Automatic and efficient tomographic reconstruction algorithms

Lagerwerf, M.J.

Chapter 4

Neural Network Feldkamp-Davis-Kress algorithm

4.1 Introduction

Circular cone-beam (CCB) Computed Tomography (CT) has become an integral part of non-destructive imaging in a broad spectrum of applications, such as industrial quality control [GUV11], materials sciences [Die+14; Bul+16] and medical imaging [For+02; GKT17]. Limitations on the scanning process caused by the need to scan a large number of objects in a short amount of time lead to measurements with a low number of projection angles or high noise levels. Additionally, CT reconstruction has become a *big data* problem due to the development of readily available high-resolution CT-scanners [TESb; TESa; Can]. This stresses the need for computationally efficient reconstruction methods that are applicable to a broad spectrum of high-resolution problems and produce accurate results from data with a high noise levels, low number of projection angles or large cone angles.

In practice, if computational efficiency is a constraint and especially for high-resolution problems, *direct methods* (*e.g.*, the filtered backprojection (FBP) algorithm [Nat01], the Feldkamp-Davis-Kress (FDK) algorithm [FDK84] and the Katsevich algorithm [Kat03]) are still the common choice of reconstruction method

This chapter is based on:

A computationally efficient reconstruction algorithm for circular cone-beam computed tomography using shallow neural networks. *MJ Lagerwerf, DM Pelt, WJ Palenstijn, KJ Batenburg*. Journal of Imaging (Submitted for publication).

[PSV09]. While *iterative methods* have been shown to be more accurate for noisy and limited data problems [ROF92; BKP10; SP08; Jia+10; Niu+14; EF03], they have a significantly higher computational cost. Consequently there have been efforts to improve the accuracy of direct methods by computing data-specific or scanner-specific filters [Zen12; Nie+12; BP12; PB14; Lag+20]. Although these strategies do improve the reconstruction accuracy, they also add significant computational effort or are specific to one modality, *e.g.*, tomosynthesis [Kun+07].

An emerging approach for improving direct methods is to use *machine learning* to remove artifacts from the reconstructions. The idea is to use high-quality reconstructions to train a neural network that removes artifacts from low-quality reconstructions using a supervised learning approach. This *post-processing* approach has shown promising results for computed tomography using deep neural networks (DNNs) [Jin+17; PBS18; Kid+18]. Deep neural network structures contain a large number of layers, leading to millions of trainable parameters and therefore require a large amount of training data [PS18]. This is problematic in CT imaging, since there is often a limited amount of training data available, *e.g.*, due to scanning time, dose, and business-related concerns. Moreover, for the available data there are often no reference datasets or annotations available [Wan+18]. The large amount of training data and large number of parameters also lead to long training times. While for standard 2D networks the training time ranges between a couple of hours and a couple of days (see **Section 4.5.1**), for 3D networks the training time becomes prohibitively long [Çiç+16] (*i.e.*, weeks). Therefore, to apply post-processing to 3D problems the reconstruction volume can be considered as a stack of 2D problems [RFB15; PBS18] for which one 2D network is trained and then applied in a *slice-by-slice* fashion to the 3D volume. Although this strategy reduces the training time and the training data constraints, applying a 2D network to all slices can still be computationally intensive due to the number of slices in the 3D volume. A more in-depth discussion on current developments related to machine learning methods in CT imaging is given in **Section 4.2**.

In this chapter we propose the Neural Network FDK (NN-FDK) reconstruction algorithm. It is a direct reconstruction method that is designed to produce accurate results from noisy data, data with a low number of projection angles, or a large cone angle, but still maintains a similar computational efficiency and scalability as the standard FDK algorithm. Moreover, the algorithm has a fast training procedure, and requires a limited amount of training data.

The NN-FDK algorithm is an adaptation of the standard FDK algorithm using a shallow multilayer perceptron network [Bis06] with one fully connected hidden layer, a low number of trainable parameters and low memory constraints. We will show it is possible to interpret the weights of the first layer of the perceptron

network as a set of learned filters for the FDK algorithm. We can then use the FDK algorithm to evaluate the network efficiently for all voxels simultaneously to arrive at an accurate reconstruction for the CCB CT problem.

The NN-FDK algorithm is an extension of the method proposed in [PB13] for the Filtered Backprojection (FBP) algorithm [Nat01]. The derivation of the approach outlined in [PB13] relies on the shift-invariance property of the FBP algorithm. We will show that, although the FDK algorithm does not have this shift-invariance property, we can derive a similar method for the FDK algorithm. Moreover, the proposed strategy can be extended to any linear filtered backprojection type reconstruction method.

Using both simulated and experimental data, we compare the proposed method with the standard FDK algorithm, SIRT [VV90] with a nonnegativity constraint (SIRT^+), which is a commonly used iterative algorithm for CT problems, and two 2D deep neural networks (U-net [RFB15] and MSD [PBS18]) trained to remove reconstruction artifacts from slices of standard FDK reconstruction. We show that the NN-FDK algorithm is faster to evaluate than all but the standard FDK algorithm and orders of magnitude faster to train than the considered DNNs, with only a slight reduction in reconstruction accuracy compared to the DNNs.

The chapter is structured as follows. In **Section 4.3** we give definitions and introduce our method. In **Section 4.4** we introduce the data and the parameters used for the experiments. The experiments and their results are shown and discussed in **Section 4.5**. The chapter is summarized and concluded in **Section 4.6**.

4.2 Related work

Using *machine learning* methods is an emerging approach in CT imaging [Wan+18]. *Deep learning* methods have shown promising results for many applications within the development of CT reconstruction methods [KMY17]. For the sake of exposition, we split these machine learning approaches into two categories: (i) Improving standard reconstruction methods by replacing components of the reconstruction method with networks specifically trained for the application; and (ii) improving the image quality of reconstructions computed with existing reconstruction methods by training neural networks to perform *post-processing* in order to remove artifacts or reduce noise.

Examples of the first strategy (improving standard reconstruction methods) applied to iterative methods are the learned primal-dual reconstruction algorithm [AÖ17; AÖ18], variational networks [Kob+17; Ham+18], plug and play priors [VBW13; REM17; RS18], and learned regularizers [LÖS18; Muk+20]. These methods achieve promising results in reconstruction accuracy and generalizability.

However, their high computational cost limits the applicability if high throughput is required. Examples for this strategy applied to direct methods are the NN-FBP method [PB13], and also the NN-FDK method introduced in this chapter. These methods are designed to improve the image quality of direct methods for data with limitations (*e.g.*, data with noise or a low number of projection angles) while maintaining their computational efficiency.

Examples of the second strategy (learned post-processing) have demonstrated substantial improvements in reconstruction quality for CT imaging [RFB15; KMY17; PS18; Jin+17]. This is aided by the fact that the post-processing problem can be viewed as a classic imaging problem — *e.g.*, denoising, segmentation, inpainting, classification — for which many effective machine learning methods have already been developed [SLD17; PCC18; Zha+17]. Although the general trend is towards deeper networks to make such networks more expressive [YHC18], this can lead to problems with scalability for large 3D image datasets.

The rise in popularity of machine learning in CT is driven by the increased computational possibilities and although these advances are sufficient to handle most 2D problems, scaling towards 3D problems can be problematic, due to memory constraints. This is illustrated in **Figure 4.5** in **Section 4.5.1**, where we plotted the memory constraints for applying a 2D and 3D U-net and MSD network in terms of gigabytes (GiB) of memory as a function of the size of the image. This shows that in theory one could apply a 2D MSD network to images of 7500×7500 pixels (with a 24GiB GPU), but in 3D this limit lies around $400 \times 400 \times 400$ voxels. Considering that CT problems range between $256 \times 256 \times 256$ (small image size) up to $4096 \times 4096 \times 4096$ images, this gives an indication that scalability can become an issue, especially for 3D problems.

When applying machine learning techniques for improving the reconstruction quality in CT, a balance must be struck between image quality, running time, and memory requirements. Here we propose a method that achieves relatively high accuracy, while also being computationally efficient and scalable.

4.3 Method

The NN-FDK algorithm is a reconstruction algorithm with a machine learning component, meaning that a number of parameters of the reconstruction algorithm are optimized through *supervised learning* [AB09]. Similar to the network presented in [PB13], the *NN-FDK network* is a two layer neural network with a hidden layer and an output layer. We design the network such that it reconstructs one single voxel, but handles all voxels in a similar manner. This means that we only have to train one network for a full reconstruction. We consider the NN-FDK algorithm to

have three parts: The *NN-FDK network*, the *NN-FDK reconstruction algorithm* and the *training process*.

We introduce the reconstruction problem, FDK algorithm, a filter approximation method and the definition of a perceptron in **Section 4.3.1**. In **Section 4.3.2** we give the *NN-FDK reconstruction algorithm* and derive from this algorithm the *NN-FDK network*. The input of the network that is needed in the *training process* is a pre-processed version of the input of the reconstruction algorithm. In **Section 4.3.3**, we discuss how to compute this pre-processing step for all voxels simultaneously and we introduce the optimization problem and related notation for the training process. Lastly, we summarize and discuss the characteristics of the method in **Section 4.3.4**.

