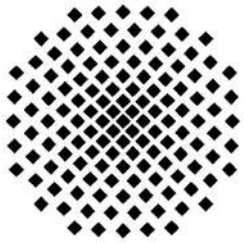

Universität
Stuttgart

École Centrale de Nantes - Universität Stuttgart

Machine Learning for Transmission Electron Microscopy Tomography with a Limited Number of Projections

---

Internship supervisors:
Guido SCHMITZ
Roham TALEI JEID

School supervisor:
Didier LIME

Student:
Alix BRANCHUT

Contents

Abstract

Covalent organic frameworks (COFs) and Metal-organic frameworks (MOFs) are porous polymers that form two- or three-dimensional structures. These materials are very sensitive to electron beams, making them difficult to observe using a transmission electron microscope (TEM). Images of these structures have already been obtained thanks to recent advances in the field, but tomography, a technique consisting of reconstructing a 3D structure with 2D projections obtained at different angles, has not yet been achieved, since only a small number of images can be obtained before the material is damaged. On the other hand, a number of articles have demonstrated the possibility of performing tomography with a limited number of projections, by implementing Machine Learning tools in the reconstruction algorithms. This report describes the concrete implementation of these algorithms, and the application of these algorithms to a MOF particle observed under an electron microscope, as well as to simulations of MOF projections at atomic resolution.

1 Introduction

1.1 Transmission electron microscopy

Microscopy is used to observe small objects. The most basic microscopy is optical microscopy: using optical lenses, a light beam is controlled to observe a specimen. With this technique, the best microscopes can achieve a magnification of 2000. For higher magnifications, other types of microscope are required. This is because the phenomenon of diffraction imposes a physical limit on the resolution that can be achieved using light rays: suppose an optical instrument has a circular aperture of diameter D, and a monochromatic wave of wavelength λ passes through it. We obtain a diffraction pattern called the 'Airy disk' (see Fig.1), the first black circle of which is at an angle to the resolution axis of:

$$\theta \approx 1.22 \, \frac{\lambda}{D} \tag{1}$$

If two points on the observed specimen are too close, their diffraction spots overlap and they cannot be distinguished from each other. The minimum distance between two distinguishable objects is called the resolution limit, and the smaller the wavelength of the beam, the smaller the Airy disk (since the Airy disk is then smaller).

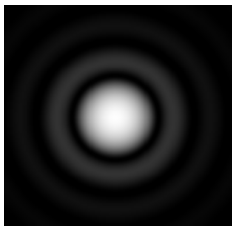

Figure 1: Computer-simulated Airy disk - contrast is exaggerated in low light. [1]

The idea is then to use electrons with short wavelengths to overcome this physical limit and obtain much higher magnifications. This wavelength is obtained using Louis de Broglie's relation: $\lambda = h/p$ where h is Planck's constant and p is the momentum. There are two main types of electron microscopy: scanning electron microscopy (SEM) and transmission electron microscopy (TEM) [2] [3]. In scanning electron microscopy, an electron beam scans the surface of the specimen to be observed and analyses the quantity of secondary and backscattered electrons, to obtain an image of the topography of the specimen at resolutions in the nanometre range. In transmission electron microscopy, the electron beam passes through the specimen - this requires more work during specimen preparation, as the specimen must be sufficiently fine, but allows resolutions of 0.04 nm to be obtained - as a reminder, the order of magnitude of the diameter of an atom is the angstrom (Å) where 1 Å = 0.1 nm. The TEM operating diagram is available in Fig.2.

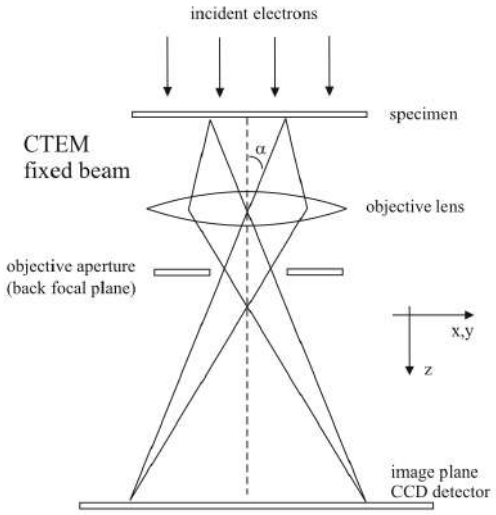

Figure 2: Simplified TEM model [3].

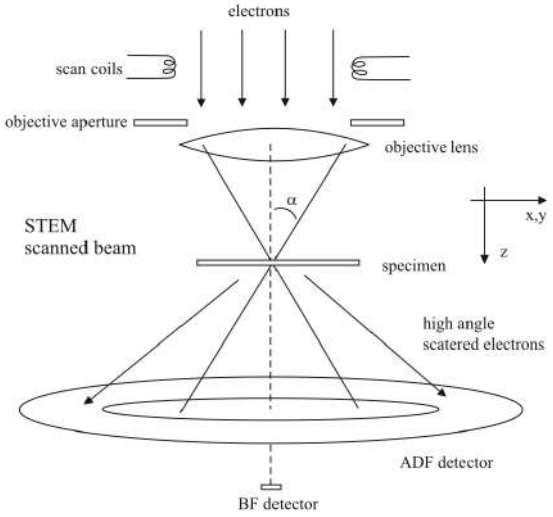

Figure 3: Simplified STEM model [3].

In the conventional TEM operating mode (CTEM), the incident electrons form a parallel beam over the entire surface of the specimen. The whole image is then formed at one time in the image plane of the objective lens. The second main mode of operation, which gives the instrument the best resolution, is scanning transmission electron microscopy (STEM). In this mode, the electrons pass through an objective lens before encountering the specimen at a given point (see Fig.3). The output scattering pattern is analysed to obtain an image of the selected point. The specimen is then scanned, repeating this process to obtain a complete image at each point. Classically, the intensity of the selected spot is obtained by counting the number of electrons reaching a specific detector: we can then obtain a bright field image (BF) by counting the non-diffracted electrons, or a dark field image with an annular detector (ADF for Annular Dark Field) by counting the scattered electrons at a certain range angle - we can thus concentrate on observing heavy atoms by choosing high angles (HAADF for High Angle ADF). Finally, the most recent techniques involve recovering complete diffraction images from each point on the specimen, which then requires computer post-processing of a 4D dataset (one 2D image for each point on the 2D specimen) - this technique is known as 4DSTEM. Examples of post-processing of this data are available in [4].

1.2 COFs and MOFs

Covalent Organic Frameworks (COFs) are porous polymers (materials made up of macromolecules) that form two- or three-dimensional structures by means of strong, covalent bonds. These materials are organic (composed of carbon, hydrogen, oxygen or nitrogen) [5]. Metal-Organic Frameworks (MOFs) are very similar to COFs, but their structures are centred around metal clusters. An example of a MOF cell, visualised with the Vesta software [6] is available in Fig.4. The main field of application

for these materials is gas storage.

One of the main problems in synthesising these materials is predicting the resulting geometry [7]. This is because their synthesis requires small molecules to interact with each other, which can sometimes, depending on how they combine, give different geometries. Hence the idea of trying to perform atomic-scale tomography on these materials, to confirm or refute the predicted geometry of a particular synthesis. Unfortunately, it is difficult to observe them under an electron microscope, because Cofs and Mofs are beam sensitive [8]. Recent advances have made it possible to obtain images with better resolutions (see Fig.5), but when it comes to making several projections, the problem becomes more complex, as the material degrades as the images are taken. The idea is therefore to carry out tomographic reconstruction with a limited number of projections.

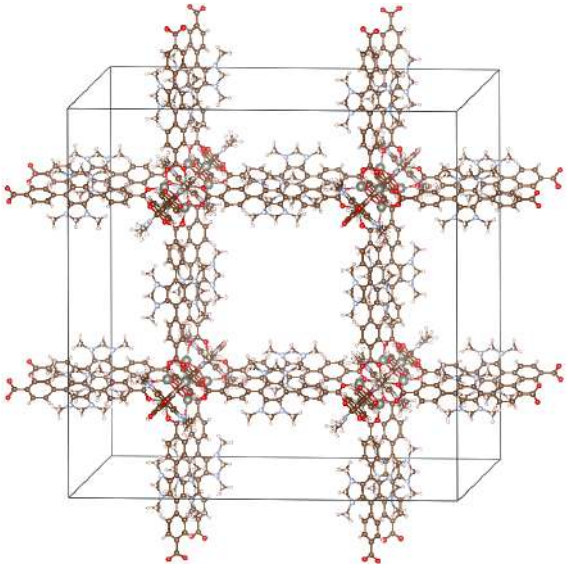

Figure 4: VESTA visualisation of an IRMOF-76 cell

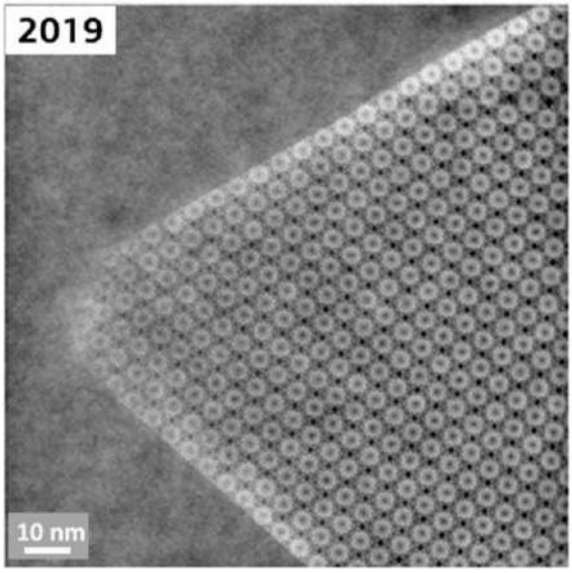

Figure 5: 2019 STEM image of MIL-101 from [8]

1.3 Tomographic reconstruction

Tomography is an imaging technique whose aim is to reconstruct the volume of an object from a series of measurements taken from outside the object. The technique is widely used in medical imaging, but is also very common in materials science. Here, we present parallel-beam tomography problems, for two-dimensional objects. Consider an unknown object modelled by an integrable function $f : \mathbb{R}^2 \to \mathbb{R}$. The projection P_θ of f at angle θ is called the Radon transform and is defined as such:

$$P_\theta(t) = \iint_{\mathbb{R}^2} f(x,y)\delta(x.cos(\theta) + y.sin(\theta) - t)dxdy \tag{2}$$

where $\delta : \mathbb{R} \to \mathbb{R}$ is the Dirac delta function and $t \in \mathbb{R}$ denotes the position of the detector (voir Fig.6). The problem is then to find the function f knowing the functions P_θ.

