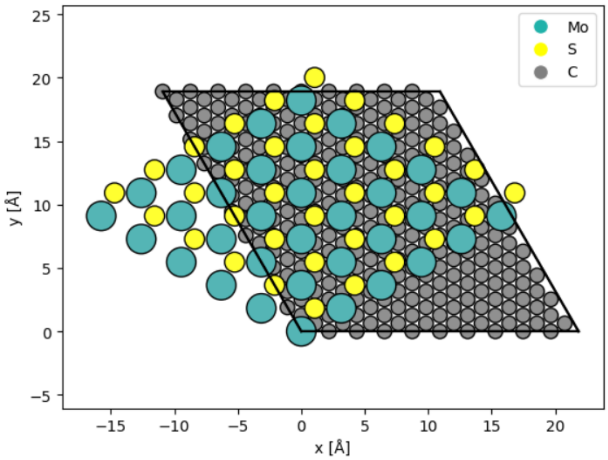


Figure 3.3

The structure shows a graphite support, with the MoS_2 slab rotated 30 degrees on top. For training purposes, we generate 1000 structures for the training set and 1000 for the test set.

Upon initializing a GraphiteSupportedMoS2Maker object, parameters such as the size of the samples, any distortions, a random seed for generation and the number of layers in the graphene support are specified.

A prototype of MoS_2 is created using the inbuilt mx2 function from the ASE library. The graphene support is created by defining its crystal structure and parameters, including the number of graphene layers, where the number of layers is a generated randomly with the upper bound of 6.

Then each of the structures, MoS_2 and graphite, are repeated in the x-y-plane until the sheets reaches a maximum of 140 Å in the x and y direction. These sheets are subsequently cut randomly to form irregular flakes. Further irregularity is introduced by creating vacancies and holes in each of the flakes. The vacancies are created by randomly removing up to 5 % of the C or S atoms and the holes are created by removing entire 5 % Mo or S-dimer columns from the specimen see figure 3.5.

The last step before stacking the structures entails rotating the MoS_2 flake around its z-axis by a random degree, either 0, 15, 30 degrees or a degree ranging from 0 to 90.

After this step the graphene and MoS_2 flake are stacked with distances between the flakes ranging from 3.3 to 7 Å. All this should aid in the generalizability of the model and improve the models ability to predict from real world HR-TEM images.

An example of such a structure is illustrated in figure 3.4.

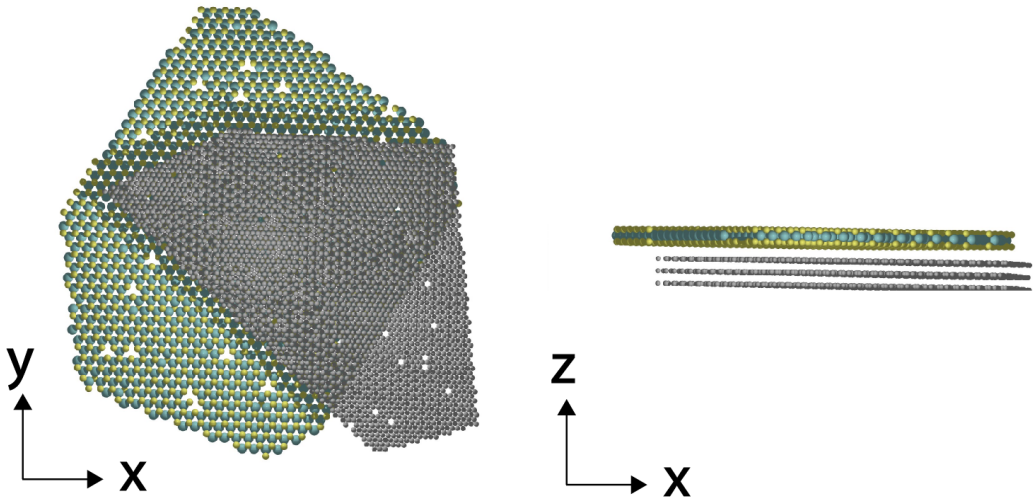


Figure 3.4: Stacked Structure of MoS_2 and Graphene. The image depicts a composite structure composed of a MoS_2 layer stacked on top of 3 graphene layers. The structure stems from the test set and is representative of those generated for training and testing purposes to simulate real-world HR-TEM images. The structures were generated using the process explained in this section.

The structures are saved into separate training and test folders, each containing 1000 structures. The structures have 4 channels, each corresponding to a different class of the structure. Columns housing a single molybdenum (Mo) atom were designated as class 0, those containing a sulfur (2S) dimer were labeled as class 1, and columns with only a single sulfur atom (referred to as a vacancy) were assigned to class 2 (1S). Class 3 represents the background. Each of these channels contains the positional information about its respective class.

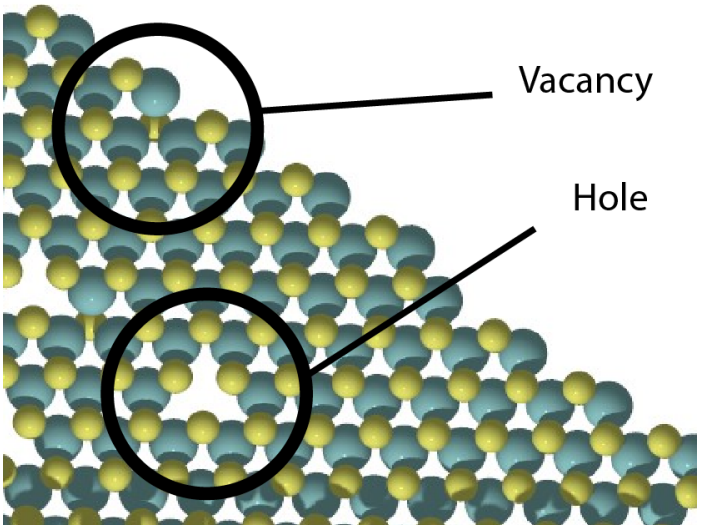


Figure 3.5: Zoom of the structure 3.5 with defects highlighted. The vacancy is represented by the single S (1S) atom missing. The hole is due to the absence of a Mo (Mo) atom.

3.2 Generating Exitwaves and Images

To simulate the exitwaves and generate the images, the Python library abTEM is used. The first argument passed to the script that generates the exitwaves and images is a parameters file. The parameters file contains the necessary physical parameters in order to simulate realistic HR-TEM images.

In this part of the process, the objective is to generate exitwaves based on the structures within both the training and test sets. Subsequently, these exitwaves are used to generate the image waves.

Before computing the exitwaves the structures are modified using the frozon phonon method in order to simulate atomic vibrations. Using the parameters from table 1 in equation 2.1.25 from section 2.1.4, the positional variance is computed. The temperature is chosen at random between 100K and 600K as seen in table 1. The postional variance is used in abTEM's FrozenPhonon method. The method randomly displaces atoms in order to emulate thermal vibrations.

For each of the structures, the exitwave is computed using the multislice method, which is the most computationally expensive part of the simulation [12].

The atomic potential is computed using the abTEM library's **Potential()** object [32]. This object is instantiated using five arguments: the **atoms**, specifying the atomic configuration of the material. **gpts** denoting the size of the wave array, which determines the grid points used for the calculation. The type of **parametrization** is set to *kirkland*, which dictates the method used to parameterize the potential.

The **Slice thickness** parameter specifies the thickness of the slices used in the multislice method. Lastly, the **projection type** is set to *finite*, which integrates the potential from each slice, as explained in section 2.1.1.

A plane wave is generated using abTEM's **Planewave()** object, which emulates the incident electron beam. The object takes the **beam energy** as an argument, which is specified in table 1. Subsequently, the exitwave is computed, using abTEM's multislice method on the **Planewave()** object with the *potential* as argument.

The image can now be generated using the exitwave.

Parameters	**Lower Bound**	**Upper Bound**
FFT Filtering Lattice Constant	2.467 Å	
Frozen Phonon Configurations	10	
Debye Temperature θ_{D}	580 K	
Phonon Temperature	100 K	600 K
Beam Energy	50 KeV	
Blur	0.05 Å	0.25 Å
2-Fold Astigmatism, C_{12}	0 Å	25 Å
Coma, C_{21}	0 Å	600 Å
Spherical aberration, C_{30}	-15 μm	15 μm
Defocus	-150 Å	50Å
Focal Spread	5 Å	10 Å
Image Epochs	10	
Dose	$10^{2.5}$ Å^{-2}	$10^{5.0}$ Å^{-2}
MTF C_1	0.2	0.6
Multislice Thickness	0.25 Å	
Normalized distance	12 Å	
Readout	0.005	0.015
Sampling	0.215 pixels/Å	0.235 pixels/Å
Wave Sampling	0.1 Å	

Table 1: Parameters used for generating exit and image waves from structure. For parameters given both upper and lower bound, a value will be chosen at random within the given range.

As the **image epochs** are set to 10 for the training set, each exitwaves results in 10 images, using the different HR-TEM parameters. For the test set, the number of image epochs are set to 1. The HR-TEM parameters is passed to the abTEM's **CTF()** object, generating 10 unique contrast transfer functions. For each image epoch, three images with different **defocus** values are generated iteratively.

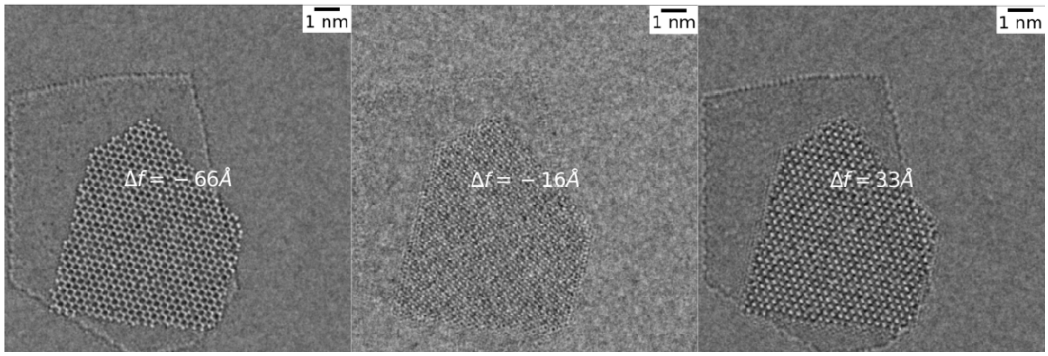
The first defocus value, Δf_1, is randomly chosen within the range of -150 Åto 50 Å. The next defocus value, Δf_2, is calculated as Δf_1 plus a random value between 49 Å and 51 Å, and the last defocus value, Δf_3, is calculated as Δf_2 plus a random value between 49 Å and 51 Å. A CTF is generated for each of the three defocuses by passing each of the defocuses, the CTF parameters, c_{12}, c_{21} and c_{30}, *focal spread* and *beam energy* to the **CTF()** object. Three images are generated using each of the resulting three **CTF()** objects, **sampling**, **blur**, **dose**, **MTF** C_1 and **readout** noise.

The usage of image epochs expands the dataset from 1000 to 10000 images. Given that the computation of exitwaves represents the most computationally intensive aspect of data generation, this approach reduces computational costs.

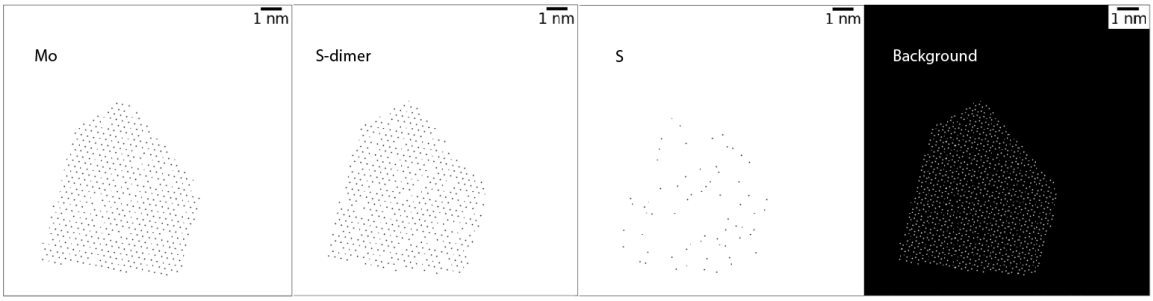
The final step of the image processing consists of applying a Fourier filter to the image. The aim of applying this filter is to remove the graphene structure from the image, as explained in section 2.1.5. **Sampling**, the **lattice constant** of graphene and the image is passed through a hexagonal Fourier filter. It Fourier transforms the image into frequency space using the fast Fourier transform (FFT). It then calculates the lattice constant per pixel based on the sampling rate. Then, it constructs a filter in the shape of a ring, centered around the origin, to target these frequencies. Three rings are applied, one for each of the three first repetitions of the pattern, in order to filter out the spatial frequencies of graphene.

The spotfilters function creates a composite filter by multiplying individual filters for each frequency peak in the list. Each individual filter, created by the spotfilter function, uses a Fermi function to ensure smooth transitions and avoid harsh edges. The fermi function is characterized by its sigmoid shape, which provides a smooth transition from 1 to 0. Parameters such as peaksize = $30 \cdot$ sampling and alpha = $20 \cdot$ sampling, determine the size and smoothness of these filters.

