A Neural Network Approach to Real-Time Discrete Tomography

K.J. Batenburg[1,2] and W.A. Kosters[1]

[1] Leiden University, Leiden, The Netherlands
kosters@liacs.nl
[2] CWI, Amsterdam, The Netherlands
K.J.Batenburg@cwi.nl

Abstract. *Tomography* deals with the reconstruction of the density distribution inside an unknown object from its projections in several directions. In *Discrete tomography* one focuses on the reconstruction of objects having a small, discrete set of density values. Using this prior knowledge in the reconstruction algorithm may vastly reduce the number of projections that is required to obtain high quality reconstructions.

Recently the first generation of *real-time* tomographic scanners has appeared, capable of acquiring several images per second. Discrete tomography is well suited for real-time operation, as only few projections are required, reducing scanning time. However, for efficient real-time operation an extremely fast reconstruction algorithm is also required.

In this paper we present a new reconstruction method, which is based on a feed-forward neural network. The network can compute reconstructions extremely fast, making it suitable for real-time tomography. Our experimental results demonstrate that the approach achieves good reconstruction quality.

1 Introduction

Tomography deals with the reconstruction of the density distribution inside an unknown object from its projections in several directions [8]. Figure 1 shows the basic principle. In this paper we look at *transmission tomography*, where the projections are obtained by sending a beam (e.g., X-rays, neutrons, etc.) through the object and measuring the attenuated beam that has passed through the object. Tomography is used extensively in medical imaging, industrial imaging and, more recently, in materials science and biology. Typically, a large number of projections is required (more than 100) to obtain good reconstruction quality.

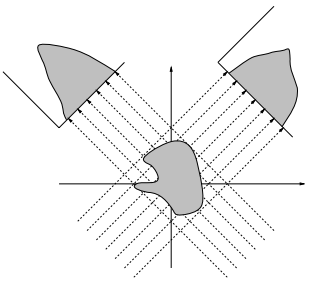

Fig. 1. Basic principle of tomography, 2 projections

When it is known in advance that the scanned object consists of only a few different materials, it may be possible to vastly reduce the number of required

U. Eckardt et al. (Eds.): IWCIA 2006, LNCS 4040, pp. 389–403, 2006.

projections by using this prior knowledge in the reconstruction algorithm. *Discrete tomography* focuses on the tomographic reconstruction of objects for which the set of pixel values in the reconstructed image is discrete and small. In particular, the reconstruction of binary images (i.e., black-and-white images) has received considerable attention [6].

Most tomographic scanners acquire *static images*, i.e., a single image, either 2D or 3D, of the object is reconstructed from the projection data. Therefore it is not possible to do imaging of *dynamic processes*, where the scanned object changes significantly in a short period of time. Recently, real-time medical tomographic scanners have emerged [9], which are capable of acquiring several images per second. Besides medical imaging, real-time tomography could prove very useful in industrial imaging.

Discrete tomography is well suited for real-time imaging, since the small number of required projections results in a substantial reduction of the scanning time. However, to compute a long series of reconstructions in reasonable time, a very fast reconstruction algorithm is required. Several authors have proposed algorithms for discrete tomography, usually for the reconstruction of binary images. All these algorithms require at least several dozens of seconds to reconstruct a single 2D 256×256 image [1, 2, 12, 13].

In this paper we present a new reconstruction method, which is based on a feed-forward neural network. The neural network is first *trained* on a set of representative images, which may take a substantial amount of time. After the training phase, the network can be used to compute a reconstruction very fast. When implemented on a *Field Programmable Gate Array* (FPGA), a piece of computer hardware, frame rates of several hundreds per second are realistic.

We focus on the reconstruction of binary images from parallel projections. Additional prior knowledge other than the binary constraint that is present in the training set, is learned by the neural network during the training phase, so it does not have to be modelled explicitly. Besides real-time tomography, our approach can also be used to compute a good start solution for more accurate, time-consuming reconstruction algorithms.