4.3.1 Preliminaries

Reconstruction problem

In this chapter we focus exclusively on the circular cone-beam (CCB) geometry, where the object rotates with respect to a point source and a planar detector, acquiring 2D cone-beam projections. The reconstruction problem for the CCB geometry can be modeled by a system of linear equations

$$W\mathbf{x} = \mathbf{y}, \tag{4.1}$$

where $\mathbf{x} \in \mathbb{R}^n$ is the vector describing the reconstruction (*i.e.*, every element coincides with a voxel value), $\mathbf{y} \in \mathbb{R}^m$ is the vector describing the measured projection data, and $W \in \mathbb{R}^{m\times n}$ is a discretized version of the *cone-beam transform* or *forward projection*. For the sake of simplicity we assume that the volume consists of $n = N\times N\times N$ voxels and the detector consists of $N\times N$ pixels. We denote the number of angles with N_a, so we have $m = N_a \times N \times N$.

FDK algorithm & filter approximation

The FDK algorithm, as presented in [FDK84], is a filtered backprojection-type algorithm that solves the CCB reconstruction problem (4.1) approximately. First, for each projection angle, it applies a *reweighting* step, $r : \mathbb{R}^{N_a\times N\times N} \to \mathbb{R}^{N_a\times N\times N}$, that adapts the cone-beam data such that it approximately behaves as fan-beam data. Second, it applies a *filtering* step, that convolves the data with a one-dimensional *filter* $\mathbf{h}$ in a line-by-line fashion, $(-*-)_{1\mathrm{D}} : \mathbb{R}^{2N} \times \mathbb{R}^{N_a\times N\times N} \to \mathbb{R}^{N_a\times N\times N}$. Last, it applies a *backprojection* step. This step transforms the filtered projection data to the image domain. Using the notation of (4.1), the FDK algorithm is given

by

$$\mathrm{FDK}(\mathbf{y}, \mathbf{h}) = W^T(\mathbf{h} * r(\mathbf{y}))_{1\mathrm{D}}, \tag{4.2}$$

with W^T the transpose of W. The operator W^T is also known as the *backprojection operator*.

In [PB13; PB14; Lag+20] exponential binning is used to approximate filters, leading to $N_e \approx \log N$ coefficients to describe a filter. This approximation can be seen as a matrix $E \in \mathbb{R}^{2N \times N_e}$ applied to a coefficient vector $\mathbf{h}_e \in \mathbb{R}^{N_e}$:

$$\mathbf{h} \approx E\mathbf{h}_e. \tag{4.3}$$

The implementation details of this filter approximation can be found in [Lag+20].

Perceptron

In a similar manner as in [Bis06] we define a *perceptron* or *node* $\mathsf{P} : \mathbb{R}^l \to \mathbb{R}$ as a non-linear activation function $\sigma : \mathbb{R} \to \mathbb{R}$ applied to a weighted sum of the input $\eta \in \mathbb{R}^l$ with the weights $\xi \in \mathbb{R}^l$ and a bias $b \in \mathbb{R}$:

$$\mathsf{P}_{\xi,b}(\eta) = \sigma(\eta \cdot \xi - b) \tag{4.4}$$

In this chapter we will only consider the sigmoid function as activation function, *i.e.*, $\sigma(t) = 1/(1 + e^{-t})$.

A multilayer perceptron is a network structure containing two types of layers with perceptrons, where each perceptron operates on the outputs of the previous layer. These layers are, in order, any number of *hidden layers*, and the *output layer*. Note that the number of hidden layers and number of hidden nodes N_h in these layers can be chosen freely.

4.3.2 Reconstruction algorithm & Network design

We formulate the NN-FDK reconstruction algorithm in a similar fashion as the NN-FBP method in [PB13]. See **Algorithm 3** for a schematic representation. The NN-FDK reconstruction algorithm consists of N_h individual FDK algorithms executed on the input data y, each using its own (exponentially binned) filter $\mathbf{h}_e^k \in \mathbb{R}^{N_e}$. It combines these N_h volumes into a single reconstruction, using point-wise application of the activation function σ and an output perceptron with parameters $b_o, b_k \in \mathbb{R}$, and $\xi \in \mathbb{R}^{N_h}$.

We use $\theta = (\xi, b_o, \mathbf{h}_e^k, b_k)$ as short-hand for the full set of parameters of the NN-FDK reconstruction algorithm. The full algorithm is then given by the following equation.

$$\text{NN-FDK}_\theta(\mathbf{y}) = \sigma\Big(\sum_{k=1}^{N_h} \xi_k \sigma\left(\text{FDK}(\mathbf{y}, E\mathbf{h}_e^k) - b_k\right) - b_o\Big) \tag{4.5}$$

The FDK algorithm is a bilinear map in the input projection data and the used filter. Therefore, for fixed input projection data $\mathbf{y}$ and an expanded exponentially binned filter $E\mathbf{h}_e$, the FDK algorithm can be written as a linear map $F_\mathbf{y}$ applied to $E\mathbf{h}_e$. The product $F_\mathbf{y}E$ can be considered as a matrix of size $N^3 \times N_e$, and the v-th voxel of the output of the FDK algorithm is given by the inner product of $\mathbf{h}_e$ with $(F_\mathbf{y}E)_{v:}$, the v-th row of the matrix $F_\mathbf{y}E$. Using the definition of a perceptron (4.4) we can show the following:

$$(\text{NN-FDK}_\theta(\mathbf{y}))_v = \sigma\Big(\sum_{k=1}^{N_h} \xi_k \sigma\left((F_\mathbf{y}E\mathbf{h}_e^k)_v - b_k\right) - b_o\Big), \tag{4.6}$$

$$= \sigma\Big(\sum_{k=1}^{N_h} \xi_k \sigma\left((F_\mathbf{y}E)_{v:}\mathbf{h}_e^k - b_k\right) - b_o\Big), \tag{4.7}$$

$$= \mathsf{P}_{\xi,b_o}\left(\left[\mathsf{P}_{\mathbf{h}_e^k,b_k}((F_\mathbf{y}E)_{v:})\right]_k\right). \tag{4.8}$$

Therefore, we define the two-layer perceptron network $\mathsf{N}_\theta : \mathbb{R}^{N_e} \to \mathbb{R}$:

$$\mathsf{N}_\theta(\mathbf{q}) = \mathsf{P}_{\xi,b_0}\left(\left[\mathsf{P}_{\mathbf{h}_e^k,b_k}(\mathbf{q})\right]_k\right). \tag{4.9}$$

This is our NN-FDK network, and as we derived above, it has the following relationship with the NN-FDK reconstruction algorithm:

$$\mathsf{N}_\theta((F_\mathbf{y}E)_{v:}) = (\text{NN-FDK}_\theta(\mathbf{y}))_v. \tag{4.10}$$

This relationship shows that we can *evaluate* the NN-FDK reconstruction algorithm efficiently on full input projection data at once, but also *train* the NN-FDK network efficiently with each *individual* voxel $(\mathbf{x}_{\text{HQ}})_v$ in a high quality reconstruction yielding a training pair with input $(F_\mathbf{y}E)_{v:}$ and target $(\mathbf{x}_{\text{HQ}})_v$. A schematic representation of the network is given in **Figure 4.1**.

Note that we arrive at the same network structure as found in [PB13] for FBP, using only the properties that the FDK algorithm is a bilinear map in the data and the filter, and that all operations can be applied point-wise. Using this reasoning we can derive a similar network structure for any FBP-type method satisfying these conditions.

Even though we use the same network structure as [PB13], the way we compute inputs to the network is different. In [PB13], the input to the NN-FBP network is

explicitly calculated by shifting and adding projection data for each reconstruction pixel. The FDK algorithm has additional weighting factors and lacks the shift-invariance property, which makes the approach presented in [PB13] not directly applicable. In the next section, we detail an alternative method to compute the input. The same approach could be applied to the NN-FBP method, similarly simplifying the network input computations.

Algorithm 3 Neural Network FDK reconstruction algorithm

1: Given a set of parameters, $\theta := \left(\xi, b_o, \mathbf{h}_e^k, b_k\right)$.
2: Compute H_k for all nodes k of the hidden layer:
3: **for** $k = \{1, 2, .., N_h\}$ **do**
4: $\quad \mathrm{H}_k(\mathbf{y}) = \sigma\left(\mathrm{FDK}(\mathbf{y}, E\mathbf{h}_e^k) - b_k\right)$
5: Compute the output of the output layer:
$\mathrm{NN\text{-}FDK}_\theta(\mathbf{y}) = \sigma\left(\sum_{k=1}^{N_h} \xi_k H_k(\mathbf{y}) - b_o\right)$

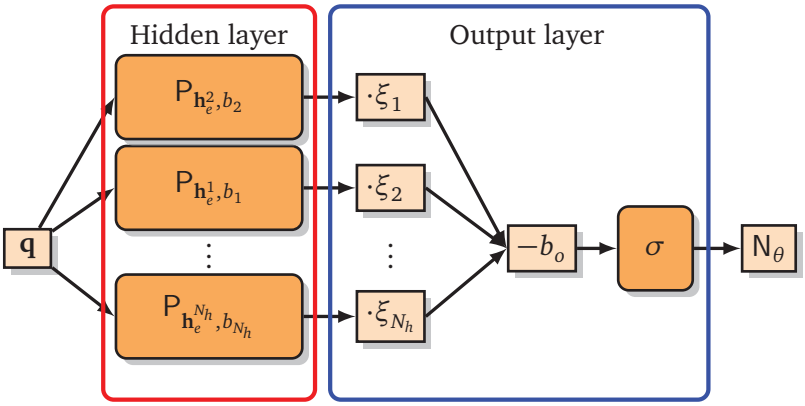


Figure 4.1: Schematic representation of the NN-FDK network, $\mathrm{N}_\theta : \mathbb{R}^{N_e} \to \mathbb{R}$, with N_h hidden nodes. Note that if we take $q = (F_{\mathbf{y}}E)_v$: we get $q \cdot \mathbf{h}_e^k = (\mathrm{FDK}(\mathbf{y}, E\mathbf{h}_e^k))_v$ in the perceptrons of the hidden layer and the output of the network is equal to the v-th voxel of the NN-FDK reconstruction algorithm.