Finally, the filtered image is obtained by inverse Fourier transforming the modified Fourier spectrum.



(a) simulated images of MoS_2 on graphene layers from the test set. The test set consists of 1000 images, each of which has 3 channels with different defocuses. Here the three different channels of image 2 is generated from the defocuses of Δf_1 = -66, Δf_2 = -16 and Δf_3 = 33. The edge on the boundary of the graphene layer is still visible.



(b) The ground truth consist of an image with four channels of the same size as the training and test images. The first, second, third and last channel acts as the ground thruth for the positions of Mo,S-dimer (2S), S (1S) and the background with the graphene support respectively.

Figure 3.6: Image of traning data and mathcing ground truth.

The labels, which represent the ground truth, is generated from the structures using the Disk label type. This approach generates binary labels that mark pixels belonging to distinct atomic columns.

A total of 10,000 images are generated for training purposes, and an additional 1,000 images are generated for testing. These images are saved into separate folders accordingly. The next phase involves training the CNNs using the generated images

3.3 The Convolutional Neural Network

In order to ensure that the neural network trains consistently, the pixel intensities are standardized before training. The specific method for standardization is called "local standardisation", which is defined as:

$$\hat{I}_{xyc} = \frac{I_{xyc} - \frac{1}{N_c}\sum_c (I_c * G)_{xy}}{\frac{1}{N_c}\sum_c \sqrt{(I_c^2 * G)_{xy}}} \quad G_{xy} = e^{-\frac{x^2+y^2}{2\sigma^2}} \tag{3.3.1}$$

The image I, contains pixels denoted by their coordinates (x, y) consisting of N_c channels each represented by c.

The parameter σ is defined by the "Normalisation Distance", which governs the "locality" of the standardization. This distance is set to be larger than the defining features of the image. The normalized distance is computed by dividing the *normalizedistance* with the *sampling* from the parameters file. This leads to a Normalisation distance ≈ 50 Å.

The CNN is constructed using the **Keras** library which acts as an interface for the **TenserFlow** library. Three different CNN architectures were used for prediction, The U-net, U-net++ and MSD-net architecture.

The input to the U-net, U-net++ and MSD-net is the locally standardized image array. As each image is 640×640 and contains 3 channels, one for each unique defocus, the dimension of the input array is $640 \times 640 \times 3$.

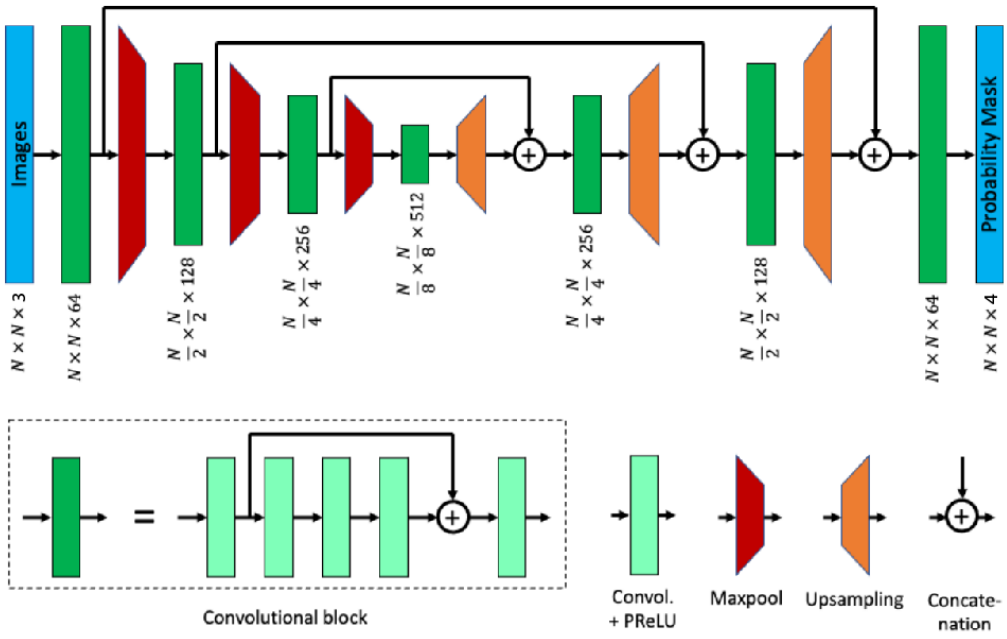


Figure 3.7: The architecture of the U-net. Information flow is from left to right as indicated by the arrows. The illustration is adapted from [33], Appendix A

The U-net architecture is comprised of a distinct down-sampling and up-sampling part, which is characteristic for the general U-net structure, see figure 3.7.

In the down-sampling part, convolutional blocks alternate with down-sampling layers. Conversely, in the up-sampling phase the convolutional blocks alternate with up-sampling layers. The convolutional blocks are constructed of 5 convolutional layers, with a skip connection between the output of the first layer and input of the fifth layer in order to preserve information of the initial layers. The

convolutional layers use the Parametric Rectified Linear Unit (PReLU) as activation function, see equation 2.2.5.

The down-sampling is done using the max pooling operation which halves the resolution. Each time the resolution is cut in half, the number of feature channels in the following convolutional block is doubled. This maintains the information flow in the network.

The upsampling is done using bilinear interpolation. The feature channels of the following convolutional blocks are now halved. After each up-sampling, information from the down-sampling part with matching resolution is added by concatenating the channels.

The resulting prediction is of the same dimensions as the ground truth $640 \times 640 \times 4$, containing one channel for each class.

U-net++ adopts the U-net architecture and incorporates additional skip connections across outputs of layers at different resolutions, see figure 3.8. This approach results in a more densely connected network, acting as a middle ground between the U-net and MSD-net architecture.

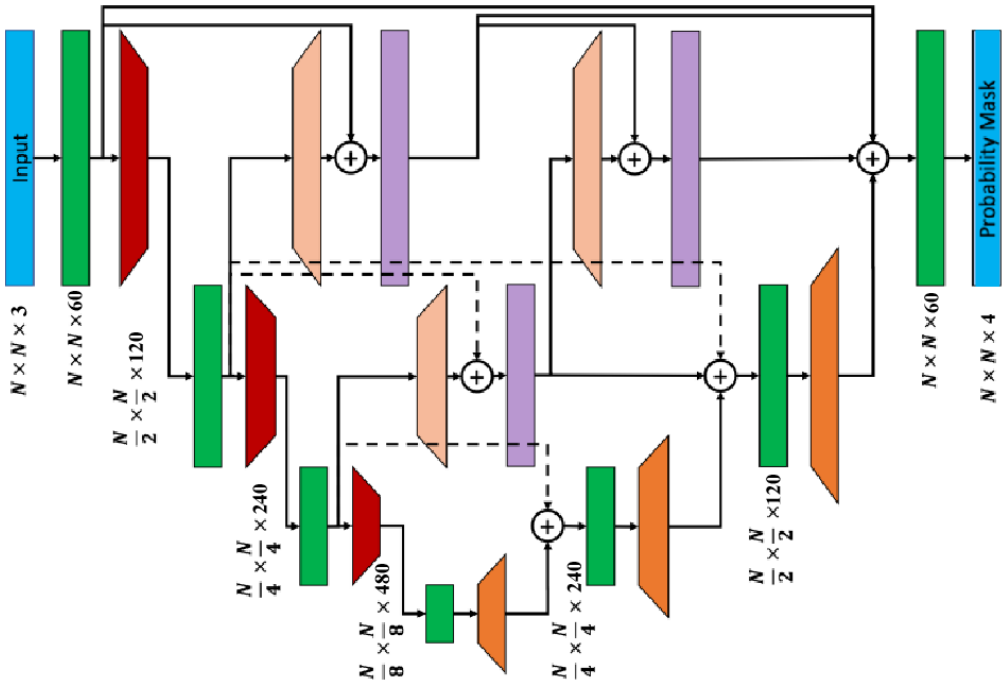


Figure 3.8: The architecture of U-net++ . Information flow is from left to right as indicated by the arrows. The illustration is adapted from [12]

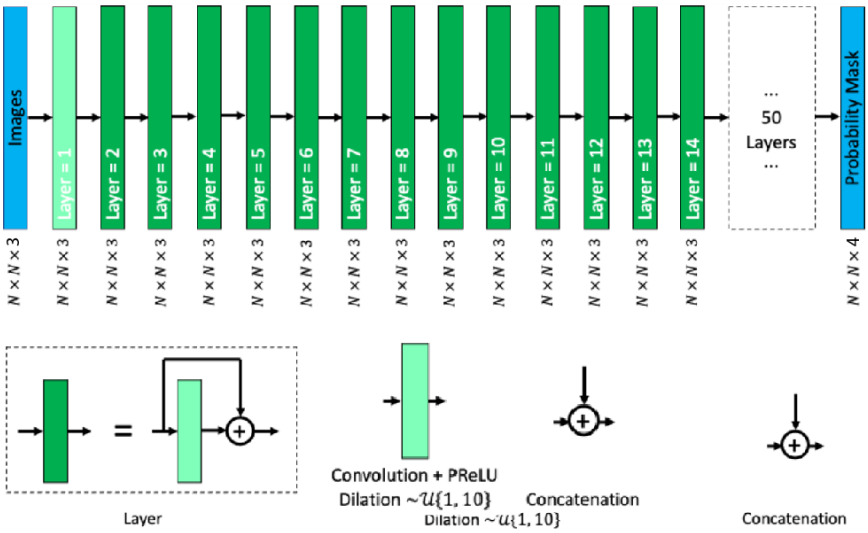


Figure 3.9: The architecture of the MSD-net. The illustration is adapted from [33], Appendix A

The MSD-net architecture consists of 50 layers, see figure 3.9. Each layer is defined by a convolution followed by is concatenating the output with the output of the previous layer. These concatenating operations are what defines the architecture as a dense net. The mixed-scale feature is introduced by dilated convolutions. The MSD-net uses dilated convolutions by implementing a 5×5 convolutional kernel which enables the network to capture features at varying length scales.

An earlier study found that hyperparameter optimization showed promising results when increasing the number of output channels, kernel size and number of layers [33]. However, this came at significant computational cost.

4 Results

In order to evaluate the CNN architectures performance, the F1-score metric is employed. This metric serves as a measure of the networks' performance, as it contains information about the ability to minimize both false positives and false negatives.

$$F1 = \frac{TP}{TP + \frac{1}{2}(FP + FN)} \tag{4.0.1}$$

Two different types of F1-scores are employed. The atomic F1-score and the pixel F1-score. In the case of the atomic F1 score, the prediction is classified true positive (TP), if the prediction contains a peak that is within a distance of 3 pixels ≈ 0.6Å from the ground truth peak. It is classified as a false positive (FP) if there is a prediction with no ground truth peak within the given distance. A false negative (FN) is the case where there is a peak in the ground truth, with no prediction within the given distance.

The pixel F1-score is computed by comparing each pixel at position (x, y) in the ground truth and prediction, for each of the channels. In some instances the background is excluded. it is classified as a true TP if the pixel value (x,y) is 1 in both the prediction and ground truth. If the pixelvalue is 0 in the ground truth and 1 in the prediction it is a FP and conversely. If the pixelvalue of the ground truth is 1 and the prediction is 0, it is a FN. The pixel F1 score is computed for all 4 channels, and for 3 channels, leaving out the background.

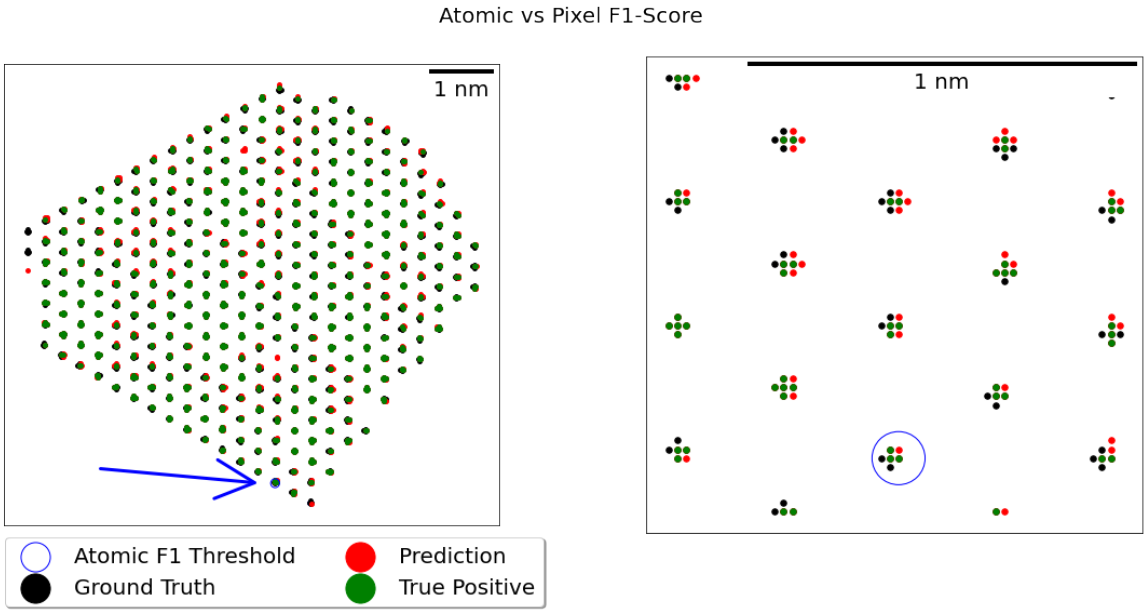


Figure 4.1: Prediction of Mo atomic columns from MSD-net trained with 500 images, plottet together with the ground truth.