Neural network reconstruction methods for other types of tomographic reconstruction have been considered in the literature, e.g., [10, 11]. Neural networks are not well suited for general transmission tomography from many projections, as the number of variables in the reconstruction problem is extremely large. Besides that, it is very difficult to outperform other available approaches. In discrete tomography the amount of projection data is much smaller, making the reconstruction problem underdetermined. Neural networks are well known for their ability to learn additional prior knowledge, which makes them suitable for discrete tomography.

Section 2 contains a short description of the tomographic reconstruction problem. In Section 3 we first propose a basic neural network approach and discuss its abilities and limitations. Subsequently we refine the approach, obtaining a so-called single-pixel neural network architecture that is capable of computing real-time reconstructions of large images. In Section 4 we provide experimental

results of the neural network approach and a brief comparison with a continuous tomography algorithm. Section 5 concludes.

2 Reconstruction Problem

Figure 2 shows the main setting of the binary tomography problem. We assume that the object of interest is contained in the disc

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

with radius R. We call this disc the *imaging area*. For the sake of convenience we assume that R is a positive integer.

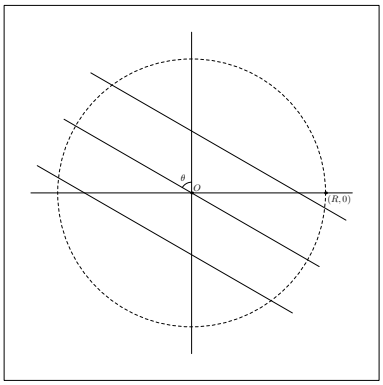

Fig. 2. Basic setting for the tomography problem in disc A; the angle between the parallel beam and the y-axis is denoted by θ

The *unknown binary image* that we would like to reconstruct is considered as a mapping $f : \mathbb{R}^2 \to \{0, 1\}$, where 0 is black and 1 is white. We assume that the support of f, i.e., the set $\{(x, y) \in \mathbb{R}^2 : f(x, y) = 1\}$, is a measurable set that is contained in A. Define the function $T_\theta : \mathbb{R}^2 \to \mathbb{R}$ as follows:

$$T_\theta(x, y) = x \cos\theta + y \sin\theta.$$

We call $T_\theta(x, y)$ the *point projection of* (x, y) *for angle* θ. Projections are measured along lines $L_{\theta,t}$ of the form

$$L_{\theta,t} = \{(x, y) \in \mathbb{R}^2 : T_\theta(x, y) = t\}.$$

The *Radon transform* P_f of f is defined (cf. [5], where also Radon's original paper is reproduced) as

$$P_f(\theta, t) = \int_{L_{\theta,t}} f(x, y)\,\mathrm{d}s \qquad \text{for } \theta \in [0, 2\pi),\ t \in \mathbb{R}.$$

We can now formulate the main reconstruction problem, where the parameter n determines the number of angles:

Problem 1. Let $D = \{\theta_1, \theta_2, \ldots, \theta_n\}$ be a given set of projection angles with $0 \leq \theta_1 < \theta_2 < \ldots < \theta_n < \pi$, and let $\phi_1, \phi_2, \ldots, \phi_n$ be given functions (the measured projections) from $\mathbb{R}$ to $\mathbb{R}$. Construct a function $f : \mathbb{R}^2 \to \{0, 1\}$ such that $P_f(\theta_i, \cdot) = \phi_i(\cdot)$ for $i = 1, 2, \ldots, n$.

When the measured projections are obtained through physical measurements, Problem 1 usually does not have an exact solution. Even if the reconstruction problem has an exact solution theoretically, we need to approximate its solution by representing it on a pixel grid.

In practice, the function $P_f(\theta, \cdot)$ is usually not measured in single points t. Instead, the total projection in a strip, covering a small t-interval (t_ℓ, t_r), is measured as

$$S_{\theta,f}(t_\ell, t_r) = \int_{t=t_\ell}^{t_r} P_f(\theta, t)\,\mathrm{d}t.$$

Typically, the value $S_{\theta,f}(t_\ell, t_r)$ is measured for consecutive strips of fixed width. Without loss of generalization we assume that all these strips have width 1. For any angle θ, $2R$ strip projections are measured. The first strip corresponds to the t-interval $(-R, -R+1)$, the last strip to $(R-1, R)$. In our neural network approach we also need to evaluate $S_{\theta,f}(t_\ell, t_r)$ for other values of (t_ℓ, t_r), often with integer width $t_r - t_\ell$. These values are computed by linear interpolation of the measured projection data.

