5 Neural network filtered backprojection

5.1 Introduction

The main problem in tomography is the reconstruction of an unknown image from its projections, acquired along a range of angles. This problem occurs in many real world applications, such as X-ray tomography in medical imaging and electron tomography in materials science. Because of its practical relevance, a large amount of research has been devoted to developing tomographic reconstruction methods (see [KS01; Nat01; Buz08] for an overview). Most common reconstruction methods can be divided into two groups: *analytical* methods and *algebraic* methods.

Analytical methods, of which filtered backprojection (FBP) is the most widely used example, are based on a continuous representation of the reconstruction problem. An analytical inverse formula of the Radon transform is discretized to obtain a reconstruction algorithm. The advantage of analytical methods is that they are usually computationally inexpensive. However, the approach is based on the assumption that the projection data is available for *all* angles, which is clearly not feasible in practice. As a result, the reconstruction quality of analytical methods tends to become unacceptable when data is only available for a small number of angles.

In several applications of tomography, practical considerations limit the number of angles for which data can be acquired. These reconstruction problems are known

This chapter is based on:
D. M. Pelt and K. J. Batenburg. "Fast tomographic reconstruction from limited data using artificial neural networks". *IEEE Transactions on Image Processing* 22.12 (2013), pp. 5238–5251.

as *limited-data problems*. For example, in electron tomography, the electron beam damages the sample, imposing a strong limitation on the number of angles [MDG95]. Furthermore, in most applications, acquiring data for more projection angles requires more time. In industrial tomography, process speed considerations limit the total scan duration, making only a limited number of angles possible [Sip93]. For such problems, algorithms are needed that can create accurate reconstructions from limited data.

Algebraic reconstruction methods, such as ART and SIRT [KS01], often handle limited-data problems better than analytical methods. They are based on a discrete representation of the problem, which leads to a system of linear equations. These equations can be solved using iterative methods. Since these methods are based on a model of the data that is actually available, they can lead to more accurate reconstructions than analytical methods. The computational cost of these methods is high however, often several orders of magnitude larger than analytical methods, even when using highly optimized implementations on graphic processor units (GPUs) [XM05].

Recently, a range of algebraic methods have been developed that exploit prior knowledge about the unknown image to solve limited-data problems even more accurately. For example, total variation minimization based methods, such as FISTA [BT09a], can compute accurate reconstructions if the image has a sparse gradient [SP08]. In discrete tomography, reconstruction methods like DART [BS11] can solve limited-data problems where the original image is known to consist of only a small number of different gray levels. Although these methods produce accurate results in many cases, they have two main disadvantages: (i) they are based on algebraic methods, sharing their high computational cost; (ii) the specific prior knowledge can limit the types of images that can be reconstructed. As an example of the second point, total variation minimization methods can only accurately reconstruct objects with a sparse gradient.

In this chapter, we present a reconstruction method for limited-data problems that is specifically designed to avoid both problems. The method is computationally similar to analytical methods, ensuring a low computational cost. Furthermore, the method learns how to use problem specific knowledge to produce more accurate reconstructions than existing analytical methods. This learning is accomplished by using an *artificial neural network* (ANN). No specific prior knowledge has to be presented to the method, making it applicable to any type of image. The result is a very general method, able to produce accurate reconstructions in short time.

Artificial neural networks have been applied to tomographic reconstruction problems by several authors (see, e.g. [SH10] for an overview in the context of medical imaging). Some previous approaches have focused on directly solving a single instance of the tomographic reconstruction problem using a Hopfield neural network as an optimization tool [SHO93; Cic+95; WW97; Cie08; Cie09]. These methods compute reconstructions by minimizing the difference between the measured projection data and projections of the reconstructed object. As such, they are essentially algebraic reconstruction methods, since the objective function that is minimized is algebraic in nature. Since the neural networks have to solve a nonlinear system instead of a linear one, the reconstruction time of these methods is often even larger than the reconstruction time of linear algebraic methods.

Other previous work on using neural networks to solve tomographic problems is based on methods with a separate training phase, where the neural network is trained on a set of example images [KB95a; KB95b; Rod+01; BK06; DKM07]. In subsequent reconstruction steps, no additional training is performed. In these methods, the neural network reconstructs the entire image from the available projection data, an approach that leads to large network sizes. Because of this large size, the training phase of these methods can take a long time. Furthermore, the number of example images that the network can be trained on is typically small, limiting the reconstruction accuracy and generalizability that can be obtained (see, for example, [Rod+01]). In particular, we have not found reports on successful application of such methods to reconstruction problems involving large images (i.e. slices of 512×512 pixels or larger).

In this chapter we present a novel neural network approach to tomography, which does not have the aforementioned drawbacks. Our approach has some similarities to previous methods, such as a separate training phase, yet we use a different network model. In our model, the network reconstructs a single pixel of the reconstruction grid, using reduced projection data. This approach leads to small network sizes, which leads to fast training times, and enables us to use advanced neural network training methods. Furthermore, in our approach, each pixel of an example image can be used as an independent example during training. Therefore, we are able to use a large number of examples to train the neural network on. As a result, the trained networks yield accurate reconstructions from limited data, as well as robustness to noise.

A somewhat similar method is given in [BK06; VKB11], where the reconstruction step is implemented by using the neural network as a black box, resulting in a slow reconstruction method. In the current chapter, a different network model is chosen, such that it can be viewed as an analytical reconstruction method, having both a low computational cost and a high reconstruction accuracy. As a result, our approach can be applied to large datasets, at a computational cost that is comparable to analytical methods.

This chapter is structured as follows. In Section 5.2, we formally define the tomographic reconstruction problem and artificial neural networks. Section 5.3 introduces the new reconstruction method, which is the key contribution of this chapter. We discuss how we implemented this method in Section 5.4. In Section 5.5, we describe the experiments that we performed to compare the reconstruction time and accuracy of the new method with existing methods. The results of these experiments are given in Section 5.6, along with a discussion of these results. We conclude the chapter in Section 5.7 with a summary and some final remarks.

5.2 Notation and concepts

In this section, we will define the mathematical notation that is used in the rest of the chapter, and introduce the relevant concepts. First, we formally define the tomographic reconstruction problem, and the popular filtered backprojection algorithm. Then, we introduce artificial neural networks, the mathematical construct on which our new method is based.

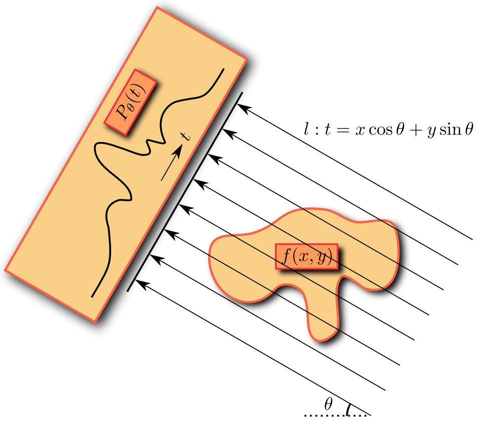


Figure 5.1: The tomography model used in this chapter. Several parallel lines, rotated by angle θ, are passing through the object f. Each line has a characteristic equation $t = x\cos\theta + y\sin\theta$, with constant t. The projection P_θ of f is given by the line integrals of f over the different parallel lines.

Problem definition

We will focus in this chapter on reconstructing two-dimensional objects from *parallel-beam* projections with a single rotation axis. The unknown object is modeled as a two-dimensional finite and integrable function $f : \mathbb{R}^2 \to \mathbb{R}$ with bounded support. We define a projection P_θ of f as the line integral of $f(x, y)$ over line l_θ:

$$P_\theta(t) = \int_{l_\theta} f(x, y)\,\mathrm{d}s \tag{5.1}$$

$$= \iint_{\mathbb{R}^2} f(x, y)\delta(x\cos\theta + y\sin\theta - t)\,\mathrm{d}x\mathrm{d}y \tag{5.2}$$

This integral transform is called the *Radon transform* of f.

Given an image $f(x, y)$, we can model the projection geometry in parallel-beam tomography as a number of parallel lines going through f, each rotated by a certain projection angle θ. A point (x, y) on one such line l_θ obeys the equation $t = x\cos\theta + y\sin\theta$. For each line l_θ, a unique constant t defines all points on that line. This model is shown graphically in Fig. 5.1. The basic tomographic problem is to reconstruct the unknown image $f(x, y)$ from the measured projections.

In practice, only discrete projection data is available, which consists of a matrix of measured values, one for each combination of N_θ projection angles $\theta \in \Theta = \{\theta_0, \theta_1, \dots, \theta_{N_\theta - 1}\}$ and N_d detectors $p \in \{0, 1, \dots, N_d - 1\}$. The position of a detector p relative to the central detector is given by τ_p:

$$\tau_p = d\left(p - \frac{N_d - 1}{2}\right), \tag{5.3}$$

where d is the width of a detector. The entire set of detector positions is given by $T = \{\tau_0, \tau_1, \ldots, \tau_{N_d-1}\}$.

The projection data is used to reconstruct f on an $N \times N$ grid of square pixels. Without loss of generality, we define that each pixel has a width and height of one, and that the center of the grid is positioned at the origin. In this case, the center of pixel (x_i, y_j) is situated in row j and column i of the pixel grid, with $i \in \{0, 1, \ldots, N-1\}$ and $j \in \{0, 1, \ldots, N-1\}$, and $x_i = y_i = i - (N-1)/2$.

