BAM

Bundesanstalt für
Materialforschung
und -prüfung

Computing the forward and inverse problem of X-ray scattering

Master's Thesis

Sofya Laskina

January 3, 2023

Advisors: Dr. Philipp Benner, Dr. Brian Pauw

Department of Mathematics and Computer Science, Freie Universität Berlin

Freie Universität Berlin

Fachbereich Mathematik, Informatik und Physik

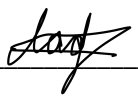
SELBSTSTÄNDIGKEITSERKLÄRUNG

Name: Laskina	(BITTE nur Block- oder Maschinenschrift verwenden.)
Vorname(n): Sofya	
Studiengang: M.Sc. Data Science	
Matr. Nr.: 5104279	

Ich erkläre gegenüber der Freien Universität Berlin, dass ich die vorliegende Masterarbeit selbstständig und ohne Benutzung anderer als der angegebenen Quellen und Hilfsmittel angefertigt habe.

Die vorliegende Arbeit ist frei von Plagiaten. Alle Ausführungen, die wörtlich oder inhaltlich aus anderen Schriften entnommen sind, habe ich als solche kenntlich gemacht.

Diese Arbeit wurde in gleicher oder ähnlicher Form noch bei keiner anderen Universität als Prüfungsleistung eingereicht.

Datum: 03.01.2023 Unterschrift: 

Acknowledgements

I would like to thank my both supervisors Dr. Philipp Benner and Dr. Brian Pauw for an opportunity to learn so many new things. I enjoyed working on this thesis a lot and found this topic extremely interesting and important. I feel like with everything I learned I have a great back- ground to continue working in academia. I would also like to express my gratitude towards the S.3 section and especially our head Dr. Thilo Muth for a chance to work on my thesis within BAM. S.3 section members were very friendly and helpful to me. I am especially thankful for our meetings, where I practised my pre- presentation skills. I would also like to acknowledge the inputs of Aakash Naik and Dr. Janine George who computed the electron density struc- ture of MOF unit cell presented in this thesis and Dr. Kuangdai Leng who gave some kick-off advice for this thesis. This work benefited from the use of the SasView application, originally developed under NSF award DMR-0520547. SasView contains code developed with fund- ing from the European Union's Horizon 2020 research and innovation programme under the SINE2020 project, grant agreement No 654000.

Abstract

Continuing progress in the field of X-ray scattering methods empowers scientists with new possibilities to capture the most important piece of information about the structure of the sample - its 3D electron density. Although the first methods appeared almost a century ago, recovering the density structure of a sample is still very problematic. Most available imaging techniques transform a 3D electron density of a realspace structure into the 2D Fourier Transform of the intensity of scattered waves in the reciprocal space. This process causes a loss of information. Firstly, instead of a 3D sample, a 2D image is created, and secondly, the phase information of the scattered waves is lost. The latter is known as the "phase problem" and poses a serious obstacle on a way to recover a 3D electron density. In this work, we draw attention to the problem of forward and inverse Small Angle X-Ray Scattering.

In the first, forward, part, we rethink the existing pipelines to computationally simulate such scattering experiments. Although there are efficient implementations of fast Fourier transformation, they often have some drawbacks. For instance, to calculate a 3D fast Fourier transform it is required to place its density in the RAM. For high-resolution structures of size $> 1024^3$, this becomes very problematic, as the whole density structure requires more than 16 GB of memory. CUDA solution allows for a very fast and parallelizable implementation of high-resolution data on hundreds of last-generation machines. Such computations are very pricy and inaccessible for most scientists. To bypass this limitation, we propose a solution for a split-up 3D fast Fourier transform, which is implemented as a sequence of 2D and 1D operations. We compare our implementation on the simulated 3D shapes and show the result of a proof-of-concept on 4096^3 Metall-organic framework density structure.

In the second, inverse problem, we train an invertible neural network, that given scattering data can predict the shape and its parameters. The architecture is built such, that the inverse problem is learned together with the forward process - the Fourier Transformation. We achieved very good results with this architecture, nonetheless, further testing is required, as the current training set only encompasses three simple shapes: sphere, hard sphere and cylinder.

All code to reproduce and analyze the results is available at https://github.com/sofyalaski/SAXS-simulations.

Zusamenfassung

Kontinuierliche Fortschritte auf dem Gebiet der Röntgenstreuungsmethoden bieten Wissenschaftlern neue Möglichkeiten, die wichtigste Information über die Struktur einer Probe zu erfassen - ihre 3D-Elektronendichte. Obwohl die ersten Methoden vor fast einem Jahrhundert entwickelt wurden, ist die Darstellung der Dichtestruktur einer Probe immer noch sehr problematisch. Die meisten verfügbaren Verfahren transformieren die 3D-Elektronendichte einer Realraumstruktur in die 2D-Fourier Transformation der Intensität der Streuung der Wellen im reziproken Raum. Dieser Prozess verursacht einen Informationsverlust. Erstens entsteht anstatt einer 3D-Probe ein 2D-Bild, zweitens geht die Phaseninformation der gestreuten Wellen verloren. Letzteres ist als "Phasenproblem" bekannt und stellt ein ernsthaftes Hindernis auf dem Weg zur Wiederherstellung der 3D-Elektronendichte dar. In dieser Arbeit wird der Fokus auf direktes und inverses Problem der Kleinwinkel-Röntgenstreuung gelegt.

Im ersten Teil überdenken wir die bestehenden Pipelines, um die Streuexperimente rechnerisch zu simulieren. Obwohl es effiziente Implementierungen der schnellen Fourier Transformation gibt, haben sie oft einige Nachteile. Zum Beispiel, um eine 3D-Fourier Transformation zu berechnen ist es erforderlich, die Elektronendichte im RAM zu platzieren. Bei hochauflösenden Strukturen der Größe $> 1024^3$ wird es sehr problematisch, da die gesamte Dichtestruktur mehr als 16 GB Speicher benötigt. Die CUDA-Lösung ermöglicht eine sehr schnelle und parallelisierbare Implementierung von solche Daten auf Hunderten von Maschinen der letzten Generation. Solche Berechnungen sind sehr teuer und unzugänglich für die meisten Wissenschaftler. Um diese Einschränkung zu umgehen, schlagen wir eine Lösung für eine aufgeteilte schnelle 3D-Fourier Transformation vor, die als Reihenfolge von 2D- und 1D-Operationen implementiert ist. Wir vergleichen unsere Implementierung auf den simulierten 3D-Strukturen und zeigen das Ergebnis eines Proof-of-Concept auf 4096^3 Metall-Organic Framework Dichtestruktur.

Im zweiten, inversen Problem trainieren wir ein invertierbares neuronales Netzwerk, das bei gegebenen Streudaten die Form und ihre Parameter vorhersagen kann. Die Architektur ist so aufgebaut, dass das inverse Problem zusammen mit dem direkten Prozess - der Fourier Transformation - erlernt wird. Wir haben mit dieser Architektur sehr gute Ergebnisse erzielt, dennoch sind weitere Tests erforderlich, da das aktuelle Trainingset nur drei einfache Formen umfasst: Kugel, Hartkugel und Zylinder.

Der gesamte Code zum Reproduzieren und Analysieren der Ergebnisse ist unter https://github.com/sofyalaski/SAXS-simulations verfügbar.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Reconstructing a 3D electron density distribution of a sample is a desired outcome of the materials research as it can help analyze the material in question from many perspectives. Most importantly, it can help infer the positions of atoms in the sample and with that, elucidate its whole structure. Further applications are possible, such as research on the chemical properties of the sample, molecular dynamics etc. Knowing the 3D electron density map of any structure could greatly proliferate research both in materials sciences and biology. Unfortunately, retrieving the whole 3D electron density of a sample is still very complicated.

Several methods exist that study 3D electron density structures like X-ray diffraction (XRD), nuclear magnetic resonance (NMR) spectroscopy or electron microscopy (EM). X-ray diffraction is the method that helped to create most of the 3D electron density maps available. It shoots with an X-ray toward the crystalline sample and the diffraction patterns are recorded on the detector. In single-crystal crystallography, the whole electron density structure can be recorded by gradually rotating a crystal. This information is a Fourier transform of the crystalline structure and retrieving a 3D electron density back from it is an issue unless the phase of the electron density structure is chosen smartly (Smyth and Martin, 2000). This technique resolves the scale of a few Ångstroms to a few nanometers. There is also powder diffraction, where diffraction peaks of the multitude of small crystals in a powder are measured at once (Holder and Schaak, 2019). It is harder to resolve structurally, but easier from a practical point of view as very is no need to grow a big enough crystal. In NMR spectroscopy an interaction between a magnetic field and excited atoms nuclei is measured (Macomber, 1997). NMR spectroscopy scale is comparable to X-ray crystallography. In EM microscopy an electron beam illuminates a thin sample creating a scattered image (Goodhew et al., 2000). An improvement of classic EM is the cryo-EM, which allows higher resolution of a few Ångstroms. Together these three methods

account for most electron density maps in Protein Data Bank[1], although by far the most samples were created with X-ray crystallography. Lately, the development of cryo-EM resulted in a great increase in the number of electron density maps retrieved with this technique. However, these methods also have their limitations.

Challenges in crystallography include the requirement for a material in a well-ordered crystalline form, which excludes the application of the technique to non-crystallizable samples. A change in the aggregate state of the sample may contradict the goals of the research, as ideally, a sample would be observed in its natural state and environment. On top of that comes the challenge of crystallizing a molecule. For some molecules, it is hard due to the instability of the solid state (Wunderlich, 1993). For complex molecules like proteins or polymers, other problems may arise: for instance, the purification of the sample in required amounts (Chayen and Saridakis, 2008) or multiple possible crystal structures (Pauw, 2009). Similar reasoning applies to EM or more specifically, cryo-EM. The procedure of the experiment requires lowering the temperature of the sample below 0°, which depending on the type of the sample can be problematic. Besides, cryo-EM only creates static maps of electron density, although this restriction can be avoided by use of its time-resolved version (Dandey et al., 2020). Finally, sensitivity is reported to be the biggest issue of the NMR spectroscopy (Emwas, 2015), such that signal strength is weaker when compared to other methods. Small-angle X-ray scattering (SAXS) technique is a good compromise to circumvent the issues of sample preparation but remain within the nanometers' resolution scale, although slightly larger than the typical XRD length scales. As such, in SAXS the scale of 1-100 nm is typically observed, although it may vary depending on the instrument.

SAXS allows studying a sample on a scale of nanometers and does not require the destruction of a sample or its cumbersome preparation. In comparison to wide-angle X-ray scattering/diffraction methods(WAXS/WAXD) SAXS allows probing of samples on a larger scale due to near 0° scattering angles. However, the result of a SAXS experiment is still not a density map, but a Fourier transform of contrast of electron density distribution in a sample. Analyzing the Fourier transformed *reciprocal* space is unintuitive, and an additional implication is posed by the surjectivity of the problem: many 3D electron density maps can result in the same Fourier transformed density divergence maps. It is often advised to use SAXS together with microscopy techniques to make some assumptions about the sample structure. With careful experiment settings, SAXS can give information about the size, shape, packing and orientation of the particles in the sample, though only one piece of information at a time, as it cannot determine all at once uniquely (Pauw, 2009).

[1] https://www.rcsb.org/stats

In this project, we do not conduct any laboratory experiments, instead, we simulate the settings and samples to provide some analytical pipelines that can help analyze data produced in a real SAXS experiment. To better convey what exactly happens in a SAXS experiment a more detailed description of scattering altogether, and especially this method is needed.

In scattering techniques, electromagnetic waves pass through the material and their electric field component polarizes the atoms of the material, causing the creation of dipoles (i.e. a pair of electric charges of a different sign). These dipoles oscillate with the frequency of the electric field induced by the initial wavelength. In X-ray scattering, the transferred energy is so high, it can polarize all electrons of the atoms in the material, such that each electron sends out a scattered wave (Glatter, 2018). When two oscillating electrons are located close to each other their scattered waves can interfere with each other resulting in a wave superposition with a different amplitude affecting the final intensity of radiation (Pauw, 2009).
In the idealized setting, the observed superposition of waves is coherent, meaning the electric fields have a fixed relation in phase and frequency. In reality, however, there are different domains in the materials, such that the superposition of any two scattered waves becomes incoherent due to the phase difference between the coherent regions in the sample (Pauw, 2009). For this project, we assume that the scattering waves are coherent.

Here, we skip the explanation for the geometry and equations required to describe the interference of waves scattered from two electrons. Instead, we use the notation of a scattering vector $\vec{q}$ defined as:

$$\vec{q} = \frac{4\pi}{\lambda} \sin\theta,$$

where λ is the wavelength of the primary radiation beam. 2θ historically designates the scattering angle between the primary and scattered beam (Guinier, 1955). The equation for the amplitude of the scattered radiation is given by:

$$A(\vec{q}) = \int \rho(\vec{x}) \cdot e^{-i2\pi\vec{q}\vec{x}}\, dx.$$

The amplitude A of the scattering angle vector $\vec{q}$ defined in the reciprocal space is an integral of the electron density vector $\rho(\vec{x})$ at a point represented by $\vec{x}$ of the scattered material. More intuitively, the scattering amplitude records the difference in the electron density of the sample in the Fourier space. The big limitation of the scattering devices, unfortunately, is that they cannot record the amplitude. Instead, the intensity of the radiation is recorded, which is mathematically represented as a square of the absolute

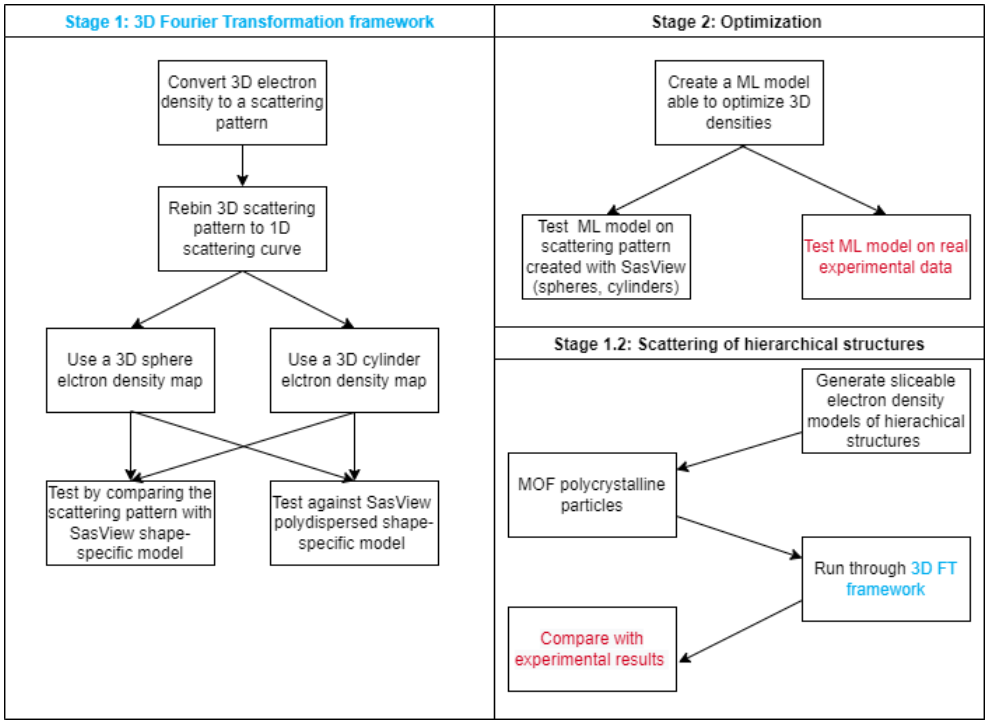


Figure 1.1: Schematic outline of the thesis

value of the amplitude of scattered waves:

$$I(\vec{q}) = |A(\vec{q})|^2.$$

Another important detail when talking about the SAXS experiment is that it does not record the whole 3D intensity of a sample but just a single slice at the center (Schmidt-Rohr, 2006). More precisely, this is not a slice, but the surface of the "Ewald" sphere, where the edges of the slice form a curved surface. At small angles, the Ewald sphere surface can be approximated by a flat surface.

In this project, we want to address two aspects of the abovementioned experiment. With that intent, we start from both "ends" of the scattering problem. In the first part, we deal with the 3D electron densities and perform 3D Fourier transform of some 3D simulated shapes. In the second part, we start at what is a result of a scattering experiment - the scattering curve. We test a few deep learning algorithms that given a scattering curve, should predict the properties of the sample. The connection might at first seem vague, so to clarify how the two are connected we prepared a schema (Figure 1.1).

The first part is called the 3D Fourier transform framework, and it does not only concern SAXS but the whole domain of scattering techniques or

even signal transmission. We propose an alternative computation of a 3D Fourier transform of 3D electron density. Computing 3D Fourier transform is not something new and effective algorithms exist already. A problem arises when computing such a transform for a high-resolution structure. In this case, most approaches reach their limitations fast depending on the platform of choice or memory availability. For instance, cuFFT[2] can compute 3D Fourier transform for a structure of size 8192^3 in just a few seconds. This, however, requires 1024 A100 NVIDIA Graphics Processor Units (GPU). Such computations are extremely costly and not all institutions can afford them. In this connection, we propose a solution for a high-resolution 3D Fourier transform that can run on simpler systems, usually available to research institutions.
The basis of the proposed solution is the discrete Fourier transform and its widely used implementation fast Fourier transform, both explained in subsection 2.1.3. We propose a "divide and conquer" approach, that changes the order of operations, explained in detail in subsection 3.1.2 and shows how it is compared in running time, memory requirements and error rates to the alternatives in subsection 4.1.2.
The high-resolution 3D FT approach developed in this thesis has been implemented as a proof-of-concept by the author of this thesis and further modified by Dr. Pauw to compute a demonstration of size 4096^3 based on a spherical metal-organic framework (MOF) particle shown in the chapter 5. For this sample, an electron density was calculated for a MOF unit cell by Aakash Naik. In the Figure 1.1, this step is designated as step 2b.

The second aspect of this thesis, marked as step 2 in the schema is an attempt to solve the inverse problem in scattering. Inverse problems often occur in the field of natural sciences and especially in its practical applications. The term "inverse" refers to the information flow in the problem addressed. In what one would on the contrary call the "forward problem", we want to predict the outcomes given the set of initial observations. In the inverse problem, on the other side, the data we possess are the outcomes of some process, and it is the initial parameters we would like to know. Depending on the application field, this process is well-studied. Examples of such applications include signal processing data. As in the application of scattering, there even might be a notion of the inverse of this process (inverse Fourier transform). Yet, it cannot be resolved that easily.
The main issue of the inverse problem is that they are often ill-posed. In particular, even if a solution is found it is often not a unique one. This has two reasons. Firstly, as we are mostly talking about the applications, in course of the "forward process" there is often a dimensionality reduction taking place. This is of particular importance for imaging techniques, as from the

[2] https://developer.nvidia.com/blog/multinode-multi-gpu-using-nvidia-cufftmp-ffts-at-scale/

3D structure a 2D image is created. Secondly, and this is not a trivial issue, there might not exist a single correct solution. This is the case in scattering: multiple 3D electron densities can result in the same 3D FT intensity, due to the loss of the phase information when measuring intensity instead of amplitude.
To conclude, in the inverse problem the forward mapping is usually surjective, but to be able to uniquely identify the parameters leading to the observation bijective mapping is desired. These types of mappings are shown explicitly in Figure 1.2.

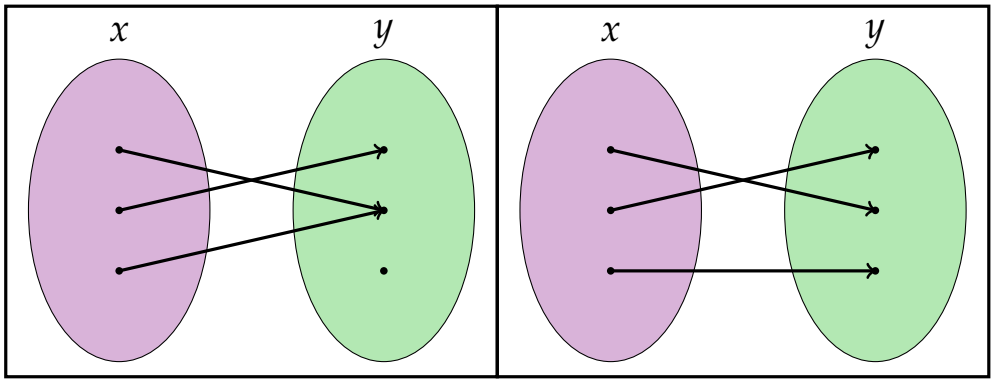


Figure 1.2: Types of function mappings. In the left box, an example of surjective mapping is shown. For each point x there is a point in y, but a few points in x may have the same image in y. On the right, the example of bijective mapping is shown. For each point x there is exactly one point in y and vice versa.

Here, we aim to resolve the surjective nature of the scattering problem by representing it in concatenation with a latent space as explained in subsection 2.3.4. We sample our training data in subsection 3.2.1; this training data is represented by the shape parameters of different samples and scattering curves created with an analytical solution for scattered shapes. We then build different neural networks in section 3.2.2 and optimize them. In section 4.2 we show that generative models produce the best results among tested architectures for our set of data.

Chapter 2

Background

2.1 Fourier transform

2.1.1 Brief introduction to Fourier transformation

For a long time, it was believed that any periodic function could be decomposed into a sum of sines. However, Joseph Fourier was the first who in his research of the heat distribution in spatial bodies introduced the idea of what now is called a Fourier transform (FT, also used as a verb in the following text) (Grafakos, 2014). With the FT a function can be broken down into components in the frequency domain. Accordingly, the former is defined in the *real space*'s direct coordinate system, whereas its Fourier transform exists in the frequency domain or the *reciprocal space*.

Mathematically, a continuous FT is given by the integral:

$$F(\xi) = \int_{-\infty}^{\infty} f(x) \cdot e^{-i2\pi\xi x}\, dx, \tag{2.1}$$

where $f(x)$ is an integrable function we want to transform, and ξ is a frequency value expressed as complex numbers. The inverse of the FT is subsequently defined as:

$$f(x) = \int_{-\infty}^{\infty} F(\xi) \cdot e^{i2\pi\xi x}\, d\xi \tag{2.2}$$

2.1.2 Fourier transformation in SAXS

A single SAXS experiment can be mathematically represented in terms of the Fourier transform. The electronic density of the probed sample is denoted as $\rho(\vec{x})$ at a point defined as vector $\vec{x}$. Then the amplitude $A(\vec{q})$ of the probed sample, or more precisely its electron density at scattering angle $\vec{q}$

in reciprocal space is given by:

$$A(\vec{q}) = \int \rho(\vec{x}) \cdot e^{-i2\pi\vec{q}\vec{x}}\, d\vec{x}. \tag{2.3}$$

In other words, the measured scattering intensity is the square of the amplitude of the FT (Guinier, 1955).