4.3.3 Training process

Training and validation data

We will train our network using supervised learning, where we assume that we have N_{TD} and N_{VD} datasets available for training and validation, respectively.

These datasets consist of low quality tomographic input data and a high quality reconstruction from which we randomly draw a total of N_{T} training pairs and N_{V} validation pairs. Note that we ensure that every drawn pair is unique and that an equal number of pairs is taken from each dataset. Moreover, to avoid selecting too many training pairs from the background we only take training pairs from a region of interest (ROI) around the scanned object. This ROI is defined from the high quality reconstruction as the voxels in the reconstructed object plus a buffer of roughly $0.2N$ voxels around it.

Recall from the previous section that given low quality tomographic data $\mathbf{y}$ and a high quality reconstruction $\mathbf{x}_{\mathrm{HQ}}$ the matrix $F_{\mathbf{y}}E$ contains each input vector $Z = \left(F_{\mathbf{y}}E\right)_{v:} \in \mathbb{R}^{N_e}$ corresponding to the target voxel $O = (\mathbf{x}_{\mathrm{HQ}})_v$. However, due to memory constraints $F_{\mathbf{y}}E$ cannot be computed directly as a matrix product. Therefore, we observe that each column of $F_{\mathbf{y}}E$ is an FDK reconstruction with a specific filter:

$$(F_{\mathbf{y}}E)_{:j} = F_{\mathbf{y}}E\mathbf{e}_j = \mathrm{FDK}(\mathbf{y}, E\mathbf{e}_j), \tag{4.11}$$

with $\mathbf{e}_j \in \mathbb{R}^{N_e}$ the unit vector with all entries equal to zero except for the j-th element.

Learning problem

The parameters of the NN-FDK network are learned by finding the set of parameters $\theta^\star$ that minimize the *loss function* $\mathscr{L}$ on the training set. We minimize the ℓ^2-distance between the network output and the target voxel for all training pairs in T:

$$\theta^\star = \operatorname*{argmin}_\theta \mathscr{L}(\theta, T) = \operatorname*{argmin}_\theta \tfrac{1}{2} \sum_{j=1}^{N_{\mathrm{T}}} \left(O_j - \mathsf{N}_\theta(Z_j)\right)^2. \tag{4.12}$$

To minimize the loss function we use a quasi-Newton optimization scheme, the *Levenberg-Marquardt algorithm* (LMA) as proposed in [Lev44; Mar63]. This is a combination of gradient descent and the Gauss-Newton algorithm, improving the stability of Gauss-Newton while retaining its fast convergence and it is specifically designed to minimize a non-linear least squares problem such as (4.12). Note that the small number of parameters of the proposed network allows us to use such a method. Lastly, to avoid overfitting we check whether every update of the parameters also reduces the loss function on the validation set. We discuss the specifics of this algorithm in **Appendix 4.7.2**.

Method comparison: Goals

	Reconstruction		Training	
Method	Time	Accuracy	Data	Time
NN-FDK	++	?	++	+++
DNN	±	+++	±	- - -
FDK	+++	- -		
SIRT$^+$	- -	+		

Table 4.1: Comparison of reconstruction methods with respect to the goals formulated in **Section 4.1**. We consider a DNN to be 2D deep convolutional neural network (U-net & MSD-net) applied in slice-by-slice fashion to a standard FDK reconstruction. Reconstruction accuracy is defined as the accuracy of a method when reconstructing low quality data, *e.g.*, data with high noise or a low number of projection angles.

4.3.4 Method characteristics & comparison

To conclude the method section we compare the characteristics of the NN-FDK algorithm to those of several other methods. These methods are two 2D post-processing DNNs (U-net [RFB15] and MSD-net [PS18]) applied in a slice-by-slice fashion, the SIRT$^+$ algorithm [VV90] and the FDK algorithm. We focus our discussion on the goals formulated in **Section 4.1** and show a summary of this comparison in **Table 4.1**. The reconstruction accuracy will be discussed in **Section 4.5**.

Computational efficiency

We approximate the reconstruction time by counting how many times it has to evaluate its most expensive computations. For simplicity we assume that a backprojection takes approximately the same time as a forward projection, T_{BP}.

- **FDK:** The FDK algorithm consist of one reweighting, filtering and backprojection step, *i.e.*, :

$$T_{\text{FDK}} \approx T_{\text{BP}}. \tag{4.13}$$

- **NN-FDK:** The NN-FDK algorithm performs one FDK reconstruction per hidden node N_h. Therefore the reconstruction time becomes:

$$T_{\text{NN-FDK}} \approx N_h T_{\text{BP}}. \tag{4.14}$$

- **SIRT$^+$:** The SIRT$^+$ method evaluates a forward and backprojection for each iteration. For N_{iter} iterations, the reconstruction time becomes:

$$T_{\text{SIRT}^+} \approx 2N_{\text{iter}} T_{\text{BP}}. \tag{4.15}$$

- **DNN:** To evaluate a DNN an FDK reconstruction is performed and a 2D network is applied per slice of the FDK reconstruction.

$$T_{\text{DNN}} \approx T_{\text{BP}} + N T_{\text{DNN}}, \tag{4.16}$$

  with T_{DNN} the time it takes to apply a 2D DNN.

On a modern GPU and with $N = 1024$ and $N_a = 360$, we found in our experiments that $T_{\text{BP}} \approx 10$ s and $T_{\text{DNN}} \approx 0.5$ s.

Comparing the reconstruction times, we see that NN-FDK is similar to FDK when the number of nodes N_h is small, which is the case since we will take N_h=4 (see **Section 4.4.3**). For DNNs the computational load of applying a 2D network leads to relatively high reconstruction times compared to the FDK algorithm. Lastly, we note that the number of iterations N_{iter} often lies between the 20 and 200, making SIRT$^+$ several times slower than the (NN-)FDK algorithm.

Number of trainable parameters

The number of trainable parameters is closely related to the amount of training data required to train a network [PS18]. From the definition of the NN-FDK network (4.5) we can compute the number of trainable parameters $|\theta|$:

$$|\theta| = (N_e + 2)N_h + 1, \tag{4.17}$$

with $N \gg N_h, N_e > 0$. Taking $N_h = 4$ and $N = 1024$ gives $|\theta| = 61$, which is several orders of magnitude lower than the typical numbers of parameters in a DNN (several tens of thousands to millions).

Training time

In the training step a solution to the minimization problem (4.12) is computed. For the NN-FDK algorithm this problem has N_{T} samples and $|\theta|$ unknowns. In a similar fashion we can formulate a least squares problem for training a DNN. Even assuming that we only take the same number of training samples to train the DNNs, this least squares problem is already orders of magnitude larger than that for NN-FDK due to the difference in the number of trainable parameters. Moreover, the LMA (the algorithm used to train NN-FDK) approaches quadratic convergence,

which means it will need fewer iterations to converge than a first order scheme such as ADAM [KB14], which is often used for training DNNs. Considering these two observations we expect the training time of the NN-FDK algorithm to be lower than the training time of the DNNs.

4.4 Experimental setup

We carried out a range of experiments to assess the performance of the NN-FDK algorithm with respect to the goals formulated in **Section 4.1** compared to several alternative methods. In this section we introduce the setup of these experiments. We describe the simulated data in **Section 4.4.1** and the experimental data in **Section 4.4.2**. In **Section 4.4.3** we discuss the specific network structure for the NN-FDK algorithm and the training parameters used. Finally, we give the quantitative measures we use to compare the reconstruction in **Section 4.4.4**.

4.4.1 Simulated data

We consider two types of phantom families for the simulated data experiments: the *Fourshape phantom family* and the *Random Defrise phantom family*. Examples are shown in **Figure 4.2** and **Figure 4.3**, respectively. The Fourshape phantom family contains three random occurrences of each of four types of objects: an ellipse, a rectangle, a Gaussian blob and a Siemens star. For evaluation and visualization of the reconstructions we fixed one realization that clearly shows at least one of all the four objects and we will refer to this phantom as the *Fourshape test phantom*. The Random Defrise phantom family is a slight adaptation of the phantom introduced in [KND98], which is a common phantom for assessing the influence of imaging artifacts due to the cone angle. Here we vary the intensities, orientations and sizes of the disks making sure they do not overlap. Again, we define a test phantom for evaluation and visualization, which is in this case the standard *Defrise phantom* without alternating intensities (right in **Figure 4.3**). To simulate realistic settings, we scale the phantoms to fit inside a 10 cm cube, and use an attenuation coefficient of $\mu = 0.22\ \text{cm}^{-1}$, approximating that of various common plastics at 40 keV [HS95]. These phantoms are defined through geometric parameters, and can therefore be generated for any desired N. For our experiments we will take $N = 1024$. Details about how we generate the data are given in **Appendix 4.7.1**.