Filtered back projection

One way of solving the reconstruction problem is to find a direct inverse of Eq. (5.2). To perform this inversion, we first convolve the projection data with a filter $h_\theta(t)$:

$$q_\theta(t) = \int_{-\infty}^{\infty} h_\theta(\tau) P_\theta(t-\tau) \mathrm{d}\tau \tag{5.4}$$

We can also perform this operation in the Fourier domain, where $\hat{P}$ and H denote the Fourier transforms of P and h:

$$q_\theta(t) = \int_{-\infty}^{\infty} \hat{P}_\theta(u) H_\theta(u) e^{2\pi \imath u t} \mathrm{d}u \tag{5.5}$$

By taking the formal adjoint of the Radon transform, it can be shown that if $H_\theta(u) = |u|$, we obtain a direct inverse of Eq. (5.2) [KS01]:

$$f(x,y) = \int_0^{\pi} q_\theta(x\cos\theta + y\sin\theta) \mathrm{d}\theta \tag{5.6}$$

In practice, Eq. (5.6) cannot be used directly, since $P_\theta(t)$ can only be measured for a finite set of angles Θ and a finite set of detector positions T. Therefore, we need to discretize both variables to obtain a usable reconstruction algorithm. Inserting Eq. (5.4) in Eq. (5.6) and discretizing, we obtain the filtered back projection method (FBP):

$$f(x,y) \approx FBP_{\mathbf{h}}(x,y) = \sum_{\theta_d \in \Theta} \sum_{\tau_p \in T} h(\tau_p) P_{\theta_d}(t - \tau_p) \tag{5.7}$$

where $t = x\cos\theta_d + y\sin\theta_d$. Because the projection data is discretized, interpolation is needed to obtain values at $t - \tau_p$, for $\tau_p \in T$. Linear interpolation is often adequate, since projection data is usually reasonably smooth.

The convolution operation in Eq. (5.7) can be performed in Fourier space, leading to an efficient implementation of FBP: first convolve the projection data with filter $\mathbf{h}$ in Fourier space in $\mathcal{O}(N_\theta N_d \log N_d)$ time and afterwards backproject the result to obtain the reconstruction in $\mathcal{O}(N_\theta N^2)$. Various discrete approximations of the ideal filter $H_\theta(u) = |u|$ are used in practice, such as the Ram-Lak (ramp), Shepp-Logan, and Hann filters [Far+97]. The Ram-Lak filter, obtained by setting $H_\theta(u)$ to 0 when $u > u_c$ for some u_c is often used. This filter is shown in real space in Fig. 5.2.

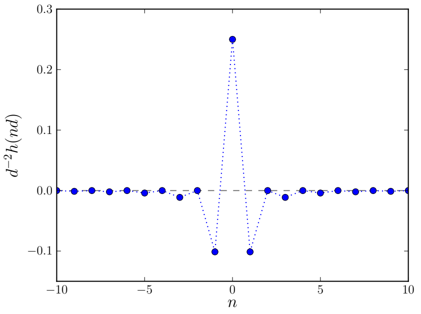


Figure 5.2: The widely used discrete Ram-Lak filter for the FBP algorithm (Eq. (5.7)). In this image, d is the distance between adjacent detector positions. This filter is an approximation of the ideal filter, obtained by taking the Fourier transform $H_\theta(u) = |u|$ of the ideal filter, and setting $H_\theta(u) = 0$ when $u > u_c$ for some u_c.

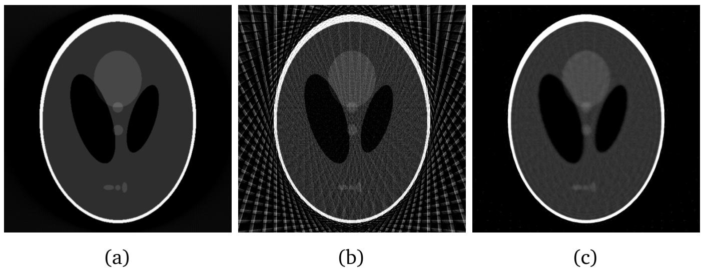


Figure 5.3: Various reconstructions of the Shepp-Logan head phantom on a 512 × 512 pixel grid. In (a) the phantom was reconstructed by FBP using 512 projection angles $\in [0, \pi)$. In (b) and (c), only 32 projection angles were used to reconstruct the phantom. FBP was used in (b), while the image (c) was obtained by using SIRT, an iterative algebraic method, with prior knowledge about the minimum and maximum possible image values.

FBP is one of the most widely used reconstruction methods in practice, because of the low computational cost compared to other methods, and good reconstruction quality if data of enough projections are available. The accuracy of the reconstructions depends on how well Eq. (5.7) approximates Eq. (5.6). If data of many projection angles are available (say, several hundreds), the approximation is often very good. When using FBP with a small number of angles, artifacts appear in the reconstructions. These artifacts can make further analysis of the reconstruction, such as segmentation, very difficult. An example of artifacts in an FBP reconstruction of limited data is shown in Fig. 5.3b. Note that a reconstruction of the same data by an algebraic method, shown in Fig. 5.3c, contains less artifacts, but takes more time to compute.

Artificial neural networks

An artificial neural network (ANN) is a computational model that processes input data using artificial neurons. The model is inspired on the workings of the human

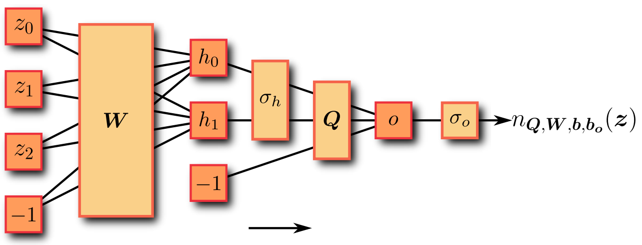


Figure 5.4: A multilayer perceptron with three input nodes z_i, two hidden nodes h_i, and one output node o. The input vector is multiplied by the weight matrix $\mathbf{W}$ to obtain hidden node inputs, and the hidden node output vector is multiplied by the weight matrix $\mathbf{Q}$ to obtain the input of the output node. Note that the biases $\mathbf{b}$ and $\mathbf{b_o}$ of Eq. (5.10) are modeled as an additional input node and hidden node of value -1. Activation functions σ_h and σ_o are applied to the hidden nodes and output node.

brain, although ANNs can also be interpreted mathematically as a class of functions. Neural networks have many uses, from simple curve fitting to complex pattern recognition [Yeg09; Hay09].

An artificial neural network can be used to model an unknown function $r : \mathbb{R}^n \to \mathbb{R}^m$. One method to accomplish this is called a *multilayer perceptron* [Hay09]. A multilayer perceptron consists of three distinct layers: the input layer, the hidden layer and the output layer. The input layer consist of n nodes, one for each input value, and the output layer has m nodes, one for each output value. The hidden layer consists of N_h hidden nodes, where N_h can be chosen freely. Generally, it is difficult to know what the optimal number of hidden nodes is for a given problem. Take too few nodes, and the network will be unable to model the unknown function. Take too many, and the resulting network will be slower and more prone to *overfitting* [TLL95]. The problem of overfitting and the way it is addressed in this chapter are explained in Section 5.4.

In a multilayer perceptron, each input node is connected to all hidden nodes, and each hidden node is connected to all output nodes. Every connection has a certain weight, and the weights can be adjusted to fit different functions r. The weights of the connections from the n input nodes to the N_h hidden nodes can be written as an $n \times N_h$ matrix $\mathbf{W}$, where the value w_{ij} in row i and column j gives the weight of the connection between input node i to hidden node j. Similarly, the weights from hidden nodes to output nodes can be written as an $m \times N_h$ matrix $\mathbf{Q}$. We denote a single column i of $\mathbf{W}$ as $\mathbf{w}_i$, and a single column i of $\mathbf{Q}$ as $\mathbf{q}_i$.

Scalar offsets $b \in \mathbb{R}$ are subtracted from the output of each hidden node and output node. Furthermore, nonlinear *activation functions* $\sigma_h : \mathbb{R} \to \mathbb{R}$ and $\sigma_o : \mathbb{R} \to \mathbb{R}$ are applied to the outputs of these nodes, making the entire model nonlinear in nature. In this chapter, we used the sigmoid function as activation function:

$$\sigma_h(t) = \sigma_o(t) = \frac{1}{1+e^{-t}} \tag{5.8}$$

The equation for the output of a multilayer perceptron, with a vector $\boldsymbol{z}$ as input, is given by:

$$n_{\mathbf{Q},\mathbf{W},\mathbf{b},\mathbf{b_o}}(\mathbf{z}) = \sigma_o\left(\sum_{i=0}^{N_h-1} \mathbf{q}_i g_{\mathbf{w}_i,b_i}(\mathbf{z}) - \mathbf{b_o}\right) \tag{5.9}$$

where the activation function σ_o is evaluated element-wise on its input vector, and $g_{\mathbf{w}_i,b_i}$ is the output of a hidden node:

$$g_{\mathbf{w},b}(\mathbf{z}) = \sigma_h(\mathbf{w}\cdot\mathbf{z} - b) \tag{5.10}$$

The question remains how to choose $\mathbf{Q}$, $\mathbf{W}$, $\mathbf{b}$, and $\mathbf{b_o}$, such that $n_{\mathbf{Q},\mathbf{W},\mathbf{b},\mathbf{b_o}}(\mathbf{z}) \approx r(\mathbf{z})$. In this chapter, *supervised learning* [AB09] is used, where we assume that, although the function r is unknown, a set of T inputs $\{Z_0, Z_1, \dots, Z_{T-1}\}$ with corresponding outputs $\{O_0, O_1, \dots, O_{T-1}\}$ of r *are* known, where $Z_i \in \mathbb{R}^n$ and $O_i \in \mathbb{R}^m$. Learning is then defined as the minimization of the sum of squared differences between the perceptron output and the correct output:

$$e(\mathbf{Q},\mathbf{W},\mathbf{b},\mathbf{b_o}) = \sum_{i=0}^{T-1}\left(n_{\mathbf{Q},\mathbf{W},\mathbf{b},\mathbf{b_o}}(Z_i) - O_i\right)^2 \tag{5.11}$$

$$\mathbf{Q}_l, \mathbf{W}_l, \mathbf{b}_l, \mathbf{b_o}_l = \underset{\mathbf{Q},\mathbf{W},\mathbf{b},\mathbf{b_o}}{\operatorname{argmin}}\; e(\mathbf{Q},\mathbf{W},\mathbf{b},\mathbf{b_o}) \tag{5.12}$$

Because of the mathematical form of a perceptron, partial derivatives of the parameters, such as $\frac{\partial e}{\partial w_{ij}}$, can be calculated quickly and accurately by applying the chain rule. The fact that these partial derivatives are easily obtained leads to efficient applications of gradient based minimization methods to train such networks. Different methods can be used for training, each with their own advantages and disadvantages. The specific method used in this chapter is given in Section 5.4.

