Internal Report 2011–03 May 2011

Universiteit Leiden
Opleiding Informatica

Discrete Tomography

A Neural Network Approach

Jonathan K. Vis

MASTER'S THESIS

Leiden Institute of Advanced Computer Science (LIACS)
Leiden University
Niels Bohrweg 1
2333 CA Leiden
The Netherlands

ABSTRACT

Tomography tries to reconstruct an object from a number of projections in multiple directions. Many application domains can be imagined, but we will focus on high throughput applications, and will therefore try to reduce the number of necessary projections, while being able to generate good quality reconstructions. We apply several forms of Neural Networks, an Artificial Intelligence method. These networks are especially suited for solving underdetermined problems, and therefore well suited to our problem.

Many different variants of Neural Networks are developed since its introduction; some simple, while others can consist of many nodes in many hidden layers increasing its training complexity. We will here focus on the simpler forms of Neural Networks: a feedforward (multilayer) perceptron.

We present some experimental results, which demonstrate the capabilities of reconstructing high quality images using relatively simple Neural Networks.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Tomography, or more especially computed tomography, is a technique used in a broad variety of research areas: from medical to industrial, and archaeological to material studies. It can be applied to investigate (non-invasively) the internal structure of many different types of objects and materials. Probably the most recognized application is the X-ray CT scanner, for diagnostic purposes, as is found in many hospitals. The main idea of tomography is to be able to visualize and analyze the internal structure of an object. Usually some source of radiation is used to generate the so-called projection data. However, other techniques can be applied such as magnetic resonance.

In order to visualize the internal structure of an object, the object is examined in various orientations, and when put together, a reconstruction is made. Typically, there is a need for a large number of projections (more than 100) to reconstruct, with adequate quality, an object. This approach, however, has many drawbacks. In areas where a high throughput is required the time needed to generate the projection data is limited, and when examining organic tissue only a limited dose of radiation might be administered without the risk of affecting the tissue. Therefore, we notice a need of good quality reconstructions based on a limited set of projections, which reduces both the scanning time as well as the amount of radiation. From here onward, we will focus on high throughput applications in favor of medical uses, and therefore we will only concentrate on reducing the number of projections, but not the dose of radiation nor any other human aspects such as comfort.

Traditional reconstruction methods tend to generate better quality reconstructions when increasing the number of projections. Furthermore, they are static and general, i.e., they cannot be adapted to a specific application domain. Here, we propose a different strategy. We apply Neural Networks, an Artificial Intelligence method [12], for generating reconstructions. These networks must be trained (which implies a one time increase in effort), but

carry the advantage of being capable of reconstructing specific images, and improve themselves.

This master thesis is written as a partial fulfillment of the requirements of the degree Master of Science in the Leiden Institute of Advanced Computer Science (LIACS) of Leiden University, and is supervised by Joost Batenburg and Walter Kosters.

1.1 Related Work

Computed Tomography is a well studied field, and there are many publications describing virtually all its aspects. In [8, 9, 4, 7] the fundamentals of computed tomography, as well as many technical aspects are covered.

The application of Neural Networks is a relatively new approach. In literature they are introduced as a reconstruction technique in [10] and [11]. In general, neural networks seem an uninteresting strategy for the general problem of tomography due to its nature of dealing with large numbers of projections, and consequently the large number of variables. It seems quite hard to outperform traditional reconstruction techniques. However, we will here focus on tomography problems consisting of a small set of projections resulting in an underdetermined problem. Neural Networks are well-known for its successful application to underdetermined problems.

In [2, 3, 1] the authors introduce Neural Networks successfully for reconstruction binary images (i.e., black and white). Two different network topologies are investigated; a full-image network, and a single-pixel network. The first variant tries to reconstruct a complete image at once from all projection data. The second variant reconstructs one pixel from a selection of the projection data. Based on their conclusions we will here focus on the single-pixel network topology. The networks used in [3] are quite large, consisting of 50–200 hidden nodes. Here, we will use much smaller networks. The disadvantage of applying a single-pixel network is its reduced ability to be trained for specific image classes. This property is, to a much greater extend, available in full-image networks at the expense of increased computational complexity.

Finally, we remark that other Artificial Intelligence methods can be applied, such as Evolutionary Algorithms and Support Vector Machines, as well as more discrete approaches such as Linear Programming.

1.2 Outline

The remainder of this thesis is organized as follows. In Chapter 2 we introduce the mathematical foundations of tomography, and its common implementation. In Section 2.3 we elaborate on the discretization problem as well as the problem of generating the projection data. Chapter 3 is dedicated to the concept and application of neural networks, and some variants are presented. In Chapter 4 we explain the experiments performed and the generated results are presented. We conclude the thesis by a summary of the main conclusions in Chapter 5.

Chapter 2

Tomography

In *tomography*, we try to reconstruct an object from a number of *projections* in multiple directions. Here, we will focus on projections obtained by parallel beams through a finite object. We assume that this object is contained in the disc

$$A = \left\{(x, y) \in \mathbb{R}^2 : x^2 + y^2 \leq R^2\right\} \tag{2.1}$$

with radius $R > 0$, see Figure 2.1. The object is an image described by the real-valued grayscale mapping $f : A \to [0, 1]$ where 0 is black and 1 is white; intermediate values can be interpreted as shades of gray. The hatching in Figure 2.1 only defines the outline of the object, and not its internal structure.

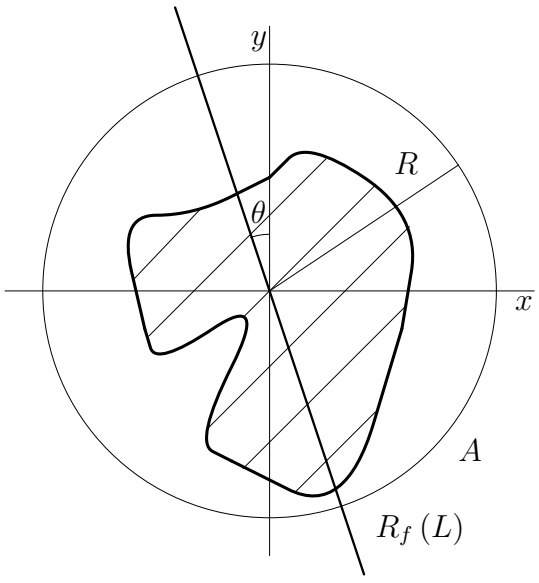

Figure 2.1: The basic principle of tomography.

The attenuations of the beams are measured on an infinite *detector*. Different projections are generated by rotating the detector around the object. The construction of the projections is performed by the so-called *Radon transform*, which is the integral transform of the function f over straight lines L:

$$R_f(L) = \int_L f(\ell)\, d\ell. \tag{2.2}$$

For angle θ we define: $L_{\theta,\tau} = \{(x,y) \in A : \tau = t\}$, with $t = x\cos\theta + y\sin\theta$. The Radon transform P_f of the function f is defined as:

$$P_f(\theta,\tau) = \int_{L_{\theta,\tau}} f(x,y)\, ds \qquad \text{for } \theta \in [0,\pi), \tau \in \mathbb{R}. \tag{2.3}$$

The reconstruction of the original image from its projections is obtained from applying the *inverse Radon transform* [6]:

$$f(x,y) = \int_{\theta=0}^{\pi}\int_{\tau=-\infty}^{\infty} h(\tau - t)\, P_f(\theta,\tau)\, d\tau\, d\theta, \tag{2.4}$$

where h is a suitable weight or *kernel* function acting as a filter, see Section 2.2.

Discrete tomography focusses on the reconstruction of images, which are reconstructed using a discrete set of pixel values or a discrete rasterization of the function f.

To find a discrete approximation, we substitute the integrals for summations. First, we choose a fixed number of angles k (equally dividing the 0 to π semicircle):

$$f(x,y) = \sum_{d=1}^{k}\int_{\tau=-\infty}^{\infty} h(\tau - t)\, P_f(\theta_d,\tau)\, d\tau, \tag{2.5}$$

where θ_d is the d-th angle. We now simplify the expression for the kernel function by introducing $\tau' = \tau - t$:

$$f(x,y) = \sum_{d=1}^{k}\int_{\tau'=-\infty}^{\infty} h(\tau')\, P_f(\theta_d,\tau' + t)\, d\tau', \tag{2.6}$$

and finally, approximate the remaining integral by choosing a finite detector size, D, so $h(\tau') = 0$ when $|\tau'| > D$:

$$f(x,y) = \sum_{d=1}^{k} S(d,t), \tag{2.7}$$

where we defined:

$$S(d,t) = \sum_{\tau'=-D}^{D} h(\tau') P_f(\theta_d, \tau' + t). \tag{2.8}$$

As the kernel is constant for equally spaced angles in the semicircle, we can precompute the summation in (2.8) before applying the kernel function h:

$$f(x,y) = \sum_{\tau'=-D}^{D} h(\tau') \sum_{d=1}^{k} P_f(\theta_d, \tau' + t). \tag{2.9}$$

2.1 Filtered Back Projection

In tomography the calculation of Equation (2.7) and Equation (2.8) are usually performed by the *filtered back projection* algorithm, where h is, usually, the Ram-Lak kernel introduced in Section 2.2.

In practical applications, filtered back projection is implemented by calculation via the *frequency domain*, or more especially the *Fourier domain*. In the Fourier domain, the convolution operator (see Equation (2.10)) translates to a much simpler multiplication, and therefore reduces the computational complexity. The convolution is performed by applying the kernel to the projection data. Note the similarity between Equation (2.4) and Equation (2.10). The convolution operator is defined as:

$$\int_{-\infty}^{\infty} \phi(\tau') \psi(t - \tau') \, d\tau', \quad 0 \leq t \tag{2.10}$$

Here, we will not use the Fourier domain, but rather calculate the convolution of the kernel with the projection data in the *spatial domain*. As is suggested in Equation (2.8) we choose the kernel to be static, and consequently shift the projection data against it. Therefore we introduce a *shift operator* which aligns the projection data for a certain image pixel x, y with corresponding τ', and a certain angle θ_d. The shift amount is denoted by t.

This implies, on a finite detector, that some projection data will be shifted outside the detector range, and some "new" projection data is shifted onto the detector. We will deal with this phenomenon as follows: the data shifted out of range is discarded, and the new data is treated as being 0, as it would be on an infinite detector.

In Figure 2.2 some reconstructions are presented generated with the filtered back projection algorithm with varying numbers of projection angles. Note the increase in the so-called image artifacts by decreasing numbers of projections.

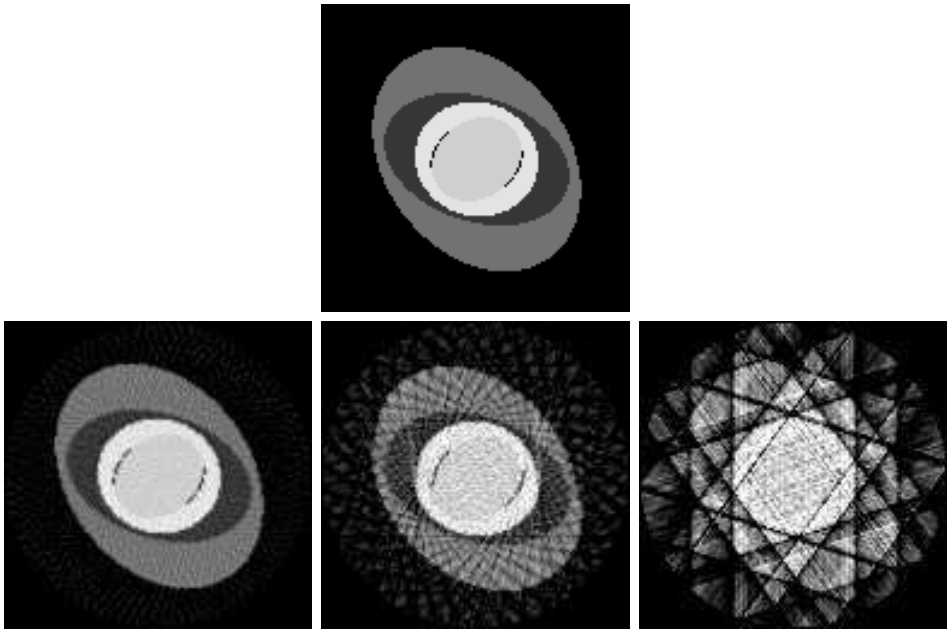

Figure 2.2: Top row: the original 128×128 image. Bottom row from left to right: filtered back projection reconstructions for the number of projection angles $k = 50, 20, 5$.