To compute a high quality reconstruction $\mathbf{x}_{\text{HQ}}$ that can be used as target for training (recall **Section 4.3.3**) we consider a simulated dataset with $N_a = 1500$ projection angles, low noise ($I_0 = 2^{20}$ emitted photon count) and cone angle of 0.6 degrees and reconstruct this problem with the standard FDK algorithm using a

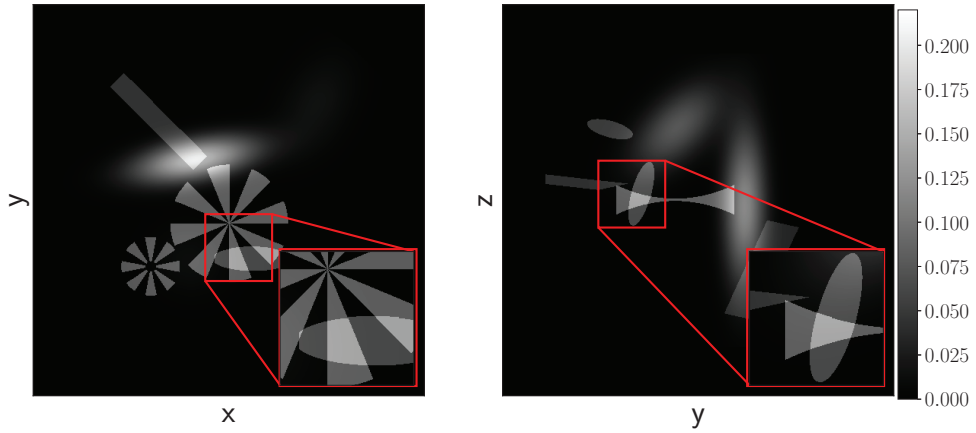


Figure 4.2: Slices, (Left) $z = 0$, (Right) $x = 0$, of the Fourshape test phantom. This phantom is designed such that at least one of all objects can clearly be observed in the slices.

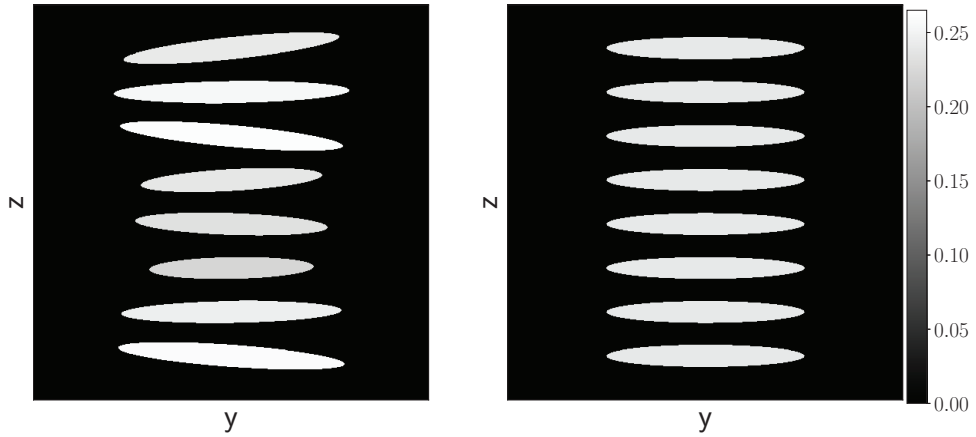


Figure 4.3: The $x = 0$ slice for a Random Defrise phantom (Left) and the standard Defrise phantom without alternating intensities from [KND98] (Right).

Hann filter [Nat01].

4.4.2 Experimental data

For experimental data we consider a set of CT scans that were recorded using the custom-built and highly flexible FleX-ray CT scanner, developed by XRE NV and located at CWI [Cob+20]. This scanner has a flat panel detector with 972 × 768

pixels and a physical size of 145.34×114.82 mm. This set of 42 scans was set up to create high noise reconstruction problems and low noise reconstruction problems with a low number of projection angles.

We acquired high-dose (low noise) and low-dose (high noise) scans of 21 walnuts. The datasets contain 500 equidistantly spaced projections over a full circle. The distance from the center of rotation to the detector was set to 376 mm and the distance from the source to the center of rotation was set to 463 mm. The scans were performed with a tube voltage of 70 kV. The high-dose scan was collected with a tube power of 45 W and an exposure time of 500 ms per projection. The low-dose scan was collected with a tube power of 20 W and an exposure time of 100 ms per projection. To create a low noise reconstruction problem with a low number of projection angles we considered the high-dose scan but only took every 16-th projection angle. As high quality reference reconstructions we used SIRT^+ reconstructions with 300 iterations (SIRT^+_{300}) of the high-dose scans with all available projection angles ($N_a = 500$). We will refer to these reconstructions as the *gold standard* reconstruction and we show such a reconstruction in **Figure 4.4**. These datasets are available at Zenodo [LCB20].

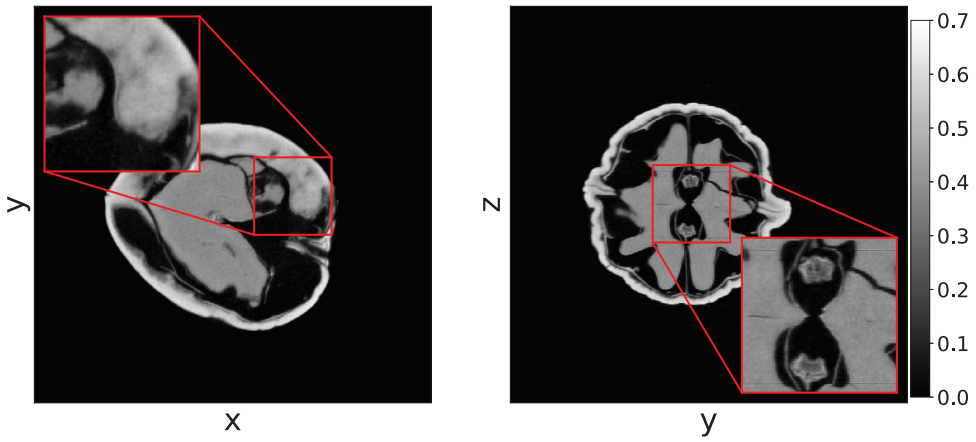


Figure 4.4: The $z = 0$ (Left) and $y = 0$ (Right) slice of the gold standard reconstruction of the high-dose dataset of the 21^{st} walnut with full number of projection angles. The projection data is acquired using the FleX-ray scanner located at the CWI [LCB20].

4.4.3 Parameter settings NN-FDK

Network structure

In our initial experiments we found that taking more FDK-perceptrons improved the accuracy of the networks, at the cost of increasing the training and reconstruction time. We found that $N_h = 4$ FDK-perceptrons led to a good balance between accuracy and reconstruction time, which is similar to the findings in [PB13].

Training data

We found that, similar to the findings in [PB13], taking $N_\mathrm{T} = 10^6$ voxels for training and $N_\mathrm{V} = 10^6$ for validation is sufficient for training an NN-FDK network.

The network structures and training procedure used for the U-nets and MSD networks are discussed in **Appendix 4.7.1**.

4.4.4 Quantitative measures

To quantify the accuracy of the reconstructions we consider two measures, the test set error (TSE) and the structural similarity index (SSIM). These measures compare the reconstructed image $\mathbf{x}_r$ to a high quality reconstruction $\mathbf{x}_\mathrm{HQ}$ on the ROI (as discussed in **Section 4.3.3**).

The TSE is the average loss[1] of the test set, where the test set is all the voxels defined in the ROI of $\mathbf{x}_\mathrm{HQ}$:

$$\mathrm{TSE}(\mathbf{x}_r, \mathbf{x}_\mathrm{HQ}) = \frac{1}{N_\mathrm{ROI}} \mathscr{L}(\mathscr{I}_\mathrm{ROI}(\mathbf{x}_\mathrm{HQ}), \theta), \tag{4.18}$$

$$= \frac{1}{2N_\mathrm{ROI}} \left\| \mathscr{I}_\mathrm{ROI}(\mathbf{x}_\mathrm{HQ} - \mathbf{x}_r) \right\|_2^2. \tag{4.19}$$

with $\mathscr{I}_\mathrm{ROI} : \mathbb{R}^{N^3} \to \mathbb{R}^{N^3}$ the masking function for the ROI and N_ROI the number of voxels in the ROI.

The SSIM [Wan+04] is implemented based on the scikit-image 0.13.1 [Wal+14] package, where all the constants are set to default and the filter is uniform with a width of 19 pixels.