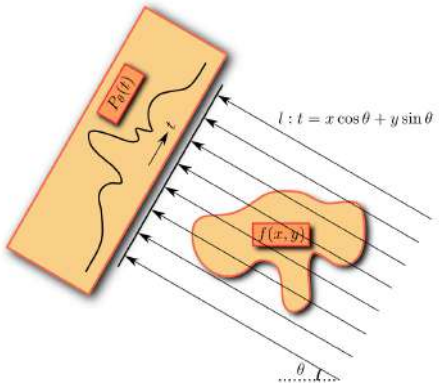

Figure 6: Radon transform of f at angle θ [9]

In practice, we can only measure a finite number of projections of a given object on a finite number of detectors. The problem is therefore often rewritten as follows: let N_θ be the number of projections and N_d the number of detectors, and let $\boldsymbol{y} \in \mathbb{R}^{N_\theta \times N_d}$ the set of measured projections, also known as the sinogram. Given that the object is represented by a grid of $N \times N$ pixels, we denote $\boldsymbol{x} \in \mathbb{R}^{N \times N}$ the object to be reconstructed. It is therefore a question of finding $\boldsymbol{x}$ knowing:

$$\boldsymbol{y} = \boldsymbol{A}\boldsymbol{x} \tag{3}$$

where $\boldsymbol{A} \in \mathbb{R}^{N_\theta N_d \times N^2}$ is the projection operator.

There are two main types of reconstruction method: direct methods and iterative methods. Direct methods are constructed by searching for a continuous analytical solution to the inverse problem of Eq.2, then discretising this solution. These methods have the advantage of being very fast for performing reconstructions, and are efficient when many projections are available. However, their performance deteriorates rapidly when the number of projections is limited. The main direct reconstruction algorithm is called Filtered BackProjection (FBP). It can be shown [11] that the function modelling the object can be written as follows:

$$f(x, y) = \int_0^\pi q_\theta(x.cos(\theta) + y.sin(\theta))d\theta \tag{4}$$

where q_θ is the convolution of P_θ with the filter $u \to |u|$ in Fourier space:

$$q_\theta(t) = \int_{-\infty}^{\infty} \hat{P}_\theta(u)|u|e^{2\pi i u t}du \tag{5}$$

and $\hat{P}_\theta$ is the Fourier transform of P_θ. To explain the need for the filter, we use the Fourier Slice Theorem, according to which the Fourier transform of a parallel projection of an image $f(x, y)$ taken at angle θ gives a slice of the two-dimensional transform, $\hat{f}(u, v)$, subtending an angle θ with the u-axis (see Fig.7). If we had an infinite number of projections, we could know $\hat{f}$ at any point in the frequency domain, but in practice we only have a finite number of measurement points, and it is necessary to interpolate to the whole space. As the density of measurement points is higher at lower frequencies (see Fig.8), a filter is added to correct the interpolation error at higher frequencies. The FBP algorithm is obtained by discretising the equations 4 and 5:

$$f(x, y) \approx \sum_{\theta_d \in \Theta} \sum_{\tau_p \in T} h(\tau_p) P_{\theta_d}(x.cos(\theta_d) + y.sin(\theta_d) - \tau_p) \tag{6}$$

where Θ is the set of angles for which projections have been measured, T is the set of detector positions and h is the filter in real space ($\hat{h}(u) = |u|$).

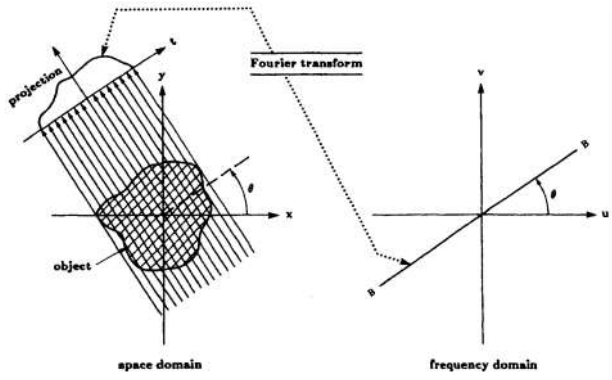

Figure 7: Illustration of the Fourier Slice Theorem θ [11]

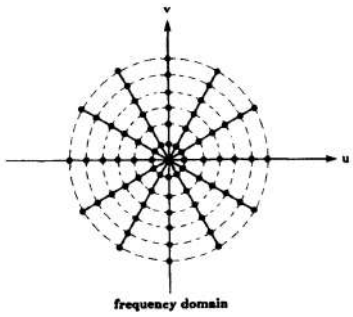

Figure 8: Density of points in the frequency domain [11]

Iterative methods attempt to solve the linear system of Eq.3 iteratively. The computation times for these methods are much longer than for direct methods, but they give better results when few projections are measured. We will briefly present here only the SIRT (Simultaneous Iterative Reconstruction Technique) method, which will be the reference method in all the studies carried out. The update equation is:

$$\begin{aligned} \boldsymbol{x}^{(0)} &= \boldsymbol{0} \\ \boldsymbol{x}^{(t+1)} &= \boldsymbol{x}^{(t)} + \boldsymbol{C}\boldsymbol{A}^{\boldsymbol{T}}\boldsymbol{R}(\boldsymbol{y} - \boldsymbol{A}\boldsymbol{x}^{(t)}) \end{aligned} \tag{7}$$

where $\boldsymbol{A}^{\boldsymbol{T}}$ is the transpose of $\boldsymbol{A}$, $\boldsymbol{C}$ and $\boldsymbol{R}$ are diagonal matrices, such that $c_{jj} = 1/\sum_i a_{ij}$ and $r_{ii} = 1/\sum_j a_{ij}$. Fig.9 shows different reconstructions of the Shepp-Logan head phantom, for the FBP and SIRT algorithms, and for different numbers of projections. The stopping condition chosen for SIRT is described in section 2.3.

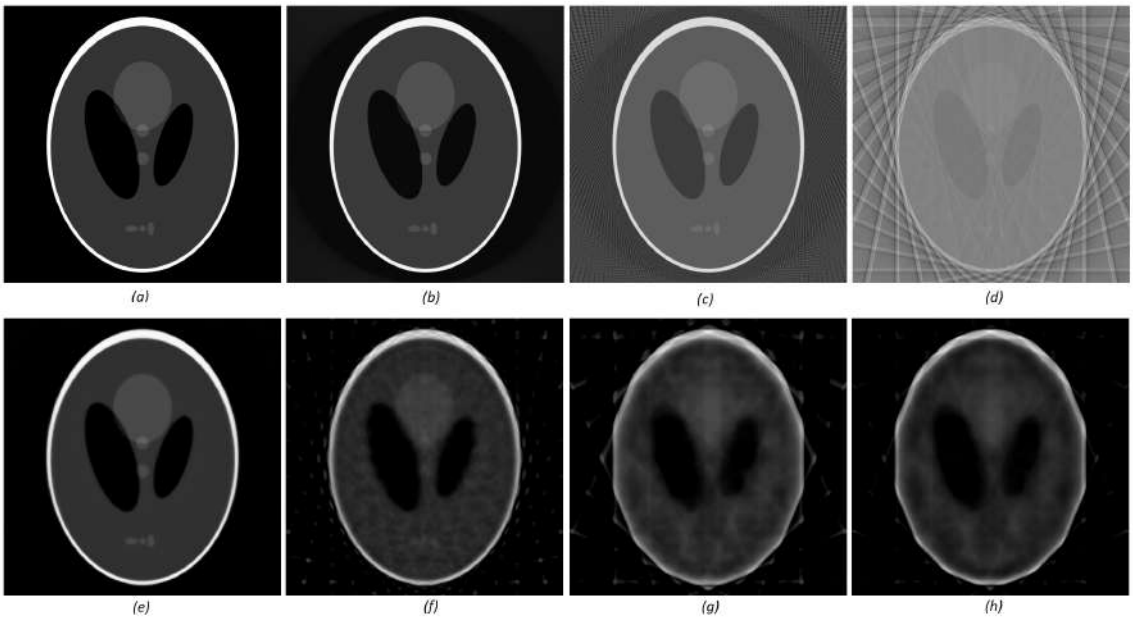

Figure 9: From left to right, top to bottom: (a) the original 800 × 800 pixels image, (b) FBP with 500 projections, (c) FBP with 100 projections, (d) FBP with 16 projections, (e) SIRT with 100 projections and 150 iterations, (f) SIRT with 16 projections and 150 iterations, (g) SIRT with 8 projections and 150 iterations, and (h) SIRT with 8 projections and 500 iterations. Projections are taken at regular intervals between -90° and 90°.

1.4 Objectives of the internship and contributions

The main objective of the internship is to implement the NN-FBP algorithm derived from [9], presented in part 2.1, the aim of which is to use a neural network to perform tomographic reconstruction from a limited number of projections. This is my main contribution - the main improvements to the algorithm developed consisted in extending the algorithm to three-dimensional structures, parallelizing certain parts of the algorithm, and making it possible to apply the algorithm to sets of projections where a range of angles is missing, which is typical in electron tomography. The code also aims to be reusable or extendable - Roham, the PhD student I worked with should be able to appropriate the code as part of his thesis. A major focus has been placed on the documentation of the code, and on its relative modularity: the implementation of other tomographic reconstruction algorithms using neural networks must not be too complex. For example, the MSDNET algorithm from [10] has also been implemented, but will not be detailed in the report. The second main objective of the course is to apply this algorithm to experimental measurements of MOFs, and to measure its performance against other more conventional algorithms. The performance of the algorithm for reconstructing an MOF particle will be evaluated and, as we have not yet succeeded in obtaining several images of an MOF from

different angles at the atomic scale, its performance for reconstructing simulated MOF images at this scale will be studied. I was also in charge of carrying out these simulations. No experimental manipulations were carried out by me during this internship, all TEM measurements were performed by Roham. The code is available on GitHub.