2.2 Kernel Function

In reconstructing an image from its Radon transform, we use a so-called *kernel function*. This function is not of arbitrary form. Each pixel is a projection onto the detector at a certain place τ (for angle θ_d). We expect that the measured value on the detector at position τ is likely to contribute highly to the reconstruction of the pixel in question, and values at a large distance from τ will not influence the reconstruction of that pixel so much. Several kernel functions can be used. Often the so-called *Ram-Lak* kernel, only defined in the integer domain, or *ramp* filter is applied, see Figure 2.3:

$$h(\sigma) = \begin{cases} \frac{\pi}{4} & \text{if } \sigma = 0, \\ -\frac{1}{\pi^2 \sigma^2} & \text{if } \sigma \text{ is odd}, \\ 0 & \text{otherwise}, \end{cases} \tag{2.11}$$

where $\sigma \in \mathbb{Z}$. Note that the kernel is symmetric around 0, as expected.

Besides the Ram-Lak kernel there are other possibilities. The Ram-Lak kernel is extremely sensitive to noise because of the emphasis on the higher frequencies. In implementation domains, subject to noise, often alternative

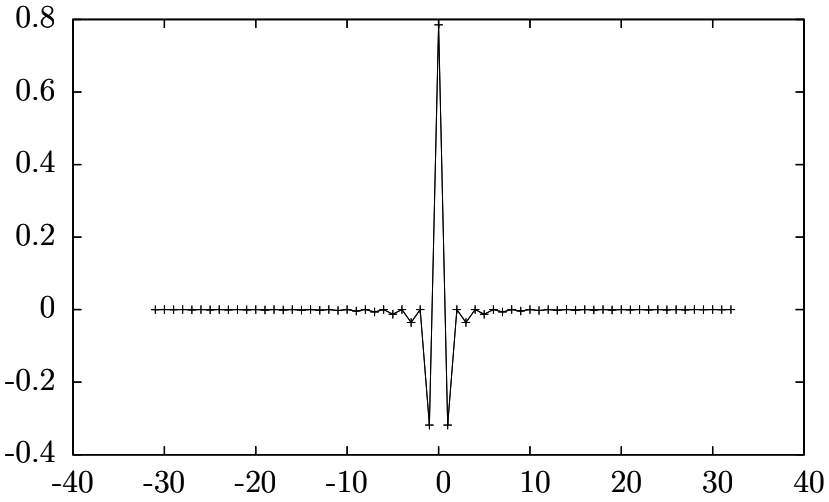

Figure 2.3: Ram-Lak kernel of size 64. The kernel is discrete, however, for better readability the discrete values are connected by linear interpolation.

kernels are used, such as the *Shepp-Logan* kernel, which is a modification of the Ram-Lak kernel by multiplication with the sine function.

2.3 Discretization

Instead of the function f we use a rasterized image I of $N \times N$ *pixels*, with integer $N > 1$. In order to perform the computations in the real-valued domain we use the following conversion rules for a pixel $I(r, c)$:

$$x = \frac{2c}{N} - 1 \qquad \text{for integer } c \in [0, N-1], \tag{2.12}$$

$$y = \frac{2r}{N} - 1 \qquad \text{for integer } r \in [0, N-1], \tag{2.13}$$

where $(x, y) \in A$ and we defined A to be the disc with radius $R = 1$. Although the image I is square, we disregard all pixels for which $x^2 + y^2 > 1$. This allows us to define the detector size D to be 1. This detector will also be rasterized into a line of N pixels.

The conversion functions map a pixel $I(r, c)$ to the interval $[-1, 1]$ which will, in turn, be mapped to a pixel on the detector. In general, a pixel will not be mapped to a single pixel on the detector, it will instead be mapped

between two pixels. Now, we have to decide how the image pixel will contribute to the possible pixels on the detector. Several strategies can be applied. Roughly, they can be divided into two categories. First, a single pixel on the detector receives the full contribution of an image pixel, for example, the *nearest neighbor* approximation. Secondly, we can distribute the contribution of an image pixel over the detector pixels, as is, for instance, done by *linear interpolation*.

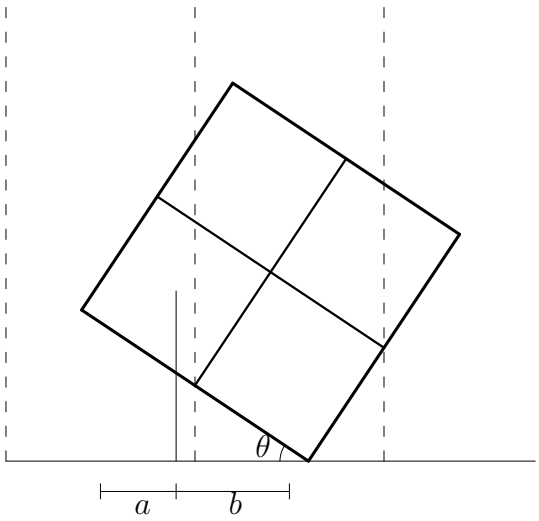

Figure 2.4: Subsampling and linear interpolation with $m = 2$.

In order to increase the accuracy of the projections, and, ultimately, the reconstruction, we apply a *subsampling* technique. This strategy is especially beneficial for small values of N. Each pixel in I is divided into $m \times m$ *subpixels*, where m is the *sampling rate*, with integer $m > 1$. Typically, $m = 2$ or $m = 3$. Each subpixel is then projected onto the detector. Again, a subpixel is, in general, mapped between two pixels on the detector, and we must now apply one of the aforementioned strategies. There are, however, some difficulties with this technique. Assume our image to be perpendicular to the detector position ($\theta = 0$). Each pixel (or its center) coincides with the centre of a pixel on the detector. So if no subsampling is applied the projection data is the sum of the pixel columns of the image. This is exactly as expected. If we now introduce a subsampling, e.g., let $m = 2$, the contribution of a pixel from the image is now distributed over three pixels on the detector, as is demonstrated in Figure 2.5. This effect may be unwanted and responsible for unpredictable results during reconstruction. To counteract this effect we propose the same subsampling technique on the detector. Each detector pixel is divided into m

subpixels, which are aggregated (summed) to a single pixel later. The centers of each subpixel in the image coincide with the centers of the subpixels on the detector. After aggregation, the result is the same as before, indifferent to the level of subsampling. This method still preserves the beneficial effects of subsampling for $\theta \neq 0$.

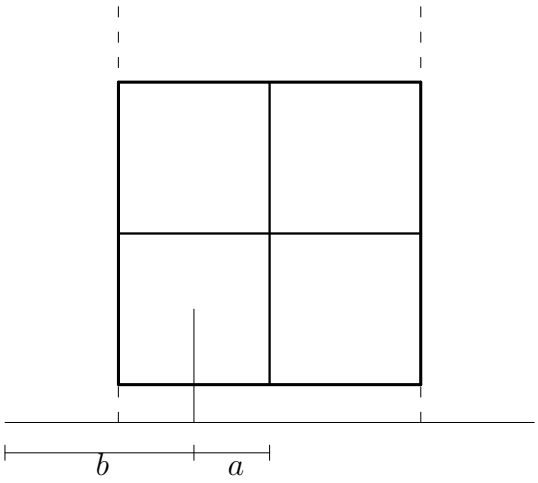

Figure 2.5: Subsampling and linear interpolation for $\theta = 0$.

A second pitfall lies within the implementation details of the subsampling algorithm. A naive approach can easily lead to several subpixels being mapped outside the detector range, while their counterparts are not.

Although the aforementioned approaches work well in practise, they do not satisfy Equation (2.7). A more accurate, but perhaps also more elaborate, projection technique involves calculating the mass "above" a certain detector pixel. As such a pixel has a certain width we overlay the image with bands and calculate the mass within a band to be the value of that detector pixel. The resolution of the detector can be chosen freely. In general these bands intersect the image pixels in all possible ways, see Figure 2.6.

An image pixel, which measures 1×1, can be divided into three separate regions: two triangular regions, and a parallelogram. The areas can be calculated as follows, for a given projection angle θ, with $0 \leq \theta \leq \frac{\pi}{4}$. Let I_a be an interval between p_1 and p_2 with width w at offset s from p_1. The area A_a of the corresponding "capped" triangle is calculated as follows:

$$A_a = \begin{cases} \frac{w(w+2s)}{\sin 2\theta} & \text{if } \theta \neq 0, \\ 0 & \text{otherwise.} \end{cases} \tag{2.14}$$

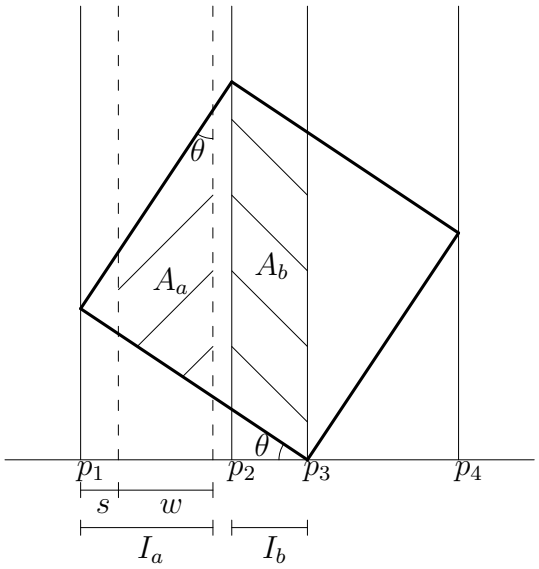

Figure 2.6: Projected mass calculation.

Note that $d(p_1, p_2) = \sin\theta$. If $s = 0$, we get the value $\frac{w^2}{\sin 2\theta}$ (if $\theta \neq 0$) for A_a, which easily generalizes to a more general formula. So, if $s = 0$ and $w = d(p_1, p_2)$, we get $A_a = \frac{1}{2}\tan\theta$.

Let I_b be an interval between p_2 and p_3 with width w. The area A_b of the parallelogram is calculated as follows:

$$A_b = \frac{w}{\cos\theta} \tag{2.15}$$

Note that $d(p_2, p_3) = \cos\theta - \sin\theta$. So, if $w = d(p_2, p_3)$, the area A_b is $1 - \tan\theta$. Combined with the areas A_a and A_c the total area equals 1 for the pixel.

Let I_c be an interval between p_3 and p_4 with width w at offset s from p_4. This interval is analogous to I_a, so the corresponding area is calculated as follows:

$$A_c = \begin{cases} \frac{w(w+2s)}{\sin 2\theta} & \text{if } \theta \neq 0, \\ 0 & \text{otherwise.} \end{cases} \tag{2.16}$$

Empirical results from comparing these projection strategies show that there is not a great difference in the average absolute error between reconstructions generated from subsampled ($m = 2$) projection data, and the projected mass approach. In order to speed up the calculations of the projection data we will use the subsampling technique. Note that in all aforementionend projection strategies the total mass of the reconstruction is always equal to the total mass of the corresponding object.

Chapter 3

Neural Networks

An *(Artificial) Neural Network* is a computational model that is inspired by the structure of a biological neural network such as the human brain [12]. It consists of interconnected neurons passing information to each other. The structure is often adaptive based on internal or external data. This concept is referred to as learning. Many different forms exist today. Here, we will first focus on a simple form of a feedforward network called a *perceptron*.

3.1 Linear Perceptron

A perceptron is the simplest of feedforward networks, i.e., a linear classifier:

$$\text{net}\,(\vec{a}) = \vec{w} \cdot \vec{a} - b, \tag{3.1}$$

where $\vec{a}$ is a real-valued input vector which is mapped to an output value (also real-valued) $\text{net}\,(\vec{a})$, $\vec{w}$ is a real-valued vector of weights, and b is a "bias", a constant term that does not depend on any input. To model the bias we define $a_0 = -1$, which is connected by the bias weight w_0, and we get $\text{net}\,(\vec{a}) = \vec{w} \cdot \vec{a}$, where we added one dimension to the vectors.