[1] Recall (4.12) in **Section 4.3.3**

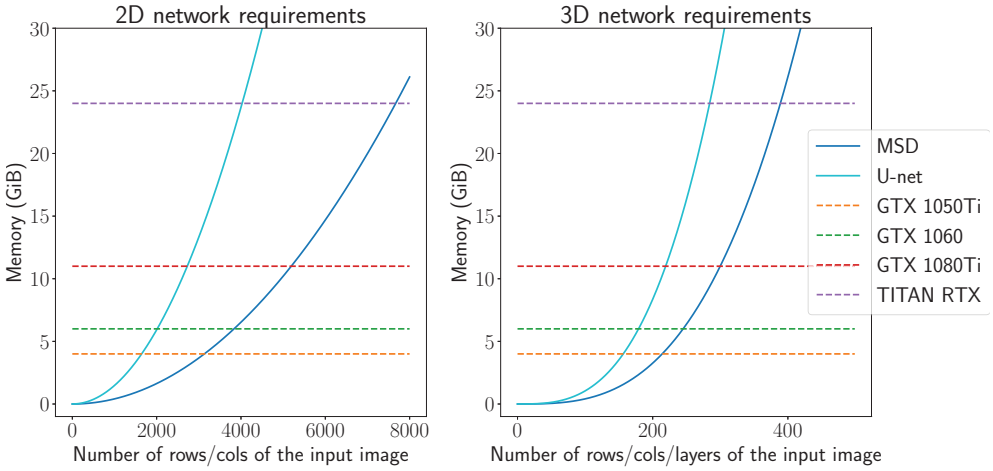


Figure 4.5: The required memory to store all intermediate images for applying a 2D and 3D U-net and MSD network as a function of the input image size.

4.5 Results and discussion

4.5.1 Scalability

Memory scaling

The required memory to store all intermediate images for a forward pass of a 2D or a 3D U-net and MSD network as a function of the input image size is shown in **Figure 4.5**. Considering that CT imaging problems typically range from $256 \times 256 \times 256$ up to $4096 \times 4096 \times 4096$ we conclude from these figures that full 3D networks do not fit into GPU memory for higher resolutions and that even for 2D U-nets not all resolutions fit on the GPU. As a forward pass of the NN-FDK algorithm requires only one additional reconstruction volume[2] compared to the FDK algorithm, the memory requirements of the NN-FDK algorithm are roughly 2 times the memory required by the FDK algorithm.

Training time

In **Figure 4.6** we compare the training processes by plotting the progress of the network training (measured by the TSE) as a function of the number of voxels that the network has seen during training. We see that the NN-FDK has seen $1.1 \cdot 10^8$

[2]Technically a forward pass of the NN-FDK algorithm can be done for every voxel separately, however, for the sake of comparison we assume a forward pass is for a full reconstruction volume.

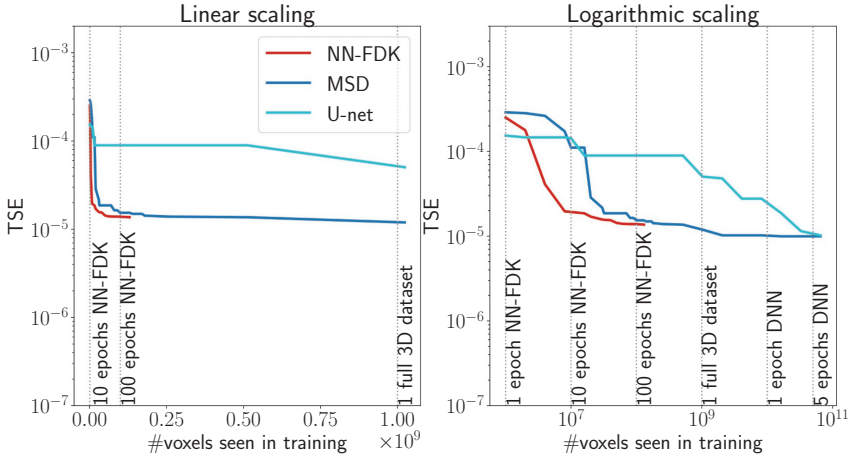


Figure 4.6: The TSE as a function of the number of voxels the training process has seen. We report the lowest TSE up till that point. The networks are trained on randomly generated Fourshape phantoms with size $N = 1024$, $N_a = 32$ projection angles and no noise. (Left) Linear scaling in the number of voxels ranging from 1 epoch for the NN-FDK (10^6 voxels), to 1 full 3D dataset (10^9 voxels). (Right) Logarithmic scaling in the number of voxels. Ranging from 1 epoch for the NN-FDK network (10^6 voxels) to 5 epochs for a DNN ($5 \cdot 10^{10}$ voxels).

voxels when it converges to TSE$= 1.4 \cdot 10^{-5}$, whereas, MSD and U-net have seen $5.1 \cdot 10^8$ voxels and $3.2 \cdot 10^9$ voxels, respectively, at the point they first achieve a similar TSE. Important to note is that both U-net and MSD are not yet converged when they match the TSE of NN-FDK, and in general the DNNs achieve lower TSEs than NN-FDK.

In **Table 4.2** we show various timings and properties with respect to the training process. These timings are recorded using one Nvidia GeForce GTX 1080Ti with 11GiB memory. We define a converged training process as 100 epochs without improvement on the validation set error and the number of epochs to converge as the epoch with the lowest validation set error during a converged training process. From these results we see that the size of the training problem influences the time per epoch as an NN-FDK epoch is sub-second and the time per epoch for DNNs is in the range of hours.

In practice, we observed that after 2 days of training for the DNNs, any additional training only achieved marginal improvements. Therefore, in the following experiments we train all DNNs for 2 days with one Nvidia GeForce GTX 1080Ti GPU, unless mentioned otherwise.

Training process

	NN-FDK$_4$	MSD	U-net
Voxels seen in one epoch	$1 \cdot 10^6$	$1.1 \cdot 10^{10}$	$1.1 \cdot 10^{10}$
Time per epoch	0.1336 (s)	0.95 (h)	2.36 (h)
Time to converge	28 (s)	± 10 (d)	± 14 (d)
Epochs to converge	110	128	42
Epochs in 2 days	-	45	18

Table 4.2: Timings and properties of the considered training processes. We define a converged training process as 100 epochs without improvement on the validation set error. The epochs to converge is therefore the epochs computed of such a process minus 100. The training was performed using one Nvidia GeForce GTX 1080Ti GPU (11 GiB).

Reconstruction time

We measured the average reconstruction times and corresponding standard deviation over 120 reconstructions with resolution $N^3 = 1024^3$ and $N_a = 360$ projection angles. These reconstructions are computed using one Nvidia GeForce GTX 1080Ti with 11 GiB memory. The results are shown in **Table 4.3**. We define the reconstruction time as the time it takes to compute the full 3D volume. This means for U-net and MSD, an FDK reconstruction needs to be computed and the network needs to be applied $N = 1024$ times to a 2D slice. Although every application can be done within a second (U-net $\approx 0.3s$, MSD $\approx 0.7s$) this leads to long reconstruction times.

Reconstruction times

FDK	SIRT$^+_{200}$	NN-FDK$_4$	U-net	MSD
28 ± 8	3225 ± 916	76 ± 8	382 ± 69	809 ± 86

Table 4.3: Average and standard deviation of the reconstruction times (in seconds) computed over 120 reconstruction problems with $N = 1024$ and $N_a = 360$ projection angles. These reconstructions are computed using one Nvidia GeForce GTX 1080Ti GPU (11 GiB).

4.5.2 Reconstruction accuracy for simulated data

For evaluating the reconstruction accuracy using simulated data, we consider 16 cases: 6 different noise levels, 5 different numbers of projection angles and 5

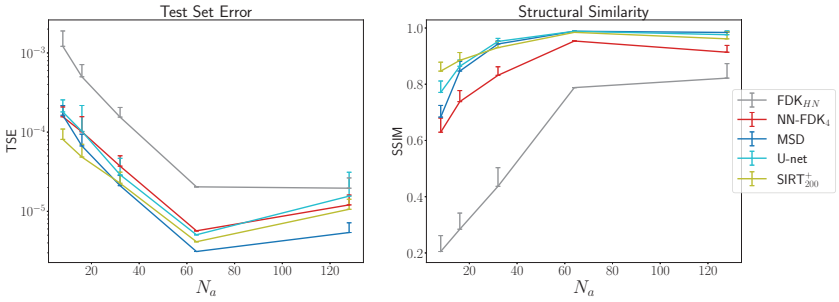
different cone angles. For each case an NN-FDK, MSD and U-net network was trained. For the training process of NN-FDK we used $N_{\mathrm{T}} = 10^6$ training voxels and $N_{\mathrm{V}} = 10^6$ validation voxels from $N_{\mathrm{TD}} = 10$ and $N_{\mathrm{VD}} = 5$ datasets, respectively. For U-net and MSD we took the same datasets for training and validation (10 for training and 5 for validation), and used all voxels in these datasets for the training process. The NN-FDK networks were trained till convergence and the DNNs were trained for 48 hours. Note that in a few cases we had to retrain the DNNs because of inconsistent results (*i.e.*, cases with more information achieving a lower reconstruction accuracy), possibly because they got stuck in local minima of the loss function.