Figure 4.1 illustrates the ground truth, predictions and True positives of Mo columns from the simulated data. In many cases, discrepancies between the prediction and ground truth is due to small positional deviations. These deviations contributes to the FP and FN and impact the pixel F1-score. However, this is accounted for in the atomic F1-score, where a prediction inside the blue circle will be classified as true positive. This mitigates the impact of the minor deviations. See B for code used to generate atomic and pixel F1-scores.

4.1 CNN Performance on Simulated Data

The U-net, U-net++ and MSD-net are trained using different sets of parameters and varied training dataset sizes. The training size provided to the CNN's ranged from 10 to 1000 structures, resulting in 10^2 to 10^4 images.

Each model were generated 3 times in order to ensure reliable results, and error bars are created using the standard error.

The U-net/ U-net++ parameters is varied by tuning the the number of channels ranging from 4 to 64/60 channels, see table 2.

Channels	U-net		U-net++	
	Parameters	Time [hours]	Parameters	Time [hours]
4	248,464	13.4	258,848	12.8
8	989,468	13.8	1,031,100	14.2
16	3,949,108	16.8	4,115,828	17.9
32	15,778,916	25.1	16,446,180	28.0
64/60	63,080,644	49.2	57,790,504	54.7

Table 2: U-net and U-net++ models for parameter tuning. Each model is generated 3 times.

The MSD-net parameter tuning entails experimenting with different numbers of channels, layers and kernel-size, see figure 3.

Kernel	Channels	Layers	Parameters	Time [hours]
5	3	25	68,329	18.0
5	3	50	277,054	45.1
5	5	50	768,004	65.1
9	3	50	894,958	54.5
5	7	50	1,503,954	-
5	9	50	2,484,904	-

Table 3: MSD-net models for parameter tuning. Each model is generated 3 times.

Increasing the number of channels, Layers and Kernel-size leads to an increase in the total number of parameters for the the model. The models marked with grey crashed due to memory limitations.

The atomic F1-score is computed for varying number of parameters for each model, defined by table 2 and 3. The model performance as a function of the number of parameters is illustrated in figure 4.2.

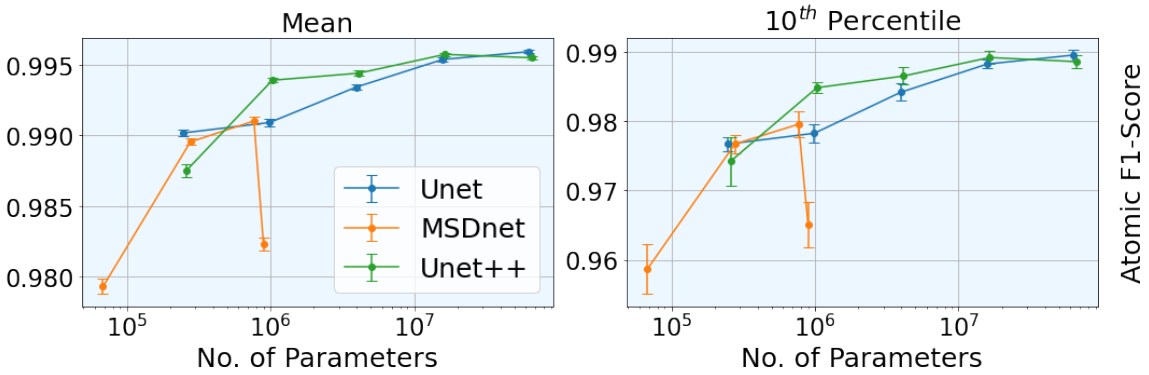


Figure 4.2: The Atomic F1-score for varying number of trainable parameters. Left: Mean of atomic F1-score. Right: 10^{th} percentile atomic F1-score.

All of the above models have been trained with a training set size of 10^4 images. The atomic F1-scores is computed on the basis of a test set with 10^3 images. All networks display high atomic F1-scores. Both U-net and U-net++ improve their atomic F1-scores when increasing the number of parameters up to a threshold of 10^7 parameters, beyond which their performance plateau. Due to the memory requirements associated with training the MSD-net, it was not feasible to assess its performance with higher parameter counts. Looking at the MSD-net performance, the first three atomic F1-scores show a positive trend when increasing the number of parameters. However, there is a decrease in model performance for the model with the highest number of parameters. This model has an increased kernel size of 9 as opposed to the other three models with a kernel size of 5. This could indicate that the kernel size is less significant than the number of channels, for model learning.

The parameters chosen for each of the three architectures is marked with green in table 2 and 3.

Network	No. of Parameters	Time [Hours]	Test Loss
Unet	63,080,644	49.2	0.0159 ± 0.002
Unet++	57,790,504	54.7	0.0131 ± 0.002
MSD-net	277,054	45.1	0.0148 ± 0.0003

Table 4: Chosen Unet, Unet++ and MSD-net, and their respective number of parameters and running time and test loss.

Figure 4.3 showcases the pixel F1-score, 10^{th} percentile pixel F1-score, and atomic F1-score corresponding to each selected model of U-net, U-net++, and MSD-net across various training sizes.

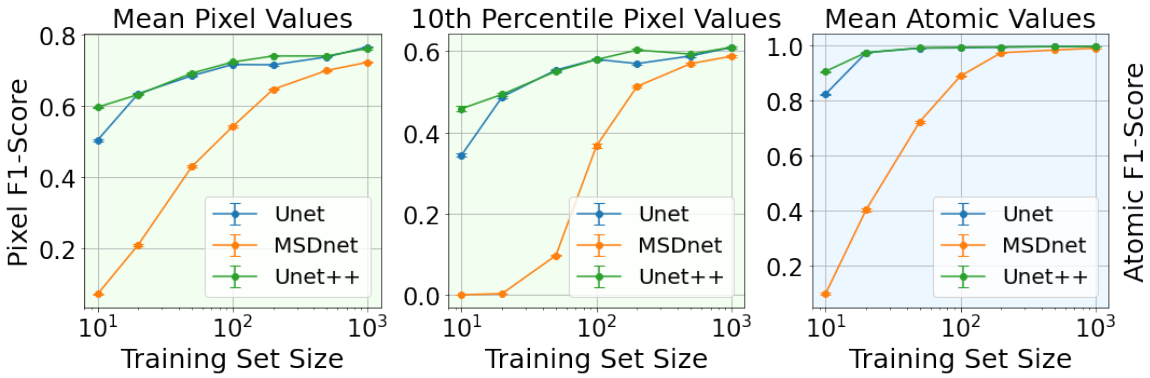


Figure 4.3: The pixel and atomic F1-score computed without background channel, for varying training set sizes. Left: Mean of pixel F1-score. Middle: 10^{th} percentile pixel F1-score. Right: Mean of atomic F1-score.

It is evident, that a larger training set size entails a higher F1-score, indicating that the models are learning. The MSD-net exhibits the most considerable improvement in both the pixel and atomic F1-scores with the expansion of the training set size. This underscores the need for a larger training size when employing the MSD-net.

The progression of the atomic F1-score for both the U-net and U-net++ models appears to plateau as the size of the training dataset approaches 10^2 structures. However, this trend is not observed in the pixel F1-scores, indicating continued improvement even with a training set size of 10^3 structures. This phenomenon suggests that the models are refining their ability to learn finer details in atomic positions.

The chosen models from table 4 are trained using a training size of 10^4 images. The atomic F1-scores and best prediction from the Unet++ is shown in figure 4.4.

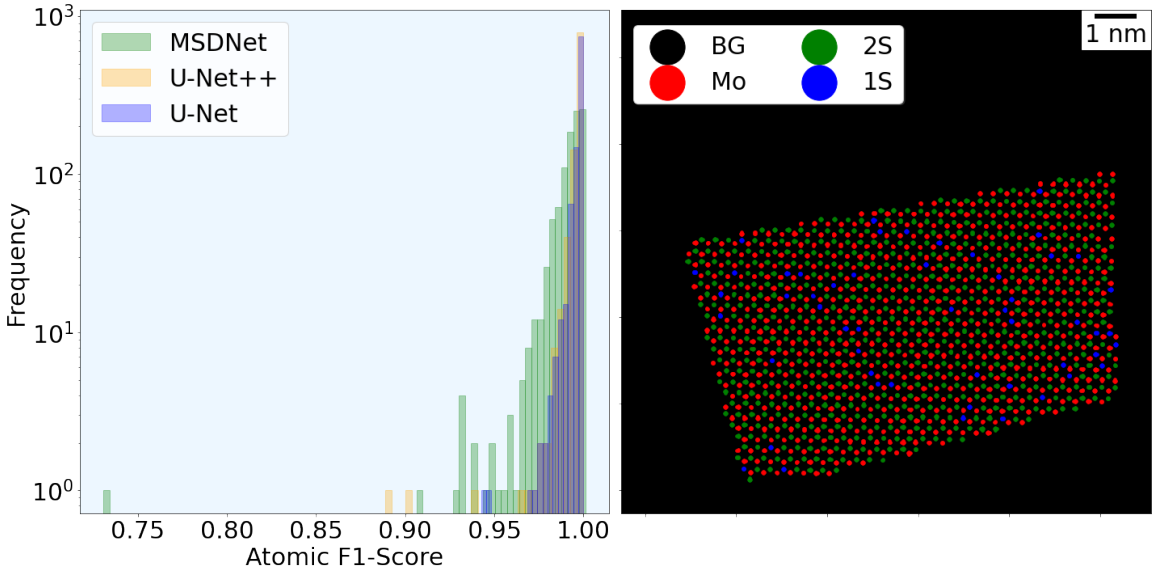


Figure 4.4: Left: Histogram over atomic F1 scores computed for all 4 channels. Right: Best prediction from Unet++.

On the left of figure 4.4 Unet++'s best prediction is illustrated. The atomic structure of MoS_2 is clearly visible.

All three models exhibit high atomic F1-scores. It appears that both Unet++ and Unet tend to have higher atomic F1-scores than the MSD-net. The pixel F1-scores, with and without background, is shown in the histograms in figure 4.5.

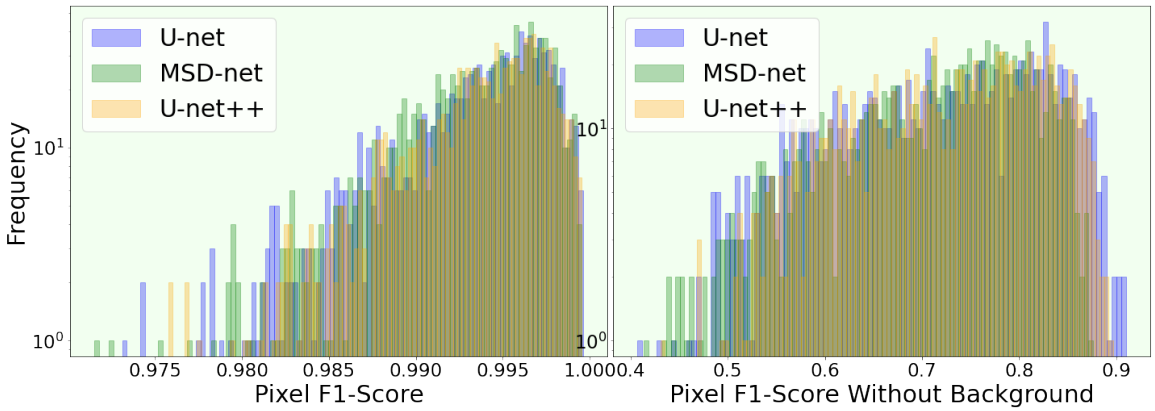


Figure 4.5: Left: Pixel F1-scores computed for all 4 channels on the U-net, MSD-net and U-net++. Right: Pixel F1-scores computed for Mo, 2S and 1S channel, leaving out the background, calculated on the predictions from the U-net, MSD-net and U-net++.

The pixel F1-scores with the background channels are higher than the ones without background. Looking at the left histogram, all three models display similar performance. On the right histogram, there might be a small tendency of U-net and U-net++ having higher pixel F1-scores than the MSD-net.

The 3 types of evaluation metrics for each of the three models are displayed in table 5.

Network	Pixel F1 with background	Pixel F1 without Background	Atomic F1
U-net	0.993	0.722	0.998
U-net++	0.994	0.730	0.997
MSD-net	0.993	0.715	0.992

Table 5: The three different kind of F1 scores for the three different network architectures.

Table 5 illustrates the different types of F1 scores for the three chosen models. Each row corresponds to a specific network architecture: U-net, U-net++, and MSD-net. Within each row, three F1 score metrics are presented: Pixel F1 with background, Pixel F1 without Background, and atomic F1.

All three models exhibit similar high pixel F1 scores when counting the background in. They all record pixel F1-scores ≈ 0.993.

However, when excluding the background channel, the pixel F1-scores drop significantly for all models. This decline occurs because a considerable number of true positives exist in the background class, thereby inflating the F1-score when included.