2.1.3 Discrete Fourier transform

The idea behind the approach in this work is to simulate scattering patterns from realistic electron density structures. This structure is placed in a 3D box of a certain size with a unit grid which bounds the problem to some scale. Such a setting of a 3D structure implies the Fourier transformation is performed on discrete points in space. In contrast to a continuous Fourier transformation formulated with the equations 2.1 and 2.2 in a discrete Fourier transformation (DFT) it is assumed that the function described within its boundaries is repeating itself periodically outside the boundaries (Butz, 2007). This condition perfectly fits the approach behind the unit box, as on each edge of a unit box one should imagine its repetition. The DFT has a similar mathematical formulation:

$$F(x) = \frac{1}{N} \sum_{k=0}^{N-1} f(k) \cdot e^{\frac{-i2\pi xk}{N}}, \tag{2.4}$$

and the inverse of the DFT is defined as:

$$f(k) = \sum_{x=0}^{N-1} F(x) \cdot e^{\frac{i2\pi xk}{N}}, \tag{2.5}$$

Compared to Equation 2.1 and Equation 2.2 there is a shift of the interval over a whole period from $[-T/2, T/2]$ to $[0, T]$, which is permitted, as function f is assumed to be periodic. The discrete points in space where the DFT is calculated are called sampling points. According to the Nyquist-Shannon sampling theorem, the accuracy of signal reconstruction depends on the sampling rate and when the latter is too low, aliases of the signal appear. To reproduce the signal accurately the sampling rate should be at least twice the signal frequency (Butz, 2007). For our purposes, the discrete sampling points will be the points on the simulation grid of the unit box. As follows, the simulation with more points in it should reproduce the process of FT more accurately.
In some sources, the DFT equation does not have a normalization constant before the sum, and it instead appears in the equation of the inverse DFT. We stick to the above notation as it otherwise does not hold that $F(x = 0)$ equals the average of all sampling points (Butz, 2007).

Implementation of discrete Fourier transformation

An approach to computing the DFT is called fast Fourier transform (FFT; also used as a verb in the following text) (Cooley and Tukey, 1965). In Python, it is implemented by packages NumPy, SciPy and PyTorch, where the latter is used in this work, as it can exploit the fast computation methods offered by Graphics Processor Units (GPUs).

There are slight differences in practical implementations that one should be aware of. These mainly involve the normalization of the sum of the Equation 2.4. As such, in the work of Cooley and Tukey, there is no normalization, but it appears in the calculation of the inverse DFT. This notation is subsequently implemented in the Python libraries but can be adapted with the proper use of the arguments.
Due to the interval shift in the notation of DFT, the result of the FFT calculation returns positive frequency terms first, which are followed by negative terms. To rearrange it back as a sequence of negative terms followed by positive, there is a function *fftshift*.

2.2 Mathematical preliminaries

The next sections concern neural network architectures and require some background (Behrmann et al., 2019; Borgwardt et al., 2006). For improved clarity, all important mathematical definitions and explanations are collected in this part.
We will begin with the Lipschitz continuity, an important concept in the field of machine learning. It explores the relation of distances between two points in the domain and codomain of the function. An intuition behind the Lipschitz continuity is that in a machine learning application we want similar data points to stay close to each other under the function we learn. We then mention a definition of a fixed point, as many methods accept numerical solutions to the problem, and state the Banach fixed point theorem, which guarantees the uniqueness of a fixed point. This introduction is necessary for the invertible residual networks introduced in subsection 2.3.2. We then continue with a presentation of a few vector spaces, such as inner product, Hilbert space and a reproducing kernel Hilbert space together with a definition of a kernel function to explain a hypothesis test called maximum mean discrepancy. This method allows for the comparison of two samples drawn from different distributions and is used to optimize the machine learning application in this work (section 3.2.2).

Lipschitz continuity

For two metric spaces (X, d_x) and (Y, d_y), with d_i representing the distance function on the respective space, a function $f : X \to Y$ is called Lipschitz

continuous if there exists a constant $k \geq 0 \in \mathbb{R}$, such that

$$d_y(f(x_1), f(x_2)) \leq k \cdot d_x(x_1, x_2),$$

for all x_1 and x_2 from X. The smallest such constant is oftentimes called the *Lipschitz constant*. If $0 \leq k < 1$ and function f maps a metric space to itself, the f is called a *contraction*.

Fixed point
A fixed point is a point x^*, that does not change upon application of a mapping f: $x^* = f(x^*)$.

Banach fixed point theorem
Let (X, d) be a non-empty complete metric space with a map $T : X \to X$ being a contraction, meaning, that there exists $q \in [0, 1)$, such that $d(T(y), T(y)) \leq d(x, y)$, for all $x, y \in X$. Then T has a unique fixed point $x^* = T(x^*)$ and a $\lim_{N\to\infty} T^N(x_0) = x^*$

Inner product
$\mathcal{H}$ is a vector space with an inner product $\langle \cdot, \cdot \rangle_{\mathcal{H}} : \mathcal{H} \times \mathcal{H} \to \mathbb{R}$. This inner product has the following properties:

1. linearity: $\langle \alpha_1 f_1 + \alpha_2 f_2, g \rangle_{\mathcal{H}} = \alpha_1 \langle f_1, g \rangle_{\mathcal{H}} + \alpha_2 \langle f_2, g \rangle_{\mathcal{H}}$
2. symmetry :$\langle f, g \rangle_{\mathcal{H}} = \langle g, f \rangle_{\mathcal{H}}$
3. positive definite: $\langle f, f \rangle_{\mathcal{H}} \geq 0$ for all f and equality is true only if $f = 0$

Hibert Space
Hilbert space $\mathcal{H}$ is an inner product space and a complete metric space, meaning that the inner product space contains the limits of the Cauchy sequence. The norm is defined as: $\|f\|^2_{\mathcal{H}} = \langle f, f \rangle_{\mathcal{H}}$.

Kernel
A kernel function $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is a real-valued inner product of elements of a non-empty set $\mathcal{X}$ with a mapping $\varphi : \mathcal{X} \to \mathcal{H}$, such that for all $x, x' \in \mathcal{X}$ it holds that $k(x, x') := \langle \varphi(x), \varphi(x') \rangle_{\mathcal{H}}$.
A symmetric kernel $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is positive definite if for all $n \geq 1$, $(a_1, \ldots, a_n) \in \mathbb{R}^n$ and for all $(x_1, \ldots, x_n) \in \mathcal{X}^n$ it holds:

$$\sum_{i=1}^{n} \sum_{j=1}^{n} a_i a_j k(x_i, x_j) \geq 0.$$

The **kernel trick** is an observation, that for all points in space $x \in \mathcal{X}$ and for

all functions in another space $f(\cdot) \in \mathcal{H}$ the inner product of both equals the function evaluated at this point $\langle f(\cdot), k(\cdot, x)\rangle_{\mathcal{H}} = f(x)$. Especially, it holds that $k(x, y) = \langle k(\cdot, x), k(\cdot, y)\rangle$ for all $x, y \in \mathcal{X}$. This property is also called the *reproducing property*.

Reproducing Kernel Hilbert Space (RKHS)
After all the important fundamentals are explained, we may proceed with the definition of the RKHS.
Let $\mathcal{H}$ denote a Hilbert space of real-valued functions on a non-empty space $\mathcal{X}$. Let $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ denote a reproducing kernel on the Hilbert space $\mathcal{H}$. $\mathcal{H}$ is called a reproducing kernel Hilbert space if:

- for all $x \in \mathcal{X}, k(\cdot, x) \in \mathcal{H}$
- for all points $\mathcal{X}$ and for all functions in $\mathcal{H}$ the reproducing property holds.

And alternative definition is that $\mathcal{H}$ is RKHS, if for all functions $f \in \mathcal{H}$ the function evaluation operator δ_x at all $x \in \mathcal{X}$ is bounded by some $\lambda_x \geq 0$:

$$|f(x)| = |\delta_x f| \leq \lambda_x \|f\|_{\mathcal{H}}.$$

This condition at the same time implies the evaluation operator δ_x is continuous.

2.2.1 Maximum Mean Discrepancy

Maximum Mean Discrepancy (MMD) is a test used to compare two samples drawn from different distributions, by comparing the distance in Hilbert space between their mean embeddings. The MMD is defined as an integral probability metric (Müller, 1997): Let $\mathcal{F}$ be a class of functions $f : \mathcal{X} \to \mathbb{R}$ and x and y be two random variables defined in $\mathcal{X}$ with the respective Borel probability measures p and q.
Let $X := \{x_1, \ldots, x_m\}$ and $Y := \{y_1, \ldots, y_n\}$ be two independent and identically distributed (iid) observations from p and q respectively. The MMD is defined as

$$\mathrm{MMD}\,[\mathcal{F}, X, Y] := \sup_{f \in \mathcal{F}} (\mathbb{E}_x[f(x)] - \mathbb{E}_y[f(y_i)]).$$

The biased empirical estimate of MMD is defined as:

$$\mathrm{MMD}_b\,[\mathcal{F}, X, Y] := \sup_{f \in \mathcal{F}} \left(\frac{1}{m}\sum_{i=1}^{m} f(x_i) - \frac{1}{n}\sum_{i=1}^{n} f(y_i)\right).$$

This notation of function $f \in \mathcal{H}$ is improved to account for the embeddings of a probability distribution $\mu_p \in \mathcal{H}$ in RKHS, such that $\mathbb{E}_x f = \langle \mu_p, f \rangle_{\mathcal{H}}$.

Such an embedding is called mean embedding and together with previous definitions and properties it follows that (Borgwardt et al., 2006):

$$\begin{aligned}\mathrm{MMD}\left[\mathcal{F}, X, Y\right] &= \sup_{f\in\mathcal{F}}(\mathbb{E}_x[f(x)] - \mathbb{E}_y[f(y_i)]) \\ &= \sup_{f\in\mathcal{F}} \left\langle f, \mu_p - \mu_q \right\rangle \\ &= \sup_{f\in\mathcal{F}} \|\mu_p - \mu_q\|_{\mathcal{H}}\end{aligned}$$

A kernel two-sample test (Gretton et al., 2012) is an unbiased estimate of the MMD, defined as:

$$\mathrm{MMD}_u\left[\mathcal{F}, X, Y\right] = \frac{1}{m(m-1)}\sum_{i=1}^{m}\sum_{j\neq i}^{m} k(x_i, x_j) + \frac{1}{n(n-1)}\sum_{i=1}^{m}\sum_{j\neq i}^{m} k(y_i, y_j) - \frac{1}{mn}\sum_{i=1}^{m}\sum_{j=i}^{m} k(x_i, y_j).$$

2.3 Machine Learning Models

For this project, we will attempt several distinct machine learning models. These are

1. Invertible residual networks (i-ResNet)
2. Real-Valued non-volume preserving transformations (RNVP)
3. Feed-forward networks.

The first two learning algorithms represent the class of models called "Normalizing flows". This type of generative modeling architecture aims to reconstruct the probability distribution of data. In the next subsection 2.3.1 an overview of this class of algorithms is given. In subsection 2.3.2 the specifics of regularization and implementation of i-ResNet are explained. Finally, subsection 2.3.3 defines the RNVP architecture.
Details on implementation and hyperparameter choices are given in section 3.2 and the results are demonstrated in section 4.2.

2.3.1 Normalizing Flows

Normalizing flows is a class of *generative models* that allows transforming a simple distribution of the data into a more complex one by concatenating multiple bijective, differentiable mappings. Because of the bijectivity, the initial density distribution of the sample can, hence, be reconstructed back and computed as the product of the value of the inverse function of the sample under the complex distribution and the product of the absolute values of the determinants of the Jacobians for each backward transformation (Kobyzev et al., 2021).
Let $Z \in \mathbb{R}^d$ be a random variable, with known probability density function

(PDF) $p_z = \mathbb{R}^d \rightarrow \mathbb{R}^d$ and g be an invertible function $X = g(Z)$. With the change of variables the formula the PDF p_x is then computable as:

$$\begin{aligned} p_x(X) &= p_z(f(X)) \cdot |\det J_f(X)| \\ &= p_z(f(X)) \cdot |\det J_g(f(X))|^{-1}, \end{aligned} \tag{2.6}$$

where $f = g^{-1}$, $J_g(Z) = J_g(f(X)) = \frac{\partial g}{\partial Z}$ is the Jacobian of the function g and $J_f(X) = \frac{\partial f}{\partial X}$ is the Jacobian of f. The prior distribution $p_z(Z)$ is chosen as a simple one, often a Normal distribution. The forward flow g is subsequently called the *generative direction* and the backward f the *normalizing direction*.
This is followed by an observation that the composition of such invertible functions $g = g_N \circ g_{N-1} \circ \ldots \circ g_N$ is itself also invertible as $f = f_1 \circ \ldots \circ f_{N-1} \circ f_N$ and the determinant of the Jacobian takes the form of a product: $\det J_f(X) = \prod_{i=i}^{N} \det J_{f_i}(X)$ (Kobyzev et al., 2021).
The advantage of Normalizing Flows is their interpretability as the maximum likelihood of observing parameters given a model. Despite this benefit, the design of the network is not a trivial problem and is subject of current research. The requirement for the functions in the application to be invertible, and for both forward and inverse functions to be differentiable poses certain constraints to the effective computation of the Jacobian matrix.

2.3.2 Invertible Residual Networks

Similarly to the Residual NNs (He et al., 2015) i-ResNet utilizes the skip connections, but draws attention to the backward flow: $x_t \leftarrow x_{t+1} - g_{\theta_r}(x_t)$, where $x \in \mathbb{R}^d$ represents the activations, t - the layer indicies and g_{θ_t} the respective residual blocks of the layers. The condition for invertibility is based on the theorem (Behrmann et al., 2019):

Theorem 2.1 *Let $F_\theta : \mathbb{R}^d \rightarrow \mathbb{R}^d$ denote a ResNet and define as $F_\theta = (F_\theta^1 \circ \ldots \circ F_\theta^T)$, consisting of blocks $F_\theta^t = I + g_{\theta_t}$. F_θ is invertible if the Lipschitz constant $Lip(g_{\theta_t}) < 1$, for all $t = 1, \ldots, T$.*

This theorem albeit holding the requirements of the generative modeling, gives no idea, about what an inverse looks like. The authors propose a numerical solution to find a fixed point of an inverse function. Theorem 2.1 together with the Banach fixed point theorem 2.2 result in a solution of a form :

$$\|x - x^n\|_2 \leq \frac{Lip(g)^n}{1 - Lip(g)} \|x^1 - x^0\|_2.$$

This concludes, that smaller Lipschitz constant converges faster and the convergence itself is exponential to the number of iterations n. In the NN it is

implemented via the estimation of the spectral norm of the weight matrices $\|W_i\|_2$ of each network layer through the power iteration, yielding an underestimate $\tilde{\sigma_i} \leq \|W_i\|_2$. The regularization for each layer is then defined as follows:

$$\tilde{W_i} = \begin{cases} cW_i/\tilde{\sigma_i}\,, \text{if} c/\tilde{\sigma_i} < 1 \\ W_i, else \end{cases},$$

where c is the scaling constant. With this network design, the Lipschitz condition guarantees stability for both forward and inverse passes.

Modelling i-ResNet

With Equation 2.6 in form of log-likelihood: $\ln\, p_x(X) = \ln\, p_z(f(X)) + \ln|\det J_f(X)|$ we can compute the likelihood of the parameters given a model. The last term $\ln|\det J_f(X)|$ is computable in $\mathcal{O}(d^3)$ time for data of dimension d, which makes it infeasible for high-dimensional data. Instead, the identity of logarithm $\ln|\det J_f(X)| = \text{tr}(\ln J_f)$ is computed (Withers and Nadarajah, 2010), such that the likelihood of the parameters under the model can be expressed as:

$$\ln\, p_x(X) = \ln\, p_z(Z) + \text{tr}(\ln(I + J_{g_\theta}(X))),$$

because $Z = f(X) = (I + g_\theta)(X)$. It is important here to distinguish the generative function $g : Z \to X$ defined in subsection 2.3.1 from the residual blocks g_θ in the definition of the i-ResNet, as introduced earlier.
Finally, the likelihood calculation can be expressed in the terms of the power series (Hall, 2013):

$$\ln\, p_x(X) = \ln\, p_z(Z) + \sum_{k=1}^{\infty} (-1)^{k+1} \frac{\text{tr}(J_{g_\theta}^k)}{k}.$$

This unbiased estimator becomes a biased one on the truncation of the series by n terms. The truncation error is dependent on the data dimensionality d, the Lipschitz constant $Lip(g_\theta)$ and the number of terms n. It is bounded as:

$$|\text{error}| \leq -d \left(\ln(1 - Lip(g_\theta)) + \sum_{k=1}^{n} \frac{Lip(g_\theta)^k}{k} \right).$$

2.3.3 Invertible coupling layers with effectively computable determinant of the Jacobian

Coupling layers are another example of the generative flow. Similarly to i-ResNet, they also estimate the maximum log-likelihood of the parameters under the model. To enable fast calculation of the Jacobian determinant in

Equation 2.6 authors use the examination that the determinant of the triangular matrix is computable as the product of its diagonal elements. This idea is implemented in terms of Non-linear Independent Component Estimation (NICE) coupling blocks (Dinh et al., 2015).
A single coupling layer represents a bijective invertible mapping. First, the input $x \in \mathbb{R}^D$ is divided into two partitions $[x_1, x_2]$. The output of the coupling layer is then defined as $y = [y_1, y_2]$:

$$y_1 = x_1$$
$$y2 = x_2 + m(x_1),$$

where m is called *the coupling function*. The Jacobian for such a transformation then takes the form of a triangular matrix:

$$\frac{\partial y}{\partial x} = \begin{bmatrix} I_{x_1} & 0 \\ \frac{\partial y_2}{\partial x_1} & \frac{\partial y_2}{\partial x_2} \end{bmatrix},$$

where I_{x_1} is the identity matrix of the size of the first partition. Hence, the determinant of the Jacobian matrix of the whole coupling layer equals the determinant of the Jacobian of the second partition y_2 with respect to the second part of the input x_2. The trick is that the Jacobian of coupling function m does not need to be computed and can be set as a complex one, for instance, a deep NN. The inverse functions are also easy to compute, especially because the inverse of m is not required (Dinh et al., 2015):

$$x_1 = y_1$$
$$x2 = y_2 - m(y_1).$$

The architecture altogether consists of multiple coupling blocks with permutations in between, such that the information in the coupling function is seen at least a few times for all inputs.

These efforts were enhanced in the following work on RNVP transformation. They obey the same principle as the NICE blocks, with the determinant of the Jacobian of the network not requiring the computation of the coupling function. The improvement lies in the function itself. A single coupling layer is subsequently defined as follows:

$$y_1 = x_1$$
$$y2 = x_2 \odot \exp(s(x_1)) + t(x_1),$$

where s and t function stand for "scale" and "translation" and $\odot$ is the element-wise product operator. The Jacobian of such a coupling layer then takes the form:

$$\frac{\partial y}{\partial x} = \begin{bmatrix} I_{x_1} & 0 \\ \frac{\partial y_2}{\partial x_1} & \text{diag}(\exp[s(x_1)]) \end{bmatrix},$$

where $\text{diag}(\exp[s(x_1)])$ is the matrix with diagonal elements being the respective values of the first part of the inputs $\exp[s(x_1)]$.
Again, the maximum likelihood of the inputs can be computed efficiently, because the determinant of the Jacobian can be computed as a product of diagonal elements of the triangular Jacobian matrix: $\exp[\sum_j s(x_1)_j])$. This consequently does not require evaluating the Jacobian for s and t, which is implemented as deep convolutional neural networks. Another advantage of this architecture is that it analogically to the NICE blocks does not require the computation of the inverse of s and t functions for the normalizing direction either and allows for multiple such coupling layers shuffled in-between to learn the accurate representation (Dinh et al., 2017).
Another interesting improvement of this work was adding condition variables to the coupling layers. This is possible due to the irrelevance of coupling functions s ad t to the inversion of the network. Following that, conditional data is mixed into the inputs of the neural networks in the coupling layer (Ardizzone et al., 2019b).

2.3.4 Invertable Neural Networks

The idea of Invertible neural networks is to restore the bijectivity of the forward process through sampling a latent space vector z, such that the mapping from the vector of parameters to the vector of scattering curves $x \rightarrow y$ is substituted via mapping of the same parameters vector to the combination of observed outcomes vector with the latent space vector: $x \rightarrow [y, z]$. It is another example of generative flow models and the architecture neural network is then constructed in a way, that in the forward process the mapping $f(x) = [y, z]$ is optimized. Because the claim is, the network is bijective, the inverse $x = f^{-1}(y, z)$ can be learned simultaneously (Ardizzone et al., 2019a). In this model, the previously discussed coupling layers or i-ResNets are used to effectively compute the inputs. The optimization is performed by minimizing the implementation of Maximum Mean Discrepancy loss.

2.4 Literature overview of methods for an inverse problem in scattering

The analysis of scattering patterns and an attempt to infer the atomic structure of a sample emerged together with the development of the first X-ray techniques. In the classic work (Porod, 1951) the "characteristic curve" was introduced, in which no reference to the exact atomic structure is made, but a probability for a point inside a particle is calculated that another point at distance r lies within the same particle. This approach was widely explored in the SAXS community. A smoothness correction of the pair-distance dis-

tribution function (PDDF) with B-spline functions was proposed (Glatter, 1977), which requires a knowledge of the maximum distance inside the particle. A similar approach using the Tikhonov and Arsenin regularization of ill-posed problems (Tikhonov and Arsenin, 1977) was applied for SAXS data resulting in faster and smoother PDDF curves (Svergun et al., 1988). This gave rise to a series of works, where the hyperparameters of PDDF were estimated with Bayesian inference. A weighted function of distances inside the particles was suggested, which resulted in a better approximation of PDDF compared to simple regularization of curves (Hansen, 2000, 2003). It additionally provided information on the inference of such parameters as noize levels, the volume fraction of a scattered particle and even an approximation of polydispersity parameter (Hansen, 2008).
Another class of developed solutions covers fitting scattering curves or scattering patterns to certain models inferring the shape parameters with Monte Carlo optimization or other non-linear optimization techniques (Bressler et al., 2015; Doucet et al.; Ilavsky and Jemian, 2009). The limitation of this approach is binding to a certain model; the user needs to have an assumption on a shape form to infer the parameters.
A further approach, which is more interesting in terms of 3D electron density maps is the reverse Monte Carlo simulations, where a unit box filled with particles is optimized until fits the scattering curves (Sarje et al., 2014; Hagita et al., 2018). This method performs very well and can predict a particle's structure resulting in a certain scattering pattern. The limitations of this method include a high number of parameters that need to be optimized, which requires parallel computing; An additional downside of this method is a rather small resolution of a unit box, where the simulation is placed. Within that simulation box, all the coarse-grained particles will be moved in course of the Metroppolis-Hastings algorithm which can be described as applying a random walk to each instance. A move is accepted if the loss improves and, in case the loss worsens, it still can be accepted with weighted probability. Reverse Monte Carlo simulations take a considerable amount of time and space to converge for a system, where each entity is positioned in a 3D space. Moreover, it requires a quite good initial guess and seems to deliver good results only for monodispersed systems.
Neural networks were applied in different forms for the scattering data analysis. As such, it is possible to extract the rotational distribution for hexagonal particles in grazing incident small angle scattering (GISAS) with dense neural networks (Herck et al., 2021) or infer the unit cell structure and orientation with convolutional neural networks (Liu et al., 2019). Finally, a similar approach to the current work was tested on the training data simulated with SasView (Do et al., 2020), which predicted the class of the shape. In another study, an autoencoder was implemented, simulating full 3D electron density of a structure of size 32^3, which performs well, but is limited in resolution and needs a lot of time to train (He et al., 2020).