Training the network is performed by updating the weight vector according to the error observed by offering a training input. The error is defined as $t - \text{net}\,(\vec{a})$, with t the target output. The update rule for a single weight w_i, with $0 \leq i \leq n$ is:

$$w_i \leftarrow w_i + \alpha\, a_i\, (t - \text{net}\,(\vec{a})) \tag{3.2}$$

or in vector form:

$$\vec{w} \leftarrow \vec{w} + \alpha\, \vec{a}\, (t - \text{net}\,(\vec{a}))\,, \tag{3.3}$$

where $\alpha \in [0, 1]$ is the learning rate, and n the number of input nodes of the perceptron excluding the bias input a_0. For linear perceptrons we will fix the

value of α to 0.01. Often a variable learning rate is used which is decreased over training or adjusted according to the current error.

The computational properties of a linear perceptron show a remarkable similarity with the computations in Equation (2.7) and Equation (2.8). We expect a linear perceptron to be able to simulate these computations. The weights should form the kernel function, and, for many projections, we expect the weight vector be very similar to the Ram-Lak kernel.

A perceptron performs best if its input vector is in a certain domain depending on the current weight vector initialization and α value. Commonly values from the interval $[0, 1]$ are used. From Equation (2.3), we can clearly observe that the generated input vectors will be outside this domain. Therefore, we introduce a linear scaling with N.

The order of the summation in Equation (2.7) and Equation (2.8) can be changed, because the kernel function h is constant for d and t. This allows us to precompute the summation of the projection data for all angles. Each projection has an equal share in the reconstruction of a certain pixel. If we perform this precomputation we can reduce the number of input nodes for our perceptron to N and therefore also the number of weights. A reduced number of weights should make training easier and faster. Therefore we propose the topology in Figure 3.1, where ΣPr is the symbolic notation for the precomputed summation for all projection angles d.

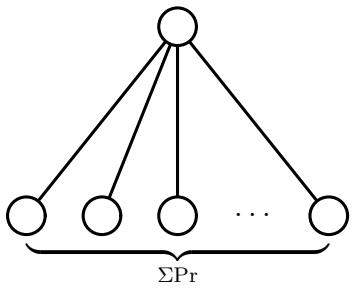

Figure 3.1: Linear perceptron with N input nodes, and one output node. The total number of weights is $N+1$. The bias node and weight are not depicted, but assumed to be present within the output node.

This topology consists of only one output node capable of reconstructing one pixel of the image. In order to reconstruct the complete image we apply the perceptron N^2 times. The input for reconstructing a single pixel is heavily dependent on that pixel. Therefore, we preprocess the Radon projection data

for depending on a certain pixel. The projection data is "shifted" such that the projection (according to Equation (2.3)) is centered around τ. Missing values are replaced by zeros (as it would be on an infinite detector).

The aforementioned topology handles all input data independently, because of the precomputation optimization. This topology is therefore only useful for projection data generated from a set of angles that equally divides the full semicircle. However, in practice, it might be the case that certain projections are flawed or missing altogether. Furthermore we can imagine situations in which it is impossible to gather projection data from every angle within the semicircle, e.g., only a section of the semicircle can be measured. In these cases an "exact" reconstruction is nearly impossible. Neural Networks, on the other hand, are especially capable of handling underdetermined data. To cope with these kinds of data we propose an alternative topology in Section 3.5.

3.2 Multilayer Perceptron

A *multilayer perceptron* is a feedforward network organized in multiple layers (an input and an output layer, as well as (several) hidden layers), see Figure 3.2. Each layer is fully connected to the next. In contrast to the perceptron model each node has a nonlinear activation function. It can be shown that each multilayer perceptron using only linear activation functions has an equivalent perceptron model. In practice two nonlinear functions are used:

$$\phi_1(x) = \tanh(\beta x), \tag{3.4}$$

and

$$\phi_2(x) = \frac{1}{1 + e^{-\beta x}}, \tag{3.5}$$

where the steepness parameter β is often chosen equal to 1. Both activation functions have easy to calculate first derivatives, which come handy in the back propagation algorithm: $\phi_1'(x) = \beta\phi_1(x)(1 - \phi_1(x))$, and $\phi_2'(x) = \beta(1 - \phi_2(x)^2)$ respectively. Here, we use Equation (3.5) as its output domain is the interval $[0, 1]$, naturally suited to our application domain. Often, Equation (3.4) is used as activation for the nodes in the hidden layers as it is symmetrical around 0. However, we use Equation (3.5) as the activation function for the nodes in the hidden layer as no significant benefit was gained by applying the other one, and it reduces the complexity of the implementation.

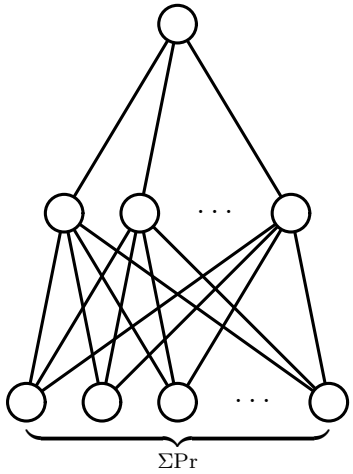

Figure 3.2: Multilayer perceptron with N input nodes, h hidden nodes, and one output node. The total number of weights is $h\,(N+1)+h+1$ (including weights for the bias nodes). The bias nodes and weight are assumed to be present in each hidden and output node.

The weight update rules for multilayer perceptrons are different from the update rule for a linear perceptron. There are two cases:

$$w_{j,i}^{(1)} \leftarrow w_{j,i}^{(1)} + \alpha \cdot \phi\left(\text{in}_j^{(1)}\right) \cdot \Delta_i^{(1)}, \quad \Delta_i^{(1)} = \text{Error}_i \cdot \phi'\left(\text{in}_i^{(0)}\right), \tag{3.6}$$

where $w_{j,i}^{(1)}$ is the weight from node j in the hidden layer to node i in the output layer, Error $= \vec{t} - \text{net}\,(\vec{a})$ analogous to the linear perceptron (note that target $\vec{t}$ and net output net$(\vec{a})$ are vectors), $\text{in}_i^{(0)} = \sum_j w_{j,i}^{(1)} \phi\left(\text{in}_j^{(1)}\right)$ is the weighted input for node i (summation over hidden nodes j), and $\text{in}_j^{(1)} = \sum_k w_{k,j}^{(2)} a_k$ (summation over input nodes k), and:

$$w_{k,j}^{(2)} \leftarrow w_{k,j}^{(2)} + \alpha \cdot a_k \cdot \Delta_j^{(2)}, \quad \Delta_j^{(2)} = \phi'\left(\text{in}_j^{(1)}\right) \sum_i w_{j,i}^{(1)} \Delta_i^{(1)}, \tag{3.7}$$

where $w_{k,j}^{(2)}$ is the weight from node k in the input layer to node j in the hidden layer, and $\sum_i w_{j,i}^{(1)} \Delta_i^{(1)}$ is the summation over the output nodes i. Note that we will commonly use only one output node.

3.3 Initialization

The training rules introduced in Section 3.1 and Section 3.2 are completely deterministic. However, commonly, a nondeterministic initialization is used

to avoid a guaranteed trap in a subglobal "optimum". The initialization is, therefore, not trivial and may have a huge impact in the convergence of the Neural Network. For linear perceptrons this effect is, generally, not too severe as the magnitude of the adjustment of the weights is always the same. A wrongly initialized network is likely to converge eventually. Note that initialization outside the interval $[-1, 1]$ may result in divergent behavior. In all cases, a suitable distribution must be chosen. Two distributions are commonly used; the uniform distribution, and the normal distribution. We will use the normal distribution. For a linear perceptron we fix $\mu = 0.5$, and $\sigma = 0.25$, where μ is the aimed for mean, and σ the standard deviation.

For nonlinear perceptrons, and consequently, multilayer perceptrons the initialization problem is harder. The introduction of nonlinearity is the main culprit. For both activation functions from Equation (3.4) and Equation (3.5) the behavior is similar. The use of the first derivative in updating the weights results in a very small update when the input of a node is very large or very small. Only in a small interval, controlled by the parameter β, the weight adjustments are similar to the linear case. Besides the danger of divergent behavior we face an additional hazard: being trapped in an inescapable region of the activation function. A good strategy is to assure that we start (from initialization) in the region where we have a linear-like behavior. We must be careful with adjusting parameter β as we might end up with a completely linear perceptron, and therefore reducing the additional capabilities of a nonlinear perceptron. We fix $\mu = 0$, and $\sigma = 0.25$ for nonlinear perceptrons.

3.4 Example Reconstructions

In this section we present some example reconstructions generated by the various perceptrons. In Figure 3.3, we give three reconstructions generated by a linear perceptron, nonlinear perceptron, and a multilayer perceptron. For each of the reconstructions 32 projection angles are used.

Compared with the filtered back projection reconstructions in Figure 2.2, we can distinguish a notable difference. Where we observe in reconstructions generated by the filtered back projection algorithm mainly image artifacts, in the perceptron generated reconstructions they are almost completely absent, however, the object boundaries are much "softer". This effect is especially observed when there are many edges in the images as is the case in this example. The multilayer perceptron behaves better with respect to the sharpness of the reconstruction. The nonlinear perceptron shows a behavior in between. There is more sharpness than in the case of a linear perceptron, however, the intensities seem less acurately reconstructed.

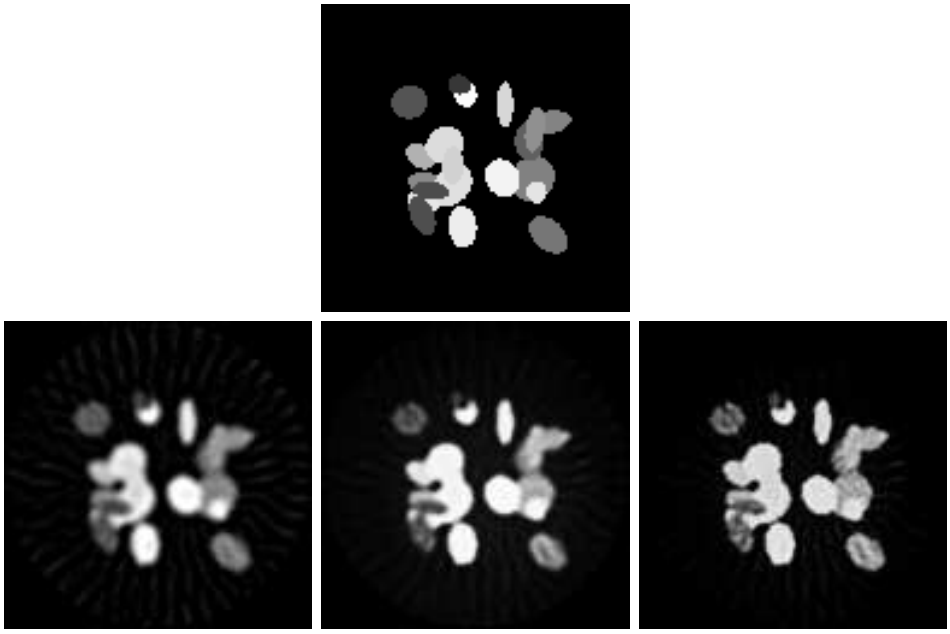

Figure 3.3: Top row: the original 128×128 image. Bottom row from left to right: reconstructions generated by a linear perceptron, nonlinear perceptron, and a multilayer perceptron respectively. In all cases 32 projection angles were used.

3.5 Alternative Topologies

The aggregation introduced in Section 3.1, the precompution of the projection data for all k angles (ΣPr), might be too restrictive for finding better reconstructions. It would be nice to offer all projection data (still shifted for the specified pixel) to the network. However, a naive approach may quickly lead to an explosion of the number of weights in the Neural Network, rendering training practically impossible. We therefore propose an alternative network topology, called *multistage perceptron*, where the number of weights does not exceed the number of weights of the multilayer perceptron, while offering all projection data separately. A multistage perceptron, cf. Section 3.4, consists of a number of perceptrons (here k) connected by another perceptron. It might also be seen as a "partitioned" multilayer perceptron where the input layer is not fully connected to the hidden layer. There are several partitions, maybe one for each projection angle, of the input nodes belonging to a certain hidden node.