5.3 Neural network filtered backprojection

In this section, we present the key contribution of this chapter: the neural network filtered backprojection method (NN-FBP). We start by defining a neural network model to reconstruct a single pixel of an image. We show that this model can be viewed as a combination of FBP steps, obtaining an efficient implementation of the method. Finally, we give examples of how the new method can be used in practice.

Neural network model

To solve the basic tomographic problem using an artificial neural network, we need to define a network model: a method of converting the given projection data to input for the neural network. As explained above, we want to be able to view the chosen model as a combination of filtered backprojection steps. Therefore, it is informative to look at the equation of the FBP method:

$$FBP_{\mathbf{h}}(x,y) = \sum_{\theta_d\in\Theta}\sum_{\tau_p\in T} h(\tau_p) P_{\theta_d}(x\cos\theta_d + y\sin\theta_d - \tau_p) \tag{5.13}$$

A first observation is that Eq. (5.13) gives the value of a single point (x,y) of the FBP reconstruction. To mimic this, we choose to use a network model that reconstructs a single pixel (x_i, y_j). The neural network only has a single output node, and the output of the network is a single value in $\mathbb{R}$.

A second observation is that the FBP method is *linear shift invariant* [KS01]. Suppose we shift an object f by δx horizontally and δy vertically to obtain a shifted object f'. The original projections P_θ shift accordingly to new projections P'_θ:

$$P'_\theta(\tau) = P_\theta(\tau - (\delta x \cos\theta + \delta y \sin\theta)) \tag{5.14}$$

For the FBP reconstruction of f, denoted by $FBP^f_{\mathbf{h}}$, and the FBP reconstruction of f', denoted by $FBP^{f'}_{\mathbf{h}}$, we have:

$$FBP^{f'}_{\mathbf{h}}(x + \delta x, y + \delta y) = FBP^f_{\mathbf{h}}(x, y) \tag{5.15}$$

To mimic the linear shift invariance of FBP, we want the neural network model to treat every pixel of the reconstruction grid the same, independent of its actual position on the grid. An additional advantage of treating each pixel the same is that we can use every pixel of the grid as an independent training example during supervised learning (Eq. (5.11)). In order to accomplish this position independence, we shift the reconstructed object such that the pixel that it currently reconstructs, (x_i, y_j), is at the origin. In other words, as input for the neural network, we use projection data of the shifted object f', which can be obtained by shifting f by $-x_i$ horizontally and $-y_j$ vertically. For the projection data of the shifted object, we have (Eq. (5.14)):

$$P'_\theta(\tau_d) = P_\theta(\tau_d + x_i \cos\theta + y_j \sin\theta) \tag{5.16}$$

Now, we combine the shifted data of all projection angles by summing them element-wise:

$$P'(\tau_p) = \sum_{\theta_d \in \Theta} P'_{\theta_d}(\tau_p) \tag{5.17}$$

Finally, we reflect the shifted and summed data about the detector center:

$$z(\tau_p) = P'(-\tau_p) = \sum_{\theta_d \in \Theta} P_{\theta_d}(x_i \cos\theta_d + y_j \sin\theta_d - \tau_p) \tag{5.18}$$

The values of $z(\tau_p)$ are used as input for the neural network, as an input vector $\mathbf{z}$ with N_d elements. Note that in Eq. (5.18), only the original projection data P_{θ_d} is used. Therefore, we do not have to explicitly shift f to f' for every pixel we reconstruct, but only have to shift the original projection data by $x_i \cos\theta_d + y_j \sin\theta_d$. The transformation from projection data to network input is shown in Fig. 5.5.

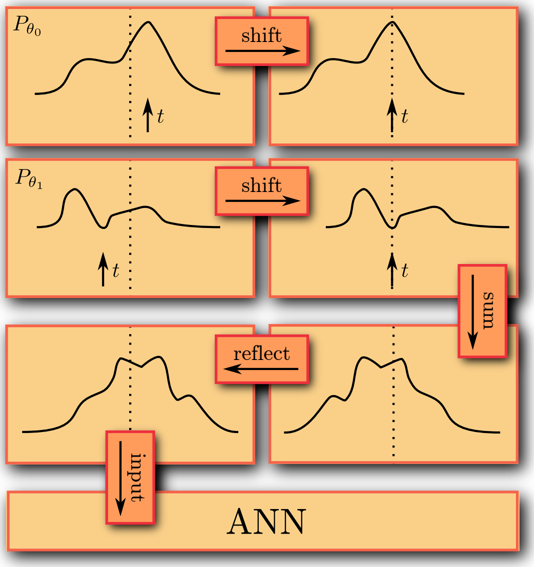


Figure 5.5: The method of transforming projection data to neural network input for pixel (x_i, y_j). For each angle θ_d, (x_i, y_j) projects onto a different point $t_d = x_i \cos\theta_d + y_j \sin\theta_d$ on the detector. We shift each projection P_{θ_d} such that the corresponding t_d is in the middle. Finally, we sum the shifted projections point by point and reflect about the center to get the network input.

Filtered back projection view

To see what the effect of the choice of network model is, we take the equation of a single hidden node $g_{\mathbf{w},b}$ (Eq. (5.10)), and insert our network model (Eq. (5.18)):

$$g_{\mathbf{w},b}(\mathbf{z}) = \sigma_h\left(\mathbf{w}\cdot\mathbf{z} - b\right) \tag{5.19}$$

$$= \sigma_h\left(\sum_k w_k \sum_{\theta_d\in\Theta} P_{\theta_d}(t-\tau_k) - b\right) \tag{5.20}$$

where $t = x_i\cos\theta_d + y_i \sin\theta_d$. Rearranging the sums and comparing with Eq. (5.7) we get:

$$g_{\mathbf{w},b}(\mathbf{z}) = \sigma_h\left(\sum_{\theta_d\in\Theta}\sum_i w_i P_{\theta_d}(t-\tau_i) - b\right) \tag{5.21}$$

$$= \sigma_h\left(FBP_{\mathbf{w}}(x_i, y_j) - b\right) \tag{5.22}$$

The entire neural network equation will now become:

$$n_{\mathbf{Q},\mathbf{W},\mathbf{b},b_o}(\mathbf{z}) = \sigma_o\left(\sum_{k=0}^{N_h-1} q_k \sigma_h\left(FBP_{\mathbf{w}_k}(x_i, y_j) - b_k\right) - b_o\right) \tag{5.23}$$

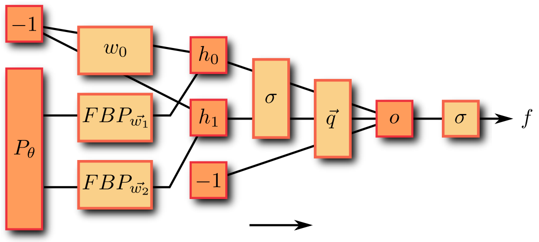


Figure 5.6: The FBP view of NN-FBP. Here, we take the projections P_θ and apply several FBP algorithms: to obtain the hidden node h_i, we apply the FBP algorithm with custom filter $\mathbf{w_i}$ and a bias. A linear combination of all hidden node images and a bias, with a sigmoid function applied to all pixels of each image, leads to a single image o. After we apply a final sigmoid function, we get an approximation of f. Note that in this case, we reconstruct the entire image f, where in the neural network view of Fig. 5.4 only a single pixel (x_i, y_j) is reconstructed.

This shows that we can view a trained network as a weighted sum of N_h FBPs with custom filters $\mathbf{w_i}$ and added biases $\mathbf{b}$. A sigmoid function is applied to the output of each FBP, and also to the final sum. The advantage of this view is that in this case, we do not have to run the network for every pixel to get the reconstruction image: we can simply apply the FBPs to obtain the entire reconstruction image in one operation.

To summarize, our new method works as follows:

Algorithm 5.1 NN-FBP reconstruction method

1. Perform N_h FBP algorithms, each with a different filter.
2. Subtract a bias from each resulting image, and apply a nonlinear activation function σ_h to each pixel of the result.
3. Multiply each resulting image with a certain weight, and add them together pixel by pixel to obtain a single image.
4. Subtract a bias from the resulting image, and apply a nonlinear activation function σ_o to each pixel to get the final reconstruction.

Note that the results of this method are identical to the results of directly applying the standard multilayer perceptron output equation (Eq. (5.9)) with Eq. (5.18) as network input. The equivalence of both methods is shown in Figs. 5.4 and 5.6.