The variation in the best pixel F1-score is more noticeable when the background channel is excluded. U-net++ achieves the highest Pixel F1-score of 0.730, whereas MSD-net obtains the lowest score of 0.715.

Regarding atomic F1-scores, all three models perform best in this metric.

While U-net has the highest atomic F1-score among the three models, both U-net and U-net++ have very similar atomic F1-scores. The MSD-net's atomic F1-score is marginally lower than that of the other two models.

The focal series with matching ground truth and prediction is illustrated in figure 4.6.

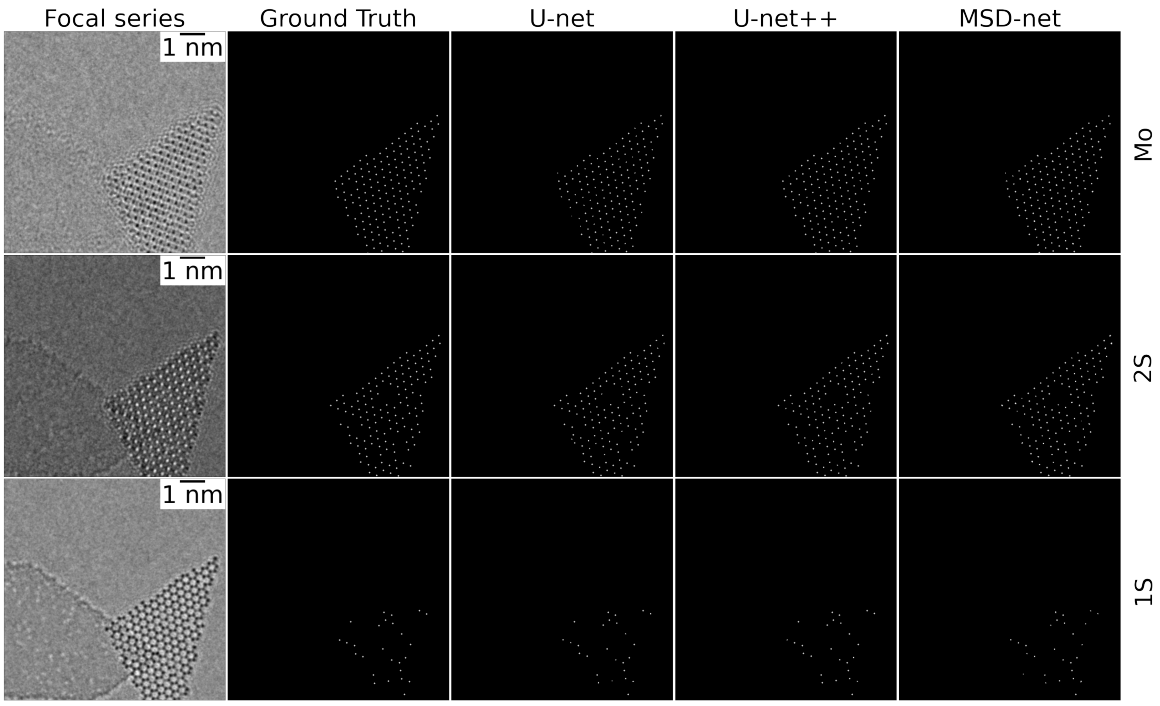


Figure 4.6: Focal series and matching ground truth. The third fourth and fifth columns represents the Predictions of The U-net's median performance for each network. The rows represents each of the three channels, Mo, 2S and 1S. In the context of the focal series, each row corresponds to a distinct defocus.

The first column represents the focal series used for testing, while the second column showcases the ground truth for each of the three channels: Mo, 2S, and 1S. Subsequent columns illustrate the predictions from the U-net, U-net++, and MSD-net.

All three models appear to predict accurately compared to the ground truth, with only minor discrepancies observed.

At the edge of the graphene layer, there is boundary phenomena stemming from the Fourier filtering. All three models exhibit minor deviations from the ground truth at the edges of the graphite support. This includes both misidentification and missed detection of atomic columns. This is primarily manifested in the edge of the MoS_2 flake.

Focusing on the 1S channel, both U-net, and U-net++ predictions aligns with the pattern of the ground truth. However, the MSD-net fails to accurately identify some atoms and also predicts 1S atoms that are absent in the ground truth.

Examining the boundaries of the MoS_2 flakes, there is instances, across the models, of extending the pattern beyond the boundaries of the ground truth.

4.2 CNN Performance on Experimental Data

Before the three models predicted the atomic positions of the three atomic columns, the real world HR-TEM focal series was Fourier filtered, see appendix C figure C.1.

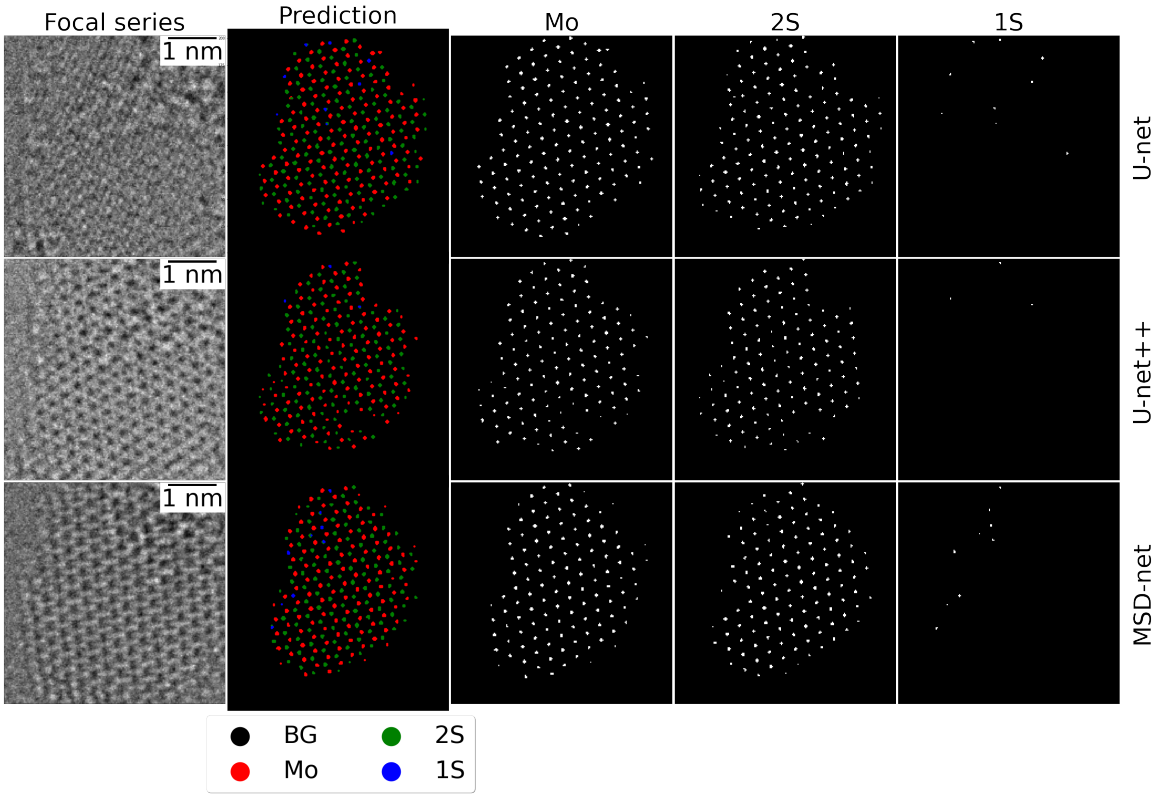


Figure 4.7: Focal series from real world HR-TEM image and matching prediction. The defocus for the images are approximately 165 Å, 115 Å and 65 Å. The third fourth and fifth columns represents the channels corresponding to the atomic columns Mo, 2S and 1S respectively. The rows represents each of the three CNN architectures, U-net, U-net++ and MSD-net.

Figure 4.7 shows a real world HR-TEM focal series alongside the three models predictions. Each column represents a different atomic column: Mo, 2S, and 1S. And each row shows the predictions from each of the three models: U-net, U-net++ and MSD-net.

As there is no ground truth for the experimental data set, the assessment of the models performances is purely qualitative.

The network are able to learn from the structural pattern of the Mo and 2S columns. However, the networks are not able to learn the positional patterns of 1S atomic columns as they appear at random. It is evident that this lack of contextual information has led to three diverse predictions of the atomic positions of 1S columns. The three models predictions have differences at the boundary of the MoS_2 flake. This indicates that while the models excel at learning patterns, they encounter difficulty in accurately predicting non-periodic features.

This is more clearly depicted in figure 4.8.

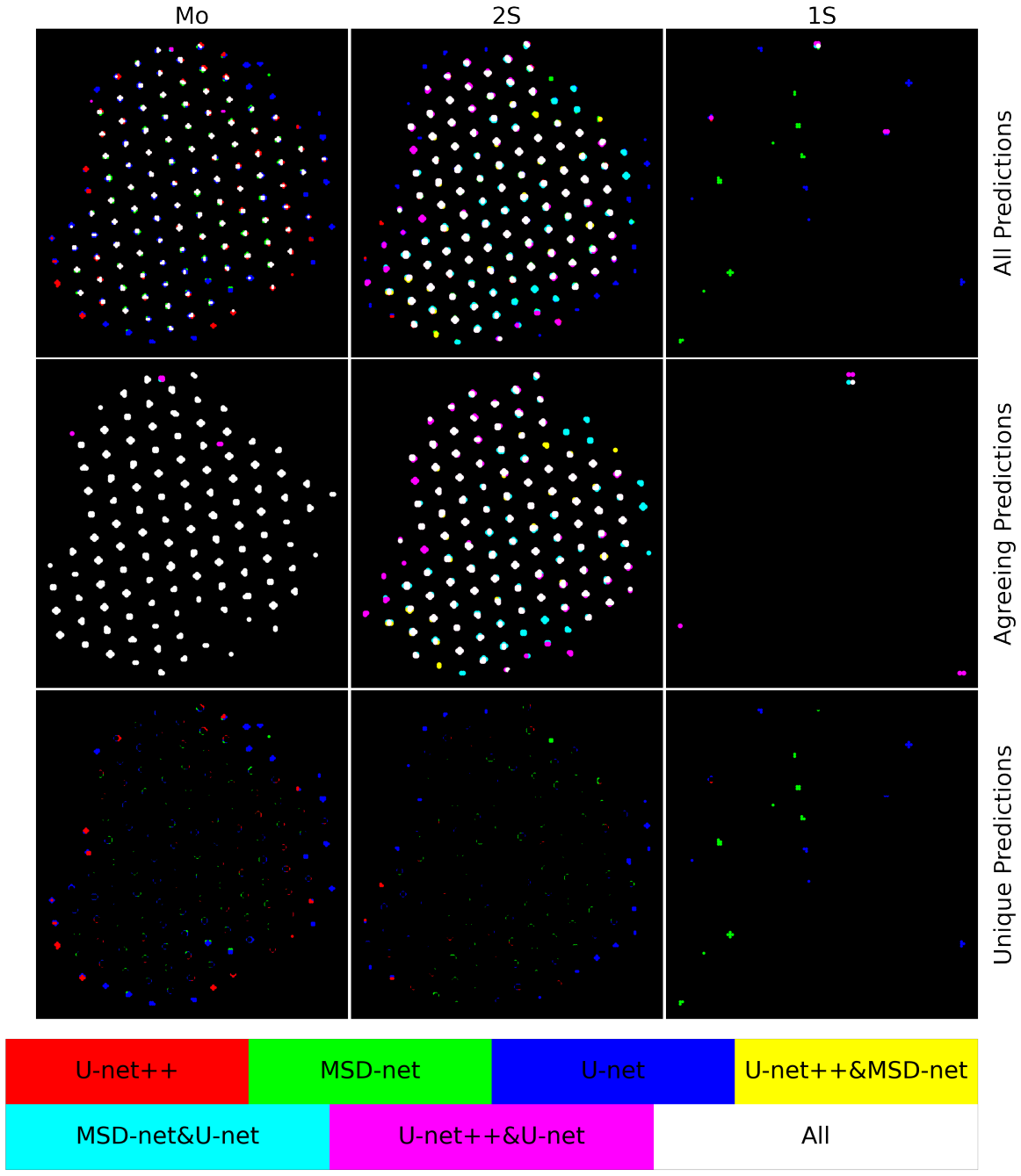


Figure 4.8: RGB coded Prediction of atomic positions of real world HR-TEM image with agreement among models. The columns represents the channels corresponding to the atomic columns Mo, 2S and 1S respectively. The first row corresponds to all predictions. The second row is only of predictions where 2 or more models are in agreement. The last row displays all unique predictions. The dot-size of unique predictions is 3 times lower than the others. For equal dot-size see appendix C figure C.2

It is observed, that the all models agree on what sub-lattice corresponds to molybdenum and sulphur. To validate this consensus, four additional real-world HR-TEM focal series were examined, see appendix C.