Chapter 3

Methods

This chapter is split into two main parts. The first part gives detailed information on the process of 3D Fourier transformation and implementation of a high-resolution version in this work. Then, the performance of this approach is compared with other state-of-the-art methods. Additionally, a comparison to methods specifically designed to calculate scattering data is made. This relies on an analytical solution of integrating scattering from certain shapes. To enable this kind of benchmarking an efficient implementation of shapes in a 3D space is required. In this work, only three shapes are considered: spheres, hard spheres (or non-overlapping, densely packed spheres) and oriented symmetric cylinders, which are then compared against the analytical solutions.
The second part of this chapter gives specifics on the creation of a training set, used to learn an inverse problem in SAXS as well as information on the machine learning model itself.
All the data used in the analysis was simulated, and the process is explained in the methods sections. This work was carried out on a GPU Nvidia RTX5000 machine with 16 GB RAM.

3.1 Split-up implementation of a 3D FFT

3.1.1 Reasons for splitting up a Fourier transform

While the 3D Fourier transform can be used to calculate the scattering of particular electron densities, it is physically limited. One hard limitation is the requirement of exponentially more memory for modest increases in the number of points in the 3D model. That leads to a practical limitation for box-shaped densities of 512^3 pixels (or 1024^3), as shown in Figure 4.10. Furthermore, such 3D FFT computations cannot easily be split up over multiple computational cores. These issues have been addressed in this work. Through modification of the 3D FFT, the aim is to be able to separate its com-

putation into slices. By splitting the calculation up, it becomes possible to parallelize the computation, which can speed up the computation, and/or enables the computation of the FFT at a much higher resolution. Lastly, this modification also allows the computation of only the information of interest. For instance, only the FFT of a central slice represents the information in the scattering pattern, and we can save time only computing this single slice. This saves time by omitting the calculation of unnecessary data points.

As shown in Figure 3.1, instead of computing a 3D FFT of the structure as a whole, here a different approach is proposed. The first thing to notice is that the simulated structure inside the box is not present in each of the 2D slices created by the grid. The calculation of the 3D FFT is broken down into, 2D FFTed slices followed by 1D FFTed vectors. As mentioned, some slices do not carry any information and can be simply disregarded in the FFT matrix, initialized to all zeros, such that the actual computation is only processed on slices, containing the density data. After the slices were computed, a final FT of 1D vectors in the remaining direction must be performed. This is explicitly shown in Figure 3.2. The 2D FFTed slices, for example in $yz-$ direction, must be completed with the 1D FFT in the direction of the $x-$axis.

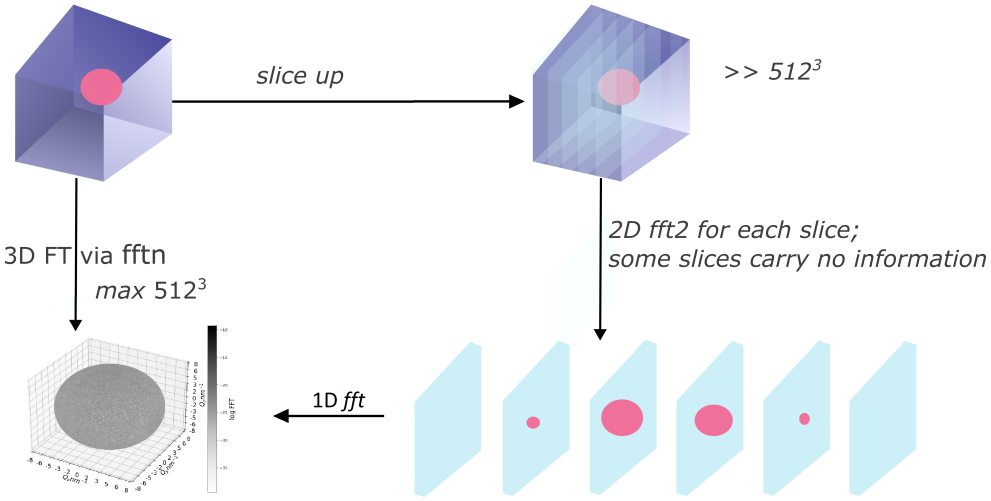


Figure 3.1: Alternative ways to calculate 3D FFT. Given the sample density (left top), the straightforward method is to FT the structure as a whole with available methods for n-dimensional DFT (left bottom). An alternative is to divide computation into 2D FFted slices and 1D FTed vectors in the remaining direction (right side), which results in the same 3D FFT.

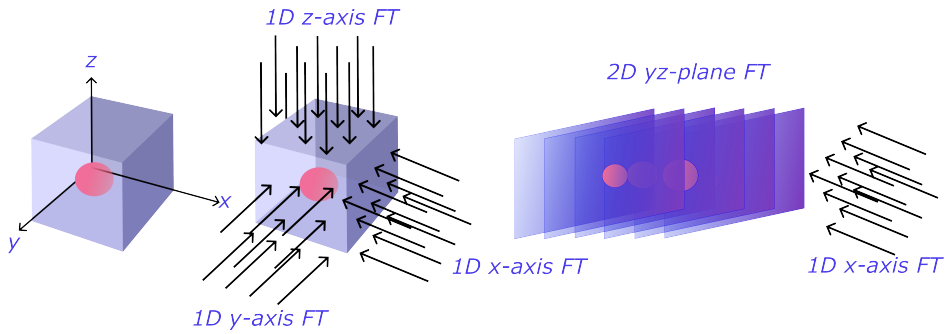


Figure 3.2: Breaking down the calculation of 3D FFT, which constitutes three consequent 1D FFT operations in the direction of each of the $x-$, $y-$ and $z-$axes. This can also be represented as FFT of 2D slices, here, for instance, yz-sices and the final 1D FFT in $x-$direction.

3.1.2 Changing order of operations in calculation of the Fourier transform

In this part, the mathematical explanation for the approach to cutting up a 3D FFT into 2D slices is demonstrated. To begin with, the definition of a 1D DFT is given. It is then extended into a 2D version and its representation in terms of 1D. Finally, we continue with a 3D FFT and its representation in terms of 2D.
The DFT of the electron density sampling points at positions $x = \{0, 1, \ldots, N-1\}$ in 1D is given by the formula:

$$F(u) = \frac{1}{M} \sum_{x=0}^{M-1} \rho_x \cdot e^{-\frac{i2\pi}{M} ux}, \tag{3.1}$$

where ρ_x is the density of sampling points at x and u is the domain in 1D space the signal is measured on.

Similarly, for a 2D system, the equation is given by the formula:

$$F(u,v) = \frac{1}{MN} \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} \rho_{xy} \cdot e^{-i2\pi(\frac{ux}{M} + \frac{vy}{N})}, \tag{3.2}$$

where the density ρ_{xy} is now given by a sampling point at (x, y) in a 2D coordinate system and u and v is the domain of a 2D space the signal is measured on.
One can rewrite equation 3.2 by pulling the exponential term out of the second summation as it is not dependent on y (Origin Lab):

$$F(u,v) = \frac{1}{M} \sum_{x=0}^{M-1} e^{-i2\pi \frac{ux}{M}} \frac{1}{N} \sum_{y=0}^{N-1} \rho_{xy} \cdot e^{-i2\pi \frac{vy}{N}}$$

This way the 2D FFT of the density is evaluated at each point in the space domain of v and then summed over the space domain of u. Practically it

can be calculated as two consequent FFTs in 1D over the space domains of v and then u. In the first FFT, the input signal will be the electron density and in the second it is the 1D FTed density.
We continue to deal with the FFT of the electron densities and now expand our structure to the 3D space with the formula:

$$F(u,v,w) = \frac{1}{MNL} \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} \sum_{z=0}^{L-1} \rho_{xyz} \cdot e^{-i2\pi(\frac{ux}{M}+\frac{vy}{N}+\frac{wz}{L})}, \tag{3.3}$$

where the density ρ_{xyz} is now given by a sampling point at (x,y,z) coordinates in 3D space and u, v and w describe each of the 3 domains in space the signal is measured on.
Analogous to the representation of 2D FFT as a series of 1D, the equation can be rewritten by pulling the exponential term depending on x out of the two summations:

$$F(u,v,w) = \frac{1}{M} \sum_{x=0}^{M-1} e^{-i2\pi\frac{ux}{M}} \left(\frac{1}{N} \sum_{y=0}^{N-1} \frac{1}{L} \sum_{z=0}^{L-1} \rho_{xyz} e^{-i2\pi(\frac{vy}{N}+\frac{wz}{L})} \right)$$

The 2D FFT from 3.2 can be recognized in the brackets of this formula and, hence, the 3D FFT can be calculated by adding up the 2D FFTs calculated in the domains of v and w for each point in the domain of u. Alternatively, the 3D FFT can be calculated as a sequence of 3 1D FFTs in each space domain:

$$F(u,v,w) = \frac{1}{M} \sum_{x=0}^{M-1} e^{-i2\pi\frac{ux}{M}} \left(\frac{1}{N} \sum_{y=0}^{N-1} e^{-i2\pi\frac{vy}{N}} \left(\frac{1}{L} \sum_{z=0}^{L-1} \rho_{xyz} e^{-i2\pi\frac{wz}{L}} \right) \right)$$

3.1.3 Implementation of split up 3D FFT

The Python PyTorch (Paszke et al., 2019) library offers all the necessary functionality to calculate the 3D FFT. It provides a function *fftn* available, that computes an n-dimensional FFT of the passed argument and applies the desired normalization method. For the approach proposed in this work, however, mostly the function *fft* and *fft2* are used. Intuitively, they calculate the FFT of 1D or 2D array respectively. PyTorch also implements the *fftshift* function, shifting the calculated FFT to zero intensity as was explained in section 2.1.3.
Most importantly, PyTorch enables the usage of GPU, allowing some great speed up for arrays of bigger size with a running time of $\mathcal{O}(n \log n)$. When used on GPU the NVIDIA's cuFFT library is invoked. Their C++ implementation of the FFT results in the abovementioned running time due to simultaneous executions of transforms. It also enables the execution of transforms on multiple GPUs. PyTorch, however, does not yet have an implementation facilitating multiple GPUs management for *fft* library.

The bottleneck for the application of 3D FFT of density structure is the resolution of the data. Consequently, the memory space a 3D structure requires is $\mathcal{O}(n^3)$. The scale of high resolution, we tried to achieve in this work, exceeds the commonly used simulations of size 512^3. This particular size is not a coincidence, but a typical size of the 3D matrix able to fit into the memory of the available GPU machine. This is where the proposed approach gains its power: by splitting up the FFT calculation into 1D arrays or 1D + 2D arrays, the required space is vastly reduced. An interplay of GPU and CPU usage is essential in this step, as the first offer fast computation and the second typically have more RAM available. The idea is to accelerate the main matrix with the calculated 3D FFT in the CPU memory and send the array to be FTed to the GPU. After the calculation is done and the result is saved in the main matrix the space on the GPU is freed up, such that the next array can be sent for the computation. Allocating only an array on the GPU and reusing it is the factor allowing us to increase the resolution of the simulated structure. Moreover, the 2D slices, naturally use $\mathcal{O}(n^2)$ space, and by breaking their computation down into all together 3 consecutive 1D FFTs the required space can be further scaled down to $\mathcal{O}(n)$.
Reducing the space comes at a cost, and computing 3D FFT as the series of 1D in turn slows the computation time. To improve it, it is possible to run the 1D computations on multiple GPUs, if available. However, in this work, this was not implemented.
Another source for the speed increasement, beneficial for the PyTorch FFT on GPU is to use an input of size of a multiple of $2^a \times 3^b \times 5^c \times 7^d$, where $a, b, c, d \in \mathbb{N}$ as recommended by the Cuda toolkit developers. Following this recommendation, we used sample sizes ($81 = 3^4$, $125 = 5^3$, $243 = 3^5$, $343 = 7^3$, $441 = 3^2 \cdot 7^2$, $625 = 5^4$, $729 = 3^6$, $875 = 5^3 \cdot 7$, $945 = 3^3 \cdot 5 \cdot 7$) as can be seen in subsection 4.1.2. We have not used any powers of 2, because the design of the box required it to have an odd number of points (subsection 3.1.5).

3.1.4 Benchmarking 3D FFT

To show the proposed approach works sufficiently, two types of benchmarking are considered. The first one is to compare it against PyTorch's *fftn*, which is only possible for simulations of size up to about 512^3. We will work with two data types: single and double precision floating points. Because the calculation of FFT takes place in the space of complex numbers, the exact data types used are 'complex64' and 'complex128'. The relative error used to compare the differences is given by formula

$$\frac{|d_1 - d_2|}{\frac{1}{2}(|d_1| + |d_2|)},$$

where d_1 and d_2 represent the calculated FFTs with the respective methods calculated point-wise.
This deviates from classical formulations of the relative error, where the denominator is the reference value or the true value of the FFT of the structure in our case. However, in this example, there is no true reference as we cannot assume that any version yields a calculation of a 3D FFT without a numerical deviation, caused by imprecise floating-point operations. Hence, we need another normalization constant. Moreover, the differences inside the matrix itself have huge orders of magnitude around $1e15$, for example at zero frequency. Following that, normalizing by the arithmetic mean of both FFTs makes the most sense. Additionally, before calculating the mean, we calculate the absolute value of the calculated FFTs. This is because for many points the magnitude might be the same, but with different signs, resulting in zero normalization constant and division by zero. A problem might arise, because as was shown the box has many points with an electron density of zero, such that the FFT at these points is also near zero.
The calculated relative error refers to each point in the calculated transforms. Because these points correlate with each other, only the maximum value of the whole relative error matrix is studied here in detail.

After it is established, that the approach proposed in this thesis adequately represents the process of the 3D FFT, we want to also show it is applicable in the case of the scattering data. With this purpose, we simulate two kinds of shapes in a 3D space, FFT them within the proposed FFT framework and compare them with the analytical solutions for these shapes. This is described in detail in the following sections.

3.1.5 Creating 3D simulation box

Before the shapes are created we need to establish an environment for them. A simulation box is a 3D cube representing a part of a material system in real space. The user must define the length of the simulation box in nanometers and the number of points to divide each axis by. This number must be odd, and it controls how fine the simulation will be. Because both the gridding and the size of the simulation can be specified, an optionally small or big shape can be placed in the box and its real size is then a matter of the scaling factor. A threshold for volume fraction can be set up to control the fullness of the box and is set by default to fill 1% of the whole box's volume with shapes. With these arguments, the symmetric box in the Cartesian coordinate system is created and centered around the origin. The smallest unit inside the box is called a voxel and itself is a cube with the dimensions of the simulation box length divided by the number of points on the grid given

by the user minus one. As follows, all the neighboring points on the grid of the same axis have the same distance from each other. The number of points in the simulation must be odd such that the number of voxels in it is an integer.
Next, the scattering angle Q is calculated for each voxel. First, it is calculated for each of the axes q_i, where $i \in x, y, z$. The scattering angle is calculated for each voxel of the simulation box at its edges. This corresponds to a vector $q_i = \{-\frac{\pi}{d}, \ldots, \frac{\pi}{d}\}$ divided evenly in the same number of points as the simulation grid itself and d is the spacing between the voxel centers. Because the simulation is symmetric all three scattering vectors are the same. In the next step, the respective scattering angles of the axes q_i are expanded onto the 3D space and the total scattering angle is calculated as $Q = \sqrt{q_x^2 + q_y^2 + q_z^2}$.
Now, after the box is set up arbitrary shapes can be placed inside to represent the electron density of the simulated structure. The idea is to set the density values inside the shapes to 1, while the box itself has a density of 0.

3.1.6 Creating spheres

A sphere is given by its radius and center in 3D space. To speed up the creation of a sphere this operation is performed slice-wise with respect to $x-$axis. Then the mask for a 2D circle is created for each slice with an equation $(y - y_0)^2 + (z - z_0)^2 < r^2$, where y and z are the points on the 2D plane. Note, that we consider a plane spanned by vectors Y and Z since the slicing is performed along the $x-$axis. The point (y_0, z_0) is the center of the circle at the current slice, and because a sphere is a rotation-invariant shape, this point is always the same with respect to Y- and Z-axes. Finally, the radius of a sphere is given by r, and it is the only value changing for each slice.
To figure out the radius at each slice it suffices to know the distance from the current slice to the central one. Each sphere is created iteratively beginning from the central slice and moving towards its poles, filling both sides at a time, since a sphere is symmetric. This way, by knowing the distance within the grid points of $x-$axis, which is defined together with a simulation box, and a current iterator number, the distance to the central slice can be calculated by multiplying the two. The radius at the current slice then can be obtained using Pythagoras Theorem: the triangle is given by 3 points as shown on Figure 3.3. Point O is the center of a sphere, point C is the center of a circle, which is placed at a distance d from the center, and point S is the point on the surface of a sphere at a current slice. The distance to point S from O is the radius r. The triangle $\triangle OCS$ is right-angled at C because each circle is created onto $Y - Z-$plane, which is perpendicular to $x-$axis. Hence, the radius at slice $r_s = \sqrt{r^2 - d^2}$.
A sphere must not necessarily slice in any direction and can be placed at

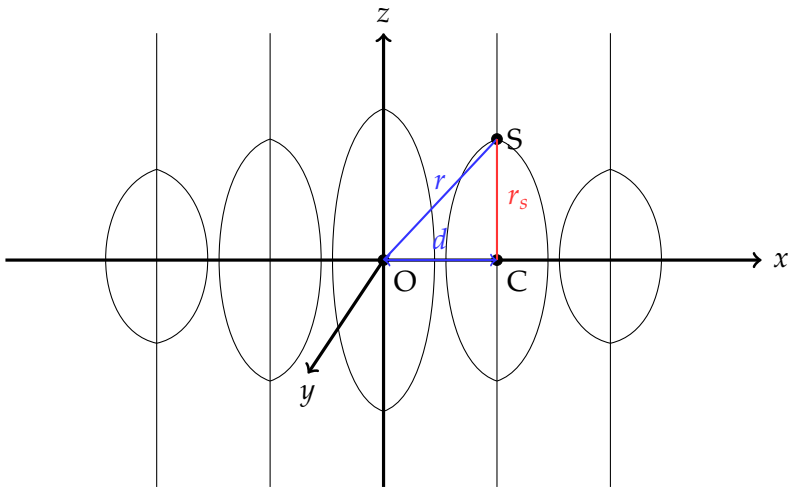


Figure 3.3: Schematic representation of a sphere as slices along $x-$axis. The scaling of spacing between slices is not true and only aims to explain the procedure. A circle mask onto $Y-Z$-plane is created for each point on $x-$axis until the radius in the direction of $x-$axis is exhausted.

the origin of the Cartesian coordinate system and on Figure 3.3 it is only depicted so for simplification, the center can be moved to any point within the simulation unit box, and it will not affect the generation of the radius, except for the calculation of a center.

Sampling variables to define a sphere

The main purpose of creating shapes in a 3D unit box is to perform the 3D FFT of this box in the following section to simulate a SAXS experiment. To verify this is representative of the real experiment the simulated scattering patterns will be compared to the analytical solution of existing shapes. To make both settings comparable, the simulation unit box should contain multiple simulated shapes and not just a single sphere. Indeed, a material sample is not consisting of just one shape example, and more importantly, it would not be representative of the polydispersity present in some materials. In pursuit of that, the simulation box will be filled with similar spheres until a threshold of a volume fraction is reached. The only two variables that can vary are the radius and center. The radius for the very first sphere is defined as one drawn from the Normal distribution $p(r) \sim \mathcal{N}(0.5\% \cdot s, 2\% \cdot s)$, where s is the size of the simulation box. All the following spheres are also drawn from the Normal distribution with the mean around the radius of the first sphere and a standard deviation of 0.1 or custom. The radius is drawn first, and then the center is sampled randomly within the boundaries of the simulation box minus the radius to prevent any placed shape from crossing the edge of the cylinder box. For the cases where shapes overlap, density is still clamped to 1, as it is more representative of the real-world

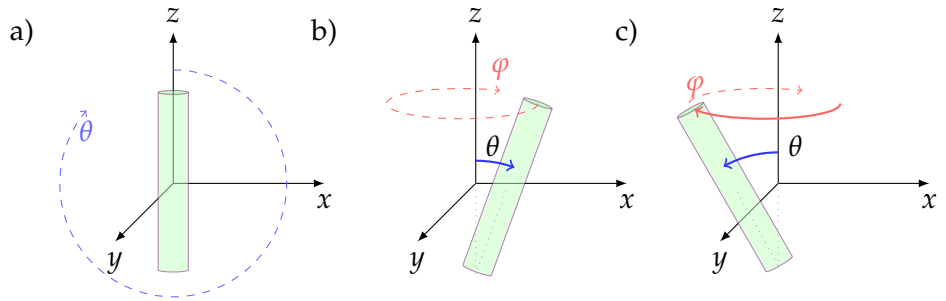


Figure 3.4: Geometry of cylinder rotations in 3D space. a) shows the cylinder that is not rotated in any direction. The first rotation to execute is θ, which turns the cylinder inside the plane parallel to $X-Z-$plane. b) shows the cylinder that has been rotated by some θ. Next, the $\varphi-$rotation will be executed, which takes place in the plane parallel to $X-Y-$plane. c) shows a cylinder that has been rotated by both θ and φ.

setting. Optionally, the user can specify the mode where placed spheres are not overlapping. This, however, implies additional checks: after each sphere is created it is checked if a minimal cube containing the sphere overlaps with the main simulation box.

3.1.7 Creating cylinders

A cylinder represents a more complex shape compared to a sphere. Apart from two obvious variables the length and the radius, different rotations describe its geometry in the 3D space as shown on Figure 3.4 To begin with, we introduce the rotation of the cylinder axis in the plane parallel to $X-Z-$plane. It is called θ in the following and can take on values between 0° and 360°. The cylinder with a $\theta-$rotation of 0° is parallel to $z-$axis. The second rotation then turns the cylinder axis about the $z-$axis, it is called $\varphi-$rotation. When $\varphi = 0^\circ$ the cylinder points in the direction of the $x-axis$. If $\theta = 0^\circ$, $\varphi-$rotation becomes invariant. A third possible rotation would be around the cylinder axis, but here only a symmetrical cylinder will be considered, meaning that rotation around the axis is invariant. The rotations must be implemented in this particular order to be consistent with existing methods. Cylinders will not be created as slices, instead, the two points at the cylinder bases will need to be specified, and then for all voxels inside the simulation box, it will be checked if they are lying within the cylinder of a given radius.