The computational complexity of Algorithm 5.1, however, is significantly lower than direct application of Eq. (5.9). In Eq. (5.9), we need to shift, sum and reflect the input data, costing $\mathcal{O}(N_d N_\theta)$ time, for each of the N^2 pixels. Additionally, applying Eq. (5.9) takes $\mathcal{O}(N_h N)$ for every pixel, since there are $N_h N$ connections between the input layer and hidden layer. Direct application will therefore take $\mathcal{O}((N_\theta + N_h)N^3)$ time to reconstruct the entire $N \times N$ image if $N \approx N_d$.

For Algorithm 5.1, we need to perform N_h FBPs, and the computation time of step 1) is $\mathcal{O}(N_h N_\theta N^2)$. The remaining operations (adding biases, applying the activation functions and weight multiplication) each take $\mathcal{O}(N^2)$ time. Therefore, step 2) and step 3) take $\mathcal{O}(N_h N^2)$ time in total, and step 4) is $\mathcal{O}(N^2)$. We see that by exploiting the FBP view, we have reduced the reconstruction time from $\mathcal{O}((N_\theta + N_h)N^3)$ to $\mathcal{O}(N_h N_\theta N^2)$. Results from Section 5.6 will show that the method can produce accurate reconstruction even when $N_h \ll N$. Furthermore, for limited-data problems $N_\theta \ll N$, so the reduction in computation time is significant.

Training

The filters, biases and weights are trained using standard training methods from neural network theory [Hay09]. The training phase is separate from subsequent reconstruction steps: we first train the network to obtain $\mathbf{Q}_l$, $\mathbf{W}_l$, $\mathbf{b}_l$ and $\mathbf{b_o}_l$, using a set of training images. Afterwards, the trained network can be used to quickly reconstruct other images by using the method described in Algorithm 5.1, without additional training.

To perform training by supervised learning, we need a set of inputs Z with corresponding correct outputs O, where $Z_i \in \mathbb{R}^{N_d}$ is shifted and summed projection data for a single pixel and $O_i \in \mathbb{R}$ is the correct value of that pixel. This means that we need a set of projection data with corresponding correct images $f(x, y)$. This presents a problem: usually, the correct image $f(x, y)$ is unknown, since that is exactly the problem we are trying to solve. However, we can take the projection data, reconstruct it using any other method, and use the reconstruction as the correct output for learning.

This training approach can be useful in two cases:

1) N_θ-REDUCTION *use-case*: Suppose that we have a scanner that can acquire projection data along a variable number of angles. Scanning with a small number of angles is preferred, because of practical considerations. To use NN-FBP in this case, we first acquire projection data along a large number of angles for a set of representative objects. We reconstruct the images using an existing reconstruction method like FBP. Then, we train NN-FBP using these reconstructions as correct output. As input during training, we only use the projection data along a small subset of angles. After training, we can scan new objects using this small set of angles, and use NN-FBP to obtain accurate reconstructions in short time. This can be useful in many practical cases, for example to increase the time resolution of tomography of dynamic systems.

2) LIMITED-DATA *use-case*: If practical considerations limit the number of angles for which projection data can be acquired, NN-FBP can be used to lower reconstruction times. In this case, we use an advanced but slow prior-knowledge based method like TV-minimization to obtain reconstructions from the limited projection data. We then train NN-FBP using these reconstructions as correct images. In other words, we train NN-FBP to mimic a slower reconstruction method. Afterwards, we can use NN-FBP to quickly reconstruct images from similar limited-data problems.

5.4 Implementation

We will now discuss our implementation of NN-FBP that was used in the computational experiments of Section 5.5. The NN-FBP method consists of two distinct parts:

the *training* phase, and subsequent *reconstruction*. In this section, we will focus on implementation of the training, since implementing the reconstruction part is fairly straightforward: it consists of several FBPs and basic image operations. More information on implementing the FBP algorithm can be found in [BB00].

Minimization method

An important part of neural network training is the minimization of the network error (Eq. (5.12)). Several minimization algorithms are well-suited for neural network training. We used the Levenberg-Marquardt algorithm (LMA) [Mar63]. LMA is a combination of the gradient descent and Gauss-Newton algorithm, improving the stability of Gauss-Newton while retaining its fast convergence. Given a function $f_{\mathbf{w}}(\mathbf{x})$ with n parameters $\mathbf{w}$ and a set of m correct input-output pairs $(\mathbf{x}_i, \mathbf{y}_i)$, the method iteratively minimizes the error $e(\mathbf{w}) = \sum_i (\mathbf{y}_i - f_{\mathbf{w}}(\mathbf{x}_i))^2$, with the parameters at iteration $j+1$ given by $\mathbf{w}^{(j+1)} = \mathbf{w}^{(j)} + \mathbf{dw}^{(j)}$. The update vector $\mathbf{dw}^{(j)}$ is obtained by solving the LMA equation:

$$(J^T J + \lambda I)\mathbf{dw}^{(j)} = J^T\left(\mathbf{y} - f_{\mathbf{w}^{(j)}}(\mathbf{x})\right) \tag{5.24}$$

where $\lambda > 0$ and J is the $m \times n$ Jacobian matrix:

$$J = \begin{bmatrix} \frac{\partial f_{\mathbf{w}}(\mathbf{x_0})}{\partial w_0} & \frac{\partial f_{\mathbf{w}}(\mathbf{x_0})}{\partial w_1} & \cdots & \frac{\partial f_{\mathbf{w}}(\mathbf{x_0})}{\partial w_{n-1}} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_{\mathbf{w}}(\mathbf{x_{m-1}})}{\partial w_0} & \frac{\partial f_{\mathbf{w}}(\mathbf{x_{m-1}})}{\partial w_1} & \cdots & \frac{\partial f_{\mathbf{w}}(\mathbf{x_{m-1}})}{\partial w_{n-1}} \end{bmatrix} \tag{5.25}$$

Since $J^T J + \lambda I$ is symmetric and positive definite if $\lambda > 0$, we can use Cholesky decomposition to solve Eq. (5.24).

The parameter λ is adjusted at each iteration to ensure convergence: if $e(\mathbf{w}^{(j+1)}) > e(\mathbf{w}^{(j)})$, we increase λ to $a\lambda$ and solve Eq. (5.24) again until $e(\mathbf{w}^{(j+1)}) < e(\mathbf{w}^{(j)})$. If no such λ can be found, $\mathbf{w}^{(j)}$ is a local minimum of the error function, and LMA terminates. After an accepted update, we decrease λ to λ/a for the next iteration. In this chapter, we take $a = 10$, and start with $\lambda = 10^4$.

In the case of neural network training, the function we are minimizing is Eq. (5.11). As parameters in the LMA method we use the collection of network parameters $\mathbf{W}$, $\mathbf{Q}$, $\mathbf{b}$ and $\mathbf{b_o}$. The initial values for the parameters are calculated randomly using the Nguyen-Widrow initialization method [NW90]. In order to apply LMA, we need to calculate the Jacobian matrix J at each iteration. For neural networks, these partial derivatives can be calculated accurately and efficiently by applying the chain rule. More information on the use and implementation of LMA for neural network training can be found in [HM94].

Overfitting

A common problem that can occur when training neural networks is *overfitting*. Overfitting occurs when the neural network learns too much information about the training set. An overfitted network will be very good at solving problem instances from the training set, but relatively bad at solving instances outside the training set. In the case of NN-FBP, the method will only be able to accurately reconstruct images used in the training set, and not other, unknown, images. Of course, this is undesirable: we already know solutions to the training set problems, and we would like to be able to solve different reconstruction problems by applying NN-FBP.

The problem of overfitting is well-known in neural network theory, and several ways of preventing the problem are available. Here, we use a relatively simple, but effective method. In addition to a training set, we also use an independent *validation* set of input-output pairs during training. We then calculate the error of the validation set using Eq. (5.11) after each iteration of LMA. When this error stops improving for N_{stop} iterations, we stop the training method and return the solution with the lowest validation error. In this chapter, we use $N_{stop} = 25$. Because the training and validation set are generated independently, this prevents the network from learning too much specific information about the training set. To summarize, the training method works as follows:

Algorithm 5.2 Training method

```
1. Initialize $\mathbf{W}$, $\mathbf{Q}$, $\mathbf{b}$, and $\mathbf{b_o}$ randomly (using [NW90])
2. Iterate:
    (a) Perform LMA iteration using training set
    (b) Calculate error of validation set
    (c) If validation error has not improved for $N_{stop}$ iterations, stop iterating
3. Return $\mathbf{W}$, $\mathbf{Q}$, $\mathbf{b}$ and $\mathbf{b_o}$ which had the lowest validation error
```

Exponential binning

Neural network training is often very effective at minimizing the error of Eq. (5.11), but training can take a long time. In the case of NN-FBP, we can greatly reduce the training time by using *exponential binning*. Exponential binning was also used effectively in [BK06] to reduce the reconstruction time of the neural network.

Looking at the Ram-Lak filter of Fig. 5.2, we note that the magnitude of $h(\tau)$ is relatively large around $\tau = 0$ and drops to zero quickly for $|\tau| \rightarrow \infty$. Therefore, during reconstruction of pixel (x_i, y_j), projection data values close to $t = x_i \cos\theta + y_j \sin\theta$ are much more important than far away values. This suggests that we can reduce the number of input values by rebinning the data with a high resolution around t and a lower resolution further away. Here, we used exponential binning, where the bin width grows exponentially away from t. Formally we can define any binning by specifying the boundary points s_i and s_{i+1} of every bin: $\beta_i = (s_i, s_{i+1})$. The width of a bin is given by $d_i = s_{i+1} - s_i$. In exponential binning, we take $d_0 = 1$ and $d_i = 2^{|i|-1}$ for $i \neq 0$. A further reduction can be achieved by making the rebinning symmetric as well, by creating new bins $B_0 = \beta_0$ and $B_i = (\beta_i \cup \beta_{-i})$ for $i \neq 0$.