1.5 The University and the Max-Planck Institute

The University of Stuttgart was founded in 1829 and is one of the oldest universities in Germany. The university is historically renowned for its technical programmes, particularly in civil, mechanical, industrial and electrical engineering, and was previously known as Technische Hochschule Stuttgart, before its name was changed in 1967, with the extension of its programmes to humanities, economics and social sciences. The university consists of two campuses: the historic one in the centre of the city, where the humanities, social sciences and architecture are taught, and the other, dating from the late 1950s, in the suburb of Vahingen in the south-west of the city, where technical subjects are taught. In the 2022/2023 academic year, the university had 22,000 students and 5,000 employees (including 270 teachers).

However, the internship did not take place on one of these campuses, but on the premises of the Max-Planck Institute for Intelligent Systems, to the north-west of Vahingen, part of which is attached to the university's Institute for Materials Science. The Max-Planck society for advancement of science (Max-Planck Gesellschaft zur Förderung der Wissenschaften) is a non-profit association funded by the German federal government, with a fundamental research mission. It was founded in 1948 and had 24,000 employees in 2021. The society comprises 83 institutes spread across Germany, with its head office in Munich. Its mission is comparable to that of the CNRS in France.

2 Reconstruction of a nanoparticle of UiO-67

In this section, the main objective is to implement the NN-FBP algorithm (for Neural Network - Filtered BackProjection) from [9], and to apply this algorithm to a first concrete example. This is one of the first algorithms to implement Machine Learning for tomographic reconstruction. The object to be reconstructed is a nanoparticle of UiO-67, a MOF whose metal clusters are zirconium (Zr) atoms arranged in octahedrons. 29 projections were made at the TEM for angles ranging from -70° to 70°, i.e. one projection every 5° (-70°, -65°, -60°, ..., 70°). This range limitation is due to technical limitations: in order to get these measurements, the sample holder is tilted, which can only be done to a certain extent. The images produced are 1024 pixels in size. Some of these projections are available in Fig.10. The main challenges are to extend the algorithm to the reconstruction of objects in three dimensions, and for limited angle ranges, to parallelize certain parts of the

algorithm to speed up reconstruction, and to determine quantitative indicators to compare reconstructions.

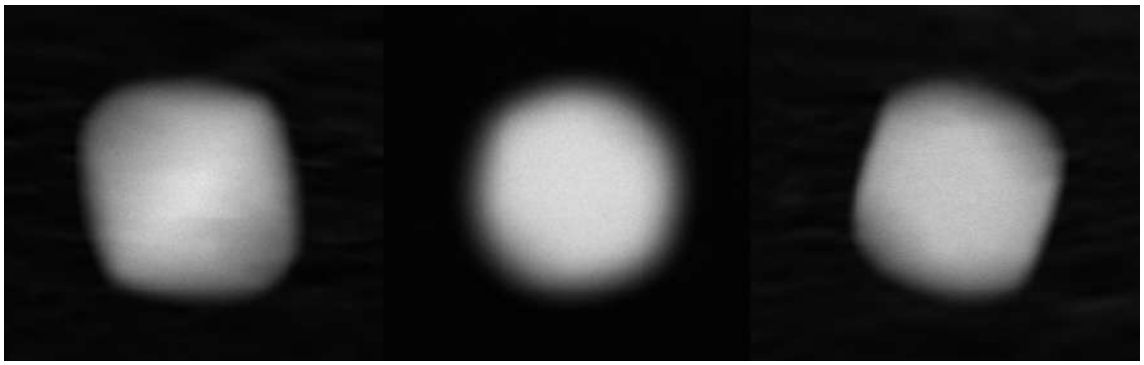

Figure 10: Measured projections of UiO-67 at -70°, 0° and 70° (from left to right).

2.1 The NN-FBP algorithm

The idea behind this algorithm is to combine different filtered backprojection reconstructions with custom filters (instead of the regular $u \to |u|$ filter), and to learn these custom filters with Machine Learning. Let us first introduce the notations specific to neural networks: the activation functions are denoted σ, the one-dimensional input vector is denoted $\boldsymbol{z}$. The proposed network has a single hidden layer, and returns a real number between 0 and 1, corresponding to the predicted value of a particular voxel. The weights and biases are $\boldsymbol{W} = (w_{ij})$ and $\boldsymbol{b} = (b_j)$ between the input layer and the hidden layer, $\boldsymbol{Q} = (q_j)$ and b_0 between the hidden layer and the output neuron - the network diagram is available in Fig.11. The question is then to determine what to choose for the vector $\boldsymbol{z}$, so that the network predicts as output the value of the associated voxel of the image at the coordinates (x_i, y_j). The value of $\boldsymbol{z}$ proposed in the article is such that:

$$z_p = z(\tau_p) = \sum_{\theta_d \in \Theta} P_{\theta_d}(x_i.cos(\theta_d) + y_j.sin(\theta_d) - \tau_p) \tag{8}$$

using the same notation as in section 1.3. Knowing that for each angle θ_d, (x_i, y_j) projects onto a different point $t_d = x_i.cos(\theta_d) + y_j.sin(\theta_d)$ on the detector, the proposed formula consists in a shift of each projection P_{θ_d} such that the corresponding t_d is in the middle, a sum of these shifted projections point by point, and finally a reflection about the centre. This formula may seem to come out of nowhere at first glance, but it achieves the original goal of using the $\boldsymbol{W}$ coefficients for FBP custom

filters. Indeed, the entire network equation becomes:

$$\begin{aligned} n_{\boldsymbol{Q},\boldsymbol{W},\boldsymbol{b},b_0}(\boldsymbol{z}) &= \sigma\left(\sum_k q_k \sigma\left(\sum_p w_{pk} z_p - b_k\right) - b_0\right) \\ &= \sigma\left(\sum_k q_k \sigma\left(\sum_p w_{pk} \sum_{\theta_d \in \Theta} P_{\theta_d}(t-\tau_p) - b_k\right) - b_0\right) \\ &= \sigma\left(\sum_k q_k \sigma\left(\sum_{\theta_d \in \Theta} \sum_p w_{pk} P_{\theta_d}(t-\tau_p) - b_k\right) - b_0\right) \\ &= \sigma\left(\sum_k q_k \sigma\left(FBP_{\boldsymbol{w_k}}(x_i, y_j) - b_k\right) - b_0\right) \end{aligned} \tag{9}$$

where $t = x_i.cos(\theta_d) + y_j.sin(\theta_d)$ and $FBP_{\boldsymbol{w_k}}$ is the filtered backprojection with filter with custom weights $\boldsymbol{w_k}$, the k-th column of $\boldsymbol{W}$. The reconstruction algorithm then becomes a combination of FBP algorithms, in which the points and biases generated by the network are used to obtain the final reconstruction.

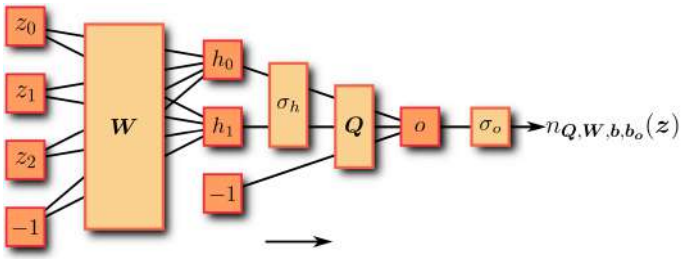

Figure 11: the network diagram as proposed in [9]

The whole process works as follows: to train the network, we use the classical view as described in Fig.11, calculating the network inputs $\boldsymbol{z}$ via Eq.8 for each voxel (x_i, y_j) selected in our training dataset. Once the network has been trained, we use Eq.9 to reconstruct the image directly from the projections, by combining FBP algorithm with custom filters.

An optimisation is also proposed in the article in order to reduce the size of the $\boldsymbol{z}$ input vector, called exponential binning. The idea behind this optimisation is to observe that, to predict the value of a certain voxel, the information contained near the centre of the detectors (after the shifting process) is more important than that contained at the ends. The procedure is illustrated in Fig.12. We denote $\bar{\boldsymbol{z}}$ the reduced input vector, whose coefficients are defined as such:

$$\begin{aligned} \bar{z}_0 &= z_0 \\ \bar{z}_i &= \sum_{j=s_{-i}}^{s_{-i+1}} z_j + \sum_{j=s_i}^{s_{i+1}} z_j \quad \forall i \neq 0 \end{aligned} \tag{10}$$

with s_i and s_{i+1} the boundary points of the bin i, of width $s_{i+1} - s_i = 2^{|i|-1}$ for $i \neq 0$.

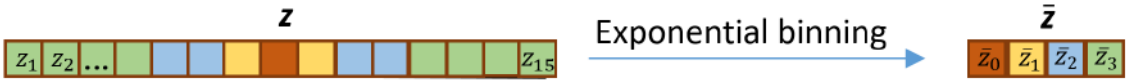

Figure 12: Illustration of the binning procedure to reduce the size of the inputs.

2.2 Description of the implementation and main challenges

The implementation has been made in such a way that it can be used as a library - the code is available on Github, and documentation can be generated from the source code, using the `make html` command in the `docs` folder. To implement this documentation, the `sphinx` and `pydata-sphinx-theme` libraries have been used. The details of the Github files and the implementation of the documentation are not described here - we will concentrate on the source code (`src/nntomo` folder). Notebooks showing examples of using the library are also available in the `scripts` folder.

The source code is notably constituted of 6 `.py` files:

- `volume.py` contains the class Volume, to represent 3d objects (wrapper of a 3d numpy array),
- `projection_stack.py` contains the class ProjectionStack, to represent stacks of projections from tomography in various formats; reconstruction methods are here,
- `network.py` contains the training functions for the two neural networks,
- `nnfbp.py` contains the pytorch Dataset and Network classes for the NN-FBP algorithm,
- `custom_interp.py` contains custom interpolation functions running on GPU for fast dataset computation and fast reconstruction,
- `utilities.py` contains utility functions.

A last file (`msdnet.py`) is also contained is the folder, which contains the pytorch implementation of another neural network for tomographic reconstruction called the MSDNET algorithm [10] [12] (for Mixed Scale Dense convolutional neural NETwork), but due to underwhelming results and lack of testing, this network will not be presented in this report. Finally, the code has been written to run on a machine with an NVIDIA GPU with at least 24 GB of memory.