Sampling variables to define a cylinder.
To define a cylinder we need the essential parameters: radius and length. The initial values are both drawn from a Normal distribution:
$p(r) \sim \mathcal{N}(0.5\% \cdot s, 2\% \cdot s)$ and $p(l) \sim \mathcal{N}(10\% \cdot s, 10\% \cdot s)$ respectively, where s is the size of the box. This means that the diameter of the cylinder is around

1% of the box, the same as the sphere. The length is sampled around 10% of the box size. Both initial variables can deviate from the named means to improve the flexibility of the model to simulate diverse settings: the radius can deviate by 2% and length by 10% of the box size.
Again, to further obtain a realistic scattering pattern we need to place multiple shapes in the box. With this purpose, the process of cylinder creation is continued until the desired volume fraction of the box is filled. To be consistent with real-world nanostructures and polydispersity in them, all the following cylinders should have dimensions correlating with the initial ones. Consequently, each next radius and length are drawn from the Normal distribution with a mean around the dimensions of the initial cylinder and a standard deviation of 0.1 and 0.15 for radius and length respectively, until defined in a custom way.
Simulating the $\theta-$rotation defines the main fiber axis of the material in question. Although a cylinder can be rotated in all 360° in θ, it makes the most practical use to rotate it such that the cylinder is either parallel or perpendicular to the $z-$axis. In the default setting, it is parallel with the $z-$axis. After the initial θ is set all the following cylinders should be rotated at similar angles. The exact value is also drawn from the Normal distribution with the mean around default θ of 0° of the initial cylinder and a standard deviation of 5° or custom. It is not advised to set it greater than 10°, because the fiber simulation becomes unrealistic.
The final $\varphi-$rotation of the cylinder is picked at random for each simulated cylinder and takes on values in the interval of $0° - 360°$.
The position for a center is sampled uniformly from the simulation box boundaries and serves as a point on one of the cylinder bases. The check for crossing box edges will be performed in the next step.

Finding the end of a cylinder.
The point at the center of one of the cylinder bases was sampled in the previous step. Now, given this point, the length and the rotations of the cylinder, and the center of the second cylinder base can be obtained. Figure 3.5 shows the geometry of the $\theta-$rotation. The point O is the sampled center on one base and is given by $O : (O_x, O_y, O_z)$. The $\theta-$ rotation is defined as one in the $X - Z-$plane and, hence, the center of the second base has the same coordinates with respect to $y-$axis and only coordinates in x and z change. As depicted, first a triangle $\triangle OAC$ will be considered. Point O is the center of the sampled base. Point C is the center of the second cylinder base after rotation θ was performed. The intersection of the line parallel to $z-$axis drawn through the point O perpendicular to the line through the point C is marked as point A. In the right-angular triangle $\triangle OAC$ angle $\angle O = \theta$ and side $OC = h$, the sampled height of the cylinder. The rest two sides of the triangle can be retrieved with the trigonometric ratios:

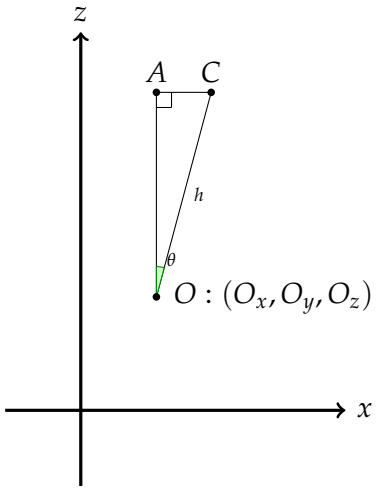


Figure 3.5: The effect of the $\theta-$rotation on position of second cylinder base. It is shifted in $x-$ and $z-$ coordinates.

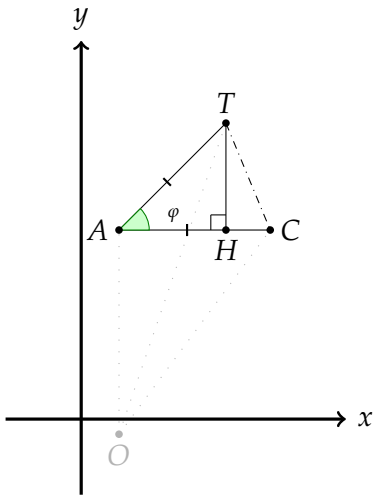


Figure 3.6: The effect of the $\varphi-$rotation on position of second cylinder base. It is shifted in $x-$ and $y-$ coordinates after it was shifted on $\theta-$rotation.

$OA = h \cdot cos(\theta), AC = h \cdot sin(\theta)$. Subsequently, the coordinates of two points are: $A : (O_x, O_y, O_z + h \cdot cos(\theta))$ and $C : (O_x + h \cdot sin(\theta), O_y, O_z + h \cdot cos(\theta))$.

Now, the $\varphi-$rotation needs to be accounted for. This rotation only changes the coordinates of point C with respect to the $x-$ and $y-$ axes. Figure 3.6 shows how the center of the second base is shifted by angle φ. In this figure points A and C have the same meaning as before and point T is the center of the second cylinder base after both rotations took place. To calculate the coordinates of this point drawing a line perpendicular to AC from T is helpful. On Figure 3.6 the intersection of the two is point H, and it suffices to calculate the distances AH and TH to know the coordinates of point T. The triangle $\triangle ATH$ is right-angled at $\angle H$ and the angle $\angle A = \varphi$, distance AT is known and equals to the distance of AC calculated in the last step. Using the trigonometric ratios then $AH = h \cdot sin(\theta) \cdot cos(\varphi)$ and $HT = h \cdot sin(\theta) \cdot sin(\varphi)$. The point T has coordinates: $T : (O_x + h \cdot sin(\theta) \cdot cos(\varphi), O_y + h \cdot sin(\theta) \cdot sin(\varphi), O_z + h \cdot cos(\theta))$

Finding what voxels lie inside a cylinder.
After the coordinate for a second cylinder base is computed we can proceed and place the whole cylinder in the box. To test if a given voxel lies inside a defined cylinder two geometrical observations will be used. First, we need to make sure that the voxel lies between two planes given by two bases of the cylinder. To ensure that it will be checked if a voxel's position is greater than one end and smaller than the other with respect to the cylinder direction.

The cylinder ends are defined with vectors $\vec{o}$ and $\vec{t}$ and any voxel to be checked is defined by a vector $\vec{q}$, and mathematically it can be expressed as follows:

$$(\vec{q} - \vec{o}) \cdot (\vec{t} - \vec{o}) \geq 0$$

$$(\vec{q} - \vec{t}) \cdot (\vec{t} - \vec{o}) \leq 0.$$

The second step is to ensure that a given voxel also lies within a curved surface of the cylinder, by checking if the distance from the voxel to the cylinder axis is smaller than the radius:

$$\frac{|(\vec{q} - \vec{o}) \times (\vec{t} - \vec{o})|}{|\vec{t} - \vec{o}|} \leq r.$$

When all three inequalities are satisfied a voxel lies inside a defined cylinder. This test needs to be performed on every voxel in the simulation box, which does not make it very practical: while a check for a single voxel has a constant running time for the whole 3D box it becomes $\mathcal{O}(n^3)$.
To reduce this running time the check will be performed on the subset of the certain voxels forming a rectangular parallelepiped containing the cylinder, with the vertices coordinates depending on both cylinder bases. Because the $\theta-$rotation is implemented as a small one, it is known that the typically longer cylinder axis is aligned near-vertically. This implies that the parallelepiped coordinates for $x-$ and $y-$ axis can be specified as the base position $\pm$ radius value. The radius value as is, however, will not suffice for the $\theta-$rotated cylinder. The value that should be added or subtracted is explained in Figure 3.7. Point O is the center of one cylinder base and point L is the outer point of the cylinder base projection onto $XZ-$ or $YZ-$plane after rotation by θ. The ellipse radius in the direction of $x-$ or $y-$axis is point K. The radius value is invariant for $x-$ and $y-$axes because the cylinder is symmetric and not elliptic. The formed triangle $\triangle OKL$ is rectangular at $\angle L$, as the cylinder axis is perpendicular to its bases, and, hence, $OK = \frac{OL}{cos\theta}$.
To calculate the exact coordinates of the parallelepiped with respect to $z-$axis the explanation from section 3.1.7 will be used. The height of the parallelepiped is the same as point A. Next, to account for the $\theta-$rotation the capping into the $z-$direction needs to be considered. The simple observation as on Figure 3.8 helps figure it out. The point O is again used as a base of the cylinder and point R is the intersection of the cylinder axis with the point on the cylinder base. The distance OR now represents the radius of the not yet rotated cylinder and is not to be confused with the previous image. Point T specifies the cylinder base projection at rotation θ on angle $\angle O$ and should be considered the end of the capping. The distance OT, naturally, also equals the radius of the base. To simplify the calculation a point F will be considered to be the end of the capping. It is constructed as an intersection of the ray OT with a line drawn through the point R, perpendicular to

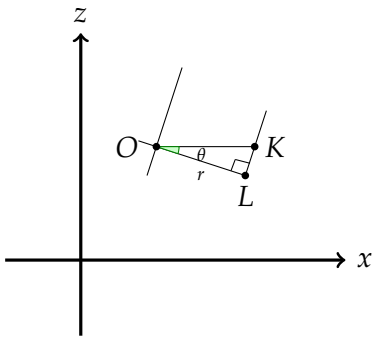


Figure 3.7: Radius of tilted cylinder

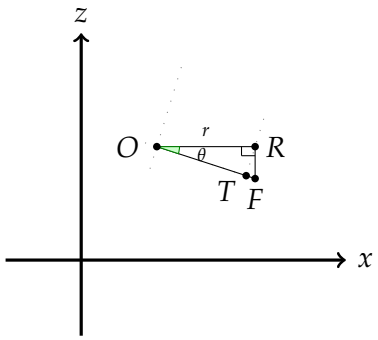


Figure 3.8: Cap of tilted cylinder

OR. Then in the rectangular triangle $\triangle ORF$, the trigonometric ratio is used to calculate the $RF = RO \cdot \tan\theta$. This capping needs to be added on each side of the parallelepiped.
Finally, for the determined parallelepiped additional two pixels will be added on each edge to ensure, the subset of voxels captures the whole cylinder.

Now, after it was established what voxels of the simulation box lie inside the cylinder, it will be checked if a cylinder intersects the boundaries of the box. For that, it is simply tested, if any voxel at the edge of the simulation box equals one. In such a case, it is assumed the periodic boundary conditions are not satisfied, and the cylinder is discarded.

3.1.8 FT of simulated structure

The simulated 3D electron densities in form of spheres and cylinders are transformed with the proposed FFT framework in the next step. There are two modes available: a full 3D FFT of the structure and the FFT of only a central slice imitating the Ewald sphere and hence the pipeline of the SAXS experiment. The central slice of FFT through the $x-$axis is the default, but the central slice in any direction can be obtained to simulate the effect of different orientations of the sample with respect to the beam and detector. In the next steps, the calculated transform is compared against the analytical solutions. Before that, it is necessary to apply the *sinc* function to the voxels of the calculated FFT.
To smooth the density decay at the discrete boundaries of the voxels it is often beneficial to convolute the calculated FFT with the width of the distance between the sampling points (Pauw, 2009). Mathematically *sinc* function is represented as

$$\frac{\sin(\vec{q}_i \cdot \frac{d}{2})}{\vec{q}_i \cdot \frac{d}{2}},$$

where $\vec{q}_i$ is the vector of scattering angles in the respective direction x, y or z and $d/2$ is the radius of the voxel. Convolution in the Fourier Space is a multiplication. Following that, the *sinc'*ed simulated intensity equals the calculated 3D FFT multiplied by the respective *sinc* values.

3.1.9 Rebinning the simulation into 2D/1D scattering pattern

While calculating a 3D FFT for high-resolution structures is beneficial to capture the electron density of smaller particles and expand the values of the scattering vector $\vec{q}$, there are no further analysis pipelines able to deal with such great quantities of data yet. To scale down its dimensionality, but preserve the most important contents - the 3D FFT of a wider range of scattering angles, there is a rebinning procedure taking place. By connecting these two steps: high-resolution 3D FFT and effective rebinning function, we aim to effectively analyze the electron density of structures up to 4000 pixels in size on the available machines.
Furthermore, as was communicated earlier, there is still a need at this point to show, that the FFT framework truthfully reflects the scattering experiment. To reduce the dimensionality of the calculated 3D FFT and compare it with the analytical solution for scattering, as mentioned above, only the central slice will be used. This will be masked by a sphere because the scattering angles in the corners of the computed FFT box are under-represented.
After masking an azimuthal averaging procedure is performed for the remaining scattering angles. The procedure aims to redefine the space of scattering angles as intervals with respective average values for both scattering angle Q and the intensity I. It additionally estimates an uncertainty of both variables. There is no golden standard for the uncertainty estimates, for this reason, different estimates are calculated and the biggest of them is used for further analysis.
The number of bins is user-defined, and it determines the intervals for scattering angles Q space. For each such interval, average values and uncertainties are calculated with schema given in the Table 3.1. The constants I_{Emin} and Q_{Emin} are user-defined, but default-set to 1% of scattering intensity/angle of the respective bin.

To further compare the simulated spheres and their scattering pattern with the analytical solution a rebinning into 1D space is performed, since this shape is symmetric and has no orientation. Cylinders, however, have an orientation along the fiber axis, such that their 2D scattering pattern preserves more information than a 1D curve.

I Estimate	Calculation	Q Estimate	Calculation
I_μ	$\frac{\sum_{i=0}^{n} I_i}{n}$	Q_μ	$\frac{\sum_{i=0}^{n} Q_i}{n}$
I_{STD}	$\sqrt{\frac{\sum_{i=0}^{n}(I_\mu - I_i)^2}{n-1}}$	Q_{STD}	$\sqrt{\frac{\sum_{i=0}^{n}(Q_\mu - Q_i)^2}{n-1}}$
I_{SEM}	$\frac{I_{STD}}{\sqrt{n-1}}$	Q_{SEM}	$\frac{I_{STD}}{\sqrt{n-1}}$
I_{SEM}	$\frac{I_{STD}}{\sqrt{n-1}}$	Q_{SEM}	$\frac{I_{STD}}{\sqrt{n-1}}$
I_{error}	$\frac{\sum_{i=0}^{n} I_i^2}{n}$	Q_{error}	$\frac{\sum_{i=0}^{n} Q_i^2}{n}$
I_σ	$max(I_{SEM}, I_{error}, I_\mu \cdot I_{Emin})$	Q_σ	$max(Q_{SEM}, Q_{error}, Q_\mu \cdot Q_{Emin})$

Table 3.1: Uncertainty estimates formulations for the rebinning procedure.

3.1.10 Comparing simulated scattering pattern to analytical solutions

To finally verify, that the proposed approach adequately represents the process of scattering, the analytical solutions with respective parameters will be created. The analytical solutions are modeled with SasView[1] sasmodels package in Python. SasView has broad functionality. Here its ability to model the inverse space for a different set of shapes and parameters is used. For each created 3D simulation a SasModels analog is also created. It is set to the same scattering angle intervals as in the rebinned simulation and reuses other parameters defining the shape of the structure. These parameters are given explicitly in Table 3.2.

Although these parameters are thought to be copied from the parameters used to create a 3D simulation, as seen in the table they sometimes differ:

1. Radius/length used in the creation of 3D simulation is divided by 10. The reason for this is that the 3D simulation box is created in the scale of nm. SasView uses the Ångstrom scale instead. The same applies to the scattering angle Q. Because in the 3D simulation the units are nm^{-1}, we divide by 10 to go from units of 1/A to 1/nm.
2. Polydispersity value for length and radius are not set as standard deviation σ but additionally normalized by the respective mean μ. The definition of the polydispersity is set this way by the SasView developers (SasView 5.0.5 User Documentation., a).

[1] http://www.sasview.org/

Parameter in SasView	Value from simulation
scattering angle Q	$Q/10$
radius	$r_\mu \cdot 10$
radius polydispersity	$\frac{r_\sigma}{r_\mu}$
radius polydispersity type	Gaussian
radius polydispersity n points	35
length	$l_\mu \cdot 10$
length polydispersity	$\frac{l_\sigma}{l_\mu}$
length polydispersity type	Gaussian
length polydispersity n points	35
$\theta_{SasView}$	90
$\varphi_{SasView}$	0
$\varphi_{SasView}$ polydispersity	θ_{3D}
$\varphi_{SasView}$ polydispersity type	Gaussian
$\varphi_{SasView}$ polydispersity n points	35
SLD	1
SLD solvent	0
background	0

Table 3.2: Parameters for creating analytical scattering patterns with SasView

3. $\theta_{SasView}$ angle is set to 90°. In the SasView setting a θ angle of zero orients the cylinder's main axis parallel to the $z-$axis. The X-Ray beam is, however, also parallel to it, such that the Fourier transformation from the perspective of $z-$axis is recorded (SasView 5.0.5 User Documentation., b). This way the cylinder under $\theta-$rotation of 0° would scatter as a sphere in SasView. As was mentioned in subsection 3.1.8, the default for the 3D simulation is a central slice from the perspective of the $x-$axis. To account for that difference angle $\theta_{SasView}$ is set this way in the SasView simulation.

4. Similar to the previous point, because the beam in SasView has another orientation of Fourier transformation, compared to the FFT framework for 3D simulations, the meaning of an angle θ_{3D} in a 3D simulation is equivalent to the rotation of the cylinder about the $z-$axis in SasView, which is given by rotation φ in the documentation. Following that, angle $\varphi_{SasView}$ should be equal to angle θ_{3D}. However, to account for

uniform rotation φ_{3D} in the 3D simulation, angle $\varphi_{SasView}$ needs to be distributed uniformly too. Importantly, the angular polydispersity is defined in degrees and does not need any further manipulations.

5. The polydispersity type used for all the parameters is set to Gaussian, except for $phi_{SasView}$, as was explained above. Such a definition agrees with the implementation of the 3D simulation.

6. The n points parameter is set to 35 for all parameters described as polydispersed. This parameter regulates the number of points used to compute the numerical distribution average and is set to a value, recommended by the developers.

7. SLD is an abbreviation for scattering length density, describing the scattering power of the material. In an experimental setting, it is calculated for each material with special calculators. In our case, there is no certain material, so we decided to set it to an adequate constant.

8. Background value represents the average scattering intensity of the solvent. In the experimental setting, the material in question is not present alone, but in some solution or may be even taped to not get lost or have some other type of background signal present. In the simulation setting, we implement only the material itself, with no additional background. Accordingly, both the background and scattering length density of the solvent are equal to zero.

SasView enables usage of the GPU for creating the analytical models with PyOpenCL (Kloeckner et al., 2022), which greatly speeds up the process, especially for 2D scattering patterns of the cylinders.

3.1.11 Benchmarking 3D FFTed simulation against SasView model

After the SasView simulation was initialized two scattering curves in 1D or scattering patterns in 2D must be compared. Another parameter that can be adapted in the SasView model is *scale*, which is a multiplier of a computed intensity value. To fit the SasView simulation to the one created by FFTing a 3D simulation a simple least square fit with the Levenberg-Marquardt algorithm is used to optimize this scale parameter. For the 2D scattering pattern, an additional mask is applied on both 3D and SasView simulations. It consists of a circular mask neglecting the under-represented angles in the corners of the simulation again, but also a spherical mask in the center. This is required because the differences at zero frequency might explode exponentially. To ultimately quantitatively compare the two simulations a χ^2 error is computed, where the uncertainties, calculated in the rebinning

step are accounted for:

$$\chi^2 = \frac{1}{n-1} \cdot \sum_{i=0}^{n} \frac{(Sas - S)^2}{\sigma^2},$$

where Sas and S represents the intensity calculated with SasView and a 3D FFTed simulations repsectively. σ is one of the uncertainty estimates calculated while rebinning the 3D simulated FFT and n is the number of points the scattering intensity is computed for.
Additionally, the same measure of the relative error as in subsection 3.1.4 is applied.
Created 3D shapes simulations, results of 3D FFT, rebinned scattering curves and scattering patterns, as well as different benchmarking methods, are demonstrated in section 4.1.

3.2 Neural Network to infer the shape parameters of a scattering pattern

3.2.1 Creating training data for machine learning problem

In the machine learning part, we want to learn the set of parameters describing a shape given a 1D scattering pattern with neural networks (NN). For a typical NN application, there is usually a real-world training set available used to learn its weights. For the type of inverse problems, this is not always necessary.
As such, in our application we can create training inputs as different shapes and parameters, describing these shapes. Then, we utilize the forward model in order to create the labels for the learning problem.
Unfortunately, we are still limited by the available RAM, as the 3D FFT calculated in the previous part or even the 2D central part requires much more memory as shown in Figure 4.10. This would require $\mathcal{O}(n^3)$ and $\mathcal{O}(n^2)$ space respectively for just one sample. For the present problem, we would require thousands of samples.
With this restriction, we can only analyze 1D scattering curves.
We could use the implementation of the 3D shapes, FFT them with the proposed framework, rebin into 1D curves and then use them as training data. However, as shown in Figure 4.8 that the 3D Fourier transformed simulations are comparable with the SasView analytical solutions, we decided on using those. In SasView it is possible to define a set of shapes and their parameters and calculate the scattering patterns in 1D with the analytical solution.
For our training set, we decided to start with three shapes: spheres, hard spheres and cylinders. For each of the shapes, we created 5000 training samples. Each scattering curve would have a resolution of 512 points. The scale

	sphere	hard sphere	cylinder
radius	$\sim \mathcal{N}(2, 0.1^2)$ nm	$\sim \mathcal{N}(2, 0.1^2)$ nm	$\sim \mathcal{N}(2, 0.1^2)$ nm
radius polydispersity	$\sim \mathcal{N}(0.05, 0.03^2)$	$\sim \mathcal{N}(0.05, 0.03^2)$	$\sim \mathcal{N}(0.05, 0.03^2)$
radius polydispersity type	Gaussian	Gaussian	Gaussian
radius polydispersity n points	35	35	35
radius effective		= sampled radius	
volume fraction		$\in \{0, 0.05, 0.1, 0.15, 0.2, 0.25, 0.3\}$	
length			$\sim \mathcal{N}(10, 10)$
length polydispersity			$\sim \mathcal{N}(0.08, 0.05^2)$
length polydispersity type			Gaussian
length polydispersity n points			35
SLD	1	1	1
SLD solvent	0	0	0
background	0	0	0
scale	1	1	1

Table 3.3: Parameter settings used to create training data for machine learning

of the simulation in the real space is set to 250 nm, to stay consistent with the 3D simulations done in the previous part. The distance between the neighboring voxels is then equal to $\frac{1}{512} \sim 0.00195$ nm. The space of the scattering angle is consequently defined as 512 evenly spaced points on the interval from $-\frac{\pi}{0.00195}\text{nm}^{-1}$ to $\frac{\pi}{0.00195}\text{nm}^{-1}$. The parameters describing shapes are defined in terms of distributions and given in the Table 3.3. Simulations here are defined in such a way, that they are not limited to the nm units and can be adjusted by the scaling factor. However, internally SasView assumes the passed data is in Ånstrongs. We consequently conserved them to agree with that.