We can reduce the number of weights even further by observing the symmetry in the kernel function. We can aggregate the inputs at the same dis-

tance from τ.

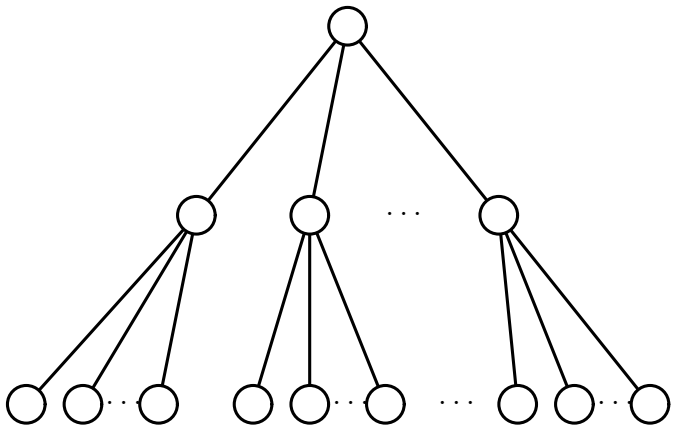

Figure 3.4: Multistage perceptron with a hidden node for each of the k projections.

Another possible reduction in weights can be achieved by a nonlinear aggregation around τ. Instead of creating an input for every detector element we choose an exponentional increase in the distance from τ supported by the observation that pixels far from the current pixel are less likely to contribute highly to its reconstruction. This reduced the number of weights to $k \log N$, where k is the number of projections, and N the dimensionality of the image.

Empirical study of this topology did not show a significant improvement over more traditional perceptron topologies. However, we are convinced that a significant improvement in reconstruction quality can only be achieved by somehow eliminating the aggregation. For the experiments performed in Chapter 4 we will not use this topology, nor will we explore the alternative nonlinear aggregation. Instead we will leave this as a promising area for further research.

Chapter 4

Experiments

In this chapter we describe the experiments performed. We start by introducing artificial image classes, and present the experiment parameters and settings. The first experiment aimes to illustrate the statement in Section 3.1 a linear perceptron is capable of simulating Equation (2.7), and Equation (2.8) We then explore the *cross class* reconstruction capabilities, and finally, experiment with a real-life case study.

4.1 Image Classes

We construct a number of artificial image classes to experiment on. For all image classes we fix the dimensions to 128×128 pixels:

- 2 Ellipses (overlay) — 2 ellipses of random intensities (maybe with the same intensity) which may or may not (partially) overlap each other. The ellipses are drawn in a nondetermisitic order. The last drawn ellipse determines the ultimate intensity,
- 20 Small ellipses (overlay) — 20 small ellipses of random intensities,
- 5 Concentric ellipses (overlay) — 5 concentric ellipses of random intensities; only their respective center points are equal,
- Random noise (1000) — 1000 pixels of a random intensity in an otherwise black image,
- Random noise (10000) — 10000 pixels of a random intensity.

The features of these images are always cropped to the disc A. The first three classes resemble objects consisting of larger homogeneous areas of constant

greylevel, while the random noise images are mainly used to validate results. Some samples of each image class are presented in Figure 4.1.

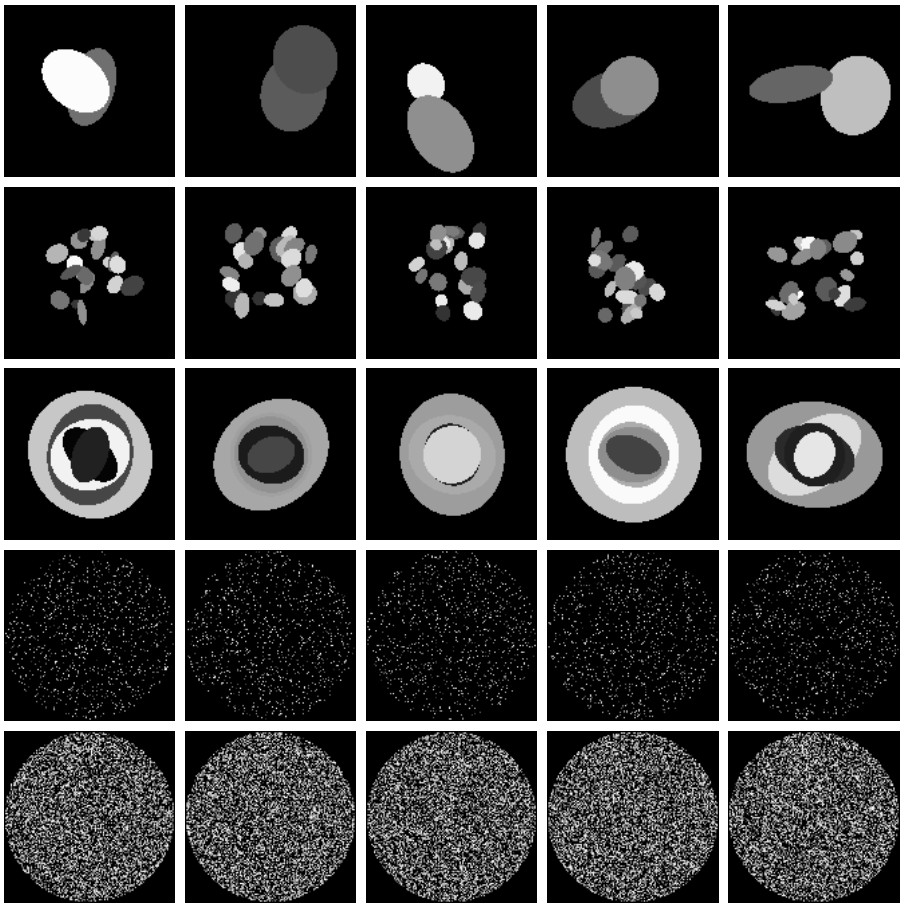

Figure 4.1: Example images from five different image classes named (from top to bottom): 2 Ellipses (overlay), 20 Small ellipses (overlay), 5 Concentric ellipses (overlay), Random noise (1000), and Random noise (10000).

4.2 Experiment Parameters

An *incremental training* approach is used. A single image of a particular image class is generated from which a number of pixels are selected randomly. Each pixel is a training instance. The weights are updated after each training instance. Alternatively, we could have used *batch training*, where a number

of training examples is offered to the network. The resulting changes are aggregated before being applied at once.

We call the usage of an image an *epoch*. In each epoch a number of pixels are selected as training instance. We refer to the number of selected pixels as *iterations*. In Appendix A, a comprehensive table of average absolute errors is given for all image classes and all experiments. Here, the number of epochs is fixed to 10,000, and the number of iterations is fixed to 3,000. A set of 50 predetermined images (for each image class) serves as *validation set*. The reported errors are measured on the validation set. The average absolute error is calculated as the absolute difference between each pixel in the original image with its corresponding reconstructed pixel, and averaged over the total number of pixels within disc A. Furthermore, the errors are averaged over the number of images in the validation set.

As a general rule training requires more examples when the Neural Network contains more weights. Therefore it seems reasonable to have a different number of examples for a linear perceptron and a multilayer perceptron. However, here we fixed the number of training instances for all networks to the aforementioned settings. For the experiments on alternative network topologies we adjusted the number of training examples to the particular network. The precise number of instances is presented with the results.

For many experiments we choose the parameter k, the number of projections, to be $2, 4, 8, 16, 32$. Note that we do not expect "reasonable" reconstructions for $k = 2$, and $k = 4$ (see for example Figure B.3 bottom row). Instead, these settings are used to validate results, and, possibly, gain some insight in the method of approach of the Neural Networks. The projections angles always equally divide the 0 to π semicircle. In Appendix B, five selected reconstructions from the validation set are shown.

4.3 Simulating Filtered Back Projection

The first experiment aims to prove a statement from Section 3.1: A linear perceptron should be able to reconstruct an image from its Radon projections (for a large number of angles) as good as Equation (2.7) and Equation (2.8). As a training set we generate a set of 10,000 (the number of epochs) random 128×128 images from the image class 2 ELLIPSES (OVERLAY), see Figure 4.1. For each image we offer the Radon transform projections for $k = 32$, and 3,000 randomly chosen pixels as target outputs for a total of 30,000,000 training examples. In Figure 4.2, the differences in the reconstructions between filtered back projection and a linear perceptron are shown. As can clearly been observed from Table 4.1, the average absolute errors obtained

from the linear perceptron approach are structually less than the errors from filtered back projection. We can therefore conclude that a linear perceptron is capable of simulating Equation (2.7) and Equation (2.8).

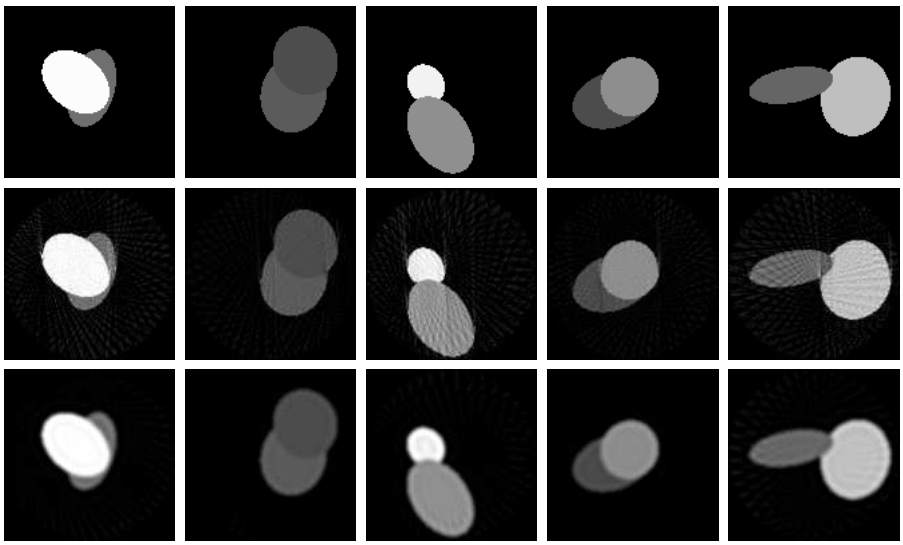

Figure 4.2: The original image (top row) and filtered back projection reconstruction (middle row) versus a linear perceptron reconstruction (bottom row) with 32 projections. The images are taken from the validation set of the 2 Ellipses (overlay) image class.

Table 4.1: Average absolute errors on the filtered back projection reconstruction and the linear perceptron reconstruction.

		Filtered back projection		Linear perceptron	
Image class	Projections	Average error	Standard deviation	Average error	Standard deviation
2 Ellipses (overlay)	32	0.0458	0.0688	0.0205	0.0402
	16	0.0751	0.0862	0.0345	0.0482
	8	0.1198	0.1262	0.0548	0.0632
	4	0.1852	0.2017	0.0861	0.0865
	2	0.2807	0.3135	0.1588	0.1401

From Figure 4.2, some differences in the reconstruction images between the filtered back projection and the Neural Network approach can be ob-

served. So-called *image artifacts* are present in the filtered back projection reconstruction giving the objects in the image a textured appearance, as well as "phantom" objects. These artifacts are strongly oriented to the projection angles. This is especially true for $k < 32$. In the Neural Network generated reconstructions there are less image artifacts at the expense of softer boundaries of the objects.

4.4 Cross Class Validation

In Appendix A, a comprehensive table of average absolute errors is given for all experiments. As can be seen in Appendix A, the average errors are by no means constant with respect to the image classes. Therefore the question arises if a network is trained in reconstructing a certain image class, and can consequently perform worse on other image classes. To investigate this phenomenon we performed a *cross validation*. The comprehensive results are presented in Appendix C. In Figure 4.3, Figure 4.4, Figure 4.5, Figure 4.6, and Figure 4.7, we show graphical representations of these results. Note that the vertical axis has a logarithmic scale.