2.2.1 The Volume and ProjectionStack objects

As the aim is to reconstruct objects in three dimensions rather than two as in the article, it is necessary to generalise the algorithm. To represent these objects in three dimensions, a wrapped numpy array in a python class named `Volume` is used. This class mainly contains utility methods to facilitate the creation of scripts in the context of tomographic reconstruction. Sinograms (which also become three-dimensional objects) are represented by the `ProjectionStack` class, which is also a wrapper of a 3D numpy array, but which is used to manage axis conventions, projection angles and reconstruction algorithms.

Before going into more detail about the implementations of the two classes, we need to talk about the conventions for the orientation and axes of our numpy 3D arrays. We chose to use the same convention as that used by the `ASTRA-toolbox` library. This library contains basic tomographic reconstruction algorithms, some of which have been optimised to run on GPUs. The FBP and SIRT algorithms were implemented in this project using this library. After a lot of trial and error, the documentation being rather short on details, the following convention can be found: N_x, N_y and N_z are the dimensions of the object to be analysed. The numpy array associated with such a volume then has a shape of size (N_z, N_y, N_x). When we try to apply a forward projection algorithm on this volume, projections are taken parallels to the axis of size N_z. So, N_z is also the number of detectors in the direction of the rotation axis. Let $N_d = max(N_x, N_y)$ be the number of detectors in the direction perpendicular to the rotation axis, and N_θ the number of projection angles, the resulting numpy array representing the sinogram will have shape (N_z, N_θ, N_d). Finally, in order to visualize projection stacks and volumes (with an isosurface tool for volumes), the `3dmod` software from `IMOD` is used [13], which reads `.mrc` files. These `.mrc` files, which are used to save raw data from three-dimensional arrays, are also the format of the TEM data output files. And, when this data is retrieved in a python script via the `mrcfile` library, the orientation convention of the projection stacks is (N_θ, N_d, N_z). These changing conventions are managed in the `ProjectionStack` class.

An instance of the `Volume` class has 4 attributes: the associated numpy array, its shape, an ID, and a file path. The ID and the file path are used if we wish to automatically save the volume in the `.mrc` format via the `.save()` method. The `Volume.retrieve()` class method is used to retrieve a previously saved volume knowing its ID. `Volume` also has 3 other class methods: `Volume.from_mrc_file()` to generate a volume from a `.mrc` file knowing its location, `Volume.random_spheres()` which allows a volume to be generated containing randomly sized and randomly positioned spheres - Fig. 13 shows an example of a generated volume -, and finally `Volume.stack_7ellipses()` to generate a volume made up of a stack of random ellipses (see Fig.14). What's the point of generating such an object, which is unlikely to make any physical sense? The interest lies in the way in which we generalise

NNFBP to three-dimensional objects: to do this, we made the simple choice of cutting the volume to be reconstructed into slices, and to consider each slice individually as a 2D reconstruction problem in order to apply NNFBP directly. In addition, such geometries were used in the article [9] to test the algorithm, hence the decision to also use these images to train the network. To speed up the generation of such geometries - for example, we want to use `Volume.random_spheres()` for a volume of $1024 \times 1024 \times 1024$ pixels - the `cupy` library was used: this library was designed as a substitute for `numpy` to carry out operations on the GPU. It is used in almost the same way, most of the `numpy` methods have a `cupy` equivalent, it is generally just a matter of replacing `np.method()` by `cp.method()` (with `import cupy as cp`). The main precaution is to ensure that the `cupy` arrays are not larger than the GPU's memory. Finally, the class contains two last utility methods: `.normalize()` to set all voxel values between 0 and 1, and `.get_segmented_volume()` which returns a volume whose voxels are either 0 or 1, depending on whether the voxel is smaller or larger than a given threshold value.

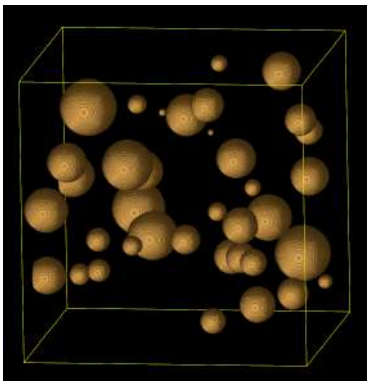

Figure 13: Volume generated via `Volume.random_spheres()`, isosurface visualisation with `3dmod`

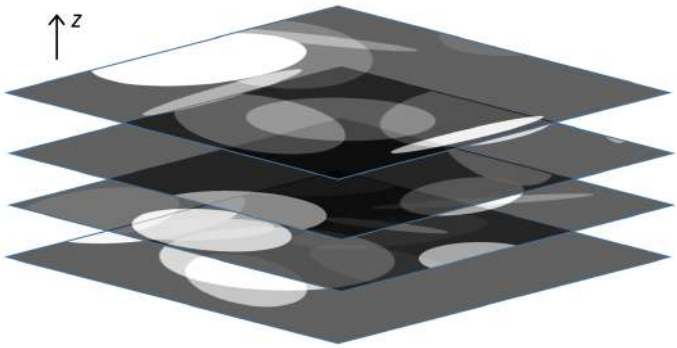

Figure 14: A visualisation of a stack of ellipses generated with `Volume.stack_7ellipses()`. The z-axis is the axis of rotation. Each slice contains 7 random ellipses.

To create an instance of `ProjectionStack`, three class methods are available. The first, `ProjectionStack.from_volume()`, takes as parameters an instance of `Volume`, a number of projections and an angle range, and applies a forward projection via the `ASTRA-toolbox` library. The angle range can take two values, `'full'` for a tilt serie going from -90° to 90°, or `'tem'` for a tilt serie going from -70° to 70°, which is the range of angles available on a real TEM. The projections are assumed to be taken at regular intervals (including -90° but not 90° for `'full'`, and including both -70° and 70° for `'tem'`). In the case where the range takes the value `'tem'`, a second numpy table is created which corresponds to a fictitious sinnogram for a full angle range, in which the projections between -90° and -70°, and between 70° and 90° are derived from an interpolation between the projections at -70° and 70° (see Fig.15). It is this fictitious sinnogram that will be used

as input to the NNFBP algorithm: indeed, as mentioned in part 2.1, this algorithm is a combination of FBPs, and for each FBP, the filter used to compensate for the variation in the density of measurement points in Fourier space is independent of θ; thus, it is necessary for the angle between two consecutive projections to be constant. In the same way as `Volume`, it is possible to create an instance of `ProjectionStack` from a `.mrc` file, using the `ProjectionStack.from_mrc_file()` method, and to save and retrieve the object in this format using the `.save()` and `ProjectionStack.retrieve()` methods. The last method to generate a stack of projections is to use `ProjectionStack.from_cif_file()` - how this works is described in part 3.1.

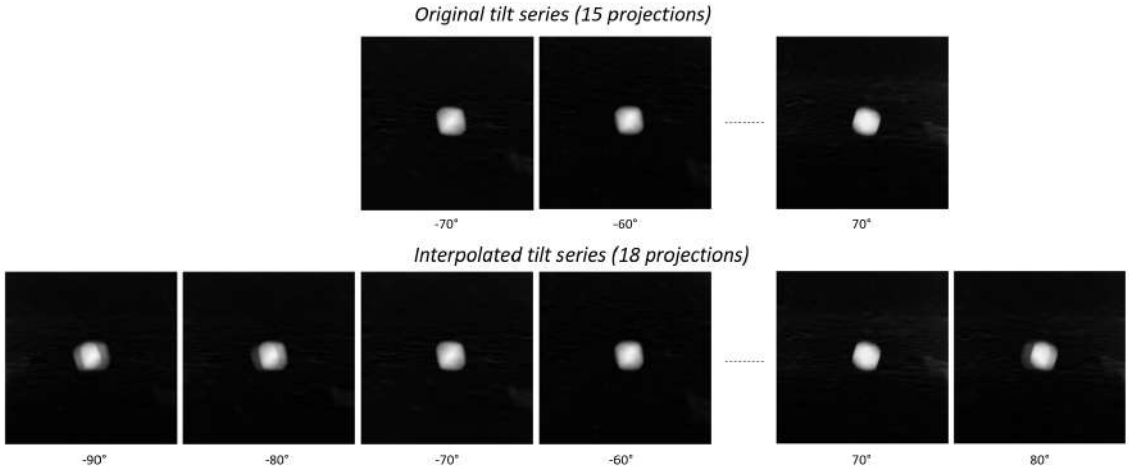

Figure 15: Illustration of the interpolation procedure for tilt series with 15 projections between -70° and 70°.

All the reconstruction algorithms are implemented in this class: these are the methods `.get_FBP_reconstruction()`, `.get_SIRT_reconstruction()`, `.get_NNFBP_reconstruction()` and `.get_MSDNET_reconstruction()`; each of these returns an instance of `Volume`. All use the GPU to speed up the calculations. The NNFBP reconstruction will be described in part 2.2.3, the MSDNET reconstruction will not be described in the report. The SIRT reconstruction is a direct application of the `ASTRA-toolbox` library, while the FBP reconstruction applies the library's 2D algorithm to each slice of the volume (the 3D version of FBP is not implemented). Finally, the class also contains utility methods: `.get_proj_subset()` to extract a projection subset from an instance of `ProjectionStack`, `.convert_full_tem()` to extract projections from -70° to 70°, `.get_resized_proj()` to resize a stack via interpolation.

2.2.2 Pytorch: nn.Module and training fuctions

To implement the neural network and the training functions, we have chosen to use the `pytorch` library. For more information on how the library works, see the tutorial in the documentation here. The network has been implemented with a `NNFBP` class inheriting from `nn.Module` - we won't go into detail about this implementation given the simplicity of the network. However, there is one difference with the article: they use the Sigmoid function for the two activation functions, and we will study the influence of replacing Sigmoid with ReLU for the activation of the hidden layer. To train the algorithm, we use the `nnfbp_training()` function in the `network.py` file, which takes as its parameters the training and validation datasets - the calculation of which is detailed in the next section -, as well as a number of hyperparameters linked to training: batch size, learning rate for the optimizer, etc. The loss used to evaluate the performances of the network is the MSELoss (Mean Square Error). Adam is chosen as the optimizer. The training method is conventional: we iterate over the datasets, with one complete iteration corresponding to one epoch. At each epoch, we use the entire training dataset to modify the network parameters, then the validation dataset to evaluate its performance. The stopping criterion used is the same as in the article [9]: the training is terminated if the computed loss didn't improve for 25 consecutive epochs. Finally, the implementation includes saving checkpoints every 30s, so that the training can be interrupted and resumed later. This also means that, once training is complete, the intermediate networks can be recovered for reconstruction and observation of the improvement in the network during training - Fig.16 (see part 2.4) was drawn using this method.