5 Discussion

The assessment of Convolutional Neural Network (CNN) architectures for identifying atomic columns in MoS_2, relied heavily on the F1-scores and qualitative analysis. In the evaluation, two types of F1-scores were employed, the atomic F1-score and pixel F1-score. The atomic F1-score, considering atomic positions, mitigates the impact of minor positional deviations by classifying predictions within a certain distance as true positives. Conversely, the pixel F1-score evaluates the networks' performance pixel-wise. This metric gives details about the models ability to learn more accurate positions of the atoms. All models performed best in the atomic F1-score which is anticipated since slight positional deviations between predictions and ground truth are classified as true positives, when using this metric.

5 different models for U-net and U-net++, and 4 different models for MSD-net were trained. It was observed, that the U-net, and U-net++ displayed minor improvements in performance after 10^5 parameters. The MSD-net displayed a positive learning curve for the first three model complexities. The last model complexity, where the kernel size were increased from 5 to 9 had a lower F1-score, comparable to the least complex MSD-net model. This indicates, that increasing the kernel size is less critical than increasing the number of channels and layers.

Because the concatenation layers consumed large amounts of memory, the model complexity of the MSD-net was limited by available memory. This resulted in the last to models in table 3 not being trained and analyzed.

It was evident, that all three chosen models showed improvements in performance when increasing the training set size, indicating that the models were learning, see figure 4.3. In terms of the atomic F1-score, both U-net and U-net++ plateaued in performance after a training set size of 10^2 structures, whereas the MSD-net required a training set size of 10^3 structures in order to plateau. When using the pixel F1-score as performance measure, all three models showed continuous improvement. This suggests, the three models were refining their ability to predict the atomic positions.

Incorporating the frozen Phonon method and Fourier filtering into the simulation of MoS_2 flake stacked on graphene layers showed promising results when testing on simulated data. The models performance in table 5 displays high pixel F1-scores when counting in the background. This metric is less interesting, as the background channel inflates the metric. The pixel F1 scores without the background was ≈ 0.7 for all three models, with U-net++ displaying the highest pixel F1-score of 0.73.

On the other hand, the atomic F1-scores was ≈ 0.99 for all models, with the U-net and U-net++ displaying the highest atomic F1-scores of 0.998 and 0.997 respectively. The MSD-net performed a atomic F1-score of 0.992. For all three metrics, the MSD-net scored the lowest, however, they all performed somewhat similar.

When comparing the different models predictions to the ground truth in figure 4.6, all three models made minor mistakes on their predictions. These mistakes often manifested themselves at the boundary of the graphene support, the edge of the MoS_2 flake and in the 1S atomic columns. This indicated that the models excelled at learning patterns, but encountered difficulty in accurately predicting non-periodic features.

In the last section of this project, the models were evaluated on real world HR-TEM focal series. As there is no ground truth, this evaluation was purely qualitative. From figure 4.8, it was observed that the discrepancies between the models mostly manifested themselves in the edge of the MoS_2 flake and in the 1S atomic column. The models predicted similar results of the periodic features, such as the Mo and 2S atomic columns positions.

As all models are trained with the same training set, the observed agreement among them may be linked to a bias introduced during training. Therefore, while the models show promising performance, it is challenging to definitively conclude their accuracy on real-world data. The discrepancies observed between the models in the MoS_2 edge and the 1S atomic column in the predictions of real-world HR-TEM focal series are notable. Additionally, errors frequently appeared at the boundary of the

Fourier filtering, at the edge of the MoS_2 flake, and in the 1S atomic columns in the predictions of simulated data. These issues suggest that the model lacks sufficient information to distinguish between 2S, 1S, Mo, and the background, despite its apparent ability to recognize patterns effectively. Understanding the 1S atomic column and edges is important, because these regions play a central role in catalytic processes [34]. Therefore, improving the model to better recognize the MoS_2 edge and 1S atomic columns is imperative.

A previous study found similar results when testing their CNN on real world HR-TEM images, however here, the CNN were not able to distinguish which atomic sub-lattice belonged to Mo and 2S [35]. As seen from figure 4.7, and appendix C, the models are agreeing on the sub-lattices, suggesting that this problem is mitigated.

6 Conclusion and Outlook

In this project, structures of graphene-supported molybdenum disulphide was constructed in order to simulate the exitwaves using the multislice method from which HR-TEM focal series images was created.

Incorporating the frozen phonon method in order to imitate thermal vibrations and Fourier filtering of the graphite support, proved as promising training data for the three convolutional neural network architectures: U-net, U-net++ and MSD-net.

The evaluation of these models demonstrated their ability to classify atomic columns (Mo, 2S and 1S) in simulated HR-TEM images. Both U-net and U-net++ showed minimal performance improvement when the training set size exceeded 10^2 structures, whereas the MSD-net required a training size of 10^3 structures in order to plateau when using the atomic F1-score as performance metric. Increasing the number of parameters for the U-net and U-net++ above 10^5 parameters showed only minor improvements. The pixel F1-score was approximately 0.7 for all models, while the atomic F1-score was around 0.99, indicating high precision in identifying atomic positions within 10^{-11}m. However, the MSD-net exhibited slightly lower performance compared to U-net and U-net++, which might be attributed to its limited model complexity due to memory constraints.

Qualitative evaluation on real-world HR-TEM focal series revealed that all models struggled with non-periodic features, such as the edges of MoS_2 flakes and the identification of 1S atomic columns. This suggests that while the models are able to learn patterns, they lack the information necessary to accurately predict the non-periodic features. However, the models were agreeing on what sub-lattice belonged to Mo and S, indicating that the CNN's correctly classified the sub-lattices.

Another study tested similar models on simulated data with defocus outside the training set range [12]. While the models identified Mo and 2S atomic columns accurately, they struggled significantly with 1S columns. Future work could consider expanding the training set to include a larger defocus range. Additionally, the models could be trained on a variety of 2D materials [35]. In addition to investigating the spatial distribution of the atomic columns, the dynamics of defects (1S atomic columns) could be researched [36].

Manually labeling real-world HR-TEM focal series would provide quantitative measures, like the atomic F1-score, to better validate model performance.

Transitioning to a framework like PyTorch could also offer better flexibility and efficiency in training more complex models. Using PyTorch could also improve with the memory constraint of the MSD-net training making it possible to train more complex MSD-net architectures [37].

In summary, while the current CNN architectures show strong potential in classifying atomic columns in simulated data, their application to real-world HRTEM images requires further refinement and validation to achieve reliable and accurate results.

7 References

[1] Mamidala Jagadesh Kumar. Is india going to be a major hub of semiconductor chip manufacturing? *IETE Technical Review*, 38:279 – 281, 2021.

[2] Tong Bor Tang. Semiconductors used in optoelectronic devices â an overview. *Journal of Electronic Materials*, 4:1229–1247, 1975.

[3] David Pollard and John M. Woodley. Biocatalysis for pharmaceutical intermediates: the future is now. *Trends in biotechnology*, 25 2:66–73, 2007.

[4] Michael Kerby, Thomas Francis Degnan, David Owen Marler, and Jeffrey Scott Beck. Advanced catalyst technology and applications for high quality fuels and lubricants. *Catalysis Today*, 104:55–63, 2005.

[5] Jun Mao, Yong Wang, Zhilong Zheng, and Dehui Deng. The rise of two-dimensional mos2 for catalysis. *Frontiers of Physics*, 13:1–19, 2018.

[6] Gloria Hyunjung Kwak and Pan Hui. Deephealth: Deep learning for health informatics. *ArXiv*, abs/1909.00384, 2019.

[7] Dan Otter, Julian R. Medina, and Jugal Kumar Kalita. A survey of the usages of deep learning for natural language processing. *IEEE Transactions on Neural Networks and Learning Systems*, 32:604–624, 2020.

[8] Wim E. Crusio, John D Lambris, Gobert N. Lee, and Hiroshi Fujita. Deep learning in medical image analysis: Challenges and applications. *Deep Learning in Medical Image Analysis*, 2020.

[9] Catherine K. Groschner, Christina Choi, and Mary C. Scott. Methodologies for successful segmentation of hrtem images via neural network. *ArXiv*, abs/2001.05022, 2020.

[10] Jakob Schiøtz, Jacob Madsen, Bjarke J. G. Østergaard, Anders S. Dreisig, Pei Liu, Stig Helveg, Ole Winther, Jens Kling, Jakob Birkedal Wagner, and Thomas Willum Hansen. Using neural networks to identify atoms in hrtem images. *Microscopy and Microanalysis*, 25(S2):216â217, 2019.

[11] Jacob Madsen. *Quantitative Image Simulation and Analysis of Nanoparticles*. PhD thesis, 2017.

[12] Matthew Helmi Leth Larsen. *Machine Learning for Assisting Atomic- Resolution Electron Microscopy*. PhD thesis, 2023.

[13] GW Bailey, RVW Dimlich, KB Alexander, JJ McCarthy, TP Pretlow, and HW Zandbergen. The Use of Phase and Amplitude Information of Reconstructed Exit Waves. *Microscopy and Microanalysis*, 3(S2):1029–1030, 08 1997.

[14] E. J. Kirkland. Advanced computing in electron microscopy third edition. Springer Cham, 2020.

[15] Earl J. Kirkland. *Calculation of Images of Thin Specimens*, pages 99–141. Springer International Publishing, Cham, 2020.

[16] David B. Williams and C. Barry Carter. Image simulation. *Transmission Electron Microscopy*, pages 533–548, 2009.

[17] Claudie Mory, Christian Colliex, and John M. Cowley. Optimum defocus for stem imaging and microanalysis. *Ultramicroscopy*, 21:171–177, 1987.

[18] O. Scherzer. The Theoretical Resolution Limit of the Electron Microscope. *Journal of Applied Physics*, 20(1):20–29, 01 1949.

[19] M. Lentzen. The tuning of a zernike phase plate with defocus and variable spherical aberration and its use in hrtem imaging. *Ultramicroscopy*, 99(4):211–220, 2004.

[20] *INTRODUCTION TO SOLID STATE PHYSICS, 7TH ED.* Wiley India Pvt. Limited, 2007.

[21] Ehren M. Mannebach, Renkai Li, Karel Alexander Duerloo, Clara Nyby, Peter Zalden, Theodore Vecchione, Friederike Ernst, Alexander Hume Reid, Tyler Chase, Xiaozhe Shen, Stephen Weathersby, Carsten Hast, Robert Hettel, Ryan Coffee, Nick Hartmann, Alan R. Fry, Yifei Yu, Linyou Cao, Tony F. Heinz, Evan J. Reed, Hermann A. DÃ¼rr, Xijie Wang, and Aaron M. Lindenberg. Dynamic structural response and deformations of monolayer mos2 visualized by femtosecond electron diffraction. *Nano Letters*, 15(10):6889–6895, 2015.

[22] Y EPELBOIN, F MORRIS, and A RIMSKY. Image-enhancement of x-ray topographs by fourier filtering. *JOURNAL OF PHYSICS D-APPLIED PHYSICS*, 26(4A):A15–A18, APR 14 1993.

[23] Michael Nielsen. Neural networks and deep learning. Website, Dec 2019. Accessed: 01/03-2024.

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

[25] Sunitha Basodi, Chunyan Ji, Haiping Zhang, and Yi Pan. Gradient amplification: An efficient way to train deep neural networks, 2020.

[26] Francois Chollet. *Deep learning with python*. Manning Publications, 2018.

[27] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation, 2015.

[28] Daniel Pelt and James Sethian. A mixed-scale dense convolutional neural network for image analysis. *Proceedings of the National Academy of Sciences*, 115:201715832, 12 2017.

[29] Ask Hjorth Larsen, Jens Joergen Mortensen, and Jakob Blomqvist et. al. The atomic simulation environmentâa python library for working with atoms. *Journal of Physics: Condensed Matter*, 29(27):273002, jun 2017.

[30] Hugh O. Pierson. 3 - graphite structure and properties. In Hugh O. Pierson, editor, *Handbook of Carbon, Graphite, Diamonds and Fullerenes*, pages 43–69. William Andrew Publishing, Oxford, 1993.

[31] Franciszek Rozploch, Jaromir Patyk, and Jan Stankowski. Graphenes bonding forces in graphite. *Acta Physica Polonica A*, 112:557–562, 2007.

[32] Madsen J and Susi T. The abtem code: transmission electron microscopy from first principles [version 2; peer review: 2 approved]. open res europe 2021, 1:24. 2021.

[33] Matthew Helmi Leth Larsen, William Bang Lomholdt, Cuauhtemoc Nuñez Valencia, Thomas W. Hansen, and Jakob Schiøtz. Quantifying noise limitations of neural network segmentations in high-resolution transmission electron microscopy. *Ultramicroscopy*, 253, 2023.

[34] Line Sjolte Byskov, Jens Kehlet Nørskov, B. S. Clausen, and H. Topsøe. Edge termination of mos2 and comos catalyst particles. *Catalysis Letters*, 64(2-4):95–99, 2000.

[35] Matthew Helmi Leth Larsen, Frederik Dahl, Lars P. Hansen, Bastian Barton, Christian Kisielowski, Stig Helveg, Ole Winther, Thomas W. Hansen, and Jakob Schi. Reconstructing the exit wave of 2d materials in high-resolution transmission electron microscopy using machine learning. *Ultramicroscopy*, 243:113641, 2023.