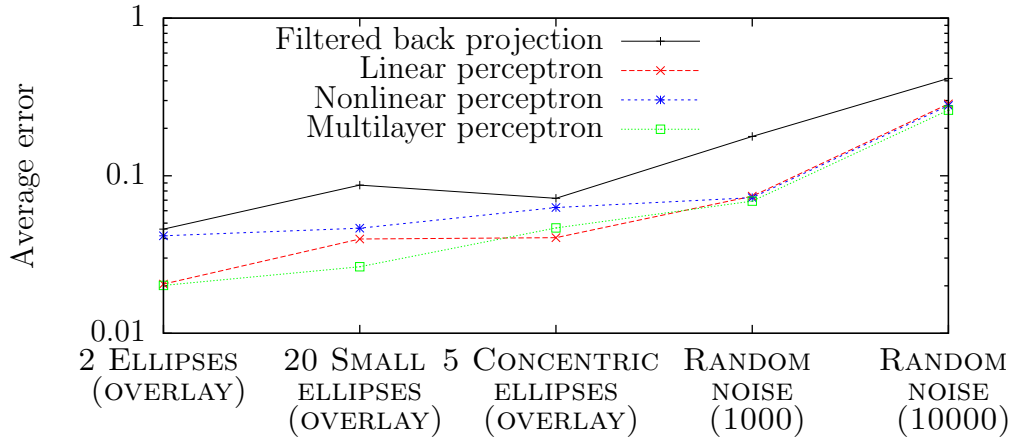

Figure 4.3: Average absolute errors for cross class validation with all image classes. Neural Networks trained on the 2 Ellipses (overlay) image class. We use a linear interpolation between the discrete results for better readability. Note that the results for filtered back projection are constant as no training is performed.

As can be observed from Figure 4.3, Figure 4.4, Figure 4.5, Figure 4.6, and Figure 4.7, there is a notable variation in reconstruction quality between the Neural Network variants. But also, there is a considerable variation in the

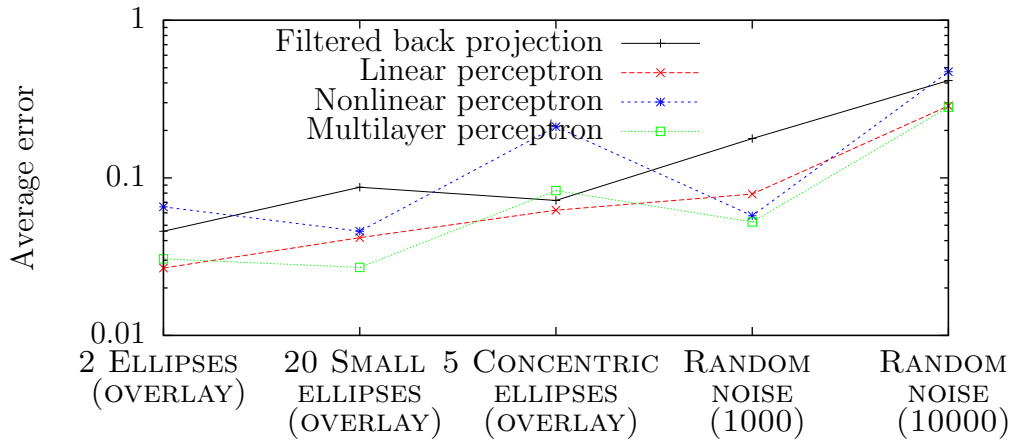

Figure 4.4: Average absolute errors for cross class validation with all image classes. Neural Networks trained on the 20 Small ellipses (overlay) image class.

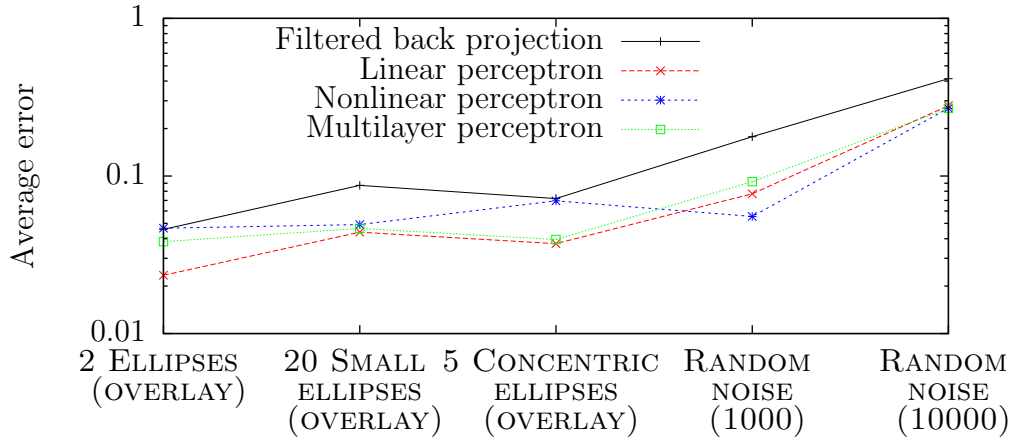

Figure 4.5: Average absolute errors for cross class validation with all image classes. Neural Networks trained on the 5 Concentric ellipses (overlay) image class.

reconstruction quality between the various image classes trained on. Overall there is no clear "winner" indicated. However, for the ellipses image classes the Neural Networks all perform generally better than filtered back projection, with, maybe, an exception for the networks trained on the 20 Small ellipses (overlay) class, and validated against the 5 Concentric ellipses (overlay) class. In general, we conclude that some image classes are better for training than others. Here the 5 Concentric ellipses (overlay) class is a good choice. This observation has a huge effect on future

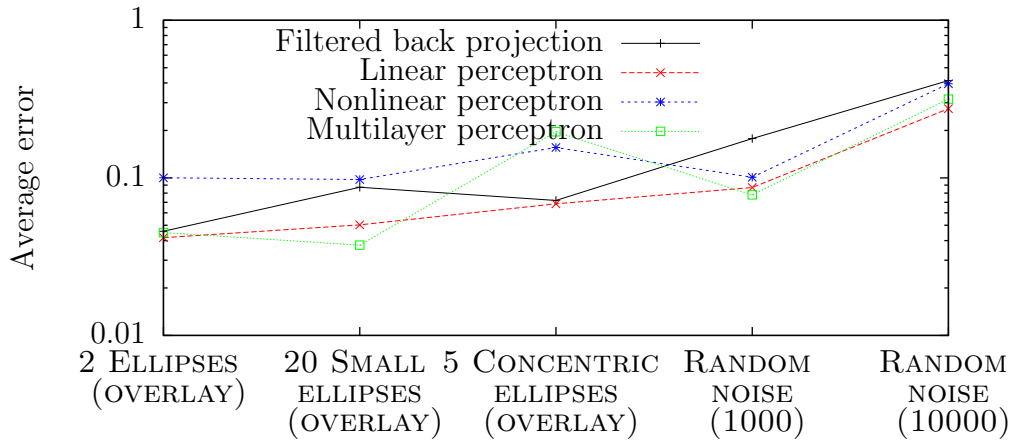

Figure 4.6: Average absolute errors for cross class validation with all image classes. Neural Networks trained on the Random noise (1000) image class.

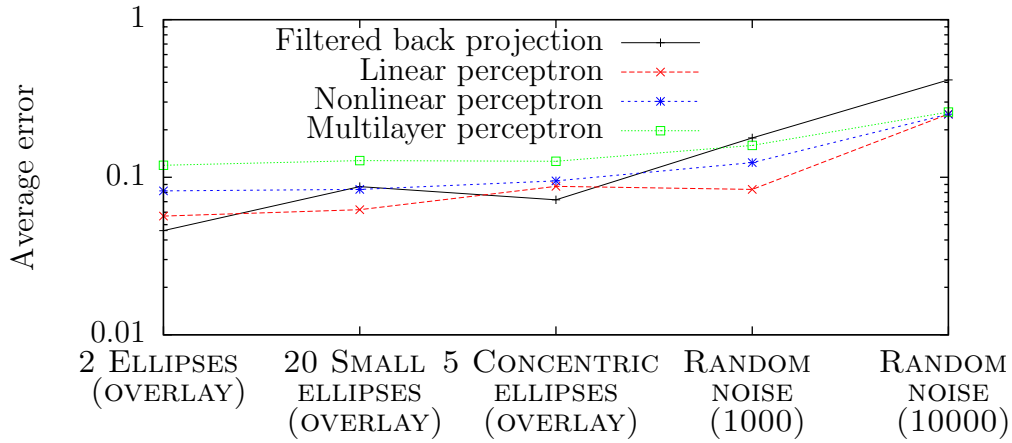

Figure 4.7: Average absolute errors for cross class validation with all image classes. Neural Networks trained on the Random noise (10000) image class.

implementations as it is essential to find a good training class.

For the random noise image classes the conclusion is slightly different, especially the Random noise (10000) image class. Here, both approaches show a more similar reconstruction quality. In general, it seems not a good idea to train Neural Networks on random noise, but when trained on a different image class, the networks are capable of reconstructing random noise considerably better than filtered back projection.

4.5 Real-life Case Study

The projection data for the real-life case study is not artificially created, but it is instead actual real-life output of a CT scanner. In this case the object is a rough diamond. The data set consists of 332 slices of the diamond, see Figure 4.8. The dimensions of the images are originally 1024×1024 pixels. We reduced this dimensionality to 384×384 pixels for two reasons; first, the diamond is rather small compared to the full image size, and secondly, it reduces the computation times significantly. Per slice 500 projections are included (equally dividing the 0 to π semicircle).

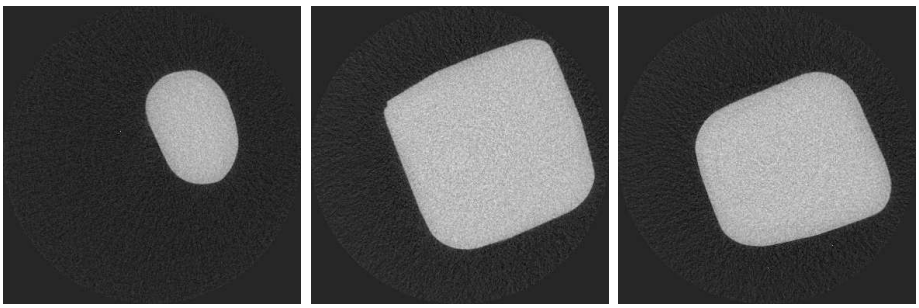

Figure 4.8: Example images from the real-life data set. Slices 50, 145, and 20 reconstructed using filtered back projection with 500 projections.

In contrast to the earlier experiments, we have no original image to train on. We used the filtered back projection reconstruction using all 500 projections as an approximation of the original image. Then we were interested in the reconstruction quality of a linear perceptron versus the traditional filtered back projection approach, of course using (much) fewer projections.

We apply a linear perceptron for several reasons. First, it is the most simple topology having the fewest number of weights which makes it easy and fast to train. The initialization is easier because of its linearity. The second motivation regards the practical implementation. The weights resulting from a trained linear perceptron are assumed to be easily embedded within existing implementations.