In **Figure 4.7** we show the average and standard deviation of the TSE and the SSIM for the considered cases. We observe that U-net and MSD achieve the most accurate results and that NN-FDK and SIRT$^+$ closely follow. The FDK algorithm is lowest in all categories. Between NN-FDK and SIRT$^+$ we see that NN-FDK performs best for the noisy reconstruction problems and SIRT$^+$ achieves better results for the reconstruction problems without noise. We visualize the noise for the lowest and highest I_0 in **Figure 4.8** by showing a line profile through the center of the $z = 0$ slice. Here we see that for the noisiest problems the amplitude of the noise can be as high as the maximum value of the phantom. In **Figure 4.9** we show 2D slices of reconstructions of the test phantoms for the three types of reconstruction problems. In all cases we still observe reconstruction artifacts, but comparing these to the baseline FDK reconstructions, the majority is removed or suppressed.

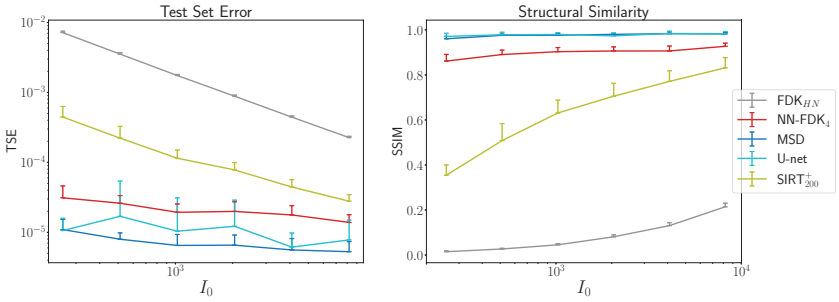
4.5.3 Reconstruction accuracy for experimental data

In this section we use the datasets discussed in **Section 4.4.2** to assess the reconstruction accuracy on experimental data. In a similar fashion as for the simulated data, we trained a network for the low-dose reconstruction problem and a network for the high-dose reconstruction problem with $N_a = 32$ projection angles with the notable exception that U-net and MSD were trained till convergence. The results are presented in **Table 4.4**.

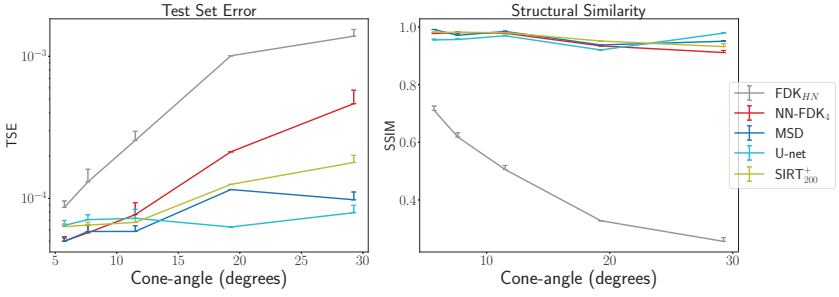
Comparing the results to the simulated data experiments we see that SIRT$^+$ performs worse on the experimental data, even with the additional regularization of early stopping. This is most likely due to the high-dose datasets still containing noise, whereas this is completely absent in the simulated data experiments. These differences are illustrated in **Figure 4.10** where 2D slices of the reconstructions for the high-dose reconstruction problem with $N_a = 32$ projection angles are shown.



(a) The average and standard deviation of the TSE and SSIM as a function of number of projection angles N_a computed over 20 randomly generated phantoms Fourshape family.



(b) The average and standard deviation of the TSE and SSIM as a function of the emitted photon count I_0 computed over 20 randomly generated phantoms of the Fourshape family.



(c) The average and standard deviation of the average TSE and SSIM as a function of the cone angle computed over 20 randomly generated phantoms of the Defrise family.

Figure 4.7: The average and standard deviation of the TSE and SSIM. These results are discussed in **Section 4.5.2**. For each number of projection angles, noise level, cone angle and training scenario one specific network is trained and used to evaluate the 20 reconstruction problems.

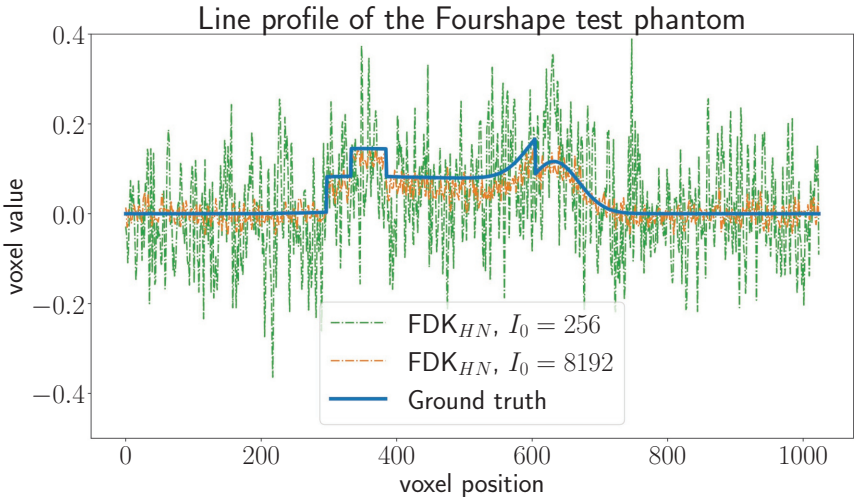


Figure 4.8: Line profile through the center of the $z = 0$ slice of the Fourshape test phantom. We show the ground truth profile, the profile of the FDK reconstruction with lowest emitted photon count $I_0 = 256$, and the profile of the FDK reconstruction with the highest emitted photon count $I_0 = 8196$.

	Experimental data			
	High-dose, low number of projection angles		Low-dose	
Method	TSE	SSIM	TSE	SSIM
FDK_{HN}	5.54±3.43e-03	0.224±0.076	1.40±0.05e-03	0.334±0.104
$SIRT^{+}_{200/20}$	9.94±0.15e-04	0.603±0.087	1.92±0.08e-03	0.584±0.083
NN-FDK_4	8.03±1.39e-04	0.946±0.010	1.14±0.23e-04	0.965±0.012
U-net	**4.10±1.06e-04**	**0.964±0.009**	1.02±0.45e-04	**0.980±0.006**
MSD	4.23±0.97e-04	**0.964±0.009**	**7.82±2.86e-05**	0.980±0.007

Table 4.4: Average and standard deviation of the quantitative measures computed over 6 walnut datasets. The high-dose low projection angle reconstruction problem has $N_a = 32$ projection angles, the low-dose reconstruction problem has $N_a = 500$ projection angles. The best results per experiment are highlighted.

4.5.4 Segmentation experiment for experimental data

To assess the performance of the different reconstruction approaches in a segmentation task, we focus here on the segmentation of the shell and kernel of walnuts, based on our experimental CT data. The review [Ber+20] provides an overview of segmentation problems in walnut imaging, and their relevance. For segmenting

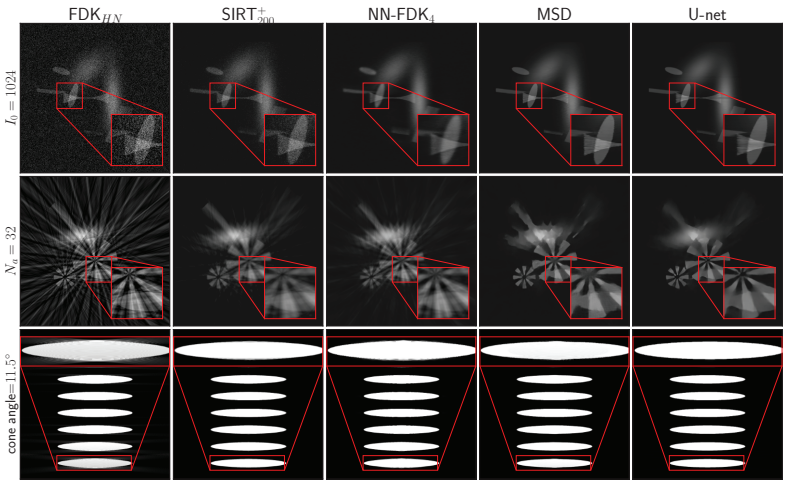


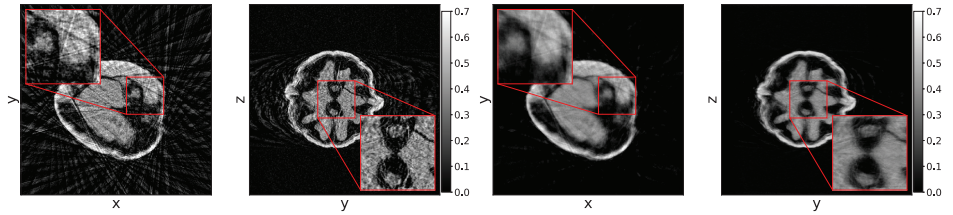
Figure 4.9: Two-dimensional slices of the reconstructions for the considered reconstruction methods. (Top) Slice $x = 0$ of the Fourshape test phantom reconstruction problem with $N_a = 360$ projection angles and $I_0 = 1024$ emitted photon count. (Middle) Slice $z = 0$ of the Fourshape test phantom reconstruction problem with $N_a = 32$ projection angles. (Bottom) Slice $x = 0$ of the Defrise reconstruction problem with $N_a = 360$ projection angles and a cone angle of 11.5 degrees.

the 3D volume after the reconstruction, we used a deterministic segmentation algorithm that combines thresholding, the watershed algorithm and prior knowledge of the scanned objects. Details of this method are discussed in **Appendix 4.7.1**.