2.2.3 Pytorch Dataset computation and NNFBP reconstruction

This section presents the dataset generation and tomographic reconstruction algorithms using NNFBP. The class representing our datasets is called `DatasetNNFBP` and inherits from the `Dataset` class of `pytorch`. A dataset contains a list of theoretical inputs and outputs for the corresponding neural network. Thus, for the NNFBP network, each data point in the dataset contains a vector $\boldsymbol{z}$ (or $\bar{\boldsymbol{z}}$ if exponential binning is applied, see section 2.1 for notations), and the theoretical value of the associated voxel (x_i, y_j). The network is trained by a succession of forward propagations and backpropagations, comparing the theoretical value of the voxel with the predicted value. To be calculated, a `DatasetNNFBP` takes two main parameters: one (or more) `ProjectionStack` and one (or more) `Volume`. The `Volume` must correspond to the theoretical reconstruction obtained from the `ProjectionStack`. First, we normalise the volume so that all the voxels take values between 0 and 1: the network ends with a Sigmoid activation and can therefore only output between these two values. The normalisation coefficients are saved so that the reverse operation can be performed at the reconstruction stage. The next step is to select a certain number of random voxels in `Volume` (by default 100,000), and, for each of these voxels, apply Eq.8 to obtain the corresponding $\boldsymbol{z}$ value. However, this

equation requires a significant number of interpolations. Indeed, the value of P_{θ_d} in $(x_i.cos(\theta_d) + y_j.sin(\theta_d) - \tau_p)$, for a certain angle of projection θ_d and a detector in position τ_p is not known in the general case, so it is necessary to interpolate between two known values of P_{θ_d}. To speed up the calculations, this part of the algorithm can be parallelized. We use the built-in functions of `cupy` for the basic operations. Ideally, we would like to perform the interpolation operation for all values of θ_d, τ_p and on several voxels at once. Unfortunately, `cupy` doesn't have a method for performing linear interpolation in 1D on several axes at once, so we had to write a custom kernel. The kernel is written in the `custom_interp_db_init()` function in the `custom_interp.py` file, using the `cp.RawKernel()` function in `cupy` which allows you to embed `CUDA C++` code directly inside Python code - you just write it in a string. The number of voxels on which the $\boldsymbol{z}$ calculation is performed is chosen carefully, so as not to exceed the 24 GB memory of the GPU used. Once all these inputs have been calculated, all that remains is the optional exponential binning step, where we simply apply Eq.10 to $\boldsymbol{z}$ to obtain $\bar{\boldsymbol{z}}$. In practice, we add a resizing step on the coefficients of $\bar{z}$ - we divide each of them by the number of coefficients of $\boldsymbol{z}$ of which they are the sum. For example, $\bar{z}_2$ is the sum of 4 coefficients of $\boldsymbol{z}$, so we divide $\bar{z}_2$ by 4 to calculate the input. Without this resizing, the network reconstructs anything.

Once the datasets are generated and the network is trained, all that remains is to perform the reconstruction. The naive approach consists of applying Eq.8 for each voxel in the `Volume` to be reconstructed, and then performing a forward propagation on each input. The resulting computational complexity for reconstructing an object of N × N × N pixels is on the order of $O(N^4(N_\theta + N_h))$, where N is the size of one side of the volume, and N_h is the number of neurons in the hidden layer (we will take $N_h = 8$ for the rest of the report, as the article shows that there is hardly any improvement for higher values). If we instead use Eq.9 for reconstruction, it can be shown that the complexity is on the order of $O(N^3 N_\theta N_h)$ [9], which is much more interesting since in our case studies, $N >> N_h$ and $N >> N_\theta$. Therefore, this is the version we implement here, using the method `.get_NNFBP_reconstruction()` from `ProjectionStack`, which takes the trained network as a parameter. The main problem arises from the fact that it is not possible to perform an FBP with custom weights via the `ASTRA-toolbox` library, so the FBPs were manually implemented. To recap, Eq.6 provides the formula to be implemented. The trick is to perform the convolution between h and P_{θ_d} in the Fourier space for each value of θ_d. Once the convolution is done, we need to evaluate $h \circledast P_{\theta_d}$ at $(x_i.cos(\theta_d) + y_j.sin(\theta_d))$, again through interpolation. Thus, another custom kernel was written to perform this interpolation, via the `custom_interp_reconstruction()` function in the `custom_inter.py` file. Once the FBPs are computed, if we look at Eq.9, we can notice that the remaining operations correspond to a forward propagation in the network, but starting from the hidden layer. Therefore, we implement a `.end_forward()` method for `NNFBP`, which performs the remaining operations to obtain the final reconstruction.

For comparison, a reconstruction using the naive method on a CPU takes 3 days when $N = 1024$ and $N_\theta = 9$, whereas it takes about 30 seconds with the algorithm implemented with FBP on a GPU.

2.2.4 Other details

To conclude this section, two final implementation details are mentioned here. First, it can be noted that we use several different libraries that implement algorithms using the GPU: `cupy`, `pytorch`, and `ASTRA-toolbox`. These libraries have the unfortunate tendency to reserve GPU memory for their own use. For example, if we use `cupy` for an operation that requires 20 GB of memory, only 4 GB will remain for the other libraries, even if the variables tied to that memory are deleted. Therefore, it's necessary to manually clear this cache memory after each use of the GPU. We implement a decorator `@empty_cached_gpu_memory` in `utilities.py` that performs this task, and it is applied to each function or method using the GPU.

A second detail concerns the long computation times we often face. To avoid wasting time reconstructing volumes that have already been previously computed, we implement in the reconstruction methods a function that automatically retrieves saved data using the reconstruction identifier. If the method detects that a volume with the same identifier already exists in the data folder, it will simply retrieve the volume without recalculating it.

2.3 Protocol

Here, we describe the protocol for evaluating the performance of NNFBP compared to FBP and SIRT during the reconstruction of the UiO-67 particle. First, a stopping criterion must be decided for SIRT. The chosen criterion is to stop the algorithm when the relative error $||\boldsymbol{A}\boldsymbol{x}^{(i)} - \boldsymbol{y}||_2$ between two successive iterations is less than a certain value ϵ. That is, we stop at the first iteration i such that:

$$\frac{||\boldsymbol{A}\boldsymbol{x}^{(i+1)} - \boldsymbol{y}||_2 - ||\boldsymbol{A}\boldsymbol{x}^{(i)} - \boldsymbol{y}||_2}{||\boldsymbol{A}\boldsymbol{x}^{(i)} - \boldsymbol{y}||_2} \leq \epsilon \tag{11}$$

We set ϵ to 0.01 throughout the report.

Next, we need to choose a quantitative indicator to compare the reconstructions, and we decide to use the Mean Square Error (MSE), calculated by determining the mean of the squared differences between corresponding voxels in the two volumes. However, to use this indicator, it is necessary to have a reference volume against which the MSE error is calculated for each reconstruction. Since we are working with experimental projections of the MOF, we do not have the actual corresponding volume. A first idea is to use SIRT, a well-established method, on the maximum available projections and consider this reconstruction as the reference

volume. However, this method presents three problems in our case study. First, this reconstruction is far from perfect: we only have 29 projections ranging from -70° to 70°, and the algorithm generates voxels with negative values, which has little physical meaning. Secondly, this method has the disadvantage of being unfair to the FBP and NNFBP algorithms: if the reference is calculated with SIRT, the performance evaluation will be biased in favor of SIRT. Finally, and most significantly, the network training is done with artificial data that has little resemblance to the object to be reconstructed: here, we use spheres or ellipses generated with the `Volume.random_spheres()` and `Volume.stack_7ellipses()` methods, where voxel values range between 0 and 1 - in comparison, the raw SIRT reconstruction voxels have values ranging from -471 to 443 .

The chosen solution is as follows: we perform a segmentation operation on this raw reconstruction using a threshold arbitrarily selected by visualizing the volume with `3dmod` (we choose 202), so that all the voxels in the reconstruction take on values of 0 or 1. This volume will serve as the reference thereafter (see Fig.18, left volume). We then calculate the associated projections using the `ASTRA` library (method `ProjectionStack.from_volume()`) - these are the projections that will be used to compute the different reconstructions.

We conduct two analyses in this section. The first analysis aims to evaluate the network's training speed depending on the activation function used (Sigmoid or ReLU) for the hidden layer and whether exponential binning is used or not. The training and validation datasets used for training are calculated from random spheres and consist of only 10,000 data points to emphasize the difference in training speed. The second analysis aims to compare the reconstruction of the volume difference between FBP, SIRT, NNFBP trained on spheres, and NNFBP trained on ellipses. The datasets this time consist of 100,000 data points. We will reconstruct from 29 projections (every 5° between -70° and 70°), 15 projections (every 10° between -70° and 70°), and 8 projections (every 20° between -70° and 70°). The networks are trained with databases respectively generated from 36, 18, and 9 projections, which are the numbers of projections resulting from the interpolation process to a full tilt series (see Fig.15).

We work everywhere with volumes of dimensions $512 \times 512 \times 512$ pixels for the particle and the random spheres (40 spheres per volume). The dimensions of the TEM projections have thus been reduced from 1024 to 512 pixels: this reduction allows for slightly faster calculations, reduces the memory required to store the data, and makes the visualization of volumes easier since the maximum visualization size in `3dmod` is $512 \times 512 \times 512$ pixels. The dimensions of the ellipse stack are $100 \times 512 \times 512$. Everywhere, the batch size is 32, the learning rate is 10^{-4}, and N_h is 8. All figures were generated in the notebook `uio67-reconstruction.ipynb` in the `scripts` folder on the Github page.