Most of the used parameters are similar to the Table 3.2 and will not be explained again. There are two new parameters for a hard sphere shape, however. The parameter "radius effective" represents the interaction radius of a particle. For instance, due to charges effective radius might be greater than the actual radius of the particle. Many such interactions exist that can adjust the effective radius compared to the real radius. "Volume fraction" just specifies the volume fraction occupied by spheres and regulates how densely the shapes are packed.
For the cylinders, the angular parameters are not specified, because in the

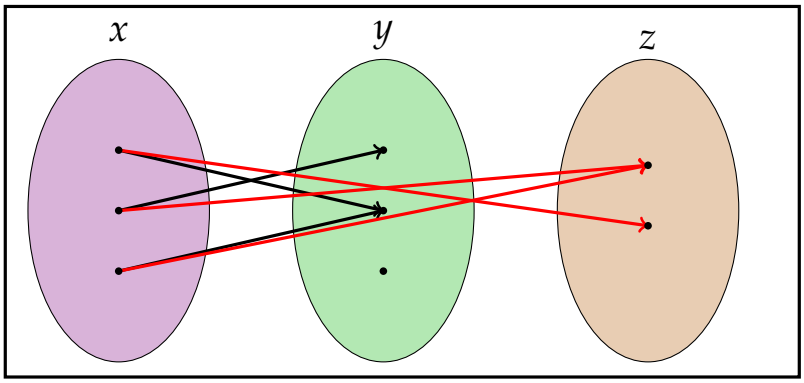


Figure 3.9: Restoration of a bijective mapping for a surjective problem via a latent space.

1D scattering curves those will not be reflected anyway.
Finally, we applied some noise to the simulations. To distinguish the noize function we tested different distributions and parameters until it looked reasonable to an experienced SAXS experimentalist. The chosen noize distribution is $\mathcal{N}(1, 0.1^2)$. After, the scattering intensity of each data point was multiplied by noize created for this data point.

3.2.2 Invertable Neural Networks

In our approach, we want to address the surjectivity of the inverse scattering problem of the scattering and generative modeling seems to be a good suggestion for it. For that, we compared predictions of a few invertible architectures, namely i-ResNet (Behrmann et al., 2019) and conditional RNVP (Ardizzone et al., 2019b), as well as different feed-forward NNs from the scattering curves to the parameters: $y \rightarrow x$. For most of the implementation, the FrEIA package was used, as it includes multiple different architectures applicable for the inverse problems (Ardizzone et al., 2018-2022).

Data preprocessing for training process

To represent the bijective mapping we sampled a latent dimension of z from the Normal multivariant distribution $\mathcal{N}(0, I_d)$, where the d is the feature space of the latent variable. Additionally, we padded both inputs space x and output space $[y, z]$, such that they obtain equal feature space.
In the cases where all three shapes were used together in the training data, we had a small complication. The inputs were both discrete and continuous variables: the shape classes and the parameter values describing these shapes, respectively. The actual parameter space is $\mathbb{R}^6$ describing the shape, radius and its polydispersity, length and its polydispersity (for the cylinders), the volume fraction (for hard spheres). The information concerning the class of the data is usually kept in one-hot-vectors. As such, the fea-

ture space defining shape parameter only was extended to three: the actual shape is marked with 1 and the false ones with 0. Furthermore, we did the same with the radius parameter, which also occurs in all three. The radius is set to 0 for indices of wrong shapes and to the real radius value for the correct shape. For instance, for a cylinder sample with a radius of 4 nm, the shapes one-hot-vector can be represented as $[0, 0, 1]$ and the radius vector as $[0, 0, 4]$. All the rest of the input features stay uncorrected.

For the training, the sampled data was randomly split into three categories "training", "validation", "testing" in a ratio $0.7 : 0.2 : 0.1$. For training with PyTorch the "training" split was used to optimize the weights through minimizing loss and the "validation" split to evaluate loss on the yet unseen data. Finally, the "test" split was used on the trained network weights and the parameters were compared with the sampled ones more precisely.
As was mentioned, the simulated parameters can be multiplied by a scaling factor to represent the different sizes of particles. To allow this flexibility the continuous inputs to the network were standardized by removing a mean and scaling to unit variance. When training the feed-forward network normalization of scattering curve values was applied.

NN Architectures

The outlook of all the architectures is given in the Table 3.4. It includes the information on the network type (inverse or simple feed-forward) and the building blocks inside the network.
In the feed-forward NNs, the inputs were the scattering curves, which match the purpose of convolutional NNs. In the inverse NNs, the inputs in the forward direction are shape parameters in $\mathbb{R}^{10}$, so the classic fully connected NN should be more appropriate as functions in the coupling blocks of RNVP or i-ResNet. Within the RNVP coupling blocks, there is an option to clamp multiplicative component that is meant to control the amplification of each level and prevents the gradients explosion.

Optimizing the learning process

In the solution proposed in the invertible neural network (INN) (Ardizzone et al., 2019b) loss of multiple functions is weighted and then accumulated. Because the NN is invertible, there are predicted outputs on each side of the network. For the forward direction, we have a pair of samples: labels of the analytical Fourier transform and the outputs of the NN. For the inverse direction, a pair is constituted by the sampled shape parameters and the output of the NN in the inverse direction.
The goal is to learn the forward and inverse Fourier transform. It can alternatively be interpreted as an intuition, that we want both samples in each of the two pairs to come from the same distributions. We conclude that the

Type	Building blocks	Details
Feed-forward architecture	fully connected (fc)	number of hidden layers and neurons are the hyperparameters
	convolution 1D with average pooling	convolution kernel size and number of hidden layers are the hyperparameters
	ResNet	residual network based on the above convolution blocks
Inverse architecture	RNVP	fc-coupling NN with permutations in-between; number of hidden layers is the hyperparameter
	i-ResNet	ResNet, based on fc-NN; number of hidden layers is the hyperparameter

Table 3.4: Overview of architectures applied in an attempt to solve the inverse scattering problem

MMD loss seems to be the right approach to optimization. More precisely, there are 4 losses calculated: for optimizing the learning problem:

- $\mathcal{L}_y(y_i, f(y_i))$ is a simple supervised loss of the forward problem f. Depending on the problem, can be cross-entropy or mean-squared error (MSE). In our case the labels y are the scattering curves, so we use the MSE
- $\mathcal{L}_z(p(y)p(z), p(y,z))$ is implemented in terms of MMD. In the previous step, we have already accounted for the labels y. With MMD we want to ensure that latent space z learns information not contained in y. For that, the predicted joint distribution of forward network output is compared with the product of the marginal distributions p_y and p_z.
- $\mathcal{L}_x(p(z), f^{-1}(y))$ is also implemented in terms of MMD and is responsible for correct predictions in the inverse direction. Two distributions are compared: the prior distribution of the sampled input parameters and the result of the backward run of the sampled scattering curves.
- (optional) reconstruction loss. It is said to speed up the convergence and is calculated from the sampled outputs of the forward direction of the INN(Ardizzone et al., 2019b). It is calculated as simple MSE loss between the network inputs (shape parameters) and the outputs of the forward direction of current NN with additional noize.

For the feed-forward network, the MMD of the network output and the sampled parameter values were compared, because the assumption that they come from the same distribution holds here too.

It is recommended to use Adam optimizer (Kingma and Ba, 2014) and ELU activation functions (Clevert et al., 2015) with the MMD loss. However, we also considered other activation functions.

MMD loss

Two losses in the implementation of the INN are calculated as MMD. To efficiently calculate its value, as was shown in subsection 2.2.1, it is necessary to choose the right kernel function. With that, another important hyperparameter in the learning process is established. Here we decided to work with the inverse multiquadric kernel of form $k(x, y) := \frac{1}{\sqrt{C^2+\|x-y\|_2^2}}$ (Schölkopf and Smola, 2002), where C is a constant. This type of kernel often works better than gaussian kernels, because it gives gradients even if distribution means are far from each other (Tolstikhin et al., 2019).

Chapter 4

Results

In this chapter, we first show the results of the 3D simulations with the consequent 3D FFT. Additionally, all the benchmarking results are demonstrated and explained. In the second part, the results of the machine learning application are presented.

4.1 3D shapes simulations and memory-efficient 3D FFT

4.1.1 3D FFT and comparison to SasView in low-resolution

Figure 4.1, 4.2, 4.3 show the simulated 3D density, the calculated 3D FFT and the comparison to the SasView simulation for spheres with 3.4 nm radius and simulation box of size 100 nm. The 3D FFTs are masked to a sphere, because, as was previously said, the scattering angles in the corners of the box are under-represented.

Analogically, the simulated electron density is shown for cylinders with a radius of 1.46 nm, length 21.4 nm and $\theta-$ rotation of 10° on Figure 4.4. The 3D FFT is shown from the perspective of each axis on its central slice on Figure 4.5. Comparison to SasView is depicted on Figure 4.6. Here is an additional circular mask placed right in the center. This area has the highest values of intensity and may negatively influence the procedure of fitting.
The resolution of these figures is quite low with the 243 points on the grid, such that the edges of the shapes can be seen on the density simulations.

In comparison to the SasView simulation, both figures show a relatively high χ^2 error of 93 for spheres and 2148 for cylinders. This is because of the low resolution of the simulations. However, to show the 3D density and FFT it was necessary to not use high-resolution figures, otherwise producing a figure was not feasible.

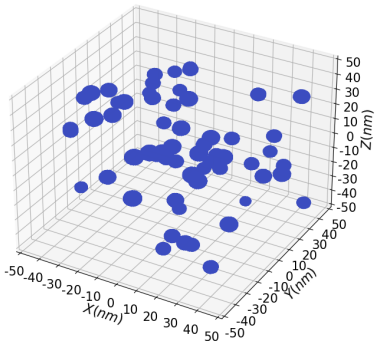


Figure 4.1: 3D electron density of simulated spheres of radius 3.4 nm in a box of size 100 nm.

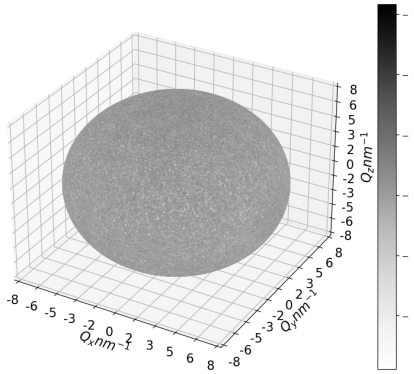


Figure 4.2: 3D FFT of the simulated spheres. The corners of the box are masked. The scale is logarithmic.

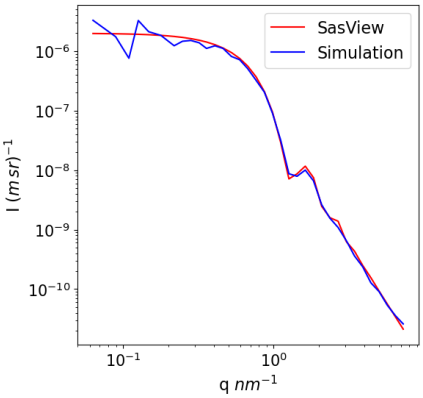


Figure 4.3: Rebinned FFTed 3D simulation compared with SasView simulation for the same scattering angles Q. χ^2 error is 93.

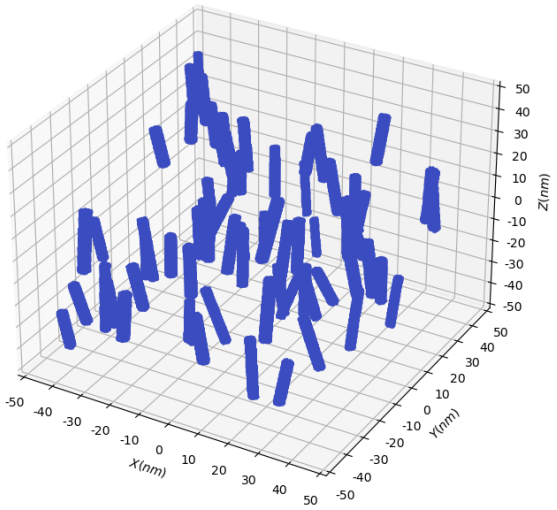


Figure 4.4: 3D electron density of simulated cylinders. Radius is 1.46 nm, length 21.4 nm rotation θ is 10°.

For the hard spheres such figures are not shown here. As was explained in the methods part, the volume fraction for them is set much higher, starting from 0.5 compared to the default volume fraction of 0.01 of normal spheres. Following that, the 3D figure of the simulated density is not representative as most of its space is colored. The representation of simulated hard spheres of the same radius 3.4 nm in the box of size 100 nm as 3D FFTed, rebinned scattering curves in the same resolution of 243 pixels can be studied on Figure 4.7. It gives an overview of the curve behavior with the growing

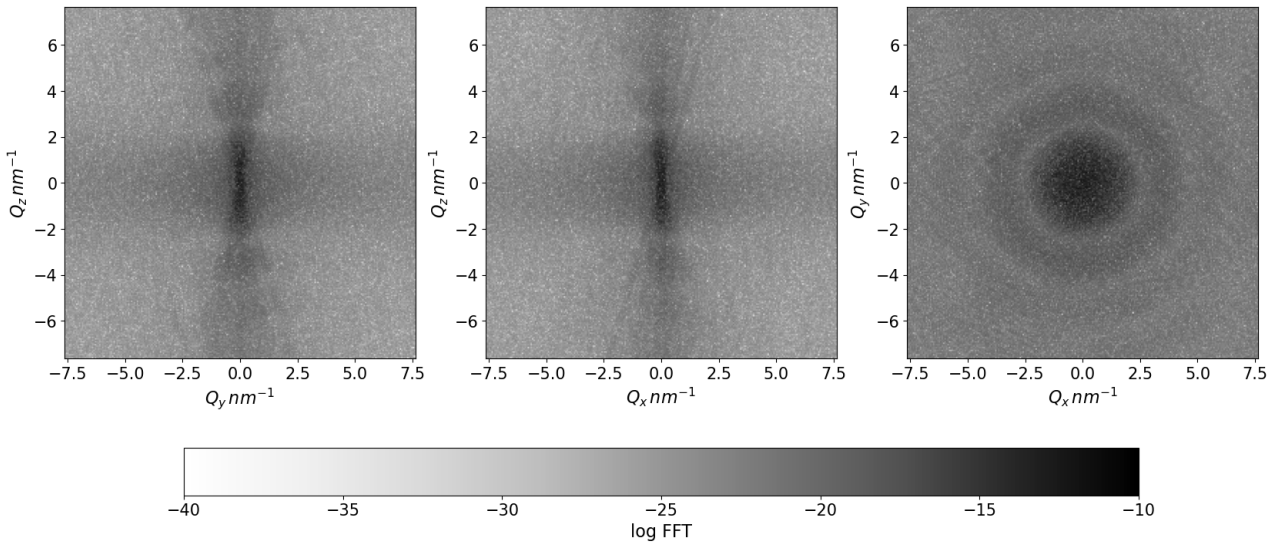


Figure 4.5: 3D FFT of simulatied cylinders. Each of the subfigures depicts the central slice of the 3D FFT in the respective dimensions. The scale is logarithmic.

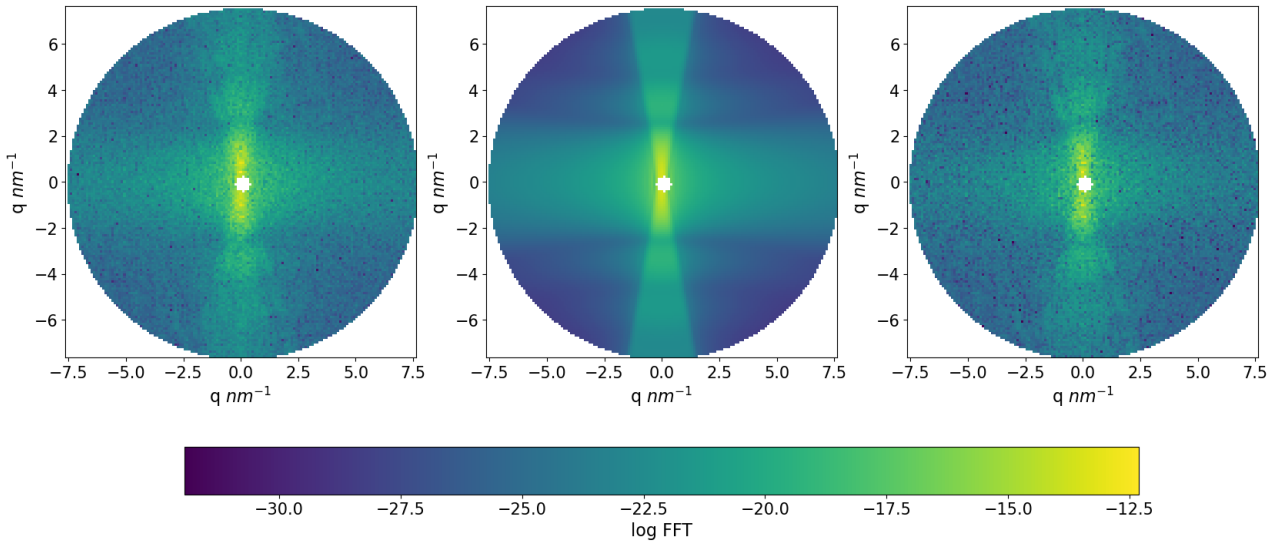


Figure 4.6: 3D simulation FFTed in the direction of $x-$axis and rebinned on the central slice is on the left. SasView simulation for the same parameters is in the middle. The logarithm of absolute difference is shown on the right. The χ^2 error is 2148.

volume fraction. The Table 4.1 also gives a reader an idea, about how many spheres get accepted or declined in the process. This is an example of just one run, but it still provides a good estimate of how many created shapes get rejected in the process.

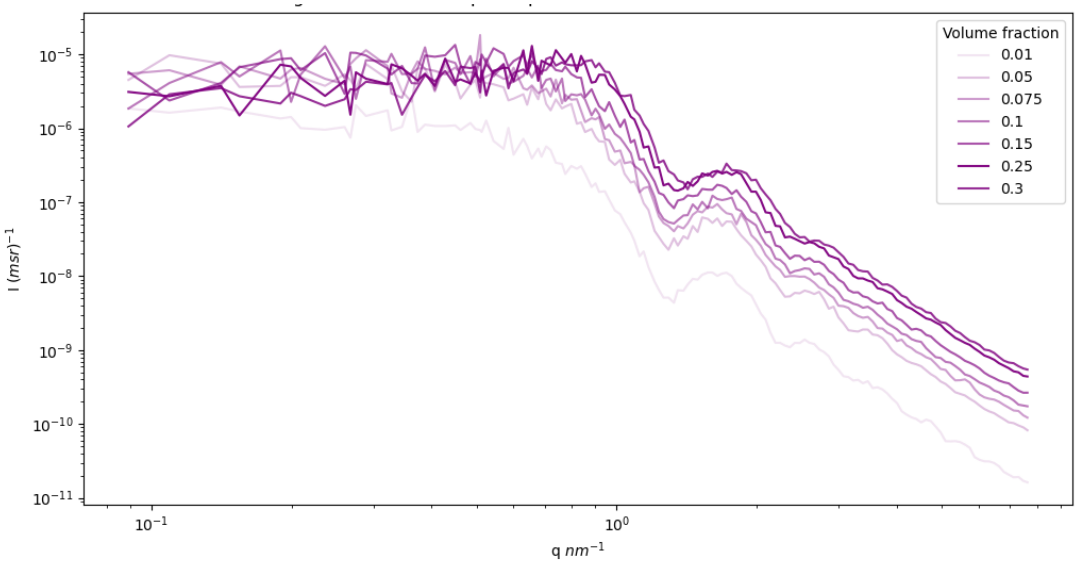


Figure 4.7: Scattering curves of hard spheres with radius of 3.4 nm in a box of 100 nm. The volume fraction differs. The intensity of the color grows together with the volume fraction.

Naturally, with the higher volume fraction more spheres are declined in the process, such that the creation of hard spheres becomes time-consuming. The 3D FFT of hard spheres looks similar to one of the spheres of the same radius to the naked eye and is also not shown here.
As the volume fraction grows, we can observe that in the range of smaller scattering angle Q, the intensity starts around the same value. With a growing scattering angle, a bump is developing (Figure 4.7). It makes the intensity value grow too until it drops again, similarly to the scattering curve of the spheres.

volume fraction	accepted	declined	% accepted
0.01	59	3	95 %
0.05	300	84	78 %
0.075	462	210	69 %
0.1	627	517	55 %
0.15	965	1629	37 %
0.25	1771	19237	9.2%
0.3	2641	114980	2.2%

Table 4.1: An example of how many shapes get accepted while creating hard spheres of certain volume fraction

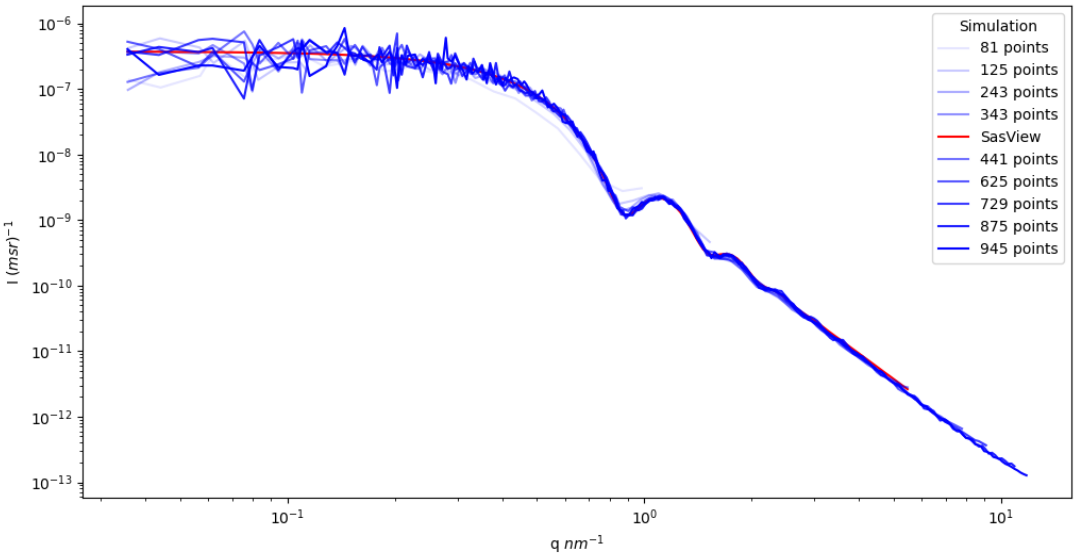


Figure 4.8: Scattering curves for 5 nm spherical particles simulated in the 250 nm box in different resolutions. The intensity of the color grows with the number of points in the resolution. The red curve is created analytically with SasView.