For this experiment we randomly select 1,000 slices, and from each selected slice we randomly select 10,000 pixels as training example resulting in a total of 10,000,000 training instances. The perceptron was trained using 50 projections equally dividing the total of 500 projections. In Figure 4.9 the resulting weight vector is shown. The symmetry can be clearly observed. Note that the resulting values are scaled down because of the preference of the input domain $[0, 1]$ of the linear perceptron. Furthermore, the features

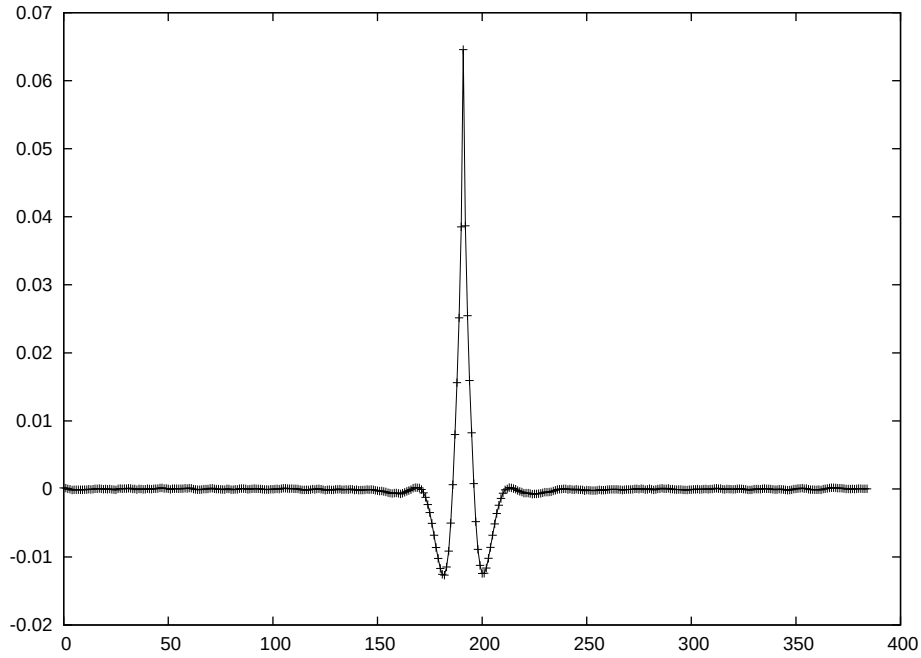

Figure 4.9: Weight vector of a linear perceptron after 10,000,000 training instances. The discrete weights are connected by linear interpolation for better readability.

are much less "sharp" as compared to for instance the Ram-Lak kernel in Figure 2.3. The resulting reconstructions, as are shown in Figure 4.10, are softer as well.

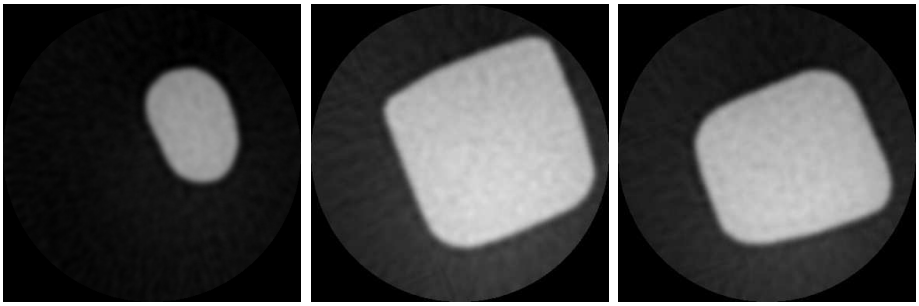

Figure 4.10: Reconstructed image by a linear perceptron. 50 projection angles where used (equally dividing the semicircle), and the perceptron was offered a total of 10,000,000 training instances.

As an alternative approach to training, we could train a perceptron on another image class, e.g., one of the artificial image classes from Section 4.1. Note that the dimensionality of the image should be equal to the dimensionality of the ultimate reconstructions. Therefore we should scale up the artifical images to 384×384 pixels. As is demonstrated in Section 4.4, the reconstruction quality depends on the selected image class for training. This choice is not trivial. Due to the appearance of the objects in the real-life data set (see Figure 4.8, we choose the 2 Ellipses (overlay) image class to train on.

Table 4.2: Real-life average absolute errors.

	Average error	Standard deviation
Filtered back projection ($k = 50$)	0.1198	0.1262
Linear perceptron	0.0548	0.0632
Linear perceptron (cross class)	0.1023	0.1103

In Table 4.2 the average absolute errors are presented. The linear perceptron trained on the real-life data set performs best as can be expected. We observe about the same difference in reconstruction quality with regard to the filtered back projection algorithm as is observed on our artifical image classes reconstructions. We might conclude that a linear perceptron can be applied in real-life. The cross class trained perceptron is still slightly better than filtered back projection.

Chapter 5

Conclusions

Here, we present a summary of the conclusions from the experiments in Chapter 4. As a general rule, neural networks are capable of being applied for reconstructing very good quality images, especially when the image resolution is low, and there are only a few projections. They lose their advantage over the traditional filtered back projection technique when there are many (over 100) projections available. In theory, a linear perceptron is able to simulate the filtered back projection strategy, i.e., we can choose the weight vector to be identical to the kernel used. Training a network does not guarantee convergence to this kernel. In fact, it is quite hard to train a Neural Network (using the aggregation approach) when many projections are used. The (aggregated) input vector shows little variation hampering its training abilities, and making the network very sensible to its initialization values.

The measurement for image quality used in Chapter 4 is the average absolute error. Clearly, Neural Networks are capable of reducing this measurement compared to the filtered back projection technique. From Figure 4.2, we can observe several differences of the approach of both techniques. While filtered back projection tends to recreate sharply defined boundaries around objects, the Neural Networks, especially the linear perceptron, make these edges softer. The reconstructions from filtered back projection suffer from many image artifacts, while they are almost absent in the reconstructions from the linear perceptron. It seems that the lower error values are mostly achieved by eliminating these artifacts, while losing some on the sharpness of the objects. This observation is supported by the, on average, higher error for the 20 Small ellipses (overlay) image class compared to other image classes with less ellipses, i.e., less boundaries.

The results from the real-life data set case study presented in Section 4.5 are encouraging. A linear perceptron is capable of generating high quality reconstructions of real-life data. It is very beneficial to train on the same

class as the objects that will be ultimately reconstructed, however, due to the limited size of our real-life data set no hard conclusions can be drawn. The resulting weight vector from the trained linear perceptron can be easily transferred to existing practical implementations, and therefore, instantly improve reconstruction qualities.

5.1 Future Research

Many areas of future research remain. As we explored the behavior of simple Neural Networks (i.e., perceptrons) other topologies could be investigated, especially regarding the elimination of the aggregation operator. We proposed in Section 3.5 an alternative topology. No significant improvement was observed on our data sets, however, and this should be investigated further.

As was concluded in Section 3.3, the initialization of the perceptrons is not trivial nor are the other parameters such as the learning rate. A comprehensive study could be performed on initialization and parameter setting. Furthermore, additional learning concepts could be introduced such as *weight decay*, or *boosting*. For multilayer perceptrons there is an additional choice for the activation function.

Appendix A

Overview Table

In Table A.1, a comprehensive overview of all average absolute errors and their corresponding standard deviations is given for all network topologies and the filtered back projection algorithm. Each strategy is performed on a decreasing number of projections $k = 32, 16, 8, 4, 2$.

Table A.1: Comprehensive overview of all average absolute errors.

Image class	Projections	Filtered back projection		Linear perceptron		Nonlinear perceptron		Multilayer perceptron	
		Average error	Standard deviation	Average error	Standard deviation	Average error	Standard deviation	Average error	Standard deviation
2 Ellipses (overlay)	32	0.0458	0.0688	0.0205	0.0402	0.0415	0.0476	0.0201	0.0416
	16	0.0751	0.0862	0.0345	0.0482	0.0409	0.0532	0.0208	0.0445
	8	0.1198	0.1262	0.0548	0.0632	0.0484	0.0596	0.0252	0.0516
	4	0.1852	0.2017	0.0861	0.0865	0.0559	0.0754	0.0453	0.0835
	2	0.2807	0.3135	0.1588	0.1401	0.0844	0.1397	0.0791	0.1269
20 Small ellipses (overlay)	32	0.0871	0.1144	0.0417	0.0633	0.0458	0.0624	0.0269	0.0536
	16	0.1401	0.1480	0.0673	0.0809	0.0529	0.0752	0.0339	0.0732
	8	0.2253	0.2194	0.1010	0.1112	0.0544	0.1097	0.0439	0.0982
	4	0.3403	0.3291	0.1211	0.1372	0.0815	0.1470	0.0669	0.1372
	2	0.5016	0.4820	0.1437	0.1572	0.1017	0.1726	0.0931	0.1628
5 Concentric ellipses (overlay)	32	0.0718	0.1016	0.0371	0.0543	0.0694	0.0779	0.0395	0.0572
	16	0.1138	0.1264	0.0538	0.0657	0.0852	0.0953	0.0420	0.0685
	8	0.1808	0.1850	0.0833	0.0864	0.1692	0.1560	0.0841	0.1185
	4	0.2785	0.2913	0.1383	0.1290	0.1961	0.1921	0.0833	0.1133
	2	0.3837	0.4451	0.5882	0.3700	0.5036	0.3791	0.2045	0.2137
Random noise (1000)	32	0.1775	0.2046	0.0867	0.1339	0.1009	0.1366	0.0779	0.1263
	16	0.2297	0.2391	0.0818	0.1430	0.0916	0.1426	0.0642	0.1425
	8	0.3048	0.3004	0.0855	0.1467	0.0911	0.1453	0.0862	0.1450
	4	0.4077	0.3929	0.0777	0.1527	0.0849	0.1500	0.0760	0.1529
	2	0.5572	0.5247	0.0728	0.1561	0.0810	0.1528	0.0763	0.1546
Random noise (10000)	32	0.4141	0.3727	0.2517	0.2186	0.2511	0.2284	0.2592	0.2173
	16	0.4966	0.4579	0.2785	0.2156	0.2824	0.2229	0.2783	0.2149
	8	0.6325	0.5911	0.2957	0.2200	0.3001	0.2277	0.2939	0.2176
	4	0.8386	0.7896	0.2937	0.2456	0.2998	0.2712	0.2932	0.2338
	2	1.1336	1.0586	0.3146	0.2295	0.3222	0.2397	0.3067	0.2218

Appendix B

Overview Reconstructions

In Figure B.1, Figure B.2, Figure B.3, Figure B.4, and Figure B.5, we present some charactistic reconstructions for each of the network topologies as well as the filtered back projection algorithm corresponding with the results from Table A.1.

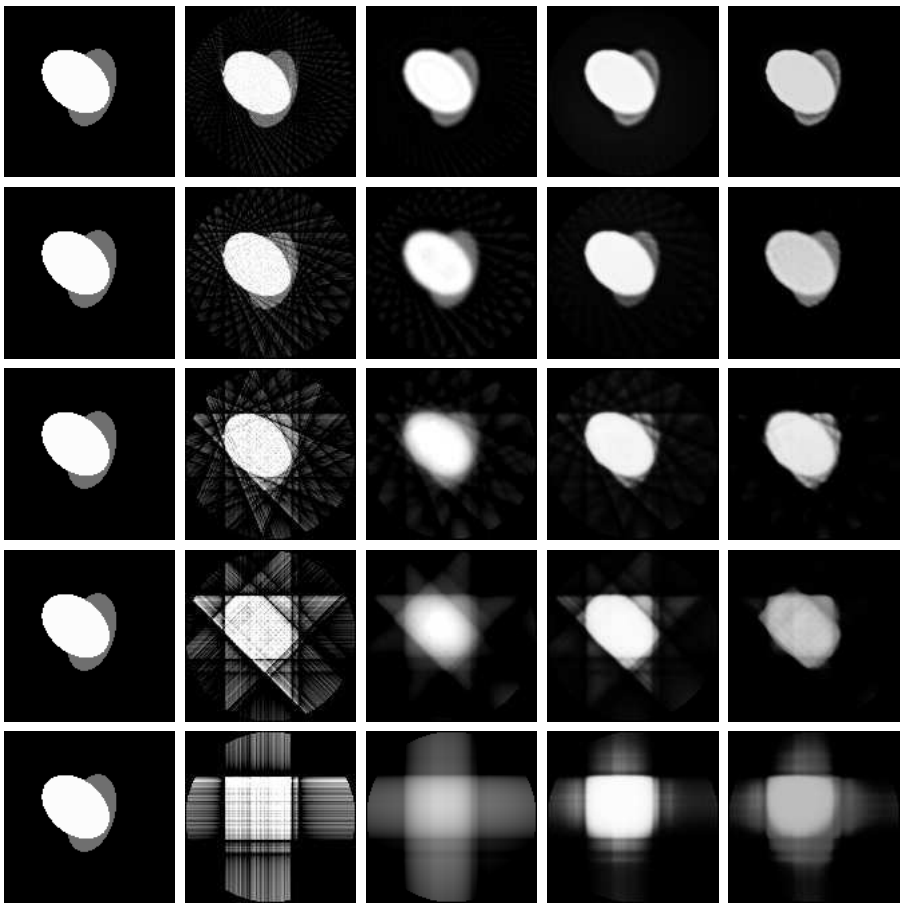

Figure B.1: Reconstructions of the 2 ELLIPSES (OVERLAY) image class. From left to right: the original image, filtered back projection, linear perceptron, nonlinear perceptron, multilayer perceptron for $k = 32, 16, 8, 4, 2$.