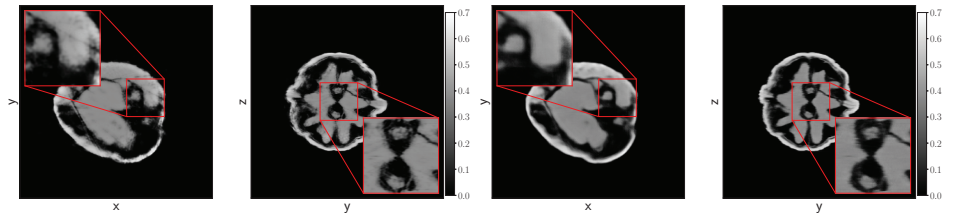
For determining the accuracy of the segmentation of an object — *i.e.*, shell, empty space and kernel of the walnut — we consider three metrics: volume error, mislabeled voxels and the Dice coefficient [Dic45]. We define a segmentation S as a reconstruction volume with value 1 if the voxel is in the object (shell, kernel or empty space) and 0 if outside the object. Furthermore we define the norm of a segmentation as the sum: $|S| = \sum_i^{N^3} (S)_i$. Using this notation we can compute the measures in the following manner:

$$V_{\mathrm{err}} = \frac{|S_{\mathrm{rec}}| - |S_{\mathrm{GS}}|}{|S_{\mathrm{GS}}|}, \qquad \mathrm{ML}_{\mathrm{err}} = \frac{|S_{\mathrm{rec}} - S_{\mathrm{GS}}|}{|S_{\mathrm{GS}}|}, \qquad \mathrm{DC} = \frac{2|S_{\mathrm{rec}} \cap S_{\mathrm{GS}}|}{|S_{\mathrm{rec}}| + |S_{\mathrm{GS}}|}, \tag{4.20}$$

with GS denoting the gold standard reconstruction.



(a) FDK_{HN} (b) SIRT^{+}_{200} reconstruction.



(c) NN-FDK_4 reconstruction. (d) MSD reconstruction.

Figure 4.10: Slices $z = 0$ and $x = 0$ of several reconstruction methods of the high-dose dataset of the 21^{st} walnut with 32 projection angles.

In **Table 4.5** we show the results for computing these metrics on the 6 walnuts not considered in the training process. We observe that MSD performs best in segmenting the shell and U-net performs best at segmenting the empty space and kernel and NN-FDK is close to both DNNs and in some cases even better than MSD for segmenting the empty space and kernel. Comparing NN-FDK to standard FDK we observe a significant improvement.

4.5.5 Data requirements

To test the influence of the amount of training data on the reconstruction quality we performed an experiment with three different training scenarios:

- **Scenario 1**. One dataset available. Here we take the training and validation data from the same dataset.
- **Scenario 2**. Two datasets available. Here we take the training and validation data from the separate datasets.
- **Scenario 3**. Fifteen datasets available. Again the training and validation data are picked from separate datasets, but now the training and validation pairs come from several datasets, specifically 10 training datasets ($N_{\text{TD}} = 10$) and 5 validation datasets ($N_{\text{VD}} = 5$). This is the scenario used in the previous experiments.

Segmentation errors			
Method	Shell	Empty space	Kernel
Volume errors			
FDK_{HN}	0.127 ± 0.078	0.146 ± 0.091	0.128 ± 0.092
$SIRT^{+}_{200}$	0.082 ± 0.047	0.104 ± 0.078	0.050 ± 0.074
$NN\text{-}FDK_4$	0.068 ± 0.035	0.045 ± 0.035	0.029 ± 0.032
U-net	0.055 ± 0.019	**0.029 ± 0.017**	**0.012 ± 0.016**
MSD	**0.028 ± 0.010**	0.059 ± 0.075	0.035 ± 0.050
Mislabeled voxels			
FDK_{HN}	0.168 ± 0.087	0.190 ± 0.98	0.144 ± 0.081
$SIRT^{+}_{200}$	0.133 ± 0.026	0.182 ± 0.118	0.101 ± 0.048
$NN\text{-}FDK_4$	0.103 ± 0.026	0.087 ± 0.023	0.072 ± 0.018
U-net	0.092 ± 0.028	**0.073 ± 0.024**	**0.059 ± 0.019**
MSD	**0.086 ± 0.038**	0.116 ± 0.094	0.061 ± 0.039
Dice coefficient			
FDK_{HN}	0.922 ± 0.036	0.895 ± 0.061	0.934 ± 0.033
$SIRT^{+}_{200}$	0.934 ± 0.016	0.908 ± 0.061	0.947 ± 0.028
$NN\text{-}FDK_4$	0.951 ± 0.012	0.955 ± 0.013	0.964 ± 0.008
U-net	0.955 ± 0.013	**0.963 ± 0.012**	**0.971 ± 0.010**
MSD	**0.957 ± 0.018**	0.939 ± 0.055	0.971 ± 0.018

Table 4.5: The average and standard deviation of the three metrics computed over the 6 low-dose walnut datasets with $N_a = 500$ projection angles. The metrics are computed using (4.20). The best results are highlighted.

We fix the number of voxels used for training and validation at $N_T = 10^6$ and $N_V = 10^6$ for all scenarios. For comparison we trained a U-net and a MSD network with the same training scenarios, with the exception that all voxels from the datasets are used. For training scenario 1 the slices are divided into a training and a validation set. More specifically, every fourth slice is used for validation.

We performed this experiment for two simulated data problems, a high noise level (emitted photon count $I_0 = 256$) and a large cone angle (29.3 degrees), and the two experimental data problems. For the sake of brevity we show only the results for the high noise simulated data reconstruction problem (**Table 4.6**) and the high noise experimental data reconstruction problem (**Table 4.7**). The results for the other reconstruction problems are given in **Appendix 4.7.3**. Comparing quantitative measures between the different scenarios we see that the reconstruction accuracy improves as more data is used for the simulated data experiment, but remains about the same for the experimental data experiment. This can be

explained by the variation in the objects used in the reconstruction problems. Recall that the Fourshape phantom family has a large variety in its phantoms, *i.e.*, three instances of four randomly generated objects, and the variety within the walnut datasets is small, *i.e.*, similar shapes, sizes and structures. This indicates that if objects are similar, one training dataset may already be sufficient to train networks that achieve a high reconstruction accuracy.

Note that although the training scenarios for NN-FDK and the DNNs use the same number of datasets, the number of voxels considered for training the NN-FDK network is constant over all three scenarios and is several orders of magnitude lower than the number of voxels considered for training the DNNs. This opens up future possibilities for reducing the training data requirements to only need a high quality reconstruction of a certain region of interest.

Simulated data, high noise

TSE			
Method	1 dataset	2 datasets	15 datasets
NN-FDK$_4$	4.97±4.68e-05	4.19±3.60e-05	2.51±1.14e-05
U-net	1.06±1.36e-05	2.45±2.87e-05	8.06±3.63e-06
MSD	1.12±0.41e-05	1.12±0.40e-05	**7.94±3.16e-06**
SSIM			
NN-FDK$_4$	0.831±0.065	0.844±0.065	0.884±0.030
U-net	0.884±0.075	0.932±0.050	**0.979±0.009**
MSD	0.961±0.013	0.962±0.013	0.974±0.008

Table 4.6: Average and standard deviation of the quantitative measures computed over 20 Fourshape phantoms for varying training scenarios. The reconstruction problems have an emitted photon count of $I_0 = 256$ and $N_a = 360$ projection angles. The best results are highlighted.

4.6 Summary and conclusion

We have proposed the Neural Network FDK (NN-FDK) algorithm, a reconstruction algorithm for the circular cone-beam (CCB) Computed Tomography (CT) geometry with a machine learning component. The machine learning component of the algorithm is designed to learn a set of FDK filters and to combine the FDK reconstructions done with these filters. This leads to a computationally efficient reconstruction algorithm, since one only needs to compute and combine the FDK reconstructions for this learned set of filters. Due to parametrization of

Experimental data, low-dose

TSE			
Method	1 dataset	2 datasets	15 datasets
NN-FDK$_4$	1.16±0.25e-04	1.23±0.25e-04	1.14±0.23e-04
U-net	1.27±0.38e-04	1.23±0.35e-04	1.02±0.45e-04
MSD	1.28±0.41e-04	1.16±0.35e-04	**7.82±2.86e-05**
SSIM			
NN-FDK$_4$	0.973±0.009	0.968±0.011	0.965±0.012
U-net	0.979±0.008	0.978±0.008	**0.980±0.006**
MSD	0.979±0.008	0.979±0.008	0.980±0.007

Table 4.7: Average and standard deviation of the quantitative measures computed over 6 walnuts for varying training scenarios. The datasets are low-dose and have $N_a = 500$ projection angles. The best results are highlighted.

the learned filters, the NN-FDK network has a low number of trainable parameters (<100) and can be trained efficiently with the Levenberg-Marquardt algorithm with approximate quadratic convergence rate.