2.4 Results and discussion

Fig.16 shows the results of the first analysis. A volume of random spheres and 9 associated projections are generated, and the MSE between the original volume and the reconstruction from the NNFBP algorithm is calculated at each epoch of the training process. For clarity, we only present the first 50 epochs, but all four algorithms converge to the same value after sufficient training time. Each curve corresponds to a different network, depending on the activation function used and whether or not binning was applied. We observe that the binning procedure has little influence on the training speed, unlike the use of ReLU instead of Sigmoid, which significantly accelerates it. The binning procedure may have a stabilizing effect on the training - the curve appears smoother - but the effect is relatively minor. For the rest of the report, we will use ReLU and the binning procedure.

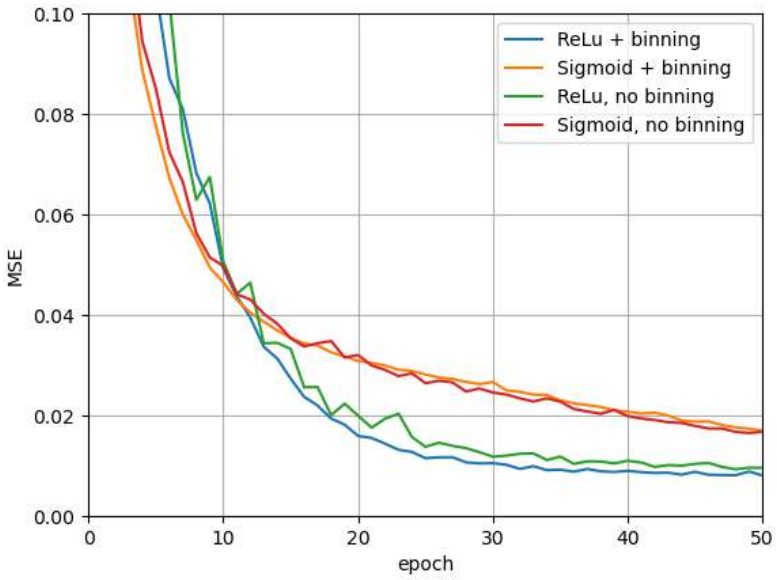

Figure 16: Evolution of MSE during training for the test volume, for NNFBP models with different activation functions and with or without exponential binning. Each model is trained to reconstruct random spheres with 9 projections. Before computing the MSE, voxels of the NNFBP are set to 0 or 1. Only the first 50 epoch are shown.

Fig.17 shows the results of the second analysis. The FBP algorithm is by far the least performant, which is consistent with the small number of projections we are working with. SIRT performs much better, but the NNFBP algorithms are the most accurate, especially in the case of the reconstruction with 8 projections, where the calculated MSE is three times lower than for SIRT. The two NNFBP training methods yield quite similar results, but training with ellipses required generating a `Volume` five times smaller ($100 \times 512 \times 512$ instead of $512 \times 512 \times 512$) - ellipses

allow for creating a more diverse dataset at a lower cost. Indeed, two adjacent slices of the spheres volume are similar and can be redundant in the input calculations. One final observation is that, for 29 projections, NNFBP-Ellipses performs much better than NNFBP-Spheres. This superior performance is likely due to a "stroke of luck," stemming from the variability in the networks' performance during training - it is also possible to observe significant underperformance for isolated points (see section 3.3).

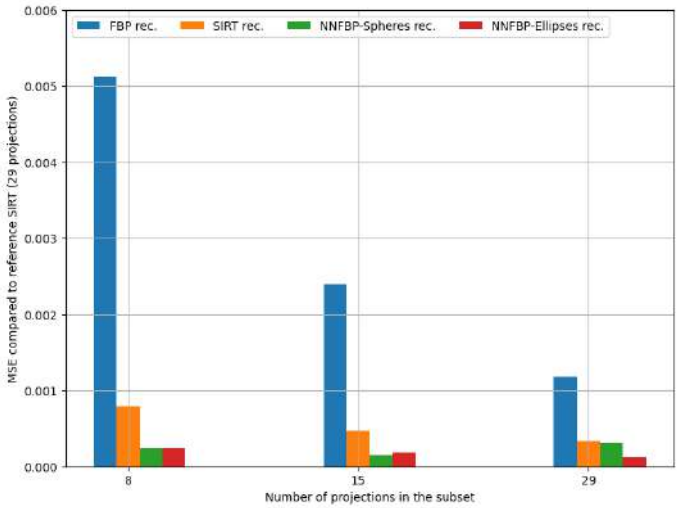

Figure 17: MSE loss between the reference volume and different reconstructions (FBP, SIRT, NNFBP trained with spheres, NNFBP trained with ellipses). The angle range of the projections is (-70°, 70°).

Lastly, the isosurface visualization of the reference volume and the SIRT and NNFBP reconstructions for 8 projections is available in Fig.18. The better performance of the neural network can be visualized qualitatively: the facets observed on the reference volume are much less pronounced in the SIRT reconstruction than in the NNFBP reconstruction.

2.5 Conclusion and outlooks

The analysis conducted shows the potential of such a neural network for tomographic reconstruction. However, we decided not to delve much further into this case study, as the final goal of the project is to perform atomic-scale reconstruction of COFs and MOFs, which will be addressed in the next section. In [14], the authors use NNFBP to reconstruct a gold nanoparticle - however, to train their network, they have access to other experimental projection stacks of particles, allowing them to directly apply the network to the raw stack (since the training data is similar to the test data). Despite this advantage, they still choose to use the arbitrarily segmented SIRT reconstruction as the reference volume. They also perform segmentation on their reconstructions before calculating the MSE (using a threshold of

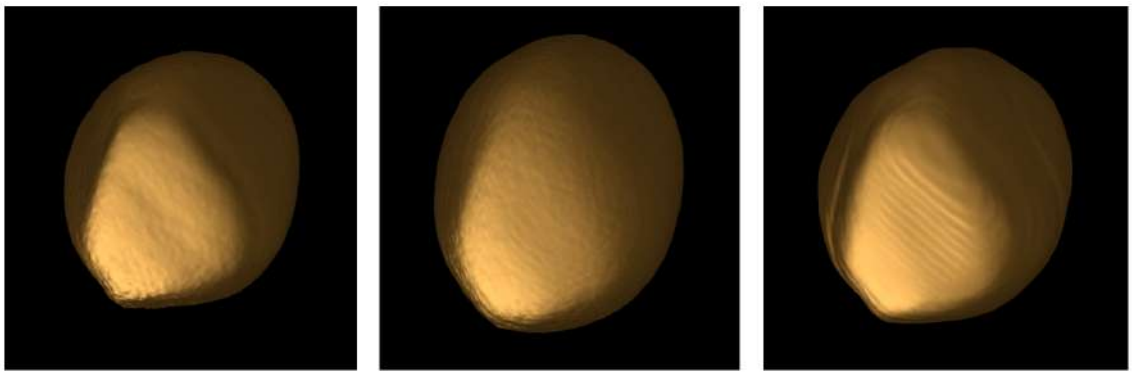

Figure 18: Visualization of reconstructions of the UiO-67 particule with the `3dmod` isosurface tool. From left to right: the reference volume, the SIRT reconstruction with 8 projections (isosurface threshold = 153/255), NNFBP reconstruction trained on ellipses with 8 projections (same threshold).

1/2 for NNFBP and the same threshold as the reference volume for SIRT). Thus, one potential improvement for this study would be to completely eliminate all segmentation operations, which is theoretically feasible if the training data is similar to the object being reconstructed.

3 Reconstruction with simulated UiO-67 projections at the atomic scale

Now that we have demonstrated the potential of using this neural network for tomographic reconstruction from experimental measurements, we will attempt in this section to apply the algorithm to atomic-scale projections of MOFs. The approach is as follows: first, simulate MOF images obtained via TEM from different angles to train the network, then, once the network is trained, use it to reconstruct from experimental projection stacks. Unfortunately, things didn't go our way during this project, as the microscope was down for a good part of the period. The images obtained by Roham are not yet suitable for tomography - the material degrades so quickly that nothing of interest can be observed after just one projection. One of the best images obtained so far is available in Fig.19. The MOF particles to which these atoms belong are shown in Fig.20. Therefore, for now, we will limit ourselves to reconstruction from other simulated images. This section presents how the STEM images are simulated using Python, followed by some initial results on the performance of NNFBP, which will be compared to SIRT.

3.1 Simulation of TEM images

To simulate images obtained via transmission electron microscopy (TEM), we primarily use the `abtem` library. The first step in these simulations is to choose the

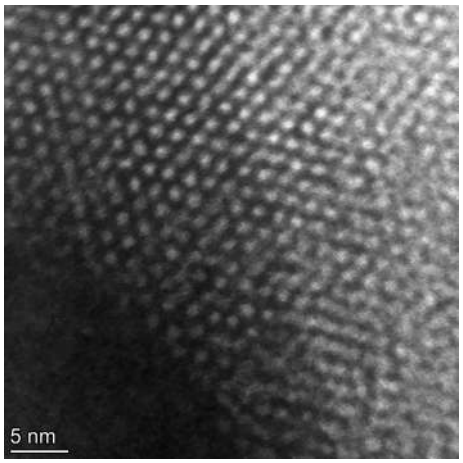

Figure 19: UiO-67 at atomic resolution, HAADF imaging.

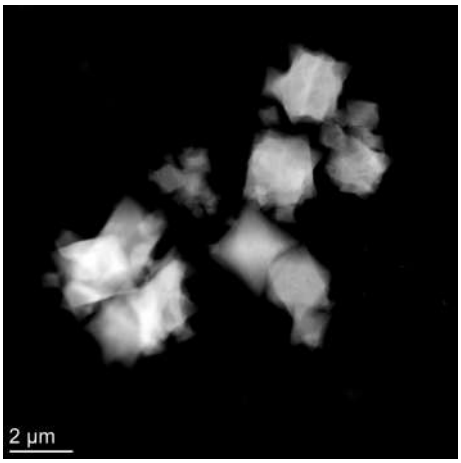

Figure 20: UiO-67 particules, HAADF imaging.

material to simulate. `abtem` is well-suited to work with `.cif` files, which describe an atomic structure and include the position and type of each atom in the crystal or cell being considered. To import and manipulate this atomic structure data, we use the `ase` (Atomic Simulation Environment) library. This library allows us to duplicate a cell in multiple directions if we want to simulate several, and to rotate the structure's atoms, which is useful for simulating the different projections needed for tomography. Once the structure is obtained (as an `Atoms` class from `ase`), we compute a potential by superimposing the potentials generated individually by each atom. Then, we perform what is called a multislice simulation: the goal is to compute the wave function corresponding to the potential and the energy level of the incident electrons. Once estimated, the wave function allows us to retrieve the intensity received by the selected detector - several detector geometries are available depending on the type of microscopy being used. The details of the multislice simulation are described in `abtem`'s documentation.