Although the reconstruction problem is defined using the projection data, the performance of reconstruction algorithms is often evaluated by considering a *known* image f and its projections $P_{\theta_1,f}, P_{\theta_2,f}, \ldots, P_{\theta_k,f}$, and comparing the reconstruction to the original image f. In practice, resemblance to the original image is often more important than perfect correspondence to the projection data. This is particularly important if the projection data by itself is not enough to determine the image f and additional prior knowledge must be used in the reconstruction algorithm, which is the case if the number of projection angles n is relatively small: the problem is then underdetermined.

3 Neural Network Approach

In this section we will discuss two neural network approaches to the discrete tomography problem. A *feed-forward neural network* consists of neurons, grouped in layers, where neurons from one layer can have a weighted connection to neurons from the next layer. The weights are trained simultaneously, hopefully toward optimal values, by presenting the network with correct input-output pairs.

Both proposed networks are feed-forward back-propagation networks (see, e.g., [4, 7]) with one input layer, one hidden layer and one output layer. The networks are fully connected. The first and probably most obvious version (referred to as a *full-image network*) has one output node for each pixel. The second version (a

so-called *single-pixel network*) has only one output node, to reconstruct one pixel, but can be used — through appropriate adaptation — to reconstruct the whole image. This type of network has several advantages over the full-image network.

3.1 A Full-Image Network for Tomography

The input nodes of the network contain the values of the projections, the output nodes contain the pixel values. So the number of input nodes equals the number of projections, while the number of output nodes equals the size of the imaging area, i.e., approximately πR^2. The hidden nodes are connected to all input nodes and all output nodes. Output values are interpreted as gray values, yielding the gray level reconstruction. If necessary, these values can be rounded in a post-processing step to 0/1-values for a "crisp" or "rounded" reconstruction. These networks were first introduced and examined in [3].

Training is performed as follows. The input pattern, consisting of the projection values, is offered to the input nodes. Every connection has a real-valued weight, that is adapted during training. The hidden nodes receive the weighted sum of their incoming connections, and generate an output through the standard sigmoid $\sigma : x \mapsto 1/(1 + e^{-x})$. Output nodes operate in a similar way. In each epoch, a number (50,000, say) of random images (sampled from a certain distribution) with their projections are presented to the network; after each epoch the learning rate α is somewhat decreased. Note that samples are used only once, unless they are by chance regenerated.

The weights are adapted using the normal back-propagation rule. A weight w_{ji} from hidden node j to output node i is adapted through

$$w_{ji} \leftarrow w_{ji} + \alpha \cdot a_j \cdot \Delta_i, \quad \Delta_i = \sigma'(\mathrm{in}_i) \cdot (t_i - n_i).$$

Here a_j is the output of node j, $\mathrm{in}_i = \sum_j w_{ji} a_j$ is the weighted input to node i, $n_i = \sigma(\mathrm{in}_i)$ is its output (for output nodes this is the net output) and t_i is the desired target value, i.e., the true pixel value. The update rule for weight w_{kj} from input node k to hidden node j is a little more complicated (cf. [4, 7]):

$$w_{kj} \leftarrow w_{kj} + \alpha \cdot a_k \cdot \Delta_j, \quad \Delta_j = \sigma'(\mathrm{in}_j) \cdot \sum_i w_{ji} \Delta_i.$$

As usual, one extra input node and one extra hidden node clamped to -1 are added, the so-called *bias nodes*.