[36] Tarak K. Patra, Fu Zhang, Daniel S. Schulman, Henry Chan, Mathew J. Cherukara, Mauricio Terrones, Saptarshi Das, Badri Narayanan, and Subramanian K. R. S. Sankaranarayanan. Defect dynamics in 2-d mos2 probed by using machine learning, atomistic simulations, and high-resolution microscopy. *ACS Nano*, 12(8):8006–8016, 2018. PMID: 30074765.

[37] Seung Won Min, Kun Wu, Sitao Huang, Mert Hidayetouglu, Jinjun Xiong, Eiman Ebrahimi, Deming Chen, and Wen mei W. Hwu. Pytorch-direct: Enabling gpu centric data access for very large graph neural network training with irregular accesses. *ArXiv*, abs/2101.07956, 2021.

A APPENDIX:Theory

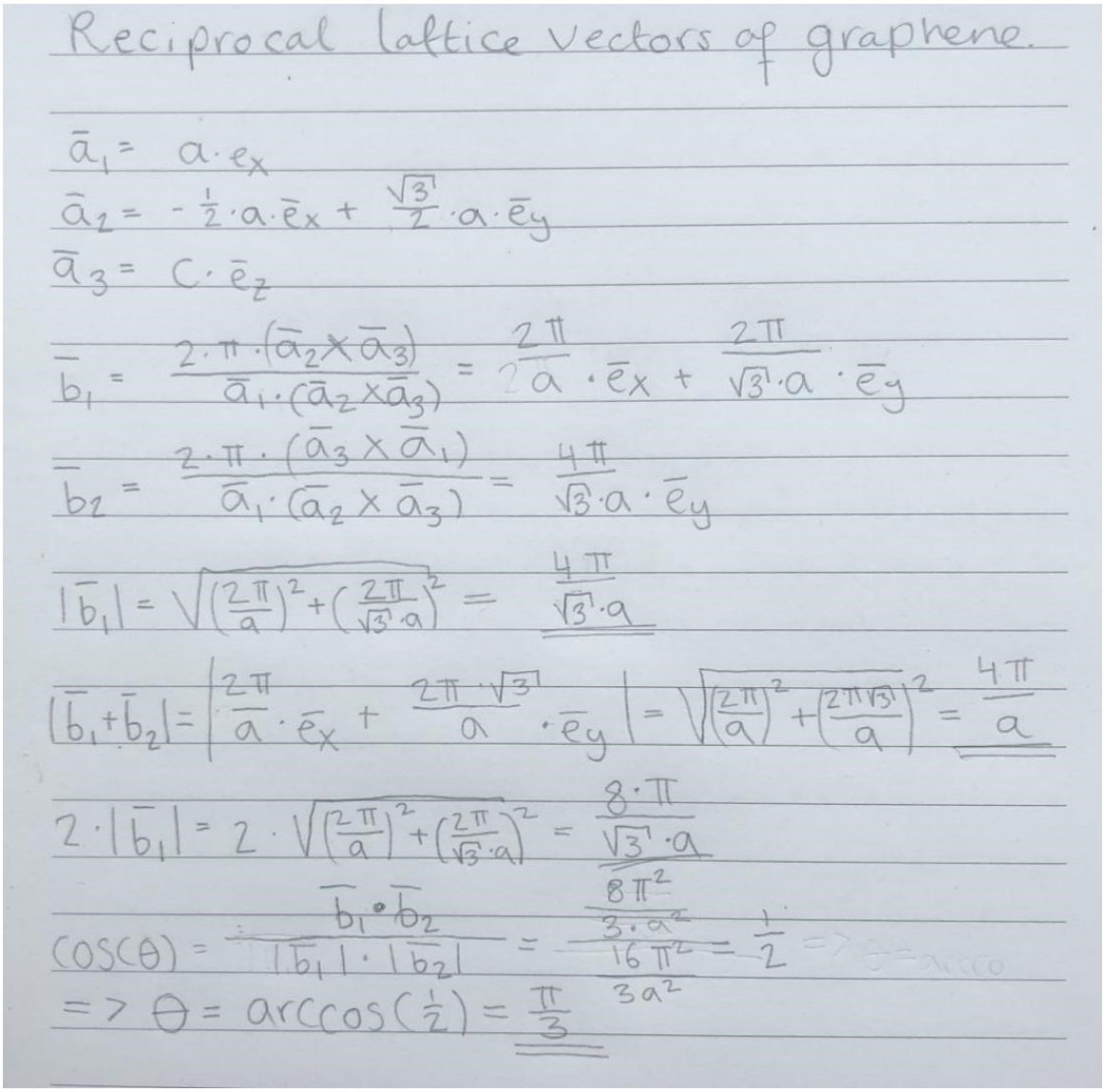


Figure A.1: Calculation of reciprocal lattice vectors of graphene.

B APPENDIX: F1-scores

```python
#Script for generating pixelwise F1-scores for one specific model, and all training sizes. ( without background)

#Every model ( defined by a training size and type of model) has a 7X3X1000 F1 score structure - each row defines which replica it is
# and the values in that row is that models pixel F1 scores. The 7 channels each represents: 0 => trainingsize = 10,
# 1 => training size = 20, 2 => training size = 50, 3 => trainingsize = 100,  4 => trainingsize = 200,  5 => trainingsize = 500,
# 6 => trainingsize = 1000,

import os
import numpy as np
from stm.feature.peaks import find_local_peaks, refine_peaks
from temnn.data.mods import local_normalize
import tensorflow.keras as keras
import json
import os
import numpy as np



class EvaluationMetrics:
    def __init__(self, predictions, ground_truth):
        self.ground_truth = (ground_truth - np.min(ground_truth)) / (np.max(ground_truth) - np.min(ground_truth)).flatten()
        self.predictions =  (predictions - np.min(predictions)) / (np.max(predictions) - np.min(predictions)).flatten()
        self.predictions = np.where(predictions > 0.5, 1, 0).flatten()
        self.ground_truth = np.where(ground_truth > 0.5, 1, 0).flatten()


    def calculate_confusion_matrix(self):
        TP = np.sum(np.logical_and(self.predictions == 1, self.ground_truth == 1))
        FP = np.sum(np.logical_and(self.predictions == 1, self.ground_truth == 0))
        FN = np.sum(np.logical_and(self.predictions == 0, self.ground_truth == 1))
        TN = np.sum(np.logical_and(self.predictions == 0, self.ground_truth == 0))
        return TP, FP, FN, TN



    def calculate_f1_score(self):
        TP, FP, FN, _ = self.calculate_confusion_matrix()
        precision = TP/(TP + FP)
        recall = TP/(TP + FN)
        f1_score = 2 * (precision * recall) / (precision + recall)
        return f1_score


    def threshold_image(self, image, threshold):
        return np.where(image > threshold, 1, 0)


    def calculate_fpr_tpr(self, thresholds):
        fpr_list = []
        tpr_list = []
        for threshold in thresholds:
            predictions_thresholded = self.threshold_image(self.predictions, threshold)
            TP, FP, FN, TN = self.calculate_confusion_matrix()  # Just using the internal variables
            FPR = FP / (FP + TN)
            TPR = TP / (TP + FN)
            fpr_list.append(FPR)
            tpr_list.append(TPR)
        return fpr_list, tpr_list



    def evaluate(self, prediction_threshold=0.5):
        TP, FP, FN, TN = self.calculate_confusion_matrix()
        f1_score = self.calculate_f1_score()
        return TP, FP, FN, TN, f1_score
```

Figure B.1: Part 1 of script used to generate Pixel F1-scores.

```
66
67 # Function to calculate evaluation metrics
68 def calculate_metrics(model, data_dir, custom_threshold=0.6):
69     metrics_collector = []
70     for i in range(1):
71         for j in range(1000):
72             imagenumber = f'{i:03d}_{j:04d}'
73             #print("analyzing:", imagenumber)
74
75             try:
76                 tmp = np.load(os.path.join(data_dir, 'images_labels', 'image_label_'+imagenumber+'.npz'))
77
78                 image = tmp['image']
79                 normimage = local_normalize(image.copy(), normalizedistance, normalizedistance)
80                 predictions = model.predict(normimage)
81                 GT = tmp['label']
82
83                 #Removing background
84                 predictions = np.delete(predictions, 3, axis=3)
85                 GT = np.delete(GT, 3, axis=3)
86
87                 eval_metrics = EvaluationMetrics(predictions, GT)
88                 TP, FP, FN, TN, f1_score = eval_metrics.evaluate(prediction_threshold=custom_threshold)
89                 metrics_collector.append({
90                     'TP': TP,
91                     'FP': FP,
92                     'FN': FN,
93                     'TN': TN,
94                     'f1_score': f1_score
95                 })
96                 tmp.close()
97             except FileNotFoundError:
98                 print(f'Image {imagenumber}.npz not found.')
99
100     return metrics_collector
101
102 # load the data
103 data_dir = "../workflow/simulation_data/MoS2_graphite_for_experimental-test/"
104 parameters = json.load(open(os.path.join(data_dir, 'parameters.json')))
105 parameters['debug'] = False
106 image_size = parameters['image_size']
107 sampling = np.mean(parameters['sampling'])
108 normalizedistance = parameters['normalizedistance'] / sampling
109 #Training set sizes
110 sizes = [10,20,50,100,200,500,1000]
111
112 Total_F1 = np.zeros((7, 3, 1000))
113 for i in range(7):
114     for j in range(3):
115         folder = '../workflow/trained_networks/MoS2_graphite_experimental_FFT_UnetPlusPlus' #U-netPlusPlus
116
117         #MoS2_graphite_experimental_FFT_UnetPlusPlus_3_1000
118         # load the network path and the parameters
119         model_path = os.path.join(f'{folder}_{j+1}_{sizes[i]}', 'model-0')
120
121         # load the network
122         #print('Loading model', model_path)
123         model = keras.models.load_model(model_path)
124
125
126         metrics_collector = calculate_metrics(model, data_dir)
127
128         Total_F1[i, j] = np.array([d['f1_score'] for d in metrics_collector])
```

Figure B.2: Part 2 of script used to generate Pixel F1-scores.

```
129
130 # Save metrics in .npz file
131
132 np.savez("pixel F1 TrainingSize UnetPlusPlus.npz", Total F1=Total F1)
```

Figure B.3: Part 3 of script used to generate Pixel F1-scores.

```python
#Script for generating atomic F1-scores for Unet++ for different training set sizes.


import os
import numpy as np
from stm.feature.peaks import find_local_peaks, refine_peaks
from temnn.data.mods import local_normalize
import tensorflow.keras as keras
import json
import os
import numpy as np
from scipy.spatial import cKDTree as KDTree
from stm.feature.peaks import find_local_peaks, refine_peaks

def calculate_metrics(model, data_dir):
    metrics_collector = []
    for i in range(1):
        for j in range(1000):
            imagenumber = f'{i:03d}_{j:04d}'
            #print("analyzing:", imagenumber)
            try:
                tmp = np.load(os.path.join(data_dir, 'images_labels', 'image_label_'+imagenumber+'.npz'))

                image = tmp['image']
                parameters = json.load(open(os.path.join(data_dir, 'parameters.json')))
                parameters['debug'] = False
                # load the data
                image_size = parameters['image_size']
                sampling = np.mean(parameters['sampling'])
                normalizedistance = parameters['normalizedistance'] / sampling

                normimage = local_normalize(image.copy(), normalizedistance, normalizedistance)
                predictions = model.predict(normimage)
                predictions = np.where(predictions > 0.5, 1, 0) # slet når det er disk og ikke gaussian
                GT = tmp['label']
                FP = 0
                TP = 0
                FN = 0
                FPP = 0
                TPP = 0

                for k in range(4):
                    peaks = find_local_peaks(predictions[0,:,:,k], min_distance=10, threshold=0.30, exclude_border=10,exclude_adjacent=True).astype(int)

                    wavepeaks = find_local_peaks(GT[0,:,:,k], min_distance=10, threshold=0.30, exclude_border=10,exclude_adjacent=True).astype(int)

                    kdt = KDTree(wavepeaks)
                    hits = kdt.query_ball_point(peaks, 3)
                    hits = np.array(hits)
                    for hit in hits:
                        if len(hit) == 0:
                            FPP += 1

                    FP += FPP
                    TP += len(hits) - FPP
                    TPP = len(hits) - FPP
                    FN += len(wavepeaks) - TPP
                    FPP = 0

                #print("TP:", TP)
                #print("FN:", FN)
                #print("FP:", FP)
                precision = TP/(TP + FP)
```

Figure B.4: Part 1 of script used to generate atomic F1-scores.

```
                recall = TP/(TP + FN)
                f1_score = 2 * (precision * recall) / (precision + recall)
                metrics_collector.append({
                    'TP': TP,
                    'FP': FP,
                    'FN': FN,
                    'f1_score': f1_score
                })
                tmp.close()
            except FileNotFoundError:
                print(f'Image {imagenumber}.npz not found.')

    return metrics_collector


#If changing network, remember to change folder path, and change the file name in the last line with np.savez

# load the data
data_dir = "../workflow/simulation_data/MoS2_graphite_for_experimental-test/"
parameters = json.load(open(os.path.join(data_dir, 'parameters.json')))
parameters['debug'] = False
image_size = parameters['image_size']
sampling = np.mean(parameters['sampling'])
normalizedistance = parameters['normalizedistance'] / sampling
#Training set sizes
sizes = [10,20,50,100,200,500,1000]

Total_F1 = np.zeros((7, 3, 1000))
for i in range(7):
    for j in range(3):
        folder = '../workflow/trained_networks/MoS2_graphite_experimental_FFT_UnetPlusPlus' #U-netPlusPlus

        # load the network path and the parameters
        model_path = os.path.join(f'{folder}_{j+1}_{sizes[i]}', 'model-0')

        # load the network
        #print('Loading model', model_path)
        model = keras.models.load_model(model_path)


        metrics_collector = calculate_metrics(model, data_dir)

        Total_F1[i, j] = np.array([d['f1_score'] for d in metrics_collector])




# Save metrics in .npz file

np.savez("Atomic_F1_TrainingSize_UnetPlusPlus.npz", Total_F1=Total_F1)
```

Figure B.5: Part 2 of script used to generate atomic F1-scores.