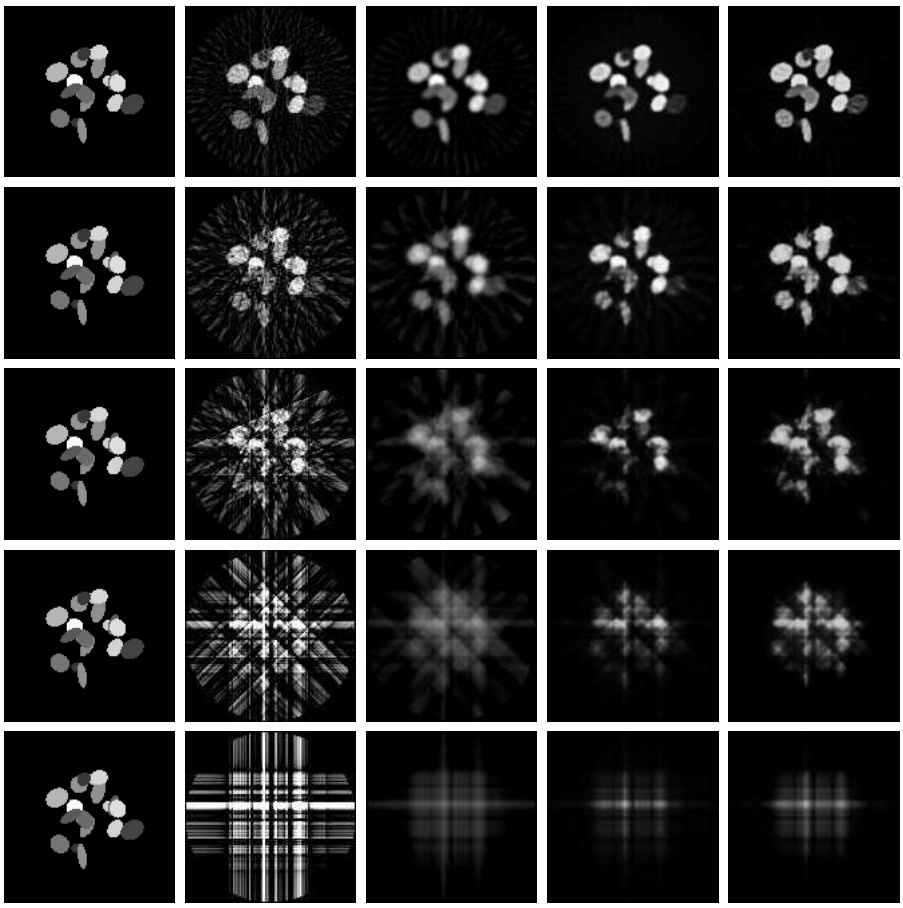

Figure B.2: Reconstructions of the 20 SMALL ELLIPSES (OVERLAY) image class. From left to right: the original image, filtered back projection, linear perceptron, nonlinear perceptron, multilayer perceptron for $k = 32, 16, 8, 4, 2$.

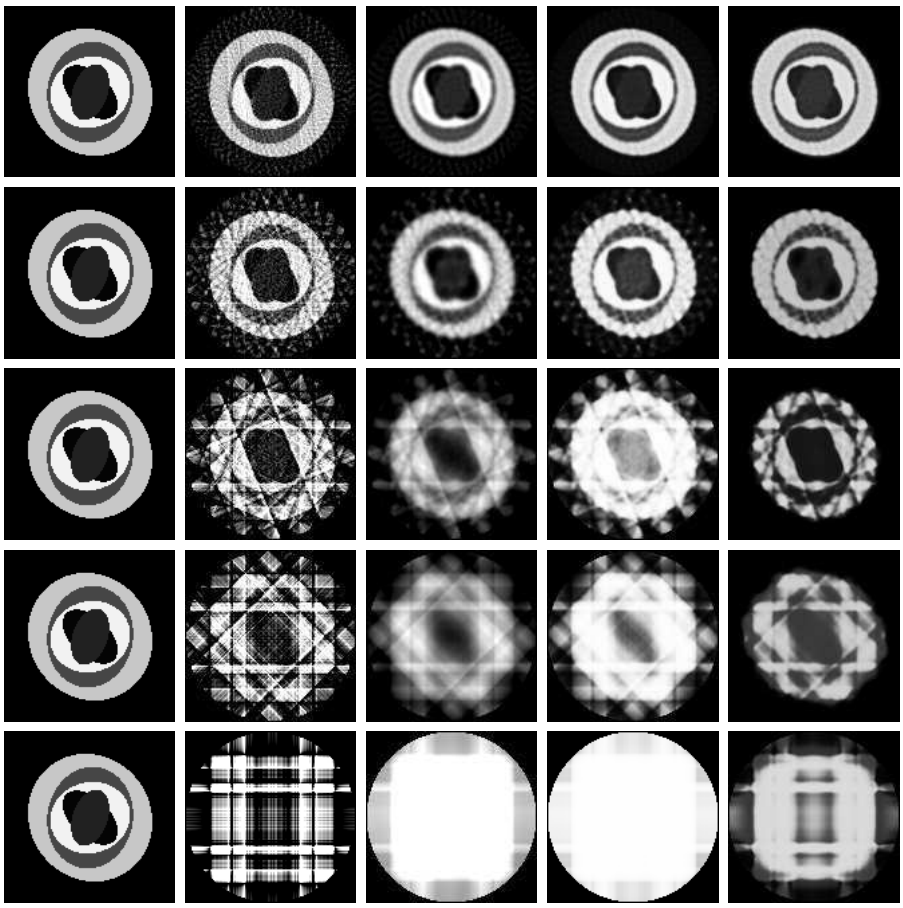

Figure B.3: Reconstructions of the 5 CONCENTRIC ELLIPSES (OVERLAY) image class. From left to right: the original image, filtered back projection, linear perceptron, nonlinear perceptron, multilayer perceptron for $k = 32, 16, 8, 4, 2$.

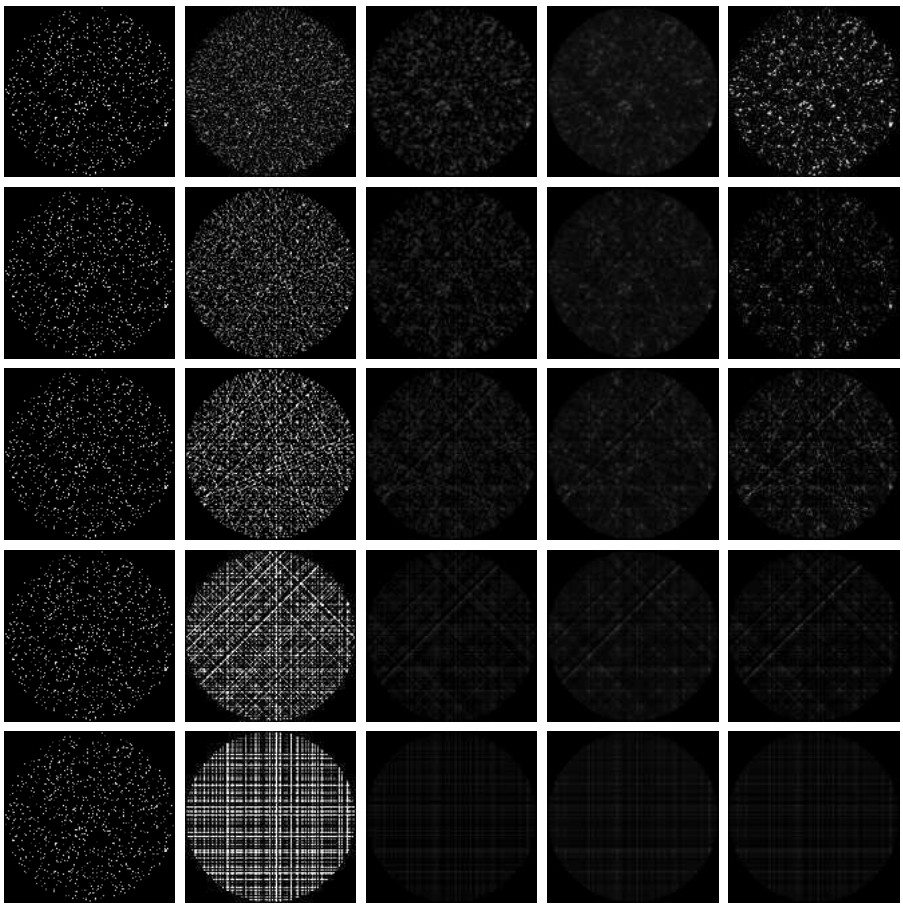

Figure B.4: Reconstructions of the RANDOM NOISE (1000) image class. From left to right: the original image, filtered back projection, linear perceptron, nonlinear perceptron, multilayer perceptron for $k = 32, 16, 8, 4, 2$.

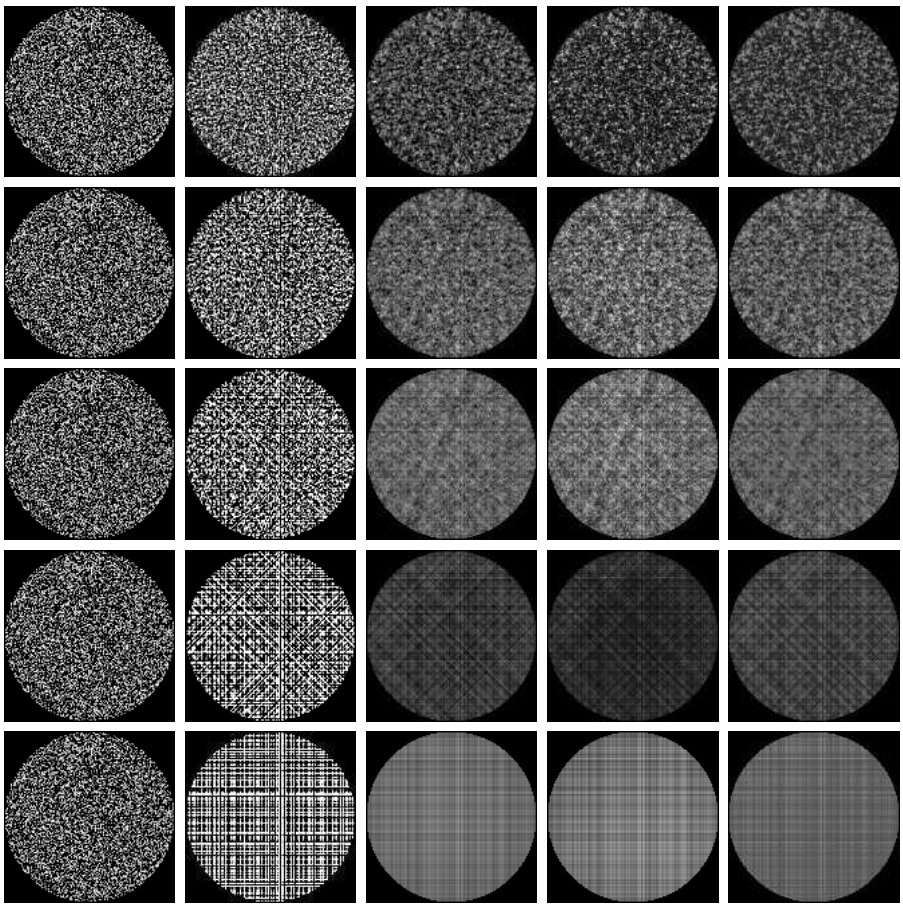

Figure B.5: Reconstructions of the RANDOM NOISE (10000) image class. From left to right: the original image, filtered back projection, linear perceptron, nonlinear perceptron, multilayer perceptron for $k = 32, 16, 8, 4, 2$.