4.1.2 Simulated scattering curves in high-resolution

After we looked at the preliminary results and what the 3D figures for density and FFT look like, it is time to see the change in the scattering curves as the number of points in the simulation grows. Such simulations were conducted for all three shapes in different resolutions: 81, 125, 243, 343, 441, 625, 729, 875 and 945 points on the grid respectively.
The effect is twofold: Figure 4.8, Figure A.1 and A.2 demonstrate, that with higher resolution the scattering curves and patterns get smoother. Additionally, because of the way the scattering angle Q is calculated, the range of the experiment expands. Scattering curves shown in Figure 4.8 and A.1 have "wiggly" values in the scattering angles of $Q < 1nm^{-1}$. If the resolution is kept low, the curves will not stabilize against its tail.
As for scattering patterns in 2D, those also benefit from wider scattering angles and generally get smoother, such that they can further be used for analysis (Figure A.2).

Error rates

In chapter 3, three main benchmarking criteria were stated. The relative error was compared between a custom 3D FFT implementation and PyTorch's *fftn* algorithm and also between a custom version and SasModels simulation. Additionally, the χ^2 metric was calculated when compared against SasView simulations using the "ISigma" uncertainty values calculated during the rebinning process. All three were calculated for two data types: single and

Figure 4.9: χ^2 error between 3D simulation and SasView model using the "Isigma" uncertainty values. On the left, the χ^2 error is shown for single precision floating points and on the right for the double. The $y-$axis is logarithmic.

double precision floating points.
Because a single run might be an outlier, for each shape, size of the simulation and data type five samples were created. The average values of the respective error metrics are shown in the Table 4.2. The simulations for high-resolution hard spheres (size of simulation 875 and 945) in double precision were not calculated as it took too much time to place the shapes into the box. The visualizations are shown in the Appendix Figures A.3 and A.4.

It is not hard to see the trend that the relative error for doubles is smaller when compared against PyTorch's *fftn*. Special attention should be paid when dealing with high-resolution data. Even for a simulation of 441-pixel size, the relative error of single precision floats lies around 1.8%, while for double precision floats it is still 0.7%. Unfortunately, it is not feasible to make such a comparison for data exceeding $\sim$500 pixels in size for double precision floats, but we at least can see that it continues to grow.
The relative error compared to SasView is around the same for both values. It is because of the way SasView computes the scattering intensity and precision used.
Similar effects can be observed in terms of χ^2 error, the precision of the floating points does not seem to be crucial here. Interestingly, the overall trend is not clear for any shape, when the size of the simulation is increased. For cylinders, the error seems to drop at first, but then it slowly starts to grow again. The spheres' error has a zigzag form and for the hard spheres, the error is huge when compared with the rest two.

In the above χ^2 calculations the uncertainty of type "Isigma" is considered only. To give a reader an outlook, on how other uncertainties change the error rates a comprehensive comparison was created (for results refer to

shape	size	custom vs PyTorch single/double	custom vs SasView single/double	custom vs SasView χ^2 single/double
cylinder	81	0.0007/0.0003	1.9990/1.9990	77814/13098
	125	0.0016/0.0007	1.9991/1.9995	5430/20024
	243	0.0108/0.0027	1.9960/1.9963	2304/599
	343	0.0119/0.0042	1.8983/1.8872	4/5
	441	0.0181/0.0070	1.8764/1.8779	4/5
	625	0.0521/nan	1.8644/1.8734	8/8
	729	0.0304/nan	1.8711/1.8530	11/10
	875	nan/nan	1.9403/1.9191	17/16
	945	nan/nan	1.9285/1.9284	21/20
hard sphere	81	0.0004/0.0002	1.5018/1.5200	57430/70669
	125	0.0010/0.0009	1.5347/1.5678	28821/28807
	243	0.0190/0.0029	1.5477/1.5552	10558/10131
	343	0.0064/0.0035	1.5851/1.5583	8558/8829
	441	0.0211/0.0100	1.5797/1.5343	11315/8215
	625	0.0176/nan	1.5829/1.5195	7687/7498
	729	0.0360/nan	1.5280/1.5333	7490/7319
	875	nan/nan	1.5467/nan	4824/nan
	945	nan/nan	1.5288/nan	8495/nan
sphere	81	0.0005/0.0004	1.0491/1.0380	666/639
	125	0.0018/0.0007	0.8281/0.8329	271/255
	243	0.0037/0.0024	0.6285/0.6387	87/115
	343	0.0059/0.0029	0.9620/0.9088	138/123
	441	0.0151/0.0096	0.6663/0.6923	219/192
	625	0.0324/nan	0.7584/0.6876	45/82
	729	0.0359/nan	0.8344/0.4904	206/72
	875	nan/nan	0.7138/0.8752	92/188
	945	nan/nan	0.8132/0.7049	74/46

Table 4.2: Error rates for computed 3D FFTs. The first column is the relative error of custom FFT and the one calculated with PyTorch's *fftn*. The second column is the relative error between custom 3D FFT after rebinning and the SasView analytical simulation. The third column is the χ^2 error between the two latter. Errors are shown first for single-bit floating points and after for double. If a value is set to "nan", it means either the 3D FFT with *fftn* was not computed because it does not fit the GPU memory available or took too much time.

section 5). We established that the used ”Isigma” uncertainty gives more balanced estimates, as standard error uncertainties ”IStd” are usually very big, making χ^2 error small and less meaningful. On the contrary, the ”IError” type gives very small uncertainty estimates, resulting in much bigger χ^2 errors (Figure A.6).

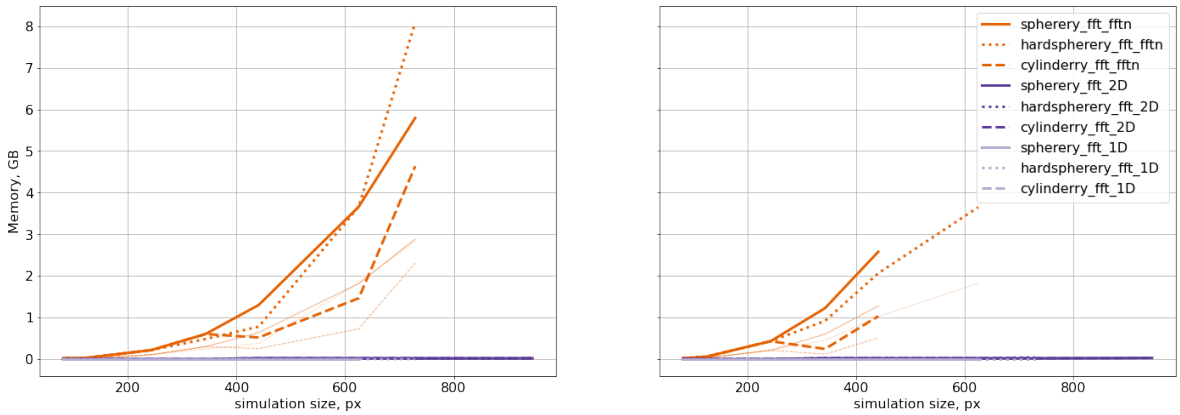


Figure 4.10: Memory on GPU required to compute 3D FFT. On the left, the requirements for single precision and on the right for double precision are shown. "fftn" in the legend of the figure represents the PyTorch's *fftn* algorithm; "2D" the custom version of 3D FFT broken down into the calculation of 2D slices and then 1D arrays; "1D" represents the version of custom 3D FFT broken down into three consecutive FFTs in each direction. Algorithm versions are color-coded. The shape information is decoded in the style of the line. A solid line represents spheres, dotted stands for hard spheres and dashed for cylinders. Wider lines in more intense color represent the memory requirements for the calculation of 3D FFT and thinner, more transparent ones the memory needed to store the data that will be 3D FFTed with the respective algorithm. For *fftn* all lines stop after there is not enough memory available to compute the FFT.

Memory requirments

The main point to divide the computation for 3D FFT was to save space in the memory, so that larger simulations could be performed. Here, it is shown how much memory on GPU is required to store the data in respective data types and to compute 3D FFT with all approaches discussed.
Figure 4.10 shows the memory required to store the data, and we observe that PyTorch's *fftn* naturally requires much more memory on the GPU as it tries to store the whole matrix compared to the custom version, both requiring much less than one GB to store data.
However, it gets even more drastic if we consider the memory requirements for the calculation of the FFT. This is exactly the double size of the memory required to store the density. PyTorch creates a copy of passed data to compute the FFT, this is why there is such a limitation regarding the space. Memory usage of custom algorithms is also doubled, as it uses PyTorch's implementation inside. Nonetheless, it is still under 1 GB for both variants, just because the density data passed for the computation was small.

Time requirments

Finally, after all other aspects concerning the implementation of 3D FFT were discussed, the aspect of time requirements will be studied. Figure 4.11 shows how much time in seconds is required for each algorithm. Here only

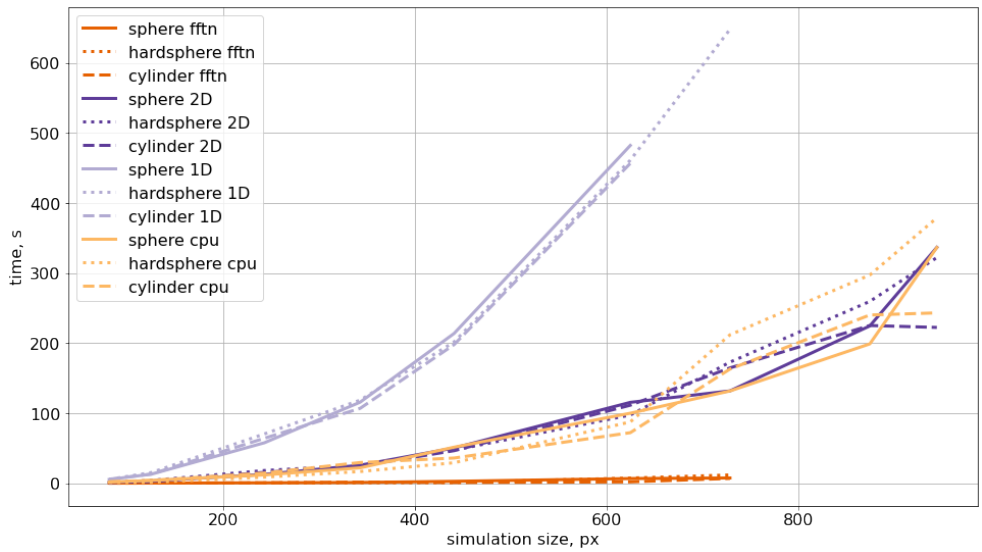


Figure 4.11: Time required to calculate 3D FFT. Only values for single precision floating points are shown. "fftn" in the legend of the figure represents the PyTorch's *fftn* algorithm computed on GPU; "cpu" represents the same algorithm computed on CPU; "2D" the custom version of 3D FFT broken down into the calculation of 2D slices and then 1D arrays; "1D" represents the version of custom 3D FFT broken down into three consecutive FFTs in each direction. Each version is color-coded. The shape information is decoded in the style of the line. A solid line represents spheres, dotted stands for hard spheres and dashed for cylinders.

the values for single precision floating points are shown, as there was not much difference compared to the double precision.
Computing it at once on GPU with *fftn* is the fastest. Next, the same algorithm calculated on CPU and 3D FFT split into slices + arrays follow. These two have around the same running time and are 10 to 100 times slower depending on the size. At last, the implementation of 3 consecutive FFTs follows.

4.2 Neural Network to infer the shape parameters of a scattering pattern

In this section, we will go through the list of architectures in Table 3.4, give some details about the hyperparameters and conclude with results. All the results are shown on the last bit of simulated data that did not participate in the training process. The visualizations of the results are shown in two formats. The first one is bar plots, where the fractions of correctly and falsely identified shapes are shown, such that each bar sums to 100% of the shapes. The second format is violin plots, as the sampled data represented distributions. On each violin plot, there are two instances per shape: the sampled and predicted ones which differ in color and label. Each such

violin illustrates the density of the respective distribution. Specifically, there is a thin line in the middle, depicting the 1.5 interquartile range and a solid one marking the interquartile range. There is also a white dot denoting the median of the distribution.

4.2.1 Feed-forward neural networks

As was earlier stated, the loss values were weighted when training NNs. In the feed-forward networks, we assigned more weight to the continuous variables, as they seemed to be harder to learn.

Fully connected NN

As was expected a fully connected neural network did not achieve good results. Because the problem is complex we tried to implement a rather deep NN with around 25 hidden layers and 1024 neurons each, activated with ReLU. No matter which changes to the hyperparameters were made, the network could at most sufficiently recognize one shape and the loss function would not decrease. The data was trained in batches of size 128 with a learning rate of 0.005 over 50 epochs. We then did not spend a lot of time tuning this network. The results of one training are shown in Figure A.7. This type of network only represents spheres. It does not recognize any other shape and independently of passed data claims, it is a sphere. It cannot, however, predict the radius value or its polydispersity even when the shape parameter is in fact a sphere. Instead, huge radius and polydispersity values are predicted around $3e14$.

Convolutional NN

Similarly, the convolutional Neural network did not perform very well. We tested different kernel sizes and a different number of hidden layers activated with ELU functions and concluded, that this activation function performs slightly better. We moreover, identified that average pooling after the convolutions improves the loss, while batch normalization seems to worsen it. Finally, the best version was able to correctly identify around 87% of shapes, when trained with the learning rate of 0.001 over 200 epochs. It consisted of 28 hidden layers with convolutional kernels of size 5 and "same" padding, yielding an output of the same size as the input. The kernel stride was set to 1, meaning that the kernel slides after each data point of the scattering curve. In the last convolutional layer, the stride was increased to 3 and the size of the kernel to 11. The final hidden layer was set as a fully-connected one. Nonetheless, the parameter values could not be identified with this setting, and we could not find a strategy to reduce the loss any further. Results of this NN can be found in Figure A.8.

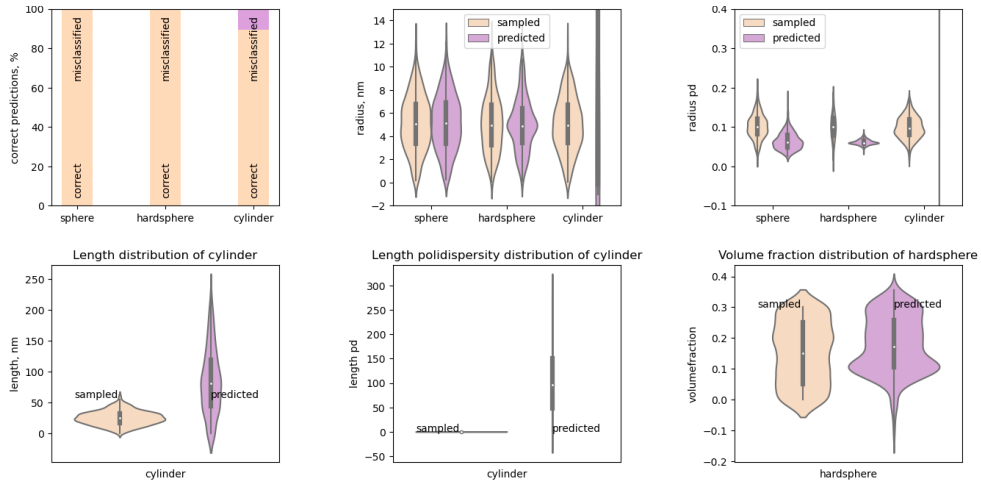


Figure 4.12: Results of convolutional ResNet NN. The top left figure shows the results of the shape prediction. Other figures show the distributions of the parameter describing shapes. The top middle and right figures depict the distributions of the radius and its polydispersity per shape. The bottom left and middle show the distribution of cylinder length and the respective polydispersity. The bottom right figure shows the volume fraction of the hard spheres.

Convolutional ResNet

In the ResNet, we implemented hidden layers based on the convolutions of the previous NN. The network was not very deep, with 4 ResNet blocks each consisting of 3 rounds of convolutions. We trained this NN with a learning rate of 0.001 over 2000 epochs. We were able to achieve high performance (shown on Figure 4.12) with the accuracy of shape prediction around 97% on the test data. Spheres and hard spheres were classified to 100% correctly. For these two shapes radius and its polydispersity were predicted very closely to the sampled ones. The volume fraction of hard spheres was also predicted very well. However, for the correctly identified cylinders, predicted values describing radius, length and their polydispersities are very far from the sampled.

The values of radius and length have much more spread-out distributions than the sampled ones. For instance, for the predicted radius values the interquartile range begins where the whole sampled distribution begins and ends much higher (it is not shown on the Figure 4.12, specifically to show that the distributions of two other shapes give good predictions). We also observe that the mean value of predicted cylinder radius values does not lie in the range of sampled distribution. A similar result is predicted for the length value. The predicted distribution takes 5 times the range of sampled values.

The polydispersity predictions of the two parameters are likewise not good. For example, the length polydispersity distribution takes on values in the

	radius	radius polydispersity	length	length polydispersity	volume fraction
sphere	0.0065	0.0018			
hard sphere	0.7824	0.0024			0.0025
cylinder	1654	372	5555	16153	

Table 4.3: MSE of parameters describing shapes obtained with convolutional ResNet NN

range $(-50, 300)$, which by far exceeds the sampled values, as these are centered around 0.15. The individual deviations of sampled and predicted parameters per shape are reflected in the Table 4.3, which points out that the cylinder's parameters prediction did not work.

In another training round, the results of which are shown in Figure A.9 we observe that this network may just not perform well for cylinders altogether. Overall prediction accuracy of prediction is still good, as well as spheres' and hard spheres' radius predictions. Cylinders' length prediction also worked well, but their radii are predicted to be even bigger than the length. We additionally looked into the differences between values for shapes that have not been identified correctly (false negatives in terms of the confusion matrix) and shapes that have been falsely identified as such shapes (false positives) in Figure A.10. Here, predictions for cylinders do not work either, although for other shapes distributions still seem to be close to each other. We finally looked at the difference in distributions between correctly predicted shapes (true positives) and misclassified predictions (false positives), which is depicted on Figure A.11 and confirms once again, that cylinders' parameters cannot be predicted with this network.

4.2.2 Inverse neural networks

While training INNs we established some important details about the hyperparameters that were true for all architectures. Firstly, the inverse multiquadric kernel seems to perform quite well in its current form. In some applications, the power term of the denominator was not $\frac{1}{2}$ as was described in section 3.2.2, but some other constants (Ardizzone et al., 2019b). In our application, this messed up the loss decay, so we stayed by the original version.
Another important hyperparameter present in the INNs is the feature space of the latent variable z. We found a rather small feature space of $\mathbb{R}^2$ to decrease the loss function the best. We weighted the MSE $\mathcal{L}_y$-loss by a very small coefficient 0.001, as it was the biggest, and forward MMD loss $\mathcal{L}_z$ by 100. Finally, the most important inverse loss $\mathcal{L}_x$ was weighted by 1000.

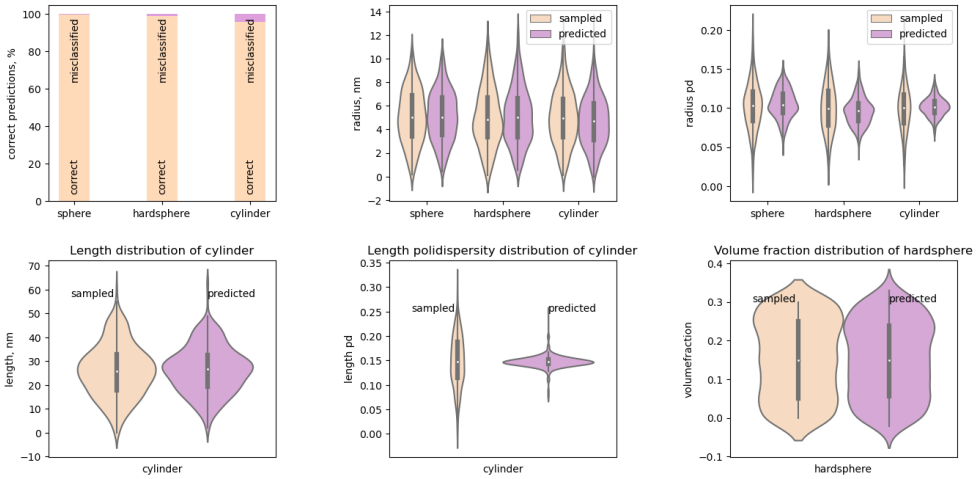


Figure 4.13: Results of RNVP-based INN. The top left figure shows the results of the shape prediction. Other figures show the distributions of the parameter describing shapes. The top middle and right figures depict the distributions of the radius and its polydispersity per shape. The bottom left and middle show the distribution of cylinder length and the respective polydispersity. The bottom right figure shows the volume fraction of the hard spheres.

We also concluded that the reconstruction loss did not help to optimize the problem.

i-ResNet

This architecture was not implemented as a deep one, because otherwise, the gradients exploded. It contained 7 Resnet blocks, each composed of four fully connected hidden layers with 32 neurons. The loss was decreasing in course of the learning process encompassing 50 epochs with a rate of $1e-4$. However, due to the fixed point iterations and truncated power series estimation (both iterative processes, see subsection 2.3.2) it takes a lot of time to train this network and we could not find hyperparameters resulting in good predictions. For this reason, we decided to stick to RNVP, because it allowed for much faster training and given the high number of hyperparameters it is a big benefit. The results of the i-ResNet can be found in Figure A.12.

RNVP

This architecture is also not deep to prevent the gradient explosion. It consisted of only 5 coupling NNs, each of which in turn had three hidden layers with 32 neurons. The exponential value was again clamped to a rather small value of e^2. The network was trained with a learning rate of 0.002 on 200 epochs, which took around 16 minutes on our machine.