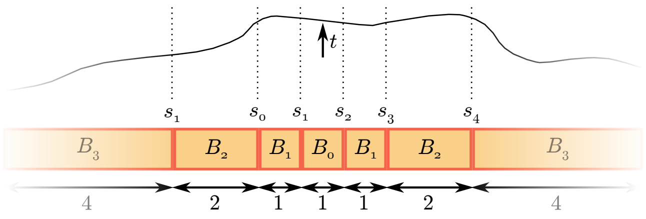


Figure 5.7: Exponential binning of the projection data, during reconstruction of a pixel (x_i, y_j), which projects onto point $t = x_i \cos\theta + y_j \sin\theta$ of the detector. Values within a bin B_i are summed to produce a single input value for the neural network. Note that the bin size increases exponentially away from t, and that the binning is symmetric, since B_i appears both to the left and to the right of t for $i \neq 0$.

To use this binning during neural network training, we apply it after the shift, sum and reflect procedure of Eq. (5.18). We sum all input values within a bin B_i to obtain the neural network input value $\bar{z}_i$:

$$\bar{z}_i = \sum_{j=s_{-i}}^{s_{-i+1}-1} z(\tau_j) + \sum_{j=s_i}^{s_{i+1}-1} z(\tau_j) \qquad \forall\, i \neq 0 \tag{5.26}$$

and $\bar{z}_0 = z(\tau_0)$. This binning procedure is shown in Fig. 5.7.

If we have N_d detectors, the output of the shift, sum and reflect procedure will have at most $2N_d$ values. We define all values outside this range to be of value 0. During binning, we only use bins that have one or both boundary points within this $2N_d$ range. Therefore, we reduce the number of input variables from $\mathcal{O}(N_d)$ to $\mathcal{O}(\log N_d)$ by using exponential binning, greatly reducing training time as well.

5.5 Experiments

In order to test the performance of the NN-FBP method, we implemented both the training and reconstruction parts using Python 2.7.3 and Numpy 1.6.3 [Oli07] built with ATLAS 3.10.0 [WP05]. We applied NN-FBP to four different problems, two for each of the two use-cases from Section 5.3. For each use-case we perform experiments on both simulation data, where the original images are known, and experimental data.

In every experiment, we are given a set of N_{im} 'correct' images, with corresponding projection data. How the correct images are obtained will be explained below for each use-case. We divide the N_{im} images into three separate groups: the *training* set, the *validation* set and the *test* set. Out of the training set, we choose N_{train} pixels to use for training, using Eqs. (5.18) and (5.26) to obtain input values for the neural network. Similarly, we take N_{val} pixels out of the validation set to use for validation, as described in Section 5.4. In this chapter, we use $N_{train} = N_{val} = 10^6$ for every experiment, unless specified otherwise.

We report results for the test set, where we use the FBP view of NN-FBP to reconstruct all test images, and report the mean absolute pixel error. The mean absolute pixel error is defined as:

$$e_p(R,O) = \frac{\langle |R - O| \rangle}{\max O - \min O} \tag{5.27}$$

where $R \in \mathbb{R}^{N\times N}$ is the reconstructed image, $O \in \mathbb{R}^{N\times N}$ the correct image, and the average is taken over all pixels that lie within the disc of radius $N/2$, centered in the image. The errors given in this chapter are the mean absolute pixel errors, averaged over all images in the test set. The results for NN-FBP are compared to results for standard FBP, with the Ram-Lak filter, and SIRT, an algebraic reconstruction method [KS01]. For both methods, we used an optimized GPU implementation from the ASTRA-toolbox [PBS11].

N_θ-reduction use-case

For the first use-case, we investigate if NN-FBP can be used to reduce the number of angles for which projection data has to be acquired. First, we reconstruct images from projection data along many angles using FBP. We then train the neural network to reconstruct these images using only a small subset of the angles. We compare the results of NN-FBP with standard FBP using the Ram-Lak filter, and with SIRT, a slower algebraic reconstruction method.

Simulation data

The simulation images used to test the performance of NN-FBP for N_θ-reduction are sampled from the threeshape family of images. Each image from the threeshape family consists of a combination of Gaussian blobs, rectangles and star-shaped objects. These components were specifically chosen to create a difficult image to reconstruct: images from the threeshape family contain both discrete and continuous areas, and both sharp edges and smooth gradients. The images are constructed as follows: starting with an image $f(x, y) = 0$, we add three Gaussian blobs, three rectangles and three star-shaped objects, each having a random shape, position, rotation and intensity. The images are then scaled, such that the darkest pixel has value 0, and the brightest has value 1. An example image of the threeshape family is shown in Fig. 5.8a.

For the training set and validation set, we generated two sets of 1000 threeshape images of 4096×4096 pixels, and calculated projection data for 4096 detector elements along 1024 equidistant angles $\in [0, \pi)$. Afterwards, we resampled the projection data to 1024 detector elements, and reconstructed on a 1024×1024 pixel grid. The test set consists of 100 images from the threeshape family. We test the network by training it to use only $N_\theta = 8, 16, 32, 64$ equidistant angles.

Experimental data

The dataset we used for experimental data stems from a small fatigue test sample made from Ti alloy VST 55531. The sample has been scanned in a parallel, monochromatic (52 keV) synchrotron X-ray beam at beamline ID11 of the European Synchrotron Radiation Facility (ESRF). The sample to detector distance was set to 40 mm and 1500 projections were acquired on a high resolution detector system. 2×2 binning resulted in projections with 1024^2 pixels and an effective pixel size of 0.56 microns. For training, validation, and testing, data from three different time steps were used, with 438 slices each.

Limited-data use-case

For the second use-case, where only a small number of projections can be acquired, we performed experiments to investigate if NN-FBP can be used to mimic advanced reconstruction algorithms that require a long computation time. We first reconstruct images using FISTA [BT09a], a TV-minimization algorithm. We train NN-FBP to approximate the result of FISTA. Afterwards, we reconstruct the test set using FISTA and NN-FBP, and report the mean absolute pixel error between the FISTA and NN-FBP reconstructions. To see the improvement of NN-FBP over standard FBP, we also report the mean absolute pixel error between FISTA and FBP.

Simulation data

For the simulation data, we sampled images from a specific family of images. Since we are investigating whether NN-FBP can mimic a TV-minimization method, images from this family should be well-suited for TV-minimization, and have a sparse gradient. Note that the threeshape family of Section 5.5 is not suitable, as the Gaussian blobs do not have a sparse gradient. Instead, we chose the 7ellipses family of images, where each image consists of 7 overlapping ellipses of random shape, position, rotation and intensity. We use 1024×1024 pixel images, randomly sampled from the 7ellipses family. The training, validation, and test sets consist of 100 images each. We calculate projection data for 1024 detector elements along $N_\theta = 8, 16, 24, 32$ equidistant angles. For reconstruction, we resampled the projection data to 256 detector elements, and obtained reconstructions using FISTA on a 256×256 pixel grid. These reconstructions were used to train the NN-FBP method, and to report errors on. An example image of the 7ellipses family is shown in Fig. 5.8c

Experimental data

Here, we use a set of experimental μCT data. These datasets were acquired by scanning raw diamonds in a Scanco 40 μCT scanner. The acquired cone-beam projection data was rebinned to a parallel beam geometry. The resulting projection data consists of 1024 detector elements along 500 projection angles, acquired for a number of two-dimensional slices through the diamonds. In total, three datasets of different diamonds were used: one for training, one for validation and one for testing. The number of slices for each dataset are 629, 358, and 375, respectively. An example of a single slice is shown in Fig. 5.8d. To test the limited-data case, we took 8, 16, 32, and 64 angles out of the available angles, resampled the projection data to 256 detector elements, and created reconstructions using FISTA on a 256×256 pixel grid. All calculations were performed using these reconstructions, thereby training NN-FBP to approximate the FISTA reconstructions.

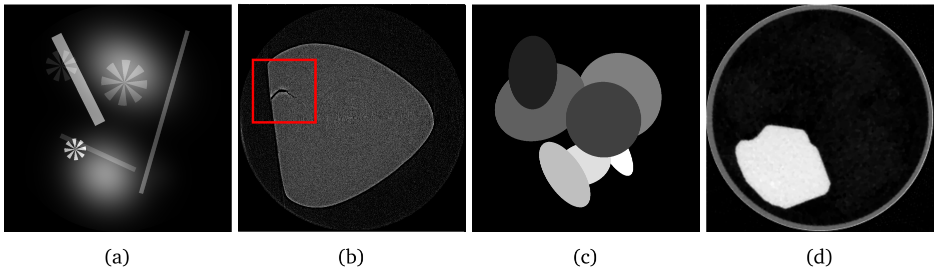


Figure 5.8: Example images of the four experiments that we performed in this chapter. The left two images were used for the N_θ-reduction use-case, and the other two for the limited-data use-case, as explained in Section 5.5. The images of (a) and (c) are computer-generated simulation images, and the images of (b) and (c) are reconstructions of experimental CT data. The area indicated in (b) is the area of which results are shown in Figs. 5.10e to 5.10h.