In [3] hidden nodes with only a small number of connections (so-called *local nodes*, as opposed to the more common *global nodes* mentioned above) are added. These local nodes are connected to a few input nodes and output nodes; they keep track of the constraints that affect a pixel and its immediate neighbours. Each local node corresponds with a unique pixel, receives input from the line projections that intersect with that pixel, and is connected to the 9 output nodes corresponding with the pixel and its immediate neighbours (6 or 4 near the boundaries). This general network architecture is depicted in Figure 3. Though this type of network was shown to perhaps have some advantages, in the sequel we will for comparison purposes just report on the version with only global nodes.

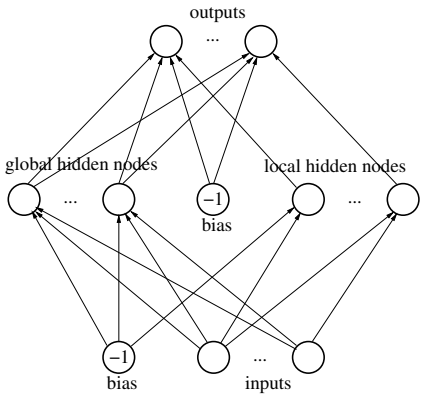

Fig. 3. General structure of the full-image network. Global hidden nodes are connected to all input nodes and all output nodes. Local hidden nodes are connected to only a few input and output nodes

3.2 A More Efficient Architecture: The Single-Pixel Network

Although the network from Section 3.1 is suitable for real-time reconstruction once the training phase is complete, it has some disadvantages:

- The network contains a large number of input/output nodes; a large number of hidden nodes is required to obtain reasonable reconstructions. Due to the large number of nodes and connections between them, training the network takes a very long time.
- Millions of training images and their projections are required to train the network. In practical applications it is usually impossible to obtain such large data sets.

In the sequel we propose an improvement by focussing on the reconstruction of a single pixel, instead of the whole image. This vastly reduces the number of hidden to output connections.

Reconstructing a Single Pixel

One of the principal goals of our neural network design is a reduction of the number of input nodes in comparison to the network from Section 3.1. When reconstructing a single pixel $p = (x_p, y_p)$ within the imaging area A, it is clear that projected lines that pass through p are more important for determining its value than the other projected lines. Also, if we assume that the image is locally smooth, projected lines that pass near p are more important than lines that pass far away from this pixel, as neighbouring pixels of p are highly relevant to the value of p.

We use this intuitive notion of relative importance between projected lines to preprocess the projection data. The inputs of the neural network from Section 3.1

correspond directly to the measured projection values. In the new single-pixel network, each input corresponds to a *strip projection*. Strips that are far away from p are much broader than strips near p.

Let k be a positive integer. Let $0 < d_0 \leq d_1 \leq \ldots \leq d_k$ be real values, the *strip sizes*. We require that the strip sizes satisfy

$$d_0/2 + \sum_{i=1}^{k} d_i \geq 2R. \tag{1}$$

Define the *strip boundaries* $s_{-k}, \ldots, s_0, \ldots, s_{k+1}$ as follows:

$$\begin{aligned} s_0 &= -d_0/2; \\ s_i &= s_{i-1} + d_{i-1} \quad (i = 1, \ldots, k+1); \\ s_i &= s_{i+1} - d_{-i} \quad (i = -k, \ldots, -1). \end{aligned}$$

Put $\tau_\theta = T_\theta(x_p, y_p)$, the point projection of p for angle θ. The set I_θ of *input strips* for angle θ can now be defined as

$$I_\theta = \bigcup_{i=-k}^{k} \{(\theta, \tau_\theta + s_i, \tau_\theta + s_{i+1})\}.$$

Each element of I_θ is a 3-tuple, consisting of the angle θ and the left and right boundary of a t-interval, which jointly define a strip through the imaging area. The constraint in Equation (1) ensures that the strips in I_θ cover at least the entire imaging area A, independent of the position of p. Given the angles $\theta_1, \theta_2, \ldots, \theta_n$, define the set I of all input strips as $I = \bigcup_{i=1}^{n} I_{\theta_i}$.