We compared the NN-FDK algorithm to SIRT with a nonnegativity constraint (SIRT$^+$), the standard FDK algorithm and two deep neural networks (DNNs), namely a 2D U-net and a 2D MSD network applied in a slice-by-slice fashion to a 3D volume. We have shown that the NN-FDK algorithm has the lowest reconstruction time after the standard FDK algorithm. We have also shown that the NN-FDK algorithm achieves a reconstruction accuracy that is similar to that of SIRT$^+$ for simulated data and a higher accuracy than that of SIRT$^+$ for experimental data. The DNNs achieved the highest reconstruction accuracy, but training those networks took between 2 days (1 training and validation dataset) and 2 weeks (15 training and validation datasets), whereas all the NN-FDK networks were trained within 1 minute.

To conclude, the NN-FDK algorithm is a computationally efficient reconstruction algorithm that can reconstruct CCB CT reconstruction problems with high noise, low projection angles or large cone angles accurately. The training process is efficient and requires a low amount of training data, making it suitable for application to a broad spectrum of large scale (up to $4096 \times 4096 \times 4096$) reconstruction problems. Specifically, the NN-FDK algorithm can be used improve image quality in high throughput CT scanning settings, where FDK is currently used to keep pace with the acquisition speed using readily available computational resources.

4.7 Appendices

4.7.1 Implementation

Data generation

For our simulated data experiments we take $N = 1024$, which means that reconstructions and reference images are defined on a 1024^3 equidistant voxel grid, and the projection data on a 1024^2 equidistant detector grid per projection angle. However, to avoid using the same operator for reconstructions as for the data generation we generate the input data at a higher resolution. More specifically, we generate a phantom at $N = 1536$, forward project this phantom to the data space with size $N_a \times 1536^2$ and apply a bilinear interpolation per projection angle to arrive at a 1024^2 detector grid, resulting in input data with the desired resolution $N_a \times 1024^2$. We set the source radius to 10 times the physical size of the phantom, resulting in a cone angle of 5.7 degrees. To generate noise we compute a noise free photon count I from clean projection data $\mathbf{y}_c$ and use that to generate a Poisson distributed photon count from which we compute $\mathbf{y}$:

$$I = I_0 e^{-\mathbf{y}_c}, \qquad I_{\text{noise}} \sim \text{Pois}(I), \qquad \mathbf{y} = -\log\left(\frac{I_{\text{noise}}}{I_0}\right), \tag{4.21}$$

with I_0 the emitted photon count. Higher I_0 implies a higher dose and therefore less noise in the data.

Deep neural networks

Application strategy We train 2D DNNs to remove artifacts from 2D slices of an FDK reconstruction. We train one network that handles all slices in the reconstructions.

Training DNNs We train the DNNs with ADAM [KB14] and stop training after 48 hours of training on a Nvidia GeForce GTX 1080Ti GPU, the network with the lowest validation set error during this training process will be used for the reconstructions.

U-net and MSD network structures For U-net we will take four up and down layers as presented in [RFB15]. For the MSD networks we take 100 layers with one input and one output layer and the dilations as suggested in [PS18].

Code-base

We implemented the NN-FDK framework using Python 3.6.5 and Numpy 1.14.5 [WCV11]. For the parameter learning we used the Levenberg-Marquardt algorithm implementation from [PB13]. The reconstruction algorithm is implemented using ODL [AKÖ17], the ASTRA-toolbox [Van+16], PyFFTW [FJ05] and the exponential binning framework for filters from [Lag+20]. For performance reasons the simulated phantoms are generated through C++ using Cython [Beh+11].

For the evaluation of U-nets we took the PyTorch [Pas+19] implementation used in [Hen+19]. The MSD-nets are implemented using the package published with [PBS18].

All the code related to this chapter can be found on Github [Lagb].

Segmentation algorithm

This algorithm consists of several steps:

1. Apply a Gaussian filter to the reconstruction.
2. Compute a histogram of the filtered reconstruction and determine the peaks relating to the background, kernel and shell.
3. Determine the shell and kernel segmentations using a threshold based on the found peaks.
4. Apply the watershed algorithm on the shell segmentation. This gives the total volume inside the walnut.
5. Remove the kernel from the total volume inside the walnut to attain the empty space segmentation.

Further details about this implementation can be found on our Github [Lagb].

4.7.2 Levenberg-Marquardt algorithm

Given the learning problem (4.12), the update rule for the Levenberg-Marquardt algorithm (LMA) ([Lev44; Mar63]) is given by:

$$\theta^{i+1} = \theta^i + \mathbf{t}^i, \tag{4.22}$$

with $\mathbf{t}^i$ the update vector. This is computed by solving the following equation for $\mathbf{t}^i$

$$\left(J_i^T J_i + \lambda_i I\right)\mathbf{t}^i = -\frac{\partial \mathscr{L}}{\partial \theta}(\theta^i, T) = -J_i^T \sum_{j=1}^{N_\mathrm{T}} \left(O_j - \mathsf{N}_\theta(Z_j)\right) \tag{4.23}$$

where $\lambda_i > 0$ is the step parameter and J_i the $m \times n$ Jacobian matrix of $\mathsf{N}_{\theta^i}(\mathbf{Z})$ with respect to θ^i, with $\mathbf{Z}$ the vector containing all inputs from the training set T. We can solve (4.23) using a Cholesky decomposition.[3]

To ensure convergence, only updates that improve the training error are accepted, *i.e.*, if the following is true:

$$\mathscr{L}(\theta^i, T) > \mathscr{L}(\theta^i + \mathbf{t}^i, T), \tag{4.24}$$

If this is not the case we change the step parameter λ_i to $a\lambda_i$ with $a > 1$ and compute a new update vector $\mathbf{t}^i$. When an update is accepted we change the step parameter to $\lambda_{i+1} = \lambda_i / a$.

We use two stopping criteria for the LMA. Firstly, we stop if we cannot find a suitable θ^{i+1}, using several indicators for this:

- The norm of the gradient $\frac{\partial \mathscr{L}}{\partial \theta}(\theta^i)$ is too small
- The step size λ_i is too big
- After N_{up} rejected updates.

The second stopping criterion checks whether the parameters θ^i improve the validation set error. More specifically, we terminate the LMA when the validation set error has not improved for N_{val} iterations.

In **Algorithm 4** the LMA is summarized. The random initialisation is done with the Nguyen-Widrow initialization method [NW90]. For our experiments we take $N_{\text{up}} = 100$, $\lambda_0 = 10^5$, $a = 10$ and $N_{\text{val}} = 100$.

Algorithm 4 Levenberg-Marquardt algorithm

1: Compute random initialization θ^0 using [NW90]
2: **repeat**
3: Compute $\mathbf{t}^i$ until we accept an update θ^{i+1}.
4: **until** N_{up} updates were rejected **or**
$\mathscr{L}(\theta^i, V)$ did not improve N_{val} times **or**
$\left\| \frac{\partial \mathscr{L}}{\partial \theta}(\theta^{i+1}) \right\|$ is too small **or** λ_{i+1} is too big.
5: Set $\theta^\star$ equal to the θ^i with the lowest validation error.

[3] $J_i^T J_i$ is positive semi-definite and $\lambda_i > 0$, therefore the left hand side of (4.23) is positive definite.

4.7.3 Results data requirement experiment

Simulated data, large cone angle

TSE			
Method	1 dataset	2 datasets	15 datasets
NN-FDK$_4$	6.47±1.19e-04	4.70±1.16e-04	4.82±1.13e-04
U-net	1.04±0.27e-04	1.02±0.17e-04	8.23±0.85e-05
MSD	2.44±1.43e-04	1.53±0.17e-04	**6.52±0.43e-05**
SSIM			
NN-FDK$_4$	0.825±0.018	0.904±0.011	0.910±0.007
U-net	**0.974±0.015**	0.971±0.021	0.973±0.010
MSD	0.954±0.006	0.937±0.004	0.966±0.002

Table 4.8: Average and standard deviation of the quantitative measures computed over 20 different Defrise phantoms for varying training scenarios. The reconstruction problems have a cone angle of 29.2 degrees and $N_a = 360$ projection angles. The best results are highlighted.

Experimental data, high-dose, 32 projection angles

TSE			
Method	1 dataset	2 datasets	15 datasets
NN-FDK$_4$	8.14±1.45e-04	8.68±1.43e-04	8.03±1.39e-04
U-net	7.56±1.52e-04	6.85±1.56e-04	**4.10±1.06e-04**
MSD	7.82±0.41e-04	6.51±0.35e-04	4.23±0.97e-04
SSIM			
NN-FDK$_4$	0.950±0.010	0.948±0.010	0.946±0.011
U-net	0.955±0.011	0.930±0.023	**0.964±0.009**
MSD	0.955±0.010	0.947±0.014	**0.964±0.009**

Table 4.9: Average and standard deviation of the quantitative measures computed over the 6 datasets for varying training scenarios. These are the high-dose datasets from [LCB20] with $N_a = 32$ projection angles. The best results are highlighted.