Appendix C

Cross Class Validation Tables

In Table C.1, Table C.2, Table C.3, Table C.4, and Table C.5, the average absolute errors and their corresponding standard deviations are given for all network topologies, while they were trained on a different image class.

Table C.1: Average absolute errors for perceptrons trained on the 2 Ellipses (overlay) image class.

Image class	Projections	Linear perceptron		Nonlinear perceptron		Multilayer perceptron	
		Average error	Standard deviation	Average error	Standard deviation	Average error	Standard deviation
20 Small ellipses (overlay)	32	0.0396	0.0663	0.0464	0.0649	0.0264	0.0569
	16	0.0675	0.0814	0.0493	0.0751	0.0363	0.0756
	8	0.1023	0.1103	0.0696	0.1026	0.0624	0.1090
	4	0.1226	0.1366	0.0896	0.1402	0.0805	0.1445
	2	0.1374	0.1662	0.0850	0.1759	0.0830	0.1700
5 Concentric ellipses (overlay)	32	0.0404	0.0588	0.0627	0.0733	0.0466	0.0675
	16	0.0566	0.0692	0.0629	0.0802	0.0492	0.0811
	8	0.0872	0.0906	0.0725	0.0852	0.0569	0.0900
	4	0.1255	0.1194	0.0949	0.1203	0.1120	0.1611
	2	0.3417	0.2483	0.2161	0.2386	0.1453	0.1573
Random noise (1000)	32	0.0742	0.1479	0.0725	0.1535	0.0689	0.1472
	16	0.0761	0.1498	0.0717	0.1546	0.0703	0.1515
	8	0.0734	0.1537	0.0733	0.1547	0.0706	0.1547
	4	0.0789	0.1531	0.0720	0.1563	0.0624	0.1608
	2	0.1911	0.1977	0.0470	0.1696	0.0469	0.1697
Random noise (10000)	32	0.2854	0.2147	0.2789	0.2427	0.2608	0.2487
	16	0.2906	0.2174	0.2866	0.2678	0.2817	0.2908
	8	0.2968	0.2200	0.2886	0.2514	0.2888	0.2741
	4	0.2964	0.2320	0.2958	0.2703	0.3184	0.3696
	2	0.4601	0.3791	0.4345	0.3710	0.3118	0.2398

Table C.2: Average absolute errors for perceptrons trained on the 20 Small ellipses (overlay) image class.

Image class	Projections	Linear perceptron		Nonlinear perceptron		Multilayer perceptron	
		Average error	Standard deviation	Average error	Standard deviation	Average error	Standard deviation
2 Ellipses (overlay)	32	0.0267	0.0391	0.0655	0.0749	0.0305	0.0520
	16	0.0365	0.0483	0.0773	0.0885	0.0319	0.0615
	8	0.0542	0.0644	0.0593	0.0915	0.0259	0.0582
	2	0.2005	0.1646	0.1806	0.2126	0.1154	0.1470
	4	0.0881	0.0888	0.1220	0.1582	0.0447	0.0862
5 Concentric ellipses (overlay)	32	0.0623	0.0606	0.2110	0.1881	0.0829	0.1193
	16	0.0671	0.0723	0.2712	0.2325	0.0874	0.1279
	8	0.0887	0.0924	0.2125	0.2091	0.0616	0.1013
	4	0.1454	0.1408	0.4304	0.3620	0.1120	0.1590
	2	0.6123	0.3798	0.5166	0.4054	0.2900	0.2480
Random noise (1000)	32	0.0789	0.1433	0.0573	0.1613	0.0525	0.1527
	16	0.0768	0.1485	0.0550	0.1636	0.0513	0.1610
	8	0.0776	0.1512	0.0485	0.1683	0.0506	0.1660
	4	0.1029	0.1444	0.0465	0.1698	0.0461	0.1700
	2	0.2276	0.2090	0.0457	0.1703	0.0470	0.1697
Random noise (10000)	32	0.2860	0.2158	0.4713	0.3900	0.2814	0.2588
	16	0.2925	0.2177	0.5455	0.4344	0.2976	0.2650
	8	0.2907	0.2269	0.4831	0.4015	0.2987	0.3209
	4	0.2992	0.2700	0.6494	0.4877	0.3731	0.3272
	2	0.7903	0.5640	0.6588	0.4920	0.4303	0.3513

Table C.3: Average absolute error for perceptrons trained on the 5 Concentric ellipses (overlay) image class.

Image class	Projections	Linear perceptron		Nonlinear perceptron		Multilayer perceptron	
		Average error	Standard deviation	Average error	Standard deviation	Average error	Standard deviation
2 Ellipses (overlay)	32	0.0234	0.0382	0.0466	0.0785	0.0382	0.0518
	16	0.0392	0.0486	0.0466	0.0837	0.0250	0.0487
	8	0.0583	0.0634	0.0549	0.0770	0.0350	0.0600
	4	0.1181	0.1073	0.0706	0.1279	0.0685	0.1040
	2	0.1967	0.1629	0.1914	0.1856	0.1198	0.1541
20 Small ellipses (overlay)	32	0.0440	0.0626	0.0491	0.0990	0.0464	0.0679
	16	0.0749	0.0827	0.0525	0.1124	0.0378	0.0735
	8	0.1104	0.1123	0.0628	0.1115	0.0598	0.1033
	4	0.1472	0.1526	0.0815	0.1820	0.1025	0.1539
	2	0.1509	0.1546	0.1367	0.1589	0.1189	0.1739
Random noise (1000)	32	0.0770	0.1453	0.0553	0.1630	0.0920	0.1396
	16	0.0753	0.1503	0.0522	0.1657	0.0683	0.1550
	8	0.0798	0.1504	0.0542	0.1650	0.0718	0.1560
	4	0.1328	0.1817	0.0473	0.1693	0.0558	0.1651
	2	0.1893	0.1975	0.0580	0.1640	0.0608	0.1629
Random noise (10000)	32	0.2788	0.2147	0.2689	0.2286	0.2685	0.2251
	16	0.2864	0.2186	0.2834	0.2360	0.2791	0.2509
	8	0.2936	0.2204	0.4018	0.3380	0.3029	0.3098
	4	0.3085	0.2272	0.4290	0.3573	0.3019	0.2706
	2	0.7535	0.5456	0.6501	0.4883	0.4406	0.3687

Table C.4: Average absolute error for perceptrons trained on the Random noise (1000) image class.

Image class	Projections	Linear perceptron		Nonlinear perceptron		Multilayer perceptron	
		Average error	Standard deviation	Average error	Standard deviation	Average error	Standard deviation
2 Ellipses (overlay)	32	0.0417	0.0424	0.1000	0.1405	0.0449	0.0953
	16	0.0468	0.0527	0.0770	0.1183	0.0843	0.1723
	8	0.0722	0.0826	0.0755	0.1105	0.0667	0.1285
	4	0.1231	0.1313	0.0615	0.0895	0.0587	0.1125
	2	0.1367	0.1740	0.0965	0.1438	0.1051	0.1813
20 Small ellipses (overlay)	32	0.0503	0.0638	0.0975	0.1474	0.0374	0.0901
	16	0.0708	0.0823	0.0786	0.1293	0.0416	0.1098
	8	0.1037	0.1181	0.0918	0.1413	0.0565	0.1127
	4	0.1273	0.1584	0.0956	0.1550	0.0748	0.1465
	2	0.1276	0.1877	0.1091	0.1771	0.1021	0.1920
5 Concentric ellipses (overlay)	32	0.0684	0.0642	0.1557	0.1862	0.1959	0.2439
	16	0.0686	0.0727	0.1043	0.1413	0.2412	0.3194
	8	0.1481	0.1395	0.1032	0.1350	0.2318	0.3005
	4	0.2941	0.2389	0.1069	0.1475	0.2145	0.2734
	2	0.2861	0.2775	0.2077	0.2533	0.2278	0.2957
Random noise (10000)	32	0.2746	0.2091	0.3957	0.3315	0.3161	0.3749
	16	0.2800	0.2147	0.2905	0.2403	0.3216	0.3819
	8	0.2933	0.2815	0.2955	0.2486	0.3215	0.3816
	4	0.4032	0.3974	0.3056	0.3255	0.3216	0.3818
	2	0.3842	0.3945	0.3163	0.3586	0.3210	0.3799

Table C.5: Average absolute errors for perceptrons trained on the Random noise (10000) image class.

Image class	Projections	Linear perceptron		Nonlinear perceptron		Multilayer perceptron	
		Average error	Standard deviation	Average error	Standard deviation	Average error	Standard deviation
2 Ellipses (overlay)	32	0.0566	0.0758	0.0818	0.0536	0.1191	0.0812
	16	0.2386	0.1520	0.3001	0.1810	0.3319	0.2003
	8	0.3933	0.2456	0.4699	0.2880	0.4701	0.2906
	4	0.3822	0.2387	0.3868	0.2413	0.5023	0.3334
	2	0.3859	0.2458	0.4117	0.2665	0.3220	0.2055
20 Small ellipses (overlay)	32	0.0622	0.0796	0.0836	0.0620	0.1275	0.0862
	16	0.2617	0.1697	0.3236	0.2063	0.3635	0.2363
	8	0.4274	0.2702	0.5076	0.3203	0.5153	0.3324
	4	0.4245	0.2671	0.4364	0.2798	0.5184	0.3428
	2	0.4149	0.2553	0.4429	0.2800	0.3444	0.2099
5 Concentric ellipses (overlay)	32	0.0874	0.0896	0.0948	0.0763	0.1262	0.0986
	16	0.1662	0.1209	0.1950	0.1331	0.2054	0.1396
	8	0.2624	0.1841	0.2946	0.2019	0.2829	0.1923
	4	0.2636	0.2031	0.2578	0.2101	0.2846	0.2165
	2	0.3131	0.2128	0.3218	0.2243	0.3050	0.2030
Random noise (1000)	32	0.0836	0.1355	0.1237	0.1217	0.1591	0.1221
	16	0.3277	0.1917	0.3998	0.2377	0.5050	0.3023
	8	0.5087	0.2939	0.6094	0.3545	0.6789	0.3935
	4	0.5300	0.3050	0.5812	0.3364	0.7361	0.4256
	2	0.4559	0.2597	0.4931	0.2828	0.3280	0.1864

Bibliography

[1] K.J. Batenburg. A Network Flow Algorithm for Reconstructing Binary Images from Discrete X-rays. *Journal of Mathematical Imaging and Vision*, 27(2):175–191, 2007.

[2] K.J. Batenburg and W.A. Kosters. Neural Networks for Discrete Tomography. In *Proceedings of the Seventeenth Belgium-Netherlands Conference on Artificial Intelligence (BNAIC)*, pages 21–27, 2005.

[3] K.J. Batenburg and W.A. Kosters. A Neural Network Approach to Real-Time Discrete Tomography. In *Combinatorial Image Analysis*, volume 4040 of *Lecture Notes in Computer Science*, pages 389–403. Springer Berlin/Heidelberg, 2006.

[4] M. Buzug. *Computed Tomography: From Photon Statistics to Modern Cone-Beam CT*. Springer Berlin/Heidelberg, 2008.

[5] S. Haykin. *Neural Networks and Learning Machines*. Prentice Hall, third edition, 2008.

[6] S. Helgason. *The Radon Transform*. Birkhäuser Boston, 1980.

[7] G.T. Herman. *Fundamentals of Computerized Tomography: Image Reconstruction from Projections*. Springer London, second edition, 2010.

[8] G.T. Herman and A. Kuba. *Discrete Tomography: Foundations, Algorithms and Applications*. Birkhäuser Boston, 1999.

[9] G.T. Herman and A. Kuba. *Advances in Discrete Tomography and Its Applications*. Birkhäuser Boston, 2007.

[10] J. Kerr and E. Bartlett. Neural Network Reconstruction of Single-photon Emission Computed Tomography Images. *Journal of Digital Imaging*, 8(3):116–126, 1995.

[11] J. Lampinen, A. Vehtari, and K. Leinonen. Application of Bayesian Neural Network in Electrical Impedance Tomography. In *Proceedings of the 1999 International Joint Conference on Neural Networks*, pages 3942–3947, 1999.

[12] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, third edition, 2010.