5.6 Results and discussion

The mean absolute error for each use-case, averaged over the entire test set, is given in Fig. 5.9. The figure shows that for all experiments and number of hidden nodes, NN-FBP produces images with lower mean absolute error than those produced by FBP *and* SIRT. An important observation is that the improvement of NN-FBP over standard FBP is significant. Furthermore, NN-FBP with one hidden node is able to produce images with significantly lower mean absolute error compared to FBP, even though their computation complexities are identical. Although FBP with the Shepp-Logan or Hann filter performed better than FBP with the Ram-Lak filter, the NN-FBP method produced significantly more accurate reconstructions than both.

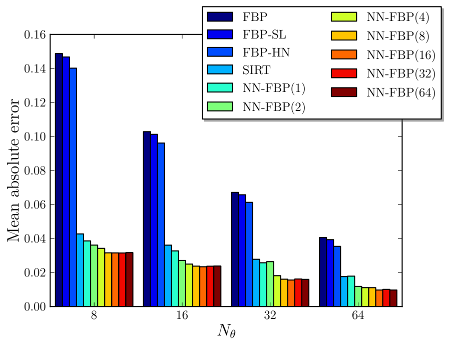
The dependence of the accuracy of NN-FBP on the number of hidden nodes N_h can be explained as follows: if not enough hidden nodes are used, the network is not able to capture all useful information during training, and the reconstruction quality suffers. If too many hidden nodes are used, the network is still able to capture all information, and reconstruction quality is still good. Since there are more weights to train, however, networks with too many hidden nodes are more difficult and time-consuming to train. With more weights, the risk of ending up in local minima of the objective function is higher, which explains why the mean absolute error sometimes increases slightly when more hidden nodes are used.

In the remainder of this section, we give detailed results for each of the use-cases, and give results of other experiments investigating the properties of NN-FBP.

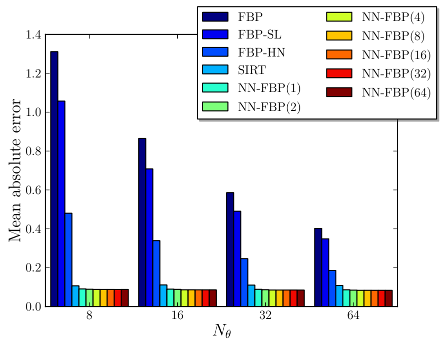
N_θ-reduction use-case

Simulations

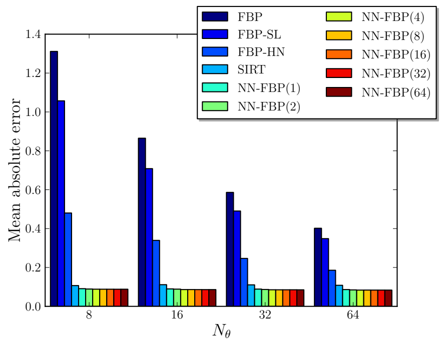
The results for the N_θ-reduction case with simulation data is shown in Table 5.1. The results show that, for all numbers of angles, NN-FBP produces more accurate reconstructions than both FBP and SIRT. The reconstruction time of NN-FBP is close



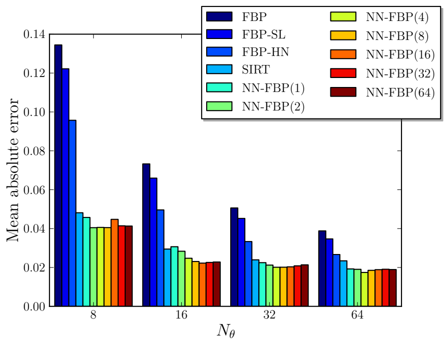
(a) N_θ-REDUCTION, simulation data



(b) N_θ-REDUCTION, experimental data



(c) LIMITED-DATA, simulation data



(d) LIMITED-DATA, experimental data

Figure 5.9: The mean absolute error, averaged over the entire test set, for each use-case. Given are results for FBP with the Ram-Lak filter (FBP), FBP with the Shepp-Logan filter (FBP-SL), FBP with the Hann filter (FBP-HN), SIRT, and NN-FBP, where the number of hidden nodes is given between parentheses.

		$N_\theta = 8$			$N_\theta = 16$			$N_\theta = 32$			$N_\theta = 64$		
	N_h	$\langle e_p \rangle$	$T_r(s)$	$T_t(s)$	$\langle e_p \rangle$	$T_r(s)$	$T_t(s)$	$\langle e_p \rangle$	$T_r(s)$	$T_t(s)$	$\langle e_p \rangle$	$T_r(s)$	$T_t(s)$
FBP		0.149	0.02		0.103	0.02		0.067	0.02		0.041	0.03	
SIRT		0.043	29.64		0.036	35.42		0.028	48.37		0.018	70.72	
NN-FBP	1	0.039	0.04	2330	0.033	0.04	2362	0.026	0.04	2428	0.018	0.04	2559
NN-FBP	2	0.036	0.06	2499	0.027	0.06	2557	0.026	0.07	2550	0.012	0.08	2815
NN-FBP	4	0.034	0.12	2532	0.025	0.12	2669	0.018	0.13	2630	0.011	0.15	2905
NN-FBP	8	0.032	0.23	2928	0.024	0.23	2873	0.016	0.25	3147	0.011	0.29	2912
NN-FBP	16	0.032	0.44	3092	0.023	0.45	3552	0.016	0.49	3940	0.010	0.56	3681
NN-FBP	32	0.032	0.88	4094	0.024	0.96	4527	0.016	1.04	5160	0.010	1.21	5801
NN-FBP	64	0.032	1.92	7101	0.024	1.79	9027	0.016	2.01	12273	0.010	2.28	11883

Table 5.1: Results for the N_θ-reduction use-case, simulation data. $\langle e_p \rangle$, T_r, and T_t denote mean absolute error, reconstruction time, and training time, respectively.

		$N_\theta = 8$			$N_\theta = 16$			$N_\theta = 32$			$N_\theta = 64$		
	N_h	$\langle e_p \rangle$	$T_r(s)$	$T_t(s)$	$\langle e_p \rangle$	$T_r(s)$	$T_t(s)$	$\langle e_p \rangle$	$T_r(s)$	$T_t(s)$	$\langle e_p \rangle$	$T_r(s)$	$T_t(s)$
FBP		1.311	0.03		0.865	0.03		0.586	0.03		0.402	0.04	
SIRT		0.107	42.97		0.111	52.59		0.111	73.15		0.108	108.17	
NN-FBP	1	0.091	0.05	750	0.090	0.05	801	0.089	0.05	777	0.086	0.06	881
NN-FBP	2	0.089	0.09	818	0.089	0.09	770	0.087	0.10	795	0.085	0.11	927
NN-FBP	4	0.088	0.16	845	0.086	0.17	855	0.085	0.18	932	0.084	0.21	1076
NN-FBP	8	0.088	0.32	912	0.086	0.33	979	0.085	0.35	1003	0.084	0.40	1148
NN-FBP	16	0.088	0.62	1181	0.086	0.64	1392	0.085	0.69	1457	0.084	0.79	1923
NN-FBP	32	0.088	1.25	2343	0.086	1.27	2551	0.085	1.39	3221	0.084	1.56	3339
NN-FBP	64	0.088	2.60	4538	0.086	2.71	6229	0.085	2.98	8265	0.084	3.11	5788

Table 5.2: Results for the N_θ-reduction use-case, experimental data. See Table 5.1 for more information.

to the reconstruction time of FBP multiplied with a factor of N_h. For example, using NN-FBP with 8 hidden nodes, the mean absolute error is, on average, roughly 75% lower than FBP and 35% lower than SIRT. The reconstruction time for that case is 11.5 times larger than that of FBP, but only 0.6% of that of SIRT. An example image with reconstructions for $N_\theta = 32$ and $N_h = 8$ is shown in Figs. 5.10a to 5.10d, where we see that the NN-FBP reconstruction is sharper than that of SIRT, and has less streak artifacts than the FBP reconstruction.

Experimental data

For the N_θ-reduction case and experimental data, results are given in Table 5.2. Again, NN-FBP produces more accurate results than both FBP and SIRT, although the differences are smaller than for the simulation data. Images of the reconstructions close to the forming crack, given in Figs. 5.10e to 5.10h, show, however, that the reconstruction of NN-FBP is visually much clearer than the FBP and SIRT reconstructions. The FBP reconstruction suffers from the combined effect of limited data and noise, resulting in a very noisy reconstruction.

Limited-data use-case

Simulations

Results for the limited-data use-case and simulation data are given in Table 5.3. Here, the reported mean absolute errors are calculated with respect to the FISTA reconstructions. Note that the images are smaller than the ones used in Section 5.6. The reconstruction time of NN-FBP is only a fraction of the reconstruction time of FISTA, while reconstructions created by NN-FBP have a relatively low mean absolute error compared to the FISTA reconstructions. Example reconstruction are given in Figs. 5.10i to 5.10l. Compared to FBP and SIRT, the NN-FBP method is able to approximate FISTA reconstruction more accurately, although NN-FBP is not able to mimic FISTA exactly.