This implementation already allows for obtaining images, but these correspond to an ideal experiment, which does not exist in practice. Several tools are provided to "degrade" the quality of the simulated images to better match real observations. It is possible to apply Gaussian blur and Poisson noise to the results. The library also allows for implementing the Frozen Phonons Model, which models the motion of atoms that are not static but vibrate around their equilibrium points. To do this, we first choose a number of configurations to calculate - 10, for example - and for each of these configurations, each atom is randomly displaced by a small offset around its equilibrium point. Finally, the wave function corresponding to the average of the wave functions from each configuration is calculated. Additionally, the library allows for implementing lens aberrations through the Contrast Transfer Function (CTF). Gaussian blur, Poisson noise, and Frozen Phonons have been implemented

in the code through certain parameters of the `ProjectionStack.from_cif_file()` method. This method is used to generate TEM images from a `.cif` file by specifying the desired configuration. Optical aberrations have not yet been implemented, but they can be very easily added.

This method includes a `mode` parameter that can take values `'haadf'` or `'idpc'`, corresponding to the STEM imaging mode used for the simulations. `'haadf'` refers to HAADF-STEM mode, as described in section 1.1, while `'idpc'` refers to iDPC-STEM mode (iDPC stands for Integrated Differential Phase Contrast), which is an imaging method based on post-processing 4DSTEM data. This post-processing is carried out using the `py4dstem` library, an extension of `abtem`. The functioning of the algorithms used in 4DSTEM is described in [4] and [15] (specifically for iDPC), as well as in the `py4dstem` tutorials. Briefly, to create an iDPC image, a virtual split detector is simulated (by integrating the 4DSTEM data over the areas of the selected detector, the geometry of which is shown in Fig.21). This detector allows for calculating the center of mass displacement due to the interaction of the beam with the material at each point of the sample (see Fig.22). To obtain the final iDPC image, the center of mass displacement data is integrated. Examples of images simulated with `abtem` and `4dstem` are shown in Fig.23.

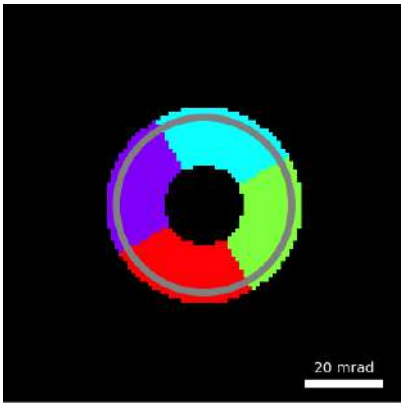

Figure 21: Geometry of the virtual iDPC detector.

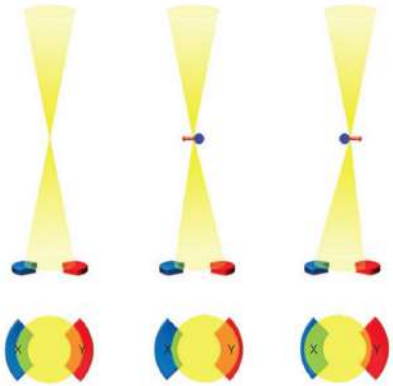

Figure 22: Visualisation of the electron trajectory when interacting with an atom [15].

Finally, to speed up the generation of these images, the PRISM algorithm has been implemented - this algorithm is also described in the `abtem` documentation. The general principle of this algorithm is to express the wave function as an expansion of plane waves and to compute the scattering matrix of the system (see [16] for details on quantum mechanics terminology). The speedup occurs when we choose not to keep every plane wave in the expansion but instead interpolate between these waves. With this optimization, it becomes possible to generate around a hundred images in an hour of computation time (compared to just one image previously).

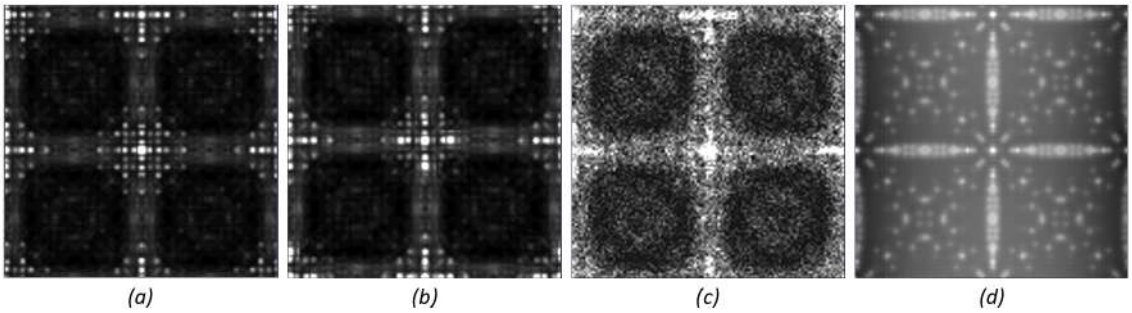

Figure 23: Simulation of TEM imaging of 1 cell of UiO-67. (a) is a perfect HAADF-STEM image, (b) is HAADF-STEM with 4 frozen phonons, (c) is HAADF-STEM with blur and noise, (d) is perfect iDPC-STEM.

3.2 Protocol

This report presents only the results for simulations without imperfections, i.e., without blur, noise, optical aberration, or frozen phonons. We evaluate the performance of the algorithm in reconstructing simulated images of UiO-67 using both HAADF-STEM and iDPC-STEM modes. In both cases, the datasets used for training are computed from MOF images generated with the same mode. The training dataset is derived from images of the following MOFs: MOF-5, PCN-61, IRMOF-76, ZIF-4, and MOF-1131. The validation dataset is calculated from UiO-66, MIL-53, and MOF-1115. All corresponding `.cif` files were downloaded from the CCDC and FIZ Karlsruhe databases available here. For each of these MOFs, 144 images were simulated for tilt angles between -90° and 90°, with a 1.25° interval. This number was chosen because it has many divisors, making it easier to extract subsets of this projection set. Thus, we studied the performance of the algorithm for reconstructions using 144, 72, 48, 36, 24, 18, 16, 12, 9, 8, 6, 4, or 1 projections between -90° and 90°, or using 113, 57, 29, 15, or 8 projections between -70° and 70°. SIRT reconstructions with all 144 projections were used as the reference volume. Unlike the previous section, no segmentation step is performed. All networks were trained using ReLU activation, exponential binning, and 8 hidden nodes, and all datasets contained 100,000 data points. For all SIRT reconstructions, we force the voxel values to be over 0, and we chose $\epsilon = 0.001$ for the stopping criteria. All figures were generated in the `atomic_reconstruction.ipynb` notebook located in the `scripts` folder on the Github page.

3.3 Results and discussion

Fig.24 and Fig.25 show the evolution of the MSE of the reconstructions with respect to the reference volume (SIRT with 144 projections), for different numbers of projections in the HAADF-STEM mode. Fig.26 shows the isosurface visualisation of the reference volume as well as the SIRT and NNFBP reconstructions for 8 projections. This is the HAADF mode: only the positions of the Zirconium atoms, the only

heavy elements in the structure, are recovered. Firstly, we note that the MSE value for the SIRT reconstruction with 144 projections is 0, which is consistent since we are comparing two identical volumes. Also for SIRT, there is a gradual increase in MSE as we reconstruct with fewer and fewer projections, which is again consistent. There are, however, some exceptions to this rule: the MSE is lower for 4 projections than for 6: this can be explained by the fact that some projections align with the atomic structure, which facilitates reconstruction. The gradual increase in MSE is also observed for NNFBP reconstructions, but in a much more erratic way: this is due to the high variability of the network's performance during training, which generates an element of chance in the evaluation of the MSE for this particular structure. This high variability can be explained by the fact that the network has difficulties to converge, because most of the voxels to predict are close to 0. In all cases, NNFBP performs much less well overall than SIRT. It should be noted, however, that the analysis carried out here is biased in favour of SIRT, since the reference volume was calculated using this method.

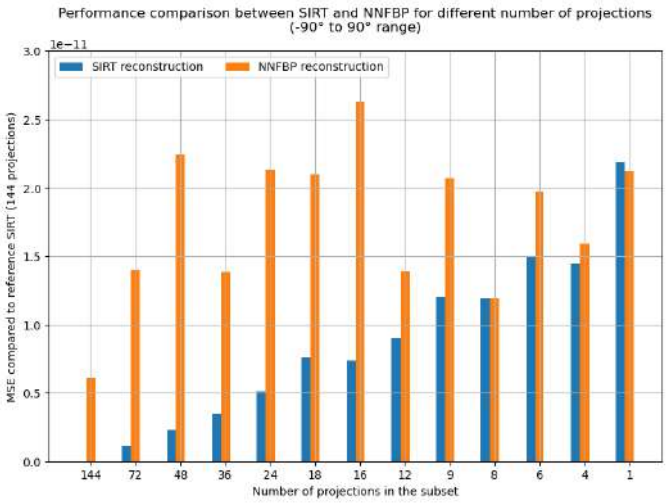

Figure 24: Performances of reconstruction algorithm with HAADF data and full range projections.

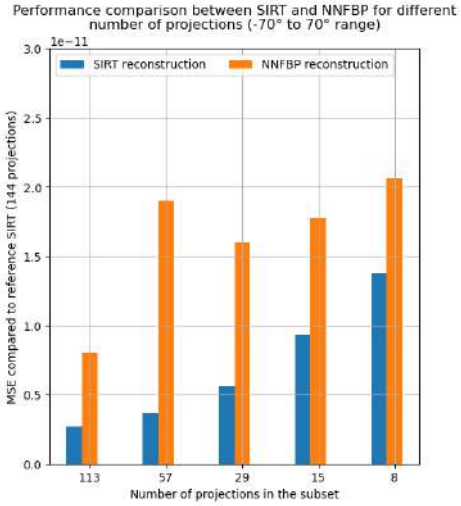

Figure 25: Performance of reconstruction algorithm with HAADF data, limited range.