For every triple $(\theta, t_\ell, t_r) \in I$ there is an input node in the neural network, giving a grand total of $(2k+1) \cdot n$ input nodes. The input for such a node is $S_{\theta,f}(t_\ell, t_r)$, the strip projection for angle θ in the t-interval (t_ℓ, t_r).

Figure 4 shows three possible choices for the set $\{d_0, d_1, \ldots, d_k\}$ of strip sizes. Setting $d_0 = d_1 = \ldots = d_k$ yields equally spaced strips (Figure 4a). Setting $d_0 = 1$ and $d_i = i$ for $i \geq 1$ yields a set of strips for which the size increases linearly as the distance from the pixel p increases (Figure 4b). In Section 4 we will show that even if we set $d_0 = 1$, $d_i = 2^{i-1}$ for $i \geq 1$ (Figure 4c), the results are still satisfactory. Using strip sizes that grow exponentially with the distance to p yields a large reduction in the number of inputs of the neural network.

Training the single-pixel network proceeds as in the case of the full-image network. However, training a separate network for each pixel would take a huge amount of time. Also, the problem that the training requires a large number of training examples remains. In the sequel we will show how both problems can be solved.

Reconstructing All Pixels

Although the binning approach from the previous section drastically reduces the number of inputs of the network, a large number of images is still required to

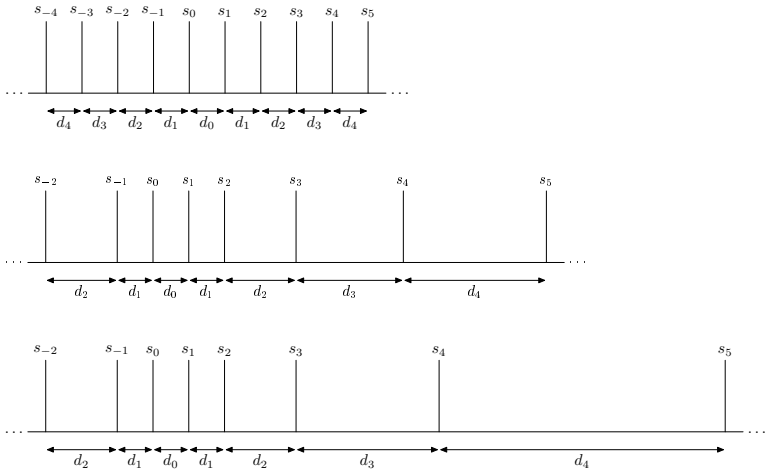

Fig. 4. Three different strip size configurations; a, top: constant, b, middle: linear, c, bottom: exponential

perform the training. If a separate network needs to be trained for each pixel in the image, the total training time may be even larger than for the basic network from Section 3.1.

Note that all inputs of the single-pixel network are *relative* to the projections of the center of the pixel. Therefore, there is no obvious reason why the network cannot be used to reconstruct a *different* pixel, elsewhere in the image. The only difference between two different pixels is that the relative position of the imaging area A is different. However, if we use a *varying* set of pixels from the training images, instead of using the same single pixel from each image, it may be possible to train the network *without* providing additional information on the relative position of the imaging area. This reconstruction task is harder, since less information is offered to the network. In Section 4 we show that one single-pixel network is capable of reconstructing arbitrarily positioned pixels. This offers some major advantages over the network from Section 3.1:

- If exponentially increasing strip sizes are used (see Figure 4c) both the number of inputs (logarithmic in R) and the number of outputs (constant, 1) is vastly reduced when compared to the network from Section 3.1, reducing training time for the single-pixel network.
- Only a single network has to be trained, instead of a new network for each pixel.
- Each training image, with its projections, now yields a new training example for each pixel in the image. The network from Section 3.1 requires a new image for each training example.

And finally, as we shall see in Section 4, the single-pixel networks seem more capable of reconstructing images from their projections, rather than learning images from certain classes.

3.3 Computational Complexity

For a feed-forward neural network having N_I input nodes, N_H hidden nodes and N_O output nodes, the time required to propagate an input pattern to the output nodes is $O(N_I N_H + N_H N_O)$. The time complexity for training a single input-output pattern is of the same order.