C APPENDIX: Experimental Plot

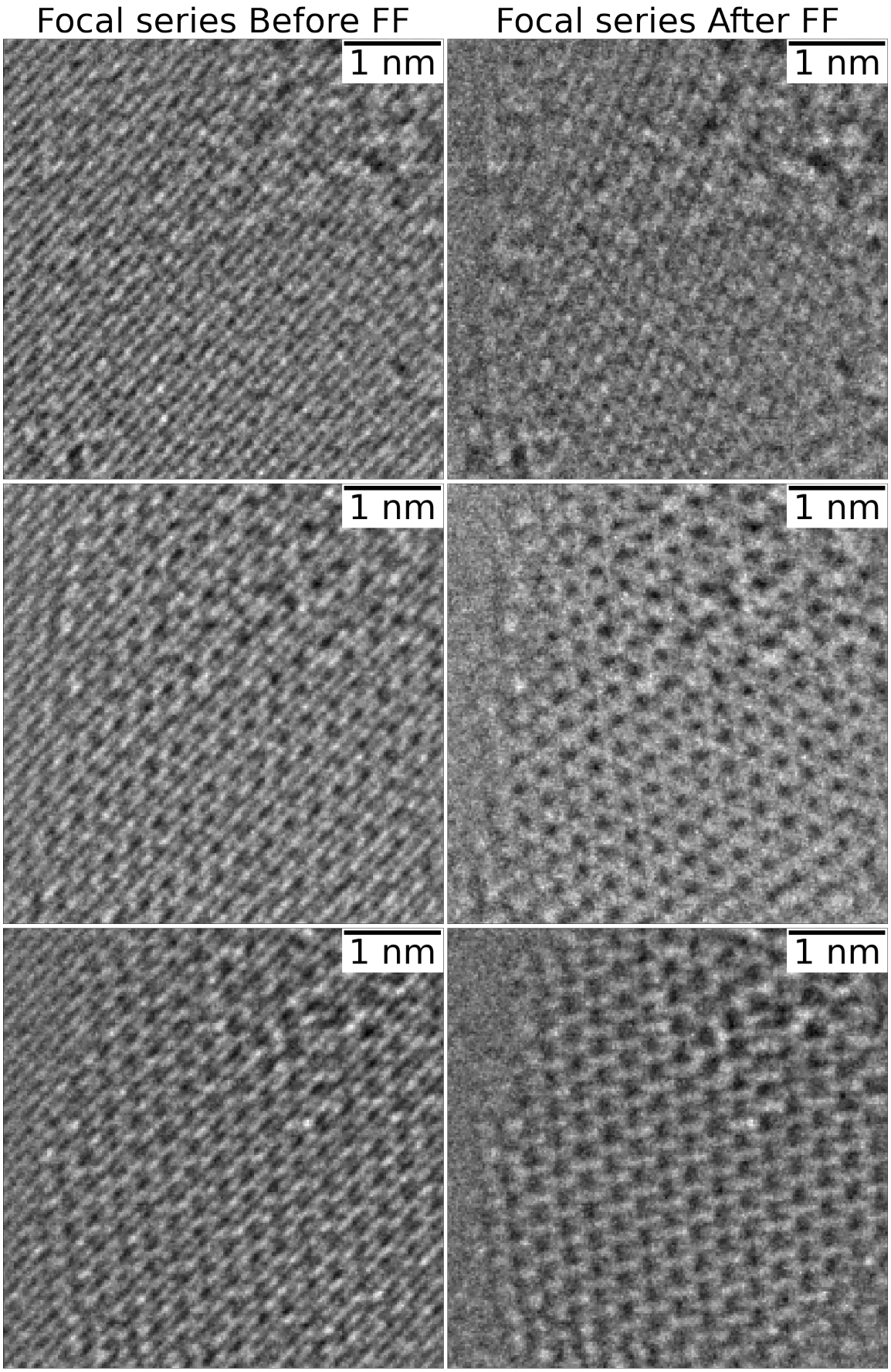


Figure C.1: Real world HRTEM focal series before and after Fourier filtering.

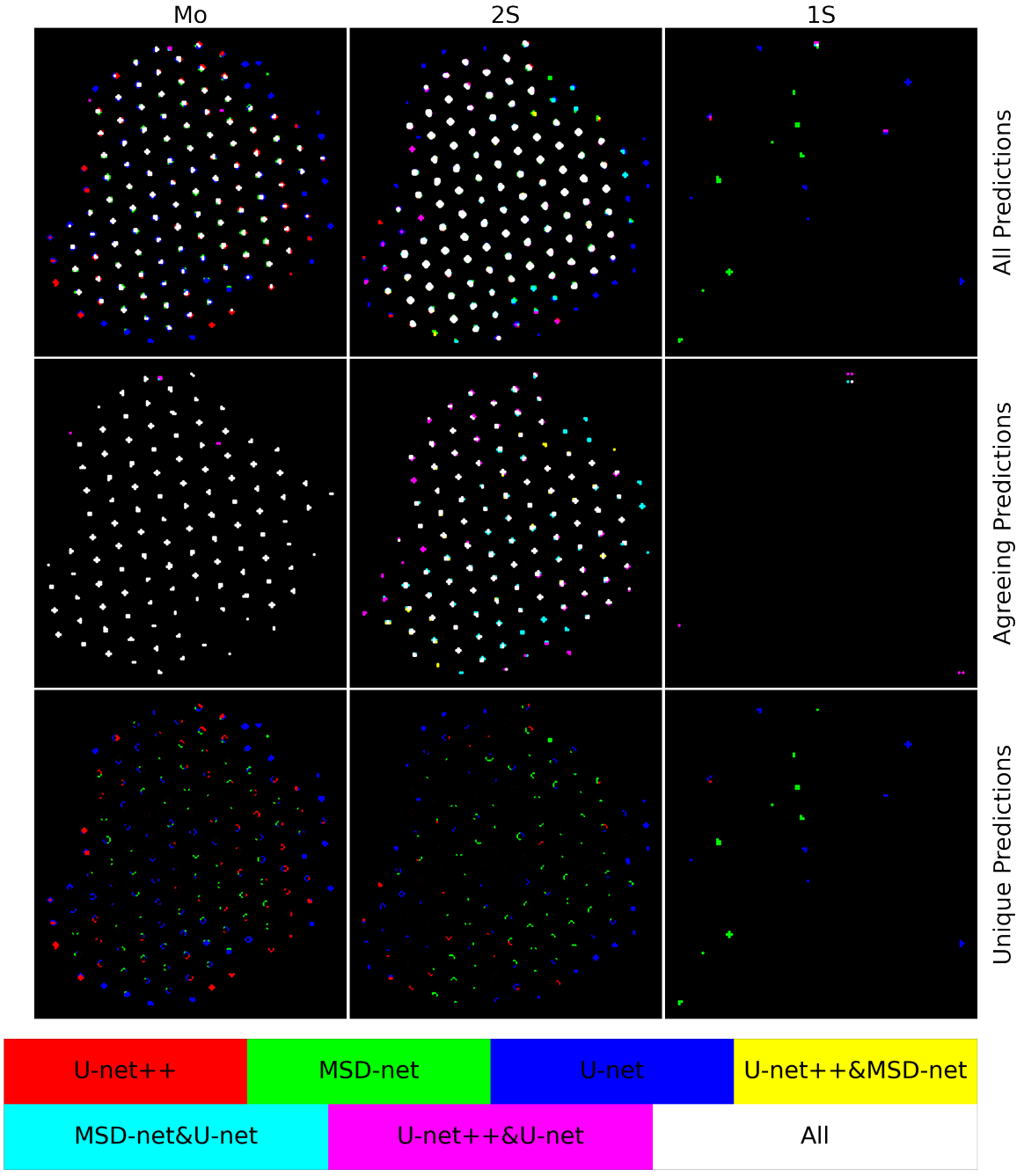


Figure C.2: RGB coded Prediction of atomic positions of real world HRTEM image with agreement among models. The columns represents the channels corresponding to the atomic columns Mo, 2S and 1S respectively. The first row corresponds to all predictions. The second row is only of predictions where 2 or more models are in agreement. The last row displays all unique predictions. All dot-sizes are equal.

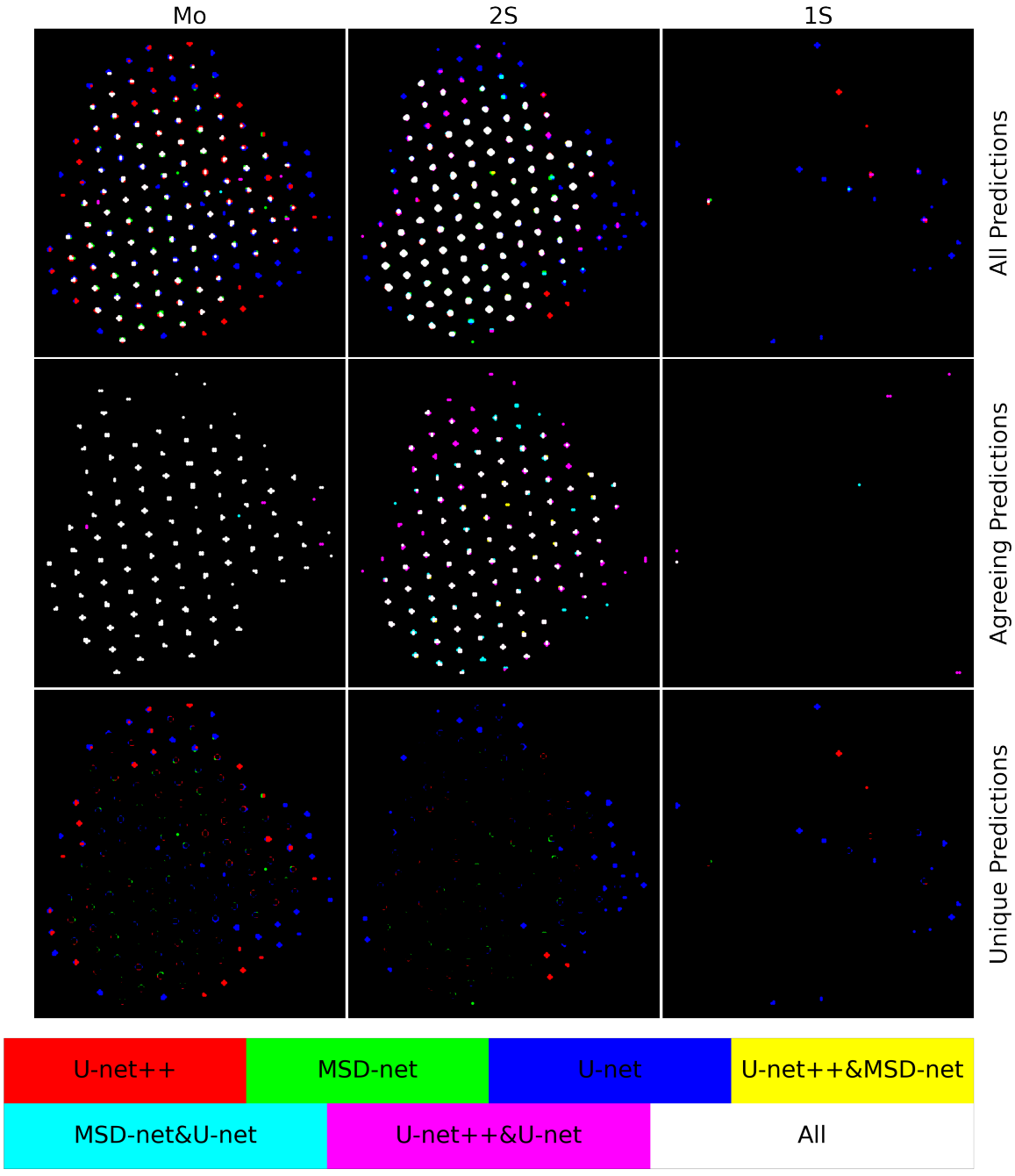


Figure C.3: The defocus for the images are approximately 170Å, 120Å and 70Å. RGB coded Prediction of atomic positions of real world HRTEM image with agreement among models. The columns represents the channels corresponding to the atomic columns Mo, 2S and 1S respectively. The first row corresponds to all predictions. The second row is only of predictions where 2 or more models are in agreement. The last row displays all unique predictions. The dot-size of unique predictions is 3 times lower than the others.