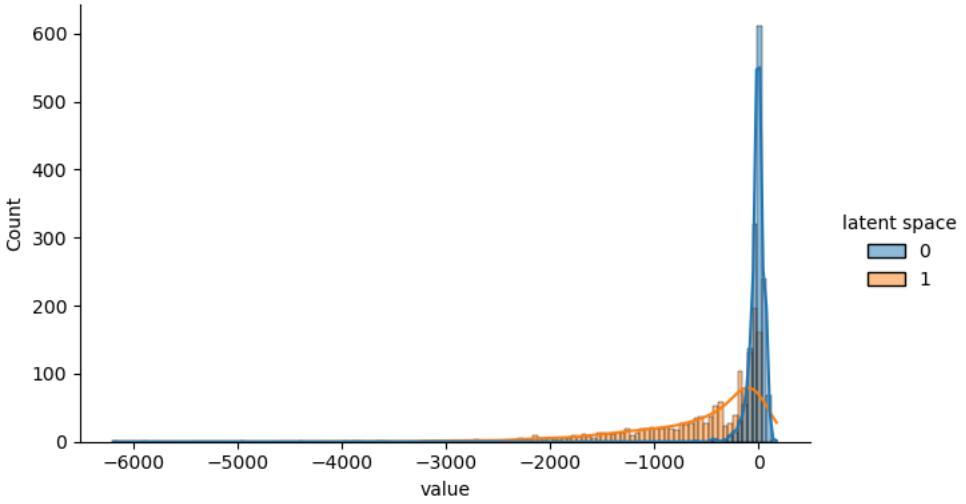


Figure 4.14: Distributions of two latent variables learned with RNVP-like INN

The achieved accuracy of shape prediction was 98%. The results of the distributions of the parameters are shown on Figure 4.13. The distributions seem to match nicely. Table 4.4 gives more detailed information on sample-to-sample error in terms of MSE. The distributions of the polydispersity and its MSE show that the network might overfit toward the mean values of the respective distributions.

	radius	radius polydis-persity	length	length polydis-persity	volume fraction
sphere	0.0574	0.0004			
hard sphere	1.3425	0.0005			0.0005
cylinder	0.4103	0.0006	10.7930	0.0027	

Table 4.4: MSE of parameters describing shapes obtained with INN based on RNVP-blocks

Another interesting characteristic that can be extracted from the INN is the space of the latent variable z. The distributions of the two latent variables are shown as a histogram on Figure 4.14. These two distributions are meant to represent the components of 3D FFT that are lost during the compression of the structure into a 1D scattering curve.

Most importantly, this network seems to be robust enough and over multiple training rounds it delivers consistent results.

Chapter 5

Discussion

In this work, we presented an approach to calculate 3D FFT with low memory requirements for high-resolution data. Here, we would like to comment on the drawbacks of the proposed method and its application in SAXS analysis development.
To begin with, in the results sections it was shown that simulations in low resolution are prone to errors and generally are not stable. Compared to the analytical solutions the error rates explode in low resolution. However, even a resolution of $\sim$512 pixels has its disadvantages.
In SAXS terms we are interested in the scattering angle Q defined in the reciprocal space. With a higher resolution, we obtain a much wider scattering angle range. This benefit might not be obvious at first, but in fact, it can help model complex hierarchical structures such as proteins or MOFs.

As for the computation of 3D FFT itself, there are many aspects to be taken into account. The bottleneck of 3D FFTing a structure with PyTorch as of now is that it is not parallelizable over available GPUs. It uses only one GPU, although the CuFFT underneath PyTorch's implementation enables parallel execution. When opting for PyTorch due to its comprehensiveness (when compared to C, C++ or Fortran supported by CUDA) this limitation becomes a big issue, as it is not able to process big quantities of data. In this context, splitting up a 3D FFT calculation into slices and then arrays seems to be a good solution to the problem.
Nonetheless, there are important points that should be addressed. There were two versions of the split-up implemented: the one that calculated 3D FFT as slices and arrays or as three consecutive arrays. The memory requirements of both algorithms are shown to be very small in the section 4.1.2. The time needed for the second version to compute the full 3D FFT is on the contrary growing exponentially with the size of the simulations. For instance, for a resolution of up to 20000^3 in double precision floating points,

calculating the 2D FFT of one such slice still requires less than 8 GB of memory. Calculating 20000^3 as 1D FFTs, on the other hand, would take forever. With that, we can conclude, that splitting a calculation of 3D FFT should be done in slices and arrays.
Another important detail exhibited in the section 4.1.2 is the relative error between the implementation of split-up 3D FFT and PyTorch's *fftn*. This artifact is an example of an unavoidable approximation of fractions as binary fractions taking place in most programming languages. The numerical error compared to the *fftn* arises since the order of operations is changed in two algorithms. In terms of mathematical operations, 3D FFT is a sequence of multiplications (the exponentiation in the DFT formula) and its summation over all dimensions. With the changed order in the custom 3D FFT version, each of the calculated 2D slice instances carries a small approximation error that is finally summed over the arrays in the last step of the algorithm. As was presented in section 4.1.2, this error can get quite significant if working with high-resolution data. From this perspective, it is advised to use double precision floating points to minimize this error.
The χ^2 error representing the error against the analytical solutions also offers interesting insights. The cylinders seem to have the smallest error compared to spheres and hard spheres. It is not surprising, since the implementation of rebinning for cylinders is carried out on a central slice of the calculated 3D FFT and produces the squared amount of the desired number of bins. The higher number of bins, in turn, allows for the creation of a more precise SasView simulation. This effect does not persist for higher resolutions, as the error rate begins to grow again. SasView numerical integrations can only offer a certain approximation and at the scale, where the χ^2 error increases, we assume the approximation error in the SasView also starts to grow. This means that the χ^2 error metric may account for both errors at the same time. The same argumentation would hold for any shape simulated with SasView on a big enough scale. Rebinned spheres show a higher but more constant error rate when compared to cylinders. As was mentioned, spheres are rebinned into fewer bins than cylinders, which possibly affects the metrics. Finally, hard spheres show error rates, which in ca. 2 orders of magnitude exceed one of the spheres. In our opinion, it also traces back to the simulation created with SasView. When dealing with polydispersity in structures with high volume fractions the numerical integration gets more imperfect.

After all drawbacks of split-up implementation of the 3D FFT were stated, a question might arise, why use it at all? As shown on Figure 4.11, passing the whole 3D electron density and computing it on the CPU takes a comparable time to execute full 3D FFT at once with PyTorch's *fftn*. However, it is not a coincidence, the 3D simulation size did not exceed 1000^3 pixels in this work. Around this size, a 3D matrix stops fitting into the RAM of the CPU on

our machine, which poses another bottleneck. Following that, the solution proposed in this work can be further developed in a way, that the whole 3D electron density instance is considered as slices. Only one slice of the data is read-in in the first place, it is then FFTed and written back. After all slices are processed this way, in the next step, a slice in another direction is read into the RAM of the CPU, then the remaining FFT in one direction is calculated and written back. At last, the rebinning procedure is executed slice-wise. This pipeline example enables the calculation of 3D FFT of high-resolution data on simpler machines. An example of such transformation is shown on Figure 5.1. The density of MOF crystal unit cell was calculated by Aakash Naik with the VASP software[1] and supersampled to the size of 4096^3 by Dr. Brain Pauw. The proof-of-concept for this high-resolution structure (including slice-wise IO operations, the 3D FFT approach explained in this thesis and the rebinning procedure) was computed by him too. The resulting scattering curve is presented on Figure 5.1. Here we see that the range of scattering angle Q is extended up to 1000 nm, compared to the lower resolution tests carried out in the scope of this thesis (Figure 4.8). The most useful result is a very detailed behavior of the intensity curve in the range of 1-10 nm. In a lower resolution, it would be smoothed out with fewer distinguishable peaks.

Concerning the problem of identifying the shapes and parameters of shapes, given the scattering curve (section 3.2), we were able to obtain good predictions with both feed-forward NN and INN in the scope of this work. However, figuring out a good architecture and especially the hyperparameters required a lot of time.
With the MMD loss function, it seemed relatively easy, once the kernel function was established correctly. Another big improvement was identifying the feature space of the latent variable z for INN.
A fully connected feed-forward NN was not able to grasp the complexity of the inverse scattering problem in the architecture that has been tested. An improvement was to use 1D convolutional kernels instead. With this change, the classification of the shapes worked much better, but further improving the NN with skip connections gave even better results, as the NN was able to predict the parameters of the shapes. An important detail in the results of the ResNet is that for a more complex shape - the cylinder, the predictions for radius and length worked badly and even more so for their polydispersity values. The sample-to-sample error given in terms of MSE (Table 4.3) confirms that the radius of spheres and hard spheres is predicted very precisely, while for cylinders these values explode. This means that such a network cannot be used to determine parameters of more complex shapes.
As for the INNs, we were not able to find hyperparameters that would result

[1] https://www.vasp.at/

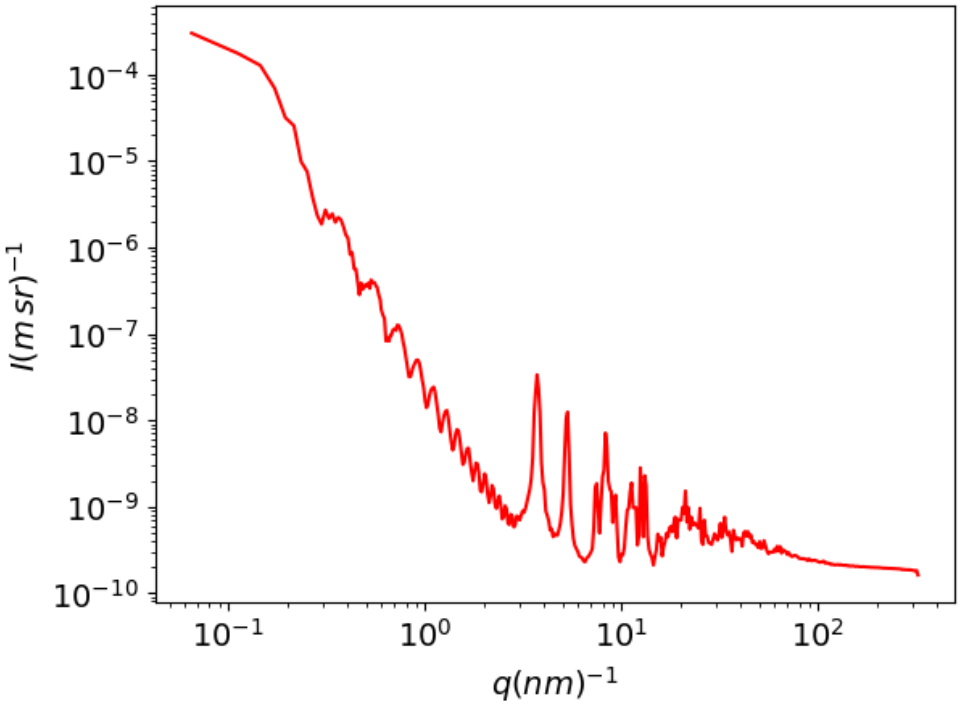


Figure 5.1: Scattering curve of the supersampled MOF density structure of size 4096^3 pixels after 3D FFT and rebinning. The $x-$ and $y-$ axes are logarithmic.

in a good prediction of i-ResNet. With RNVP coupling blocks we achieved surprisingly good results for both classification and regression tasks within our problem. A concern here is the overfitting of the polydispersity values toward the mean of the distribution. Table 4.2 shows, that the errors for radius and length are a bit bigger compared to ResNet results, but polydispersity errors are very small, which happens due to overfitting and initially sampled to be small values of the polydispersity. The latent variables that were learned with the INN give no clear insights but may include some information on the phase problem.

In the scope of this work, only three shapes were considered. However, there are many other shapes, for which analytical solution for scattering exists. One could continue the same strategy and extend the shapes in the training set. It seems that the most suitable NN architecture would be an RNVP-like INN, as the only other candidate - classic ResNet with convolutional layers in-between, could not predict the parameter values of a rather simple shape - the cylinder.
A further limitation to encounter is posed by irregular shapes that cannot be described by the set of parameters. Another big drawback of this work is

the persisting limitation to the properties of the shape. As was mentioned in the introduction, a 3D electron density is what a materials researcher would like to have. Keeping track of the exact 3D location of the densities remains the bottleneck of this study.

Bibliography

L. Ardizzone, T. Bungert, F. Draxler, U. Köthe, J. Kruse, R. Schmier, and P. Sorrenson. Framework for Easily Invertible Architectures (FrEIA), 2018-2022.

L. Ardizzone, J. Kruse, S. Wirkert, D. Rahner, E. W. Pellegrini, R. S. Klessen, L. Maier-Hein, C. Rother, and U. Köthe. Analyzing inverse problems with invertible neural networks. *arXiv:1808.04730 [cs, stat]*, Feb 2019a. arXiv: 1808.04730.

L. Ardizzone, C. Lüth, J. Kruse, C. Rother, and U. Köthe. Guided image generation with conditional invertible neural networks. (arXiv:1907.02392), Jul 2019b. arXiv:1907.02392 [cs].

J. Behrmann, W. Grathwohl, R. T. Q. Chen, D. Duvenaud, and J.-H. Jacobsen. Invertible residual networks. In *Proceedings of the 36th International Conference on Machine Learning*, page 573–582. PMLR, May 2019.

K. M. Borgwardt, A. Gretton, M. J. Rasch, H.-P. Kriegel, B. Scholkopf, and A. J. Smola. Integrating structured biological data by kernel maximum mean discrepancy. *Bioinformatics*, 22(14):e49–e57, Jul 2006. ISSN 1367-4803, 1460-2059. doi: 10.1093/bioinformatics/btl242.

I. Bressler, B. R. Pauw, and A. F. Thünemann. McSAS: software for the retrieval of model parameter distributions from scattering patterns. *J. Appl. Crystallogr.*, 48(Pt 3):962–969, June 2015.

T. Butz. *Fourier Transformation für Fussgänger*. B. G. Teubner, Germany, 2007.

N. E. Chayen and E. Saridakis. Protein crystallization: from purified protein to diffraction-quality crystal. *Nature Methods*, 5(2):147–153, Feb 2008. ISSN 1548-7105. doi: 10.1038/nmeth.f.203.

D.-A. Clevert, T. Unterthiner, and S. Hochreiter. Fast and accurate deep network learning by exponential linear units (elus), 2015.

J. W. Cooley and J. W. Tukey. An algorithm for the machine calculation of complex Fourier series. *Math. Comput.*, 19:297–301, 1965.

V. P. Dandey, W. C. Budell, H. Wei, D. Bobe, K. Maruthi, M. Kopylov, E. T. Eng, P. A. Kahn, J. E. Hinshaw, N. Kundu, C. M. Nimigean, C. Fan, N. Sukomon, S. A. Darst, R. M. Saecker, J. Chen, B. Malone, C. S. Potter, and B. Carragher. Time-resolved cryo-em using spotiton. *Nature Methods*, 17(9):897–900, Sep 2020. ISSN 1548-7105. doi: 10.1038/s41592-020-0925-6.

L. Dinh, D. Krueger, and Y. Bengio. Nice: Non-linear independent components estimation. (arXiv:1410.8516), Apr 2015. arXiv:1410.8516 [cs].

L. Dinh, J. Sohl-Dickstein, and S. Bengio. Density estimation using real nvp. (arXiv:1605.08803), Feb 2017. arXiv:1605.08803 [cs, stat].

C. Do, W.-R. Chen, and S. Lee. Small angle scattering data analysis assisted by machine learning methods. *MRS Advances*, 5(29-30):1577–1584, June 2020. doi: 10.1557/adv.2020.130.

M. Doucet, J. Cho, G. Alina, Z. Attala, J. Bakker, W. Bouwman, P. Butler, K. Campbell, T. Cooper-Benun, C. Durniak, et al. Sasview version 5.0. 4, 2021. *Google Scholar There is no corresponding record for this reference*.

A.-H. M. Emwas. *The Strengths and Weaknesses of NMR Spectroscopy and Mass Spectrometry with Particular Focus on Metabolomics Research*, pages 161–193. Springer New York, New York, NY, 2015. ISBN 978-1-4939-2377-9. doi: 10.1007/978-1-4939-2377-9_13.

O. Glatter. A new method for the evaluation of small-angle scattering data. *Journal of Applied Crystallography*, 10(5):415–421, Oct 1977. ISSN 00218898. doi: 10.1107/S0021889877013879.

O. Glatter. *Scattering Methods and their Application in Colloid and Interface Science*, page i–iii. Elsevier, 2018. ISBN 978-0-12-813580-8. doi: 10.1016/B978-0-12-813580-8.00016-X.

P. J. Goodhew, J. Humphreys, and R. Beanland. *Electron microscopy and analysis, third edition*. Taylor & Francis, London, England, 3 edition, Nov. 2000.

L. Grafakos. *Classical Fourier Analysis*. "Graduate texts in mathematics". Springer, New York, NY, 3 edition, jul 2014.

A. Gretton, K. M. Borgwardt, M. J. Rasch, B. Schölkopf, and A. Smola. A kernel two-sample test. *The Journal of Machine Learning Research*, 13(null): 723–773, 2012. ISSN 1532-4435.

A. Guinier. *SMALL-ANGLE SCATTERING OF X-RAYS*. JOHN WILEY & SONS, Inc. , New York, 1955.

K. Hagita, T. Tominaga, and T. Sone. Large-scale reverse monte carlo analysis for the morphologies of silica nanoparticles in end-modified rubbers based on ultra-small-angle x-ray scattering data. *Polymer*, 135:219–229, 2018. ISSN 0032-3861. doi: https://doi.org/10.1016/j.polymer.2017.12.018.

B. C. Hall. Lie groups, lie algebras, and representations. In *Quantum Theory for Mathematicians*, pages 333–366. Springer, 2013.

S. Hansen. Bayesian estimation of hyperparameters for indirect fourier transformation in small-angle scattering. *Journal of Applied Crystallography*, 33(6):1415–1421, Dec 2000. ISSN 0021-8898. doi: 10.1107/S0021889800012930.

S. Hansen. Estimation of chord length distributions from small-angle scattering using indirect fourier transformation. *Journal of Applied Crystallography*, 36(5):1190–1196, Oct 2003. ISSN 0021-8898. doi: 10.1107/S0021889803014262.

S. Hansen. Simultaneous estimation of the form factor and structure factor for globular particles in small-angle scattering. *Journal of Applied Crystallography*, 41(2):436–445, Apr 2008. ISSN 0021-8898. doi: 10.1107/S0021889808004937.

H. He, C. Liu, and H. Liu. Model reconstruction from small-angle x-ray scattering data using deep learning methods. *iScience*, 23(3):100906, 2020. ISSN 2589-0042. doi: https://doi.org/10.1016/j.isci.2020.100906.

K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

W. V. Herck, J. Fisher, and M. Ganeva. Deep learning for x-ray or neutron scattering under grazing-incidence: extraction of distributions. *Materials Research Express*, 8(4):045015, Apr. 2021. doi: 10.1088/2053-1591/abd590.

C. F. Holder and R. E. Schaak. Tutorial on powder x-ray diffraction for characterizing nanoscale materials. *ACS Nano*, 13(7):7359–7365, 2019. doi: 10.1021/acsnano.9b05157. PMID: 31336433.

J. Ilavsky and P. R. Jemian. Irena: tool suite for modeling and analysis of small-angle scattering. *Journal of Applied Crystallography*, 42(2):347–353, Apr 2009. doi: 10.1107/S0021889809002222.

D. P. Kingma and J. Ba. Adam: A method for stochastic optimization, 2014.

A. Kloeckner, Y. Yu, M. Wala, I. Fernando, M. Bencun, K. Kulkarni, M. Diener, H. Gao, A. Fikl, Z. Weiner, M. Weigert, R. Palmer, S. Latham, G. Magno, H. Fuller, J. Mackenzie, S. Niarchos, S. Gill, C. Gohlke, A. Bhosale, A. Rothberg, E. Ey, H. Rapp, S. van der Walt, G. Thalhammer, J. Kieffer, N. Poliarnyi, D. Bollinger, A. Nitz, and G. Bokota. Pyopencl, Mar. 2022.

I. Kobyzev, S. J. D. Prince, and M. A. Brubaker. Normalizing flows: An introduction and review of current methods. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 43(11):3964–3979, Nov 2021. ISSN 0162-8828, 2160-9292, 1939-3539. doi: 10.1109/TPAMI.2020.2992934. arXiv:1908.09257 [cs, stat].

S. Liu, C. N. Melton, S. Venkatakrishnan, R. J. Pandolfi, G. Freychet, D. Kumar, H. Tang, A. Hexemer, and D. M. Ushizima. Convolutional neural networks for grazing incidence x-ray scattering patterns: thin film structure identification. *MRS Communications*, 9(2):586–592, 2019. doi: 10.1557/mrc.2019.26.

R. S. Macomber. *A complete introduction to modern NMR spectroscopy*. A Wiley-Interscience publication. John Wiley & Sons, Nashville, TN, Dec. 1997.

A. Müller. Integral probability metrics and their generating classes of functions. *Advances in Applied Probability*, 29:429 – 443, 1997.

Origin Lab. 18.11.1.2 Algorithms (2D FFT). https://www.originlab.com/doc/Origin-Help/FFT2-Algorithm. [Online; accessed 09.08.2022].

A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

B. R. Pauw. *The nanostructure of high-performance fibres*. PhD thesis, DTU Chemical Engineering, Department of Chemical and Biochemical Engineering, Kgs. Lyngby, 2009.

G. Porod. Die röntgenkleinwinkelstreuung von dichtgepackten kolloiden systemen. *Kolloid-Zeitschrift*, 124(2):83–114, Nov 1951. ISSN 1435-1536. doi: 10.1007/BF01512792.

A. Sarje, X. S. Li, and A. Hexemer. High-performance inverse modeling with reverse monte carlo simulations. In *2014 43rd International Conference on Parallel Processing*, pages 201–210, 2014. doi: 10.1109/ICPP.2014.29.

SasView 5.0.5 User Documentation. Polydispersity & Orientational Distributions: Gaussian Dictribution. https://www.sasview.org/docs/user/qtgui/Perspectives/Fitting/pd/polydispersity.html#gaussian-distribution, 2022a. [Online; accessed 23.11.2022].

SasView 5.0.5 User Documentation. Cylinder. https://www.sasview.org/docs/user/models/cylinder.html, 2022b. [Online; accessed 23.11.2022].

K. Schmidt-Rohr. Simulation of small-angle scattering curves by numerical fourier transformation. *Journal of Applied Crystallography*, 40, 10 2006. doi: 10.1107/S002188980604550X.

B. Schölkopf and A. J. Smola. *Learning with kernels: support vector machines, regularization, optimization, and beyond*. Adaptive computation and machine learning. MIT Press, Cambridge, Mass, 2002. ISBN 978-0-262-19475-4. Table 2.1.

M. S. Smyth and J. H. Martin. X ray crystallography. *Mol. Pathol.*, 53(1):8–14, Feb. 2000.

D. I. Svergun, A. V. Semenyuk, and L. A. Feigin. Small-angle-scattering-data treatment by the regularization method. *Acta Crystallographica Section A Foundations of Crystallography*, 44(3):244–250, May 1988. ISSN 0108-7673. doi: 10.1107/S0108767387011255.

A. N. Tikhonov and V. Y. Arsenin. Solutions of ill-posed problems. 1977.

I. Tolstikhin, O. Bousquet, S. Gelly, and B. Schoelkopf. Wasserstein auto-encoders. (arXiv:1711.01558), Dec 2019. arXiv:1711.01558 [cs, stat].

C. S. Withers and S. Nadarajah. log det a = tr log a. *International Journal of Mathematical Education in Science and Technology*, 41(8):1121–1124, 2010. doi: 10.1080/0020739X.2010.500700.