Figs.27 and 28 show the evolution of the MSE for different numbers of projections in the iDPC-STEM mode. Fig.26 shows the isosurface visualisation of the reference volume as well as the SIRT and NNFBP reconstructions for 4 projections. In this mode, the light atoms are more contrasted, and the complete structuring of the MOF can be observed. Most of the comments made for the HAADF simulations also apply here, except that there is notably less randomness involved i.e. the convergence is better. However, it can be seen in Fig.29 that the geometry of the material structure can be recovered with only 4 projections, both with SIRT and NN-FBP.

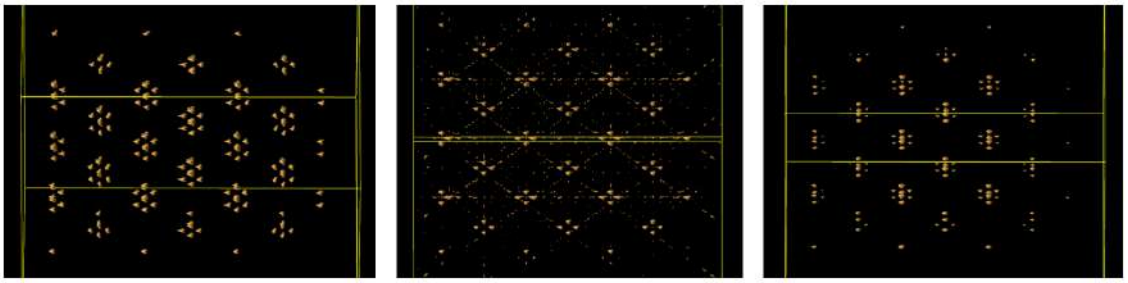

Figure 26: Reconstructions of the UiO-67 atomic structure with HAADF. From left to right: reference SIRT reconstruction (144 projs), SIRT reconstruction with 8 projections, NNFBP reconstruction with 8 projections.

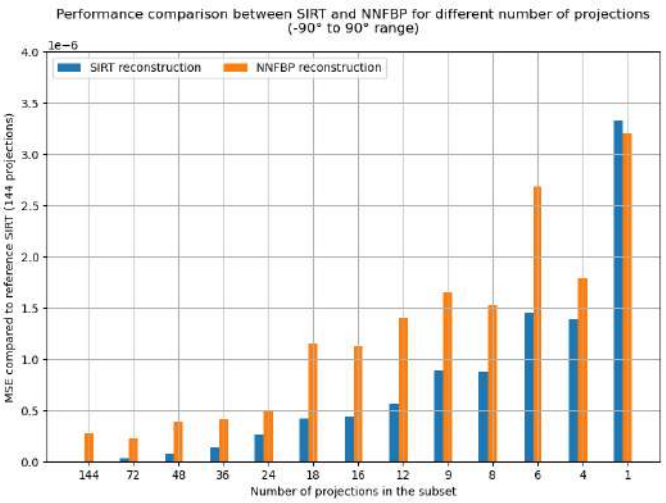

Figure 27: Performances of reconstruction algorithm with iDPC data and full range projections.

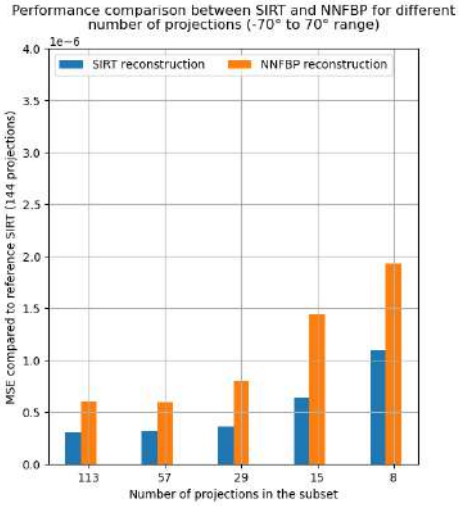

Figure 28: Performance of reconstruction algorithm with iDPC data, limited range.

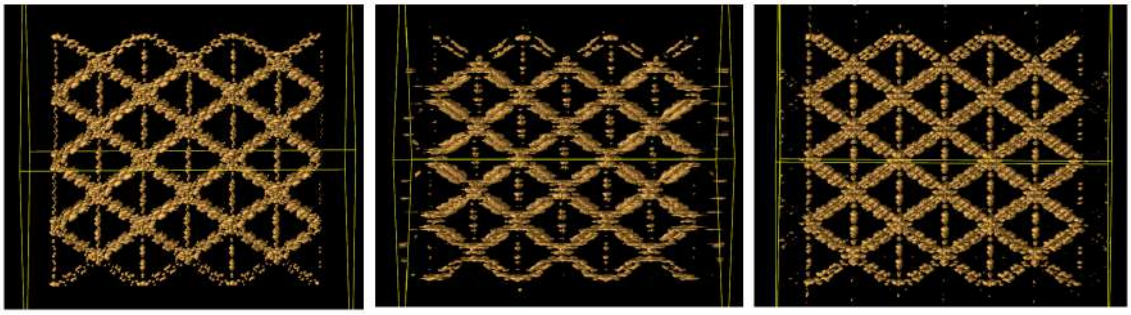

Figure 29: Reconstructions of the UiO-67 atomic structure with iDPC. From left to right: reference SIRT reconstruction (144 projs), SIRT reconstruction with 8 projections, NNFBP reconstruction with 8 projections.

3.4 Conclusion and outlooks

To conclude, here are a few ideas for improving these somewhat disappointing results. A first idea would be to find another way of determining the reference volume, either via an algorithm other than SIRT or directly from the atomic configuration (`.cif` file): a first approach, unfortunately unsuccessful, was to place a sphere of intensity 1 at the position of each atom, with a radius proportional to the radius of the atom. This would remove the bias in the analysis in favour of SIRT. A second, slightly less subtle idea would be to increase the size of the Datasets, by using the structures of a larger number of MOFs.

In all cases, and irrespective of the performance of the neural network, we can observe that SIRT manages to reconstruct rather well with a limited number of projections, and we generally manage to guess the structure of the MOF under consideration using isosurface visualisation 29. It remains to be seen whether SIRT performs just as well on experimental images: if SIRT is good enough, it would render the neural network obsolete.

4 General conclusion

During this internship, I had the opportunity to develop a Python project from start to finish, with nearly complete autonomy. The environment in which I worked, the Materials Science Department, allowed me to greatly deepen my knowledge in this field, particularly in electron microscopy and tomography - tools with which I had previously had little experience. A significant part of my work involved adapting my code so that it could be reused in the future to continue this study without requiring advanced programming expertise. I would like to thank Guido Schmitz for his guidance throughout the internship, the thought-provoking suggestions and improvements, as well as Roham Talei Jeid for his expertise in TEM and the experimental measurements conducted.

References

[1] Wikipedia, https://fr.wikipedia.org/wiki/Tache_d%27Airy, consulted on 17/09/2024

[2] A. Kaech *An Introduction to Electron Microscopy Instrumentation, Imaging and Preparation*, 2013

[3] E. J. Kirkland, *Advanced Computing in Electron Microscopy*, 2010, DOI: 10.1007/978-1-4419-6533-2

[4] G. Varvanides and al., *Iterative Phase Retrieval Algorithms for Scanning Transmission Electron Microscopy*, 2024, DOI: 10.48550/arXiv.2309.05250

[5] Christian S. Diercks, Omar M. Yaghi, *The atom, the molecule, and the covalent organic framework*, Science355, eaal1585(2017), DOI: 10.1126/science.aal1585

[6] VESTA Software 3.5.8, https://jp-minerals.org/vesta/en/, consulted on 17/09/2024

[7] S. L. James *Metal-organic frameworks*, Chem. Soc. Rev., 2003,32, 276-288, DOI: 10.1039/B200393G

[8] Liu, L., Zhang, D., Zhu, Y. et al. *Bulk and local structures of metal–organic frameworks unravelled by high-resolution electron microscopy*, Commun Chem 3, 99 (2020). DOI: 10.1038/s42004-020-00361-6

[9] D. M. Pelt and K. J. Batenburg, *Fast Tomographic Reconstruction From Limited Data Using Artificial Neural Networks*, in IEEE Transactions on Image Processing, vol. 22, no. 12, pp. 5238-5251, Dec. 2013, DOI: 10.1109/TIP.2013.2283142.

[10] Pelt, D. M., Sethian, J. A. (2018). *A mixed-scale dense convolutional neural network for image analysis. Proceedings of the National Academy of Sciences*, 115(2), 254-259.

[11] A. C. Kak and M. Slaney, *Principles of Computerized Tomographic Imaging*, Philadelphia, PA, USA: SIAM, Jul. 2001.

[12] Pelt, D. M., Batenburg, K. J., Sethian, J. A. (2018). *Improving Tomographic Reconstruction from Limited Data Using Mixed-Scale Dense Convolutional Neural Networks*, Journal of Imaging, 4(11), 128.

[13] IMOD 4.11, https://bio3d.colorado.edu/imod/, consulted on 17/09/2024

[14] Eva Bladt, Daniël M. Pelt, Sara Bals, Kees Joost Batenburg, *Electron tomography based on highly limited data using a neural network reconstruction technique*, Ultramicroscopy, Volume 158, 2015, Pages 81-88, ISSN 0304-3991, DOI: 10.1016/j.ultramic.2015.07.001.

[15] Shibata, N., Findlay, S., Kohno, Y. et al. *Differential phase-contrast microscopy at atomic resolution*, Nature Phys 8, 611–615 (2012), DOI: 10.1038/nphys2337

[16] D. J. Griffiths, *Introduction to quantum mechanics*, 1995

A Acronyms

4DSTEM - 4D Scanning Transmission Electron Microscope
ADF - Annular Dark Field
BF - Bright Field
CCD - Charged Coupled Device
COF - Covalent Organic Framework
FBP - Filtered BackProjection
GPU - Graphics Processing Unit
HAADF - High Angle Anular Dark Field
iDPC - integrated Differential Phase Contrast
MOF - Metal Organic Framework
MSDNET - Mixed Scale Dense convolutional neural NETwork [10]
MSE - Mean Square Error
NNFBP - Neural Network Filtered BackProjection [9]
ReLU - Rectified Linear Unit
SEM - Scanning Electron Microscope
SIRT - Simultaneous Iterative Reconstruction Technique
STEM - Scanning Transmission Electron Microscope
(C)TEM - (Conventional) Transmission Electron Microscope