		$N_\theta = 8$			$N_\theta = 16$			$N_\theta = 32$			$N_\theta = 64$		
	N_h	$\langle e_p \rangle$	$T_r(s)$	$T_t(s)$	$\langle e_p \rangle$	$T_r(s)$	$T_t(s)$	$\langle e_p \rangle$	$T_r(s)$	$T_t(s)$	$\langle e_p \rangle$	$T_r(s)$	$T_t(s)$
FISTA		0.000	23.5		0.000	38.0		0.000	51.2		0.000	58.7	
FBP		0.180	0.00		0.111	0.00		0.066	0.00		0.039	0.00	
SIRT		0.068	1.60		0.048	1.66		0.034	1.85		0.024	2.19	
NN-FBP	1	0.048	0.00	46	0.040	0.00	40	0.030	0.00	44	0.021	0.00	46
NN-FBP	2	0.040	0.02	108	0.031	0.01	122	0.023	0.01	192	0.016	0.01	240
NN-FBP	4	0.038	0.01	246	0.028	0.01	295	0.021	0.01	258	0.015	0.02	165
NN-FBP	8	0.038	0.02	343	0.027	0.02	500	0.019	0.03	436	0.014	0.04	436
NN-FBP	16	0.037	0.04	829	0.027	0.04	643	0.019	0.05	847	0.014	0.07	673
NN-FBP	32	0.037	0.08	1560	0.027	0.08	1557	0.019	0.11	1579	0.015	0.14	1627
NN-FBP	64	0.036	0.16	4710	0.027	0.17	4113	0.019	0.22	3904	0.015	0.29	4375

Table 5.3: Results for the limited-data use-case, simulation data. See Table 5.1 for more information.

		$N_\theta = 8$			$N_\theta = 16$			$N_\theta = 32$			$N_\theta = 64$		
	N_h	$\langle e_p \rangle$	$T_r(s)$	$T_t(s)$	$\langle e_p \rangle$	$T_r(s)$	$T_t(s)$	$\langle e_p \rangle$	$T_r(s)$	$T_t(s)$	$\langle e_p \rangle$	$T_r(s)$	$T_t(s)$
FISTA		0.000	24.1		0.000	30.6		0.000	41.1		0.000	54.6	
FBP		0.134	0.00		0.073	0.00		0.051	0.00		0.039	0.00	
SIRT		0.048	1.60		0.029	1.66		0.024	1.85		0.023	2.19	
NN-FBP	1	0.046	0.00	156	0.031	0.00	146	0.022	0.00	161	0.019	0.01	183
NN-FBP	2	0.040	0.01	187	0.028	0.01	211	0.021	0.01	199	0.019	0.01	208
NN-FBP	4	0.041	0.01	377	0.025	0.01	371	0.020	0.01	261	0.017	0.02	373
NN-FBP	8	0.041	0.02	585	0.023	0.02	327	0.020	0.03	779	0.019	0.04	583
NN-FBP	16	0.045	0.04	1247	0.022	0.04	1166	0.020	0.05	1058	0.019	0.07	896
NN-FBP	32	0.041	0.08	2835	0.023	0.08	3586	0.021	0.11	2605	0.019	0.14	3058
NN-FBP	64	0.041	0.16	13190	0.023	0.17	3901	0.021	0.22	5226	0.019	0.29	4831

Table 5.4: Results for the limited-data use-case, experimental data. See Table 5.1 for more information.

Experimental data

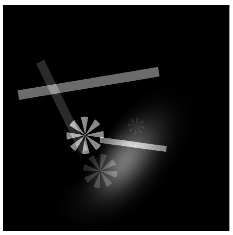
Results for the limited-data use-case and experimental data, given in Table 5.4, show similar results, where the NN-FBP reconstructions approximate the FISTA reconstructions more accurately than both FBP and SIRT. Again, it takes significantly more time to reconstruct the images using FISTA than to reconstruct them using NN-FBP. Reconstructions of a single slice of the data are given in Figs. 5.10m to 5.10p.

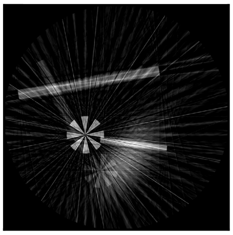
Other experiments

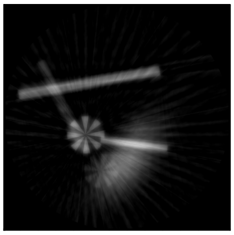
We will now discuss other experiments we performed to determine the properties of the NN-FBP reconstruction method.

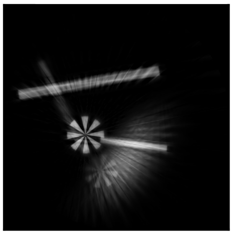
Size of the training and validation set

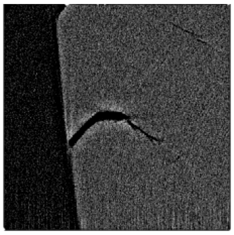
To investigate the training and reconstruction properties of the NN-FBP method, we took the N_θ-reduction use-case with simulation data over 16 angles, and the NN-FBP method with 8 hidden nodes. We trained the method 10 times, starting each time with random weights, for different sizes of the training and validation set, and calculated

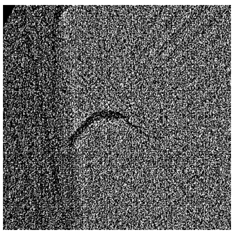

(a) Original

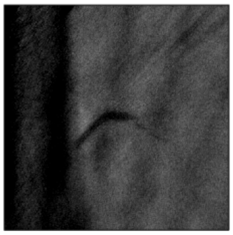

(b) FBP

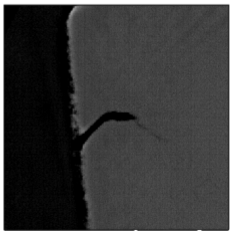

(c) SIRT

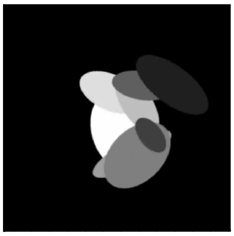

(d) NN-FBP

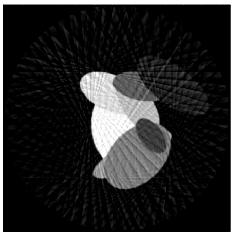

(e) Original

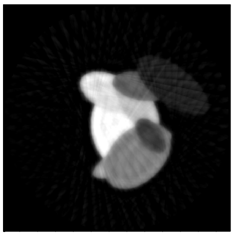

(f) FBP

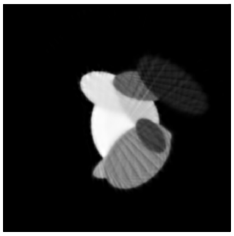

(g) SIRT

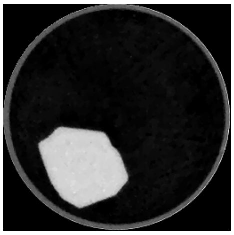

(h) NN-FBP

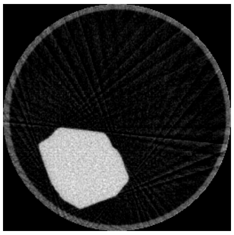

(i) Original

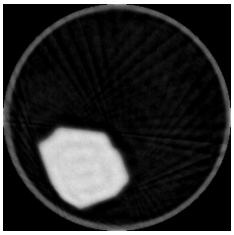

(j) FBP


(k) SIRT


(l) NN-FBP


(m) Original


(n) FBP


(o) SIRT

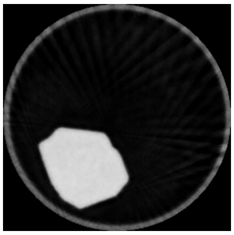

(p) NN-FBP

Figure 5.10: Reconstructions of the objects in the left column, obtained from projection data over 32 angles by FBP, SIRT, and NN-FBP with 8 hidden nodes. In the bottom two rows, the original object was obtained by applying the FISTA algorithm on the full set of 32 available projections.

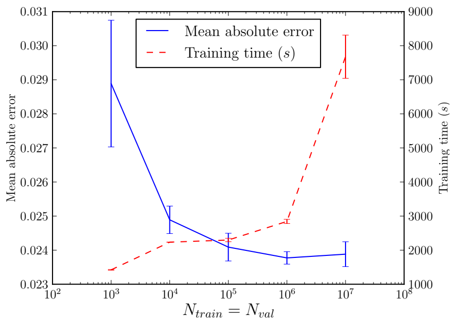


Figure 5.11: The average mean absolute error and training time for 10 runs of the N_θ-reduction use-case, simulation data, with $N_\theta = 16$ and $N_h = 8$. The error bars indicate standard deviation.

the resulting mean absolute error with the test set, and measured the training time. Results are given in Fig. 5.11.

The results show that the mean absolute error of the resulting trained network decreases with increasing training set and validation set size. After a certain size, however, increasing the size further does not seem to lower the error significantly. The time it takes to train NN-FBP becomes larger with increasing set size. Figure 5.11 also shows that for sufficient set sizes, the standard deviation of the mean absolute error is low. This is important for practical applications, since it shows that one has to train NN-FBP only once, without risk of obtaining a badly trained network.

Noise in the projection data

To investigate the effect of noise in the projection data on NN-FBP, we added different levels of Poisson noise to the simulation data of the N_θ-reduction use-case. FBP reconstructions of the noisy projection data with 1024 projection angles were used as training examples for training the NN-FBP method. After training NN-FBP to reconstruct using only 32 projection angles of the noisy data, we reconstructed a single image of the test set, and calculated the mean absolute error of the reconstruction, compared to the noiseless phantom image (Fig. 5.10a). Results are given in Fig. 5.12. The reconstructions obtained by NN-FBP are more accurate than both FBP and SIRT for all noise levels, with the mean absolute error being much lower than FBP. The artifacts in the FBP reconstructions would make further analysis of the object difficult, especially at high noise levels.