B. Wunderlich. *Unsolved Problems of Crystallization and Melting of Flexible Macromolecules*, pages 237–248. Springer Netherlands, Dordrecht, 1993. ISBN 978-94-011-1950-4. doi: 10.1007/978-94-011-1950-4_19.

Appendix A: Simulated scattering curves in high-resolution

Comparison of 3D FFTed simulated shapes in different resolution

Here we show the results of FFTed simulated hard spheres (Figure A.1) and cylinders (Figure A.2) in different resolutions. We observe, that similar to spheres, curves get smoother and the scattering vector q becomes large. In cylinders' scattering patterns, the scattering angle values also become larger.

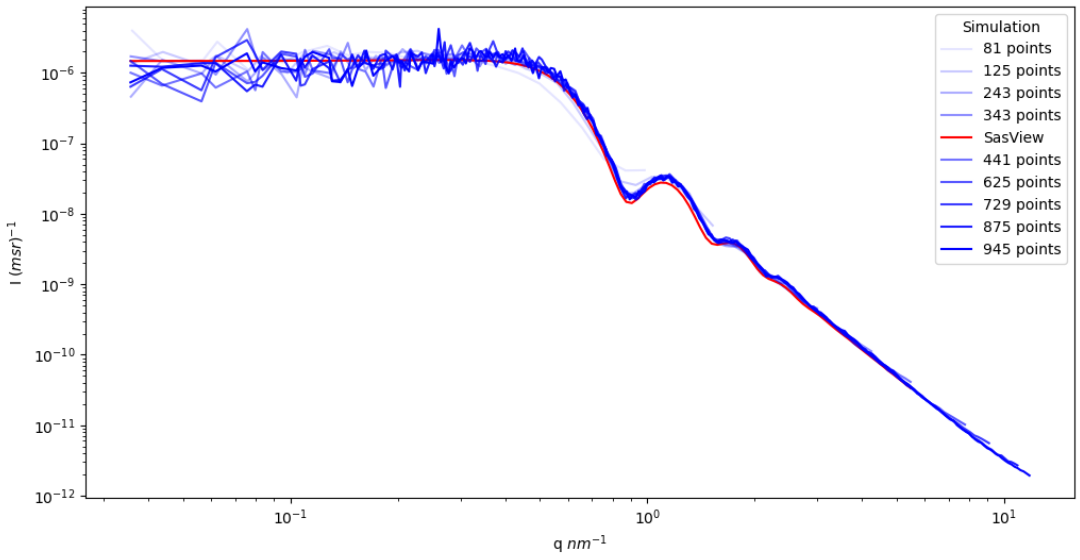


Figure A.1: Scattering patterns for hard sphere particles of radius 5 nm simulated in the 250 nm box in different resolutions. The intensity of the color grows with the number of points in the resolution. The red curve is created analytically with SasView.

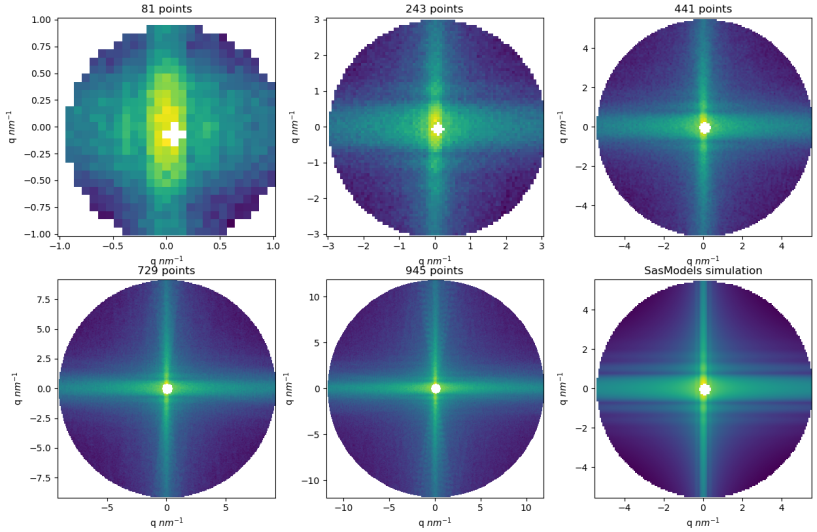


Figure A.2: Scattering patterns for cylindric particles of radius 5 nm and length 25 nm, rotated in θ by 10° simulated in the 250 nm box in different resolutions. The intensity of the color grows with the number of points in the resolution. The red curve is created analytically with SasView.

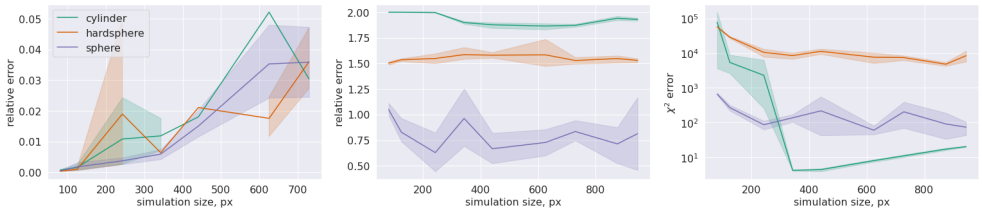


Figure A.3: Error rates for simulations in single precision floating point precision.

The error rates for single precision floating point are shown on Figure A.3. The relative error grows fast with the resolution of the simulations for all shapes, while compared to SasView simulations it is more or less on a constant level for each shape. A very similar result is obtained for double precision floating points except in the relative error values when compared against PyTorch's *fftn*.

Effect of uncertainty estimates choice on χ^2 metrics for different shapes

Here we show how different uncertainty estimates scale in comparison to the scattering curve (Figure A.5). For cylinders and hard spheres, this looks very similar. The first thing that one can notice, is a very high uncertainty

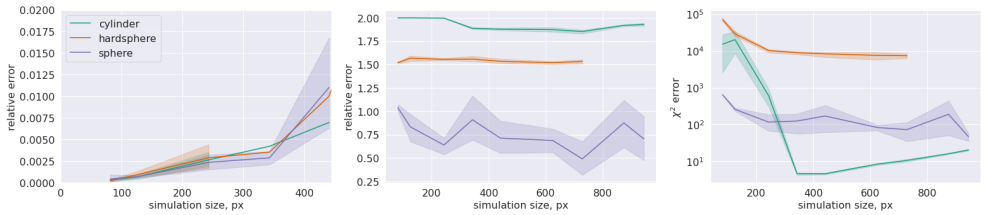


Figure A.4: Error rates for simulations in single precision floating point precision.

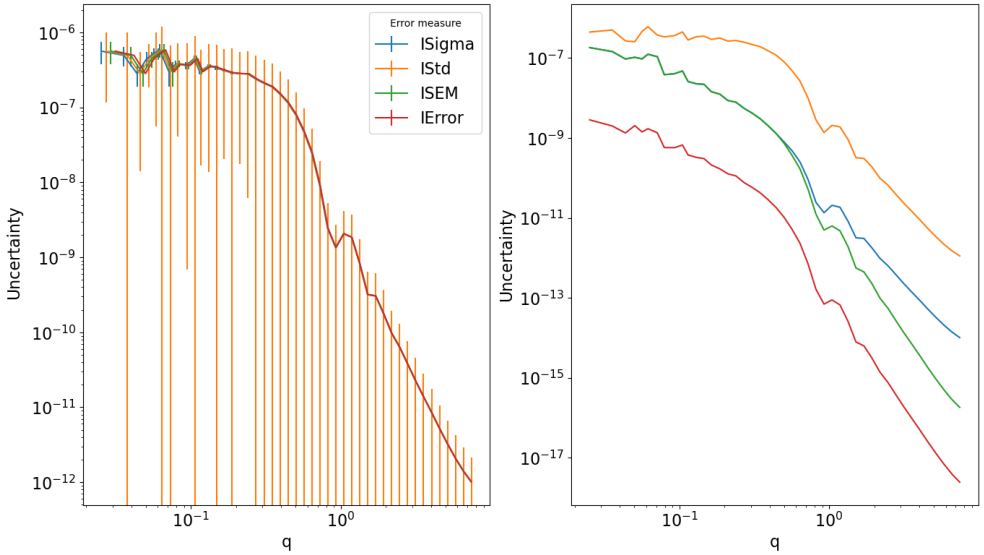


Figure A.5: Effect of uncertainty choice shown on the left as error bars on the scattering curve of a sphere. Different colors represent the respective uncertainty metric. On the scattering curve, the errorbars are shifted from each other a little to become distinguishable. They, however, still represent the uncertainty value at the same point. On the right the absolute values of uncertainties are shown.

estimate of "IStd", which naturally results in very low χ^2 metrics when compared to the SasView simulations. The "ISigma" uncertainty used in this work lies between all metrics and at the beginning agrees with the "ISEM" metric.

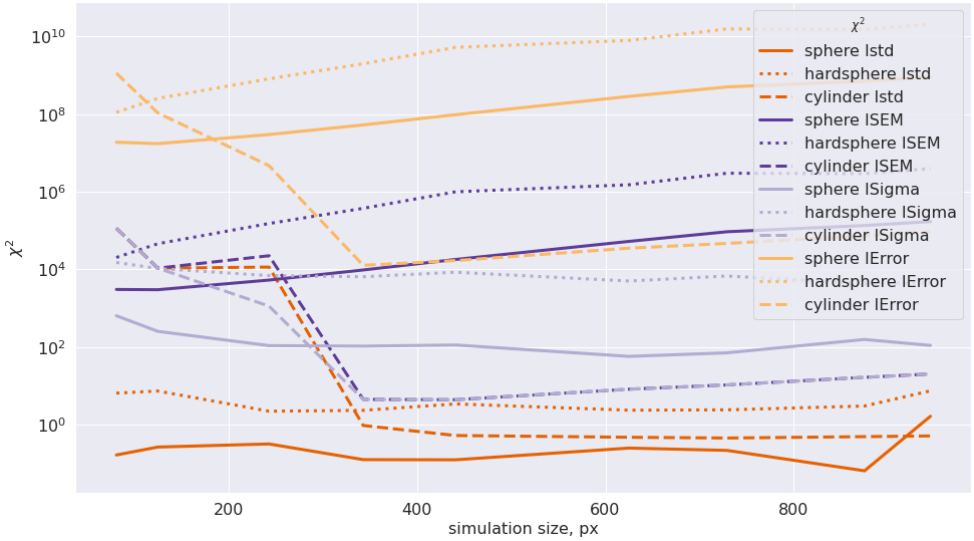


Figure A.6: Effect of the uncertainty choice on the χ^2 metric for all shapes in different resolution. $x-$axis represents the size of the simulation and $y-$axis measures χ^2 error when compared to SasView simulation. The color of the line depicts the uncertainty metric and the line style stands for a certain shape.

Appendix B: Neural Network to infer the shape parameters of a scattering pattern

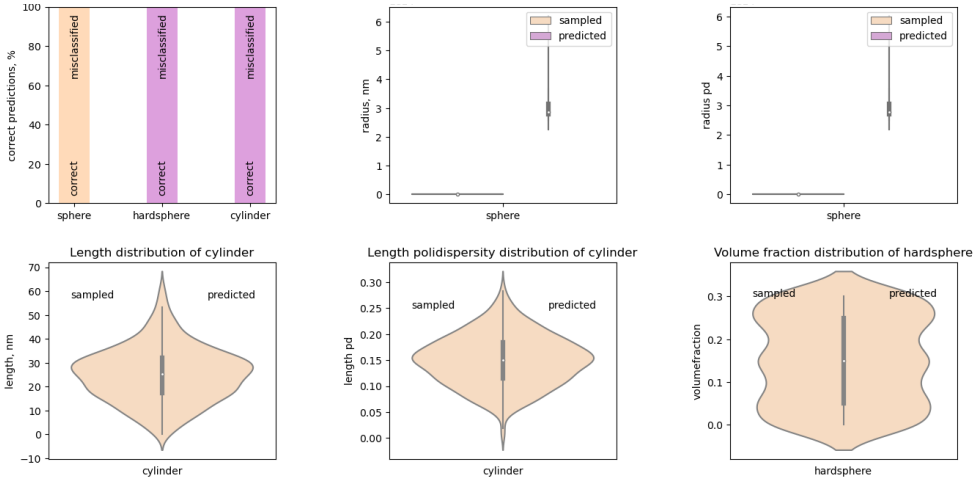


Figure A.7: Results of feed-forward fully-connected NN. The top left image represents the results of the classification task. The top middle image shows the sampled and predicted radius distribution per shape for correctly classified shapes. The top right image shows the distribution results of radius polydispersity per shape when classified correctly. The bottom left image shows the results of cylinder length, and The bottom middle image the cylinder length polydispersity when identified as a cylinder. The bottom right image shows the sampled and predicted distribution of hard spheres' volume fraction for correctly identified shapes.

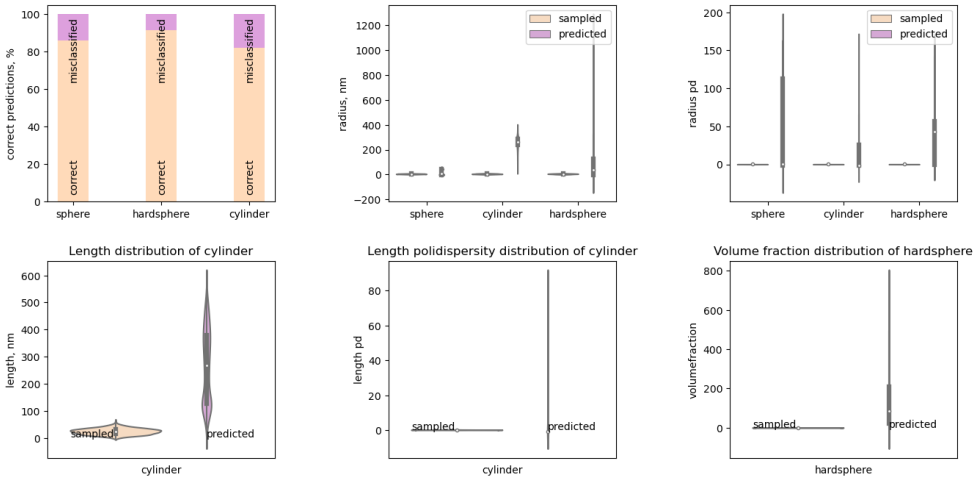


Figure A.8: Results of feed-forward convolutional NN. The top left image represents the results of the classification task. The top middle image shows the sampled and predicted radius distribution per shape for correctly classified shapes. The top right image shows the distribution results of radius polydispersity per shape when classified correctly. The bottom left image shows the results of cylinder length, and The bottom middle image the cylinder length polydispersity when identified as a cylinder. The bottom right image shows the sampled and predicted distribution of hard spheres' volume fraction for correctly identified shapes.

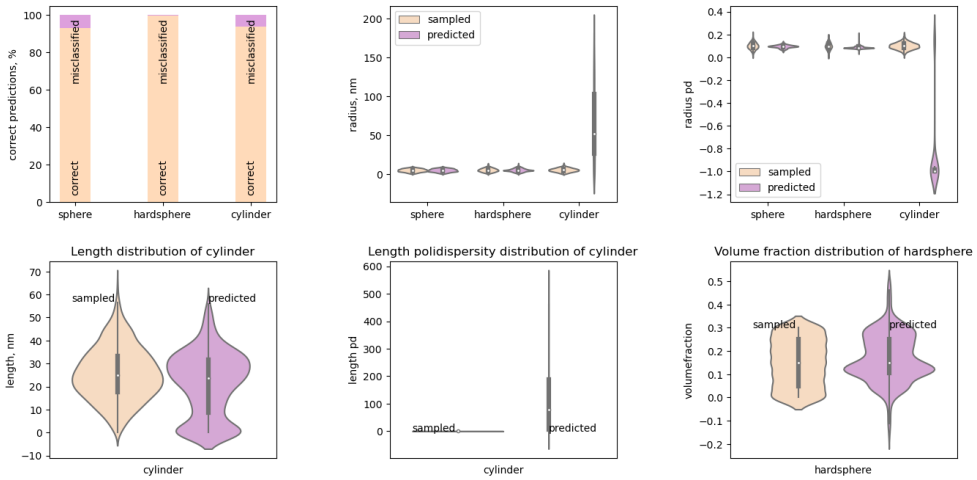


Figure A.9: Results of feed-forward convolutional ResNet on correctly identified shapes. The top left image represents the results of the classification task. The top middle image shows the sampled and predicted radius distribution per shape for correctly classified shapes. The top right image shows the distribution results of radius polydispersity per shape when classified correctly. The bottom left image shows the results of cylinder length, and The bottom middle image the cylinder length polydispersity when identified as a cylinder. The bottom right image shows the sampled and predicted distribution of hard spheres' volume fraction for correctly identified shapes.

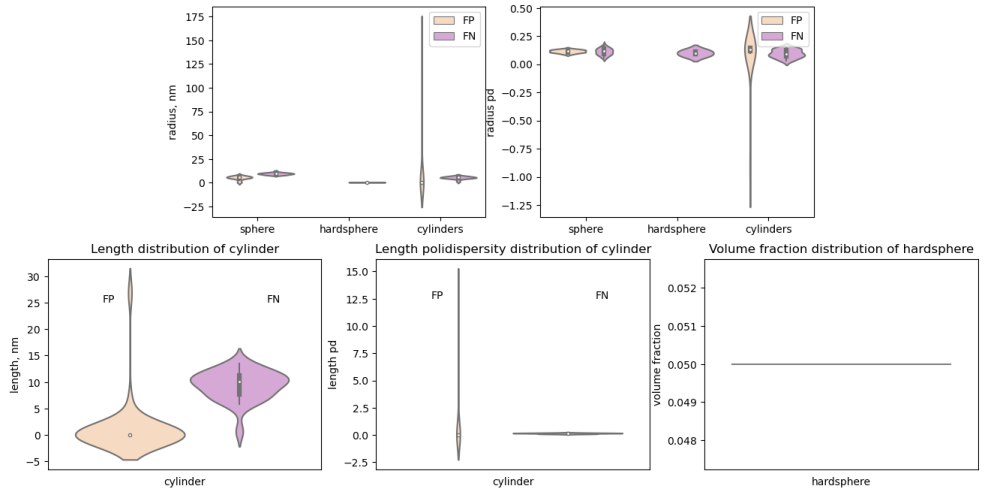


Figure A.10: Results of feed-forward convolutional ResNet on unidentified shapes. False negatives are sampled shapes that have been misclassified, and false positives are shapes that were wrongly predicted. For some plots, hard sphere data is missing. It is because there were only two data points for false negatives and none for false positives, such that the violin plot is not created for them. The top left image shows the sampled and predicted radius distribution per shape. The top right image shows the distribution results of radius polydispersity per shape when classified correctly. The bottom left image shows the results of cylinder length and The bottom middle image the cylinder length polydispersity. The bottom right image shows the sampled distribution of hard spheres' volume fraction.

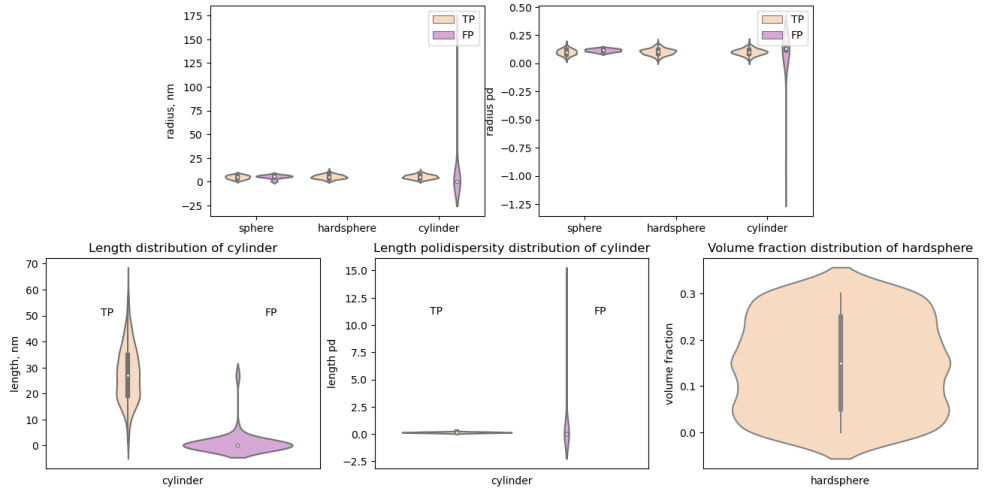


Figure A.11: Results of feed-forward convolutional ResNet. Comparison of wrongly and correctly identified shapes. True positives are sampled shapes that have been classified correctly, and false positives are shapes that were wrongly predicted. For hard spheres, there were no false positives, such that the violin plot is not created for them. The top left image shows the sampled and predicted radius distribution per shape. The top right image shows the distribution results of radius polydispersity per shape when classified correctly. The bottom left image shows the results of cylinder length and The bottom middle image the cylinder length polydispersity. The bottom right image shows the sampled distribution of hard spheres' volume fraction.

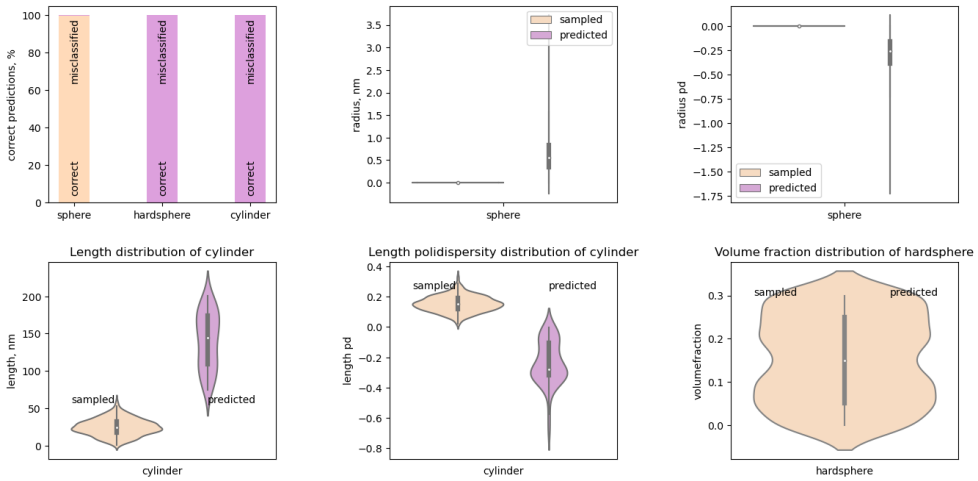


Figure A.12: Results of i-ResNet. The top left image represents the results of the classification task. The top middle image shows the sampled and predicted radius distribution per shape for correctly classified shapes. The top right image shows the distribution results of radius polydispersity per shape when classified correctly. The bottom left image shows the results of cylinder length, and The bottom middle image the cylinder length polydispersity when identified as a cylinder. The bottom right image shows the sampled and predicted distribution of hard spheres' volume fraction for correctly identified shapes.