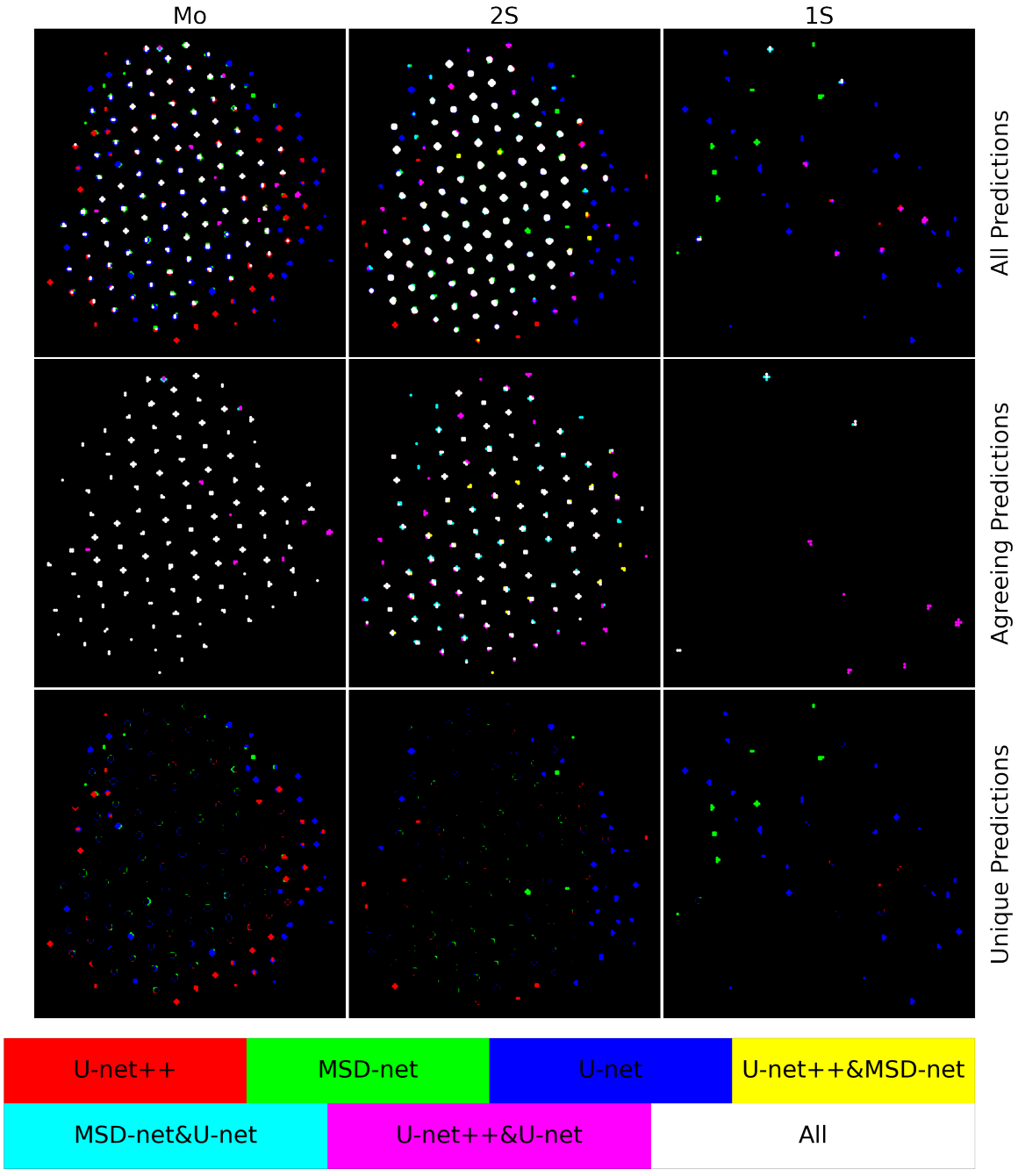


Figure C.4: The defocus for the images are approximately 115Å, 65Å and 15 Å. RGB coded Prediction of atomic positions of real world HRTEM image with agreement among models. The columns represents the channels corresponding to the atomic columns Mo, 2S and 1S respectively. The first row corresponds to all predictions. The second row is only of predictions where 2 or more models are in agreement. The last row displays all unique predictions. The dot-size of unique predictions is 3 times lower than the others.

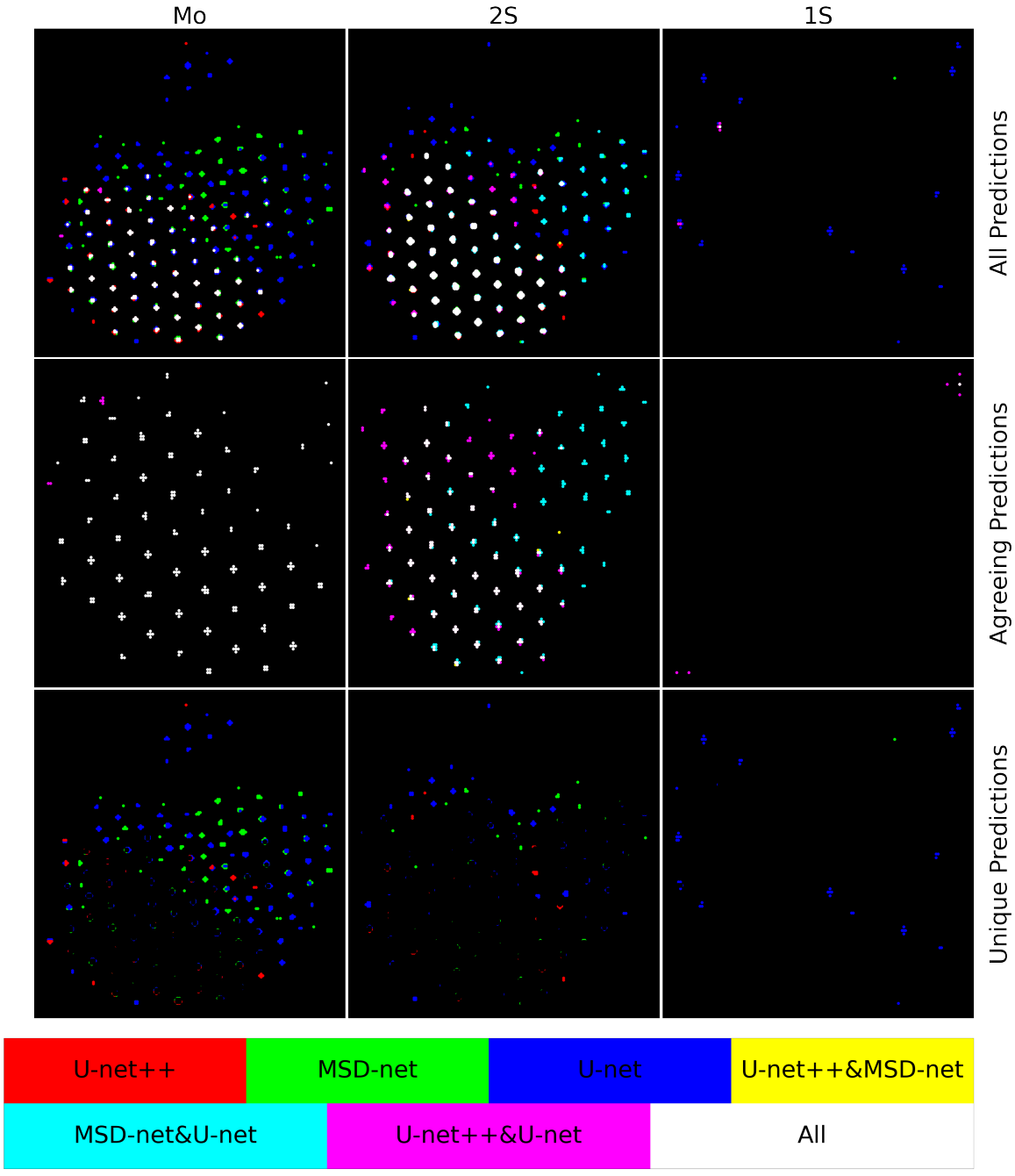


Figure C.5: The defocus for the images are approximately -30Å, -80Å and -130Å.. RGB coded Prediction of atomic positions of real world HRTEM image with agreement among models. The columns represents the channels corresponding to the atomic columns Mo, 2S and 1S respectively. The first row corresponds to all predictions. The second row is only of predictions where 2 or more models are in agreement. The last row displays all unique predictions. The dot-size of unique predictions is 3 times lower than the others.

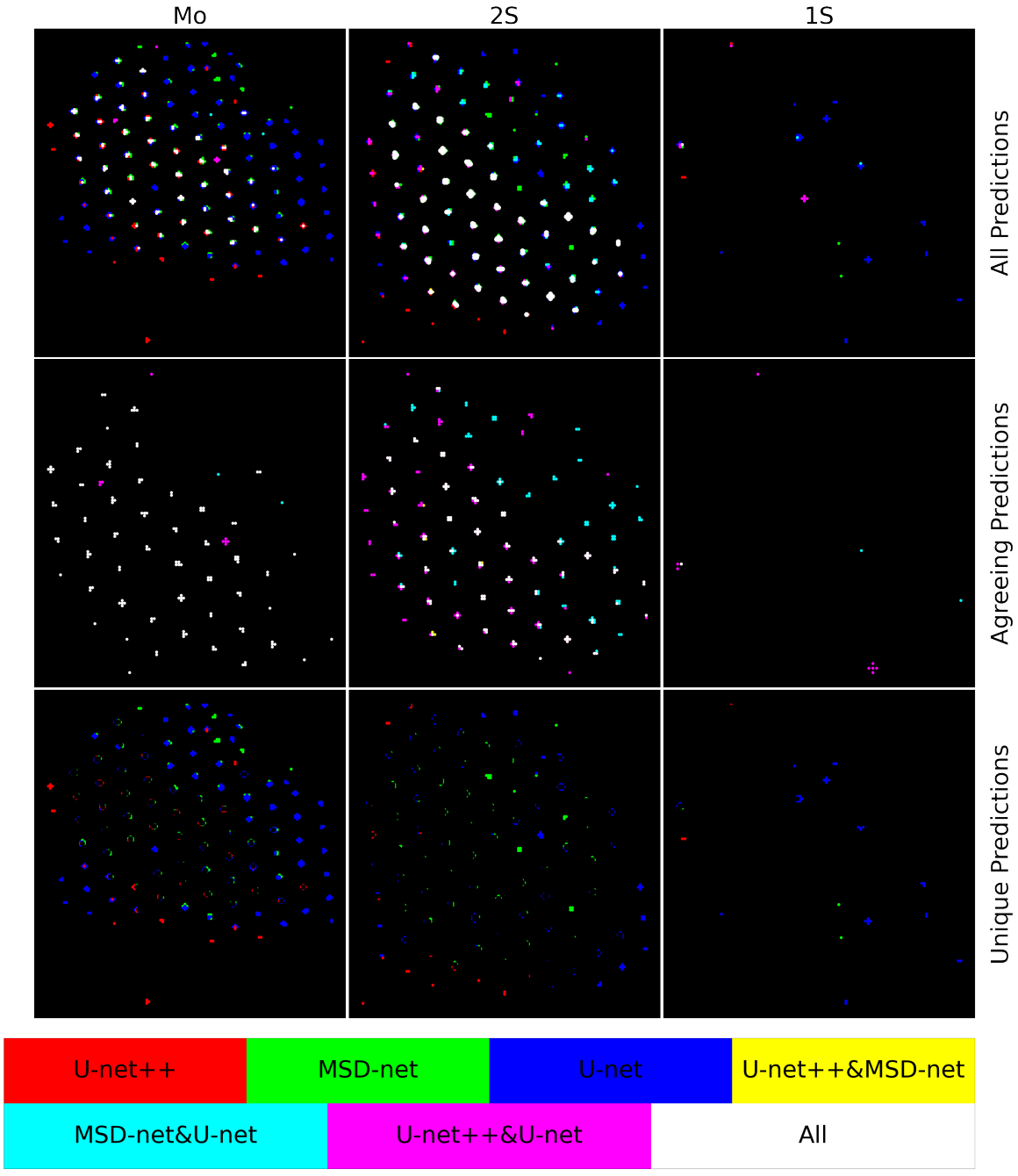


Figure C.6: The defocus for the images are approximately 81Å, 31Å and -19Å. RGB coded Prediction of atomic positions of real world HRTEM image with agreement among models. The columns represents the channels corresponding to the atomic columns Mo, 2S and 1S respectively. The first row corresponds to all predictions. The second row is only of predictions where 2 or more models are in agreement. The last row displays all unique predictions. The dot-size of unique predictions is 3 times lower than the others.

D APPENDIX: Project Plan

2 Objectives

2.1 Objectives

- Setup Computer to Niflheim
- Understand code and theory
- Genereate simulation data of two-dimensional MoS2
- Train convolutional network on generated data
- Data analysis

Table 1: Task Schedule

Task	Feb	March	April	May	June
Task 1: Setup comp. To niflheim	x				
Task 2: Understand code and theory	x				
Task 3: Generate simulations of MoS2		x			
Task 4: Train CNN on simulation		x			
Task 5: Optimise simulations			x		
Task 6: Optimise arch. Of CNN			x	x	

Figure D.1: Project plan.