To investigate the effect of noise on the training phase of NN-FBP, we trained the NN-FBP method 10 times on a single data set, each time with independently generated noise applied. In every run, the network was trained on 10^6 pixels from a training and validation set of 100 images of the threeshape family, generated on a 1024×1024 pixel grid, with projection data of 32 angles, rebinned to 256 detectors. FBP reconstructions of the noisy projection data with 1024 projection angles were used as training examples. For a test set of 100 images similar to the training set, we

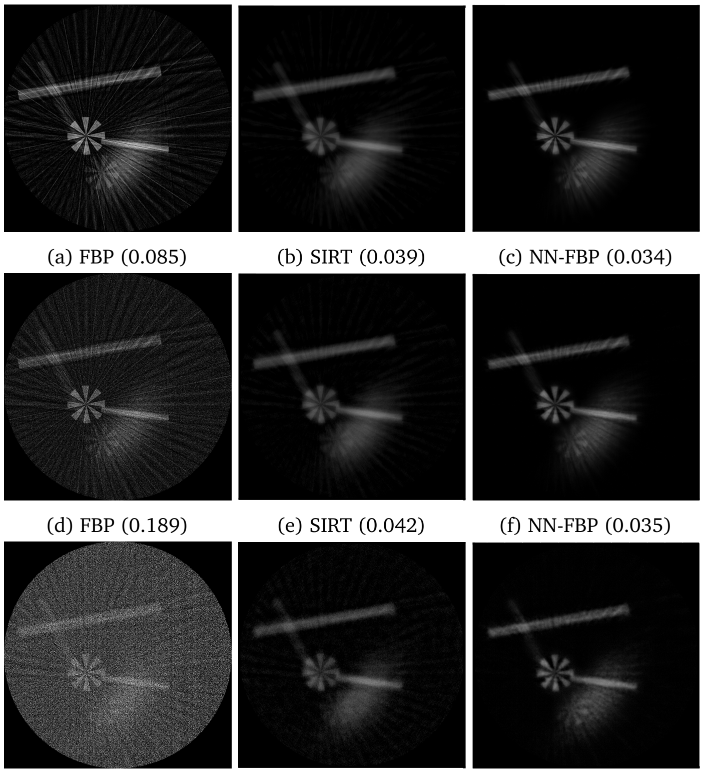


Figure 5.12: Reconstructions of the object from Fig. 5.10a, obtained from projection data over 32 angles with Poisson noise by FBP, SIRT, and NN-FBP with 8 hidden nodes. Each row has an increasing amount of added noise. The mean absolute error of the reconstructions, compared to Fig. 5.10a, is given between parentheses. The errors of FBP with the Shepp-Logan filter are 0.078, 0.157, and 0.452, for increasing amount of added noise. For FBP with the Hann filter, errors are 0.062, 0.090, and 0.216, respectively.

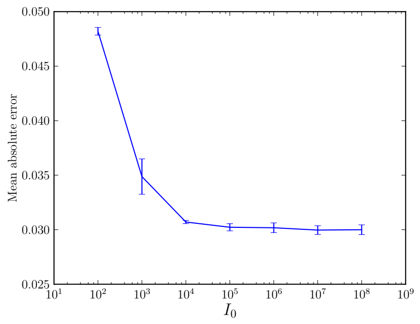


Figure 5.13: The average mean absolute error for 10 runs of the N_θ-reduction use-case, simulation data (256×256 pixels), with $N_\theta = 32$, $N_h = 8$, and Poisson noise. The Poisson noise is generated independently for each of the 10 runs. Lower values of I_0 correspond to larger amounts of noise. Error bars indicate standard deviation.

report the average mean absolute error of the noiseless phantom with the NN-FBP reconstructions, which calculated using noisy projections.

Results are given in Fig. 5.13. These results show that the mean absolute error decreases smoothly with decreasing noise levels. Furthermore, the standard deviation of the mean absolute error is relatively small compared to the error itself, for all noise levels. This indicates that noise in the projection data does not have a large impact on the ability of NN-FBP to find filters that minimize the training error. One reason for this robustness could be that we are able to use a large number of training examples, thereby reducing the influence of the noise by averaging its effect on each example.

Hidden node output

To gain a better insight in how NN-FBP is able to produce accurate reconstructions, we can look at the output of the hidden nodes of the network. Since the neural network of NN-FBP reconstructs a single pixel, we can view the output value of a single hidden node as a pixel of an image. In other words, we can look at the FBP reconstructions of each hidden node of Eq. (5.23). To obtain the final output of NN-FBP, these individual reconstructions are added together with an additional constant offset, and the sigmoid function is applied to each pixel value.

Figure 5.14 shows four of the eight hidden node output images, resulting from a reconstruction of Fig. 5.10i with data for 32 projection angles and NN-FBP with 8 hidden nodes. The results show that each hidden node reconstructs a different feature of the final reconstruction: some focus on the broad shape of the object, while others focus on the edges. Furthermore, the relative contrast of the different ellipses in the reconstructed object differs for each hidden node output image. These results, in addition to the other results in this section, show that there is something to gain by using multiple nonlinear FBPs to reconstruct an object, compared to using a single standard FBP.

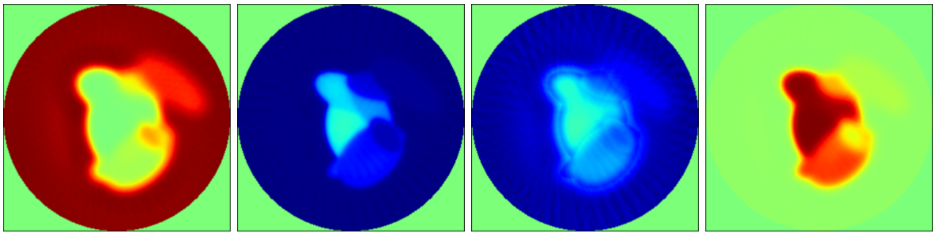


Figure 5.14: Hidden node output images of a reconstruction of the object from Fig. 5.10i, obtained from projection data over 32 angles, with 8 hidden nodes. Blue, green, and red indicate negative, zero, and positive values, respectively.

Exponential binning

To test the influence of exponential binning on both the reconstruction quality and training time of the NN-FBP method, we trained NN-FBP both with and without exponential binning. Both times, the network was trained on 10^6 pixels from a training and validation set of 100 images of the threeshape family, generated on a 1024×1024 pixel grid, with projection data of 32 angles, rebinned to 256 detectors. After training, both networks were used to reconstruct a test set of 100 images similar to the training and validation images.

With exponential binning, training the network took 673 seconds, and the resulting mean absolute error with the test set was equal to 0.0246. Without exponential binning, the mean absolute error with the test set was 0.0239, which is 3% lower. The time to train the network, however, increased to 55178 seconds, 82 times longer than with exponential binning. These results show that, although exponential binning can slightly impact the reconstruction quality of NN-FBP, it greatly reduces the time it takes to train the method.

5.7 Conclusion

In this chapter, we presented a new reconstruction method, the neural network filtered backprojection method (NN-FBP), for limited-data 2D parallel-beam tomography problems. The method is based on artificial neural networks, which allows it to learn problem specific knowledge to improve its reconstruction quality. Furthermore, we showed that NN-FBP can be viewed as a combination of several standard FBP operations, each with a custom filter. This property ensures that the computation complexity of NN-FBP is low compared to algebraic reconstruction methods.

In order to train the NN-FBP method, a set of reconstructions with corresponding projection data is needed. Although this requirement presents a problem, it can be satisfied in several practical applications. Here, we focused on two such applications. In one, we first acquire projection data over a large number of angles, and use reconstructions obtained by standard FBP to train NN-FBP on, while using limited data of only a small subset of angles. Afterwards, NN-FBP can reconstruct limited data of similar objects accurately. In the second use-case, we assume that we are not able to acquire data over a large number of angles, but are given limited-data of only a small number of angles. In this case, we can use NN-FBP to imitate a much slower prior-knowledge based reconstruction method, such as TV-minimization methods.

Results for simulation data and experimental data of both use-cases show that NN-FBP is able to produce significantly more accurate reconstructions than standard FBP. The reconstruction time of NN-FBP is slightly higher than the reconstruction time of FBP multiplied by the number of hidden nodes. The results show that even for a low number of hidden nodes, NN-FBP is able to outperform FBP. Interestingly, NN-FBP is also able to produce more accurate reconstructions than SIRT, a much slower iterative algebraic method. Additional experiments show that the method is more robust than FBP when faced with noisy projection data.

The current study focused on two-dimensional parallel-beam tomographic problems, but a similar method can in theory be applied to other tomographic problems. In these cases, the method will be related to other filter-based analytical reconstruction methods. For example, in three-dimensional cone-beam tomography problems, we can design a neural network that can be viewed as a combination of Feldkamp-David-Kress (FDK) operations [FDK84] with custom filters. Similarly, the current method can also be applied to fan-beam problems, with the fan-beam variant of FBP [KS01]. The reconstruction quality of these new methods remains subject of further research.

NN-FBP can also be used to combine the reconstruction of an object with subsequent analysis of the reconstruction. This can be achieved by training NN-FBP using analyzed images of the correct reconstructions as training images. For example, we can train NN-FBP on segmented images of the training data, thereby training it to perform both the reconstruction *and* segmentation in a single step. Other analyses, such as highlighting areas of interest, are also possible. Which type of object analyses can be accurately performed by NN-FBP remains subject of further research.

Since NN-FBP consists only of FBP operations and image addition and multiplication, implementation of the method in current applications is straightforward. Many hardware CT-scanners currently use FBP as their main reconstruction method, which would make replacement with NN-FBP easy, provided that the user is able to specify custom filters. If a heavily optimized version of FBP is available, NN-FBP will be able to use the same optimizations to reduce execution time. The results from this chapter show that NN-FBP can be a significant improvement over FBP for practical applications.