For the full-image network from Section 3.1 we have $N_I = O(nR)$ and $N_O = O(R^2)$, so that propagating a pattern takes $O(N_H(nR + R^2))$ time. (Remember that n is the number of projection angles or directions, and R is the radius of the imaging area.) Typically R is much larger than n. Once the training phase is complete, the time complexity of computing a reconstructed image from a set of projection data is of this same order.

For the single-pixel network with logarithmic strip sizes from Section 3.2 we have $N_I = O(n \log R)$ and $N_O = 1$, yielding a time complexity of $O(nN_H \log R)$ for propagating a pattern. To use the network for reconstruction after the training phase, a new input projection pattern has to be propagated through the network for every pixel in the image, yielding a time complexity of $O(nN_H R^2 \log R)$. Also, for this network the projection data must be preprocessed first to obtain the input data for the network, with time complexity $O(R)$. Note that the number N_H can vary heavily between different types of networks.

Since the value that is computed at each (non-input) node of a feed-forward network only depends on the values in the previous layer, the values for all nodes in a layer can be computed in parallel. Moreover, the computation that needs to be performed at each node is very simple. This allows for very efficient parallel implementations.

4 Experimental Results

In this section we present experimental results for the full-image network from Section 3.1 and the single-pixel network from Section 3.2. We restrict ourselves to two classes of synthetic images.

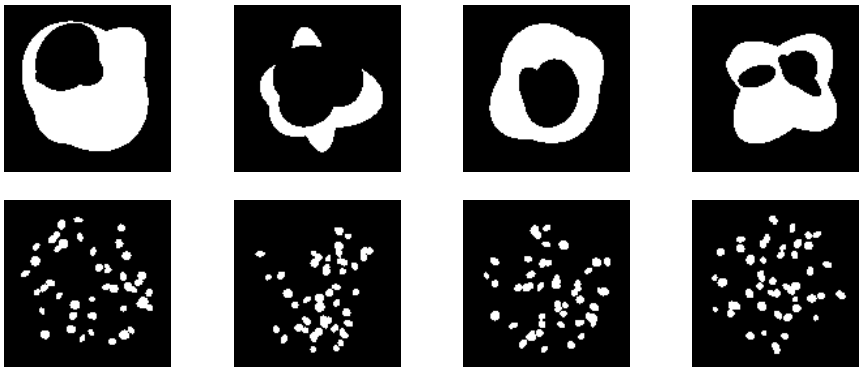

Fig. 5. Top: 128×128 images from the 7-class; bottom: images from the 50-class

The first class of examples consists of images with four large white ellipses and three smaller black ones inside, in a dark background. This class is referred to as the *7-class*, or just "7". The second class consists of images with fifty small white ellipses in a dark background, and is referred to as the *50-class*, or just "50". Figure 5 contains some examples.

Table 1. Results from experiments for the full-image network; 32×32 and 64×64 images; training runtime in hours

image width	angles	image class	hidden nodes	gray error	0–1 error	run-time
				average over 3 runs		
32	4	7	50	0.074	0.052	2.15
32	4	7	100	0.058	0.042	5.05
32	4	7	200	0.046	0.044	9.85
32	4	50	50	0.069	0.044	2.15
32	4	50	100	0.056	0.037	5.05
32	4	50	200	0.054	0.036	10.00
32	10	7	50	0.104	0.076	3.30
32	10	7	100	0.066	0.050	6.30
32	10	7	200	0.040	0.040	12.10
32	10	50	50	0.065	0.042	3.40
32	10	50	100	0.044	0.030	6.35
32	10	50	200	0.019	0.017	12.15
64	4	7	50	0.148	0.109	12.80
64	4	7	100	0.118	0.092	27.30
64	4	7	200	0.063	0.062	58.30
64	4	50	50	0.145	0.098	12.75
64	4	50	100	0.131	0.091	27.20
64	4	50	200	0.126	0.087	58.00
64	10	7	50	0.199	0.151	14.80
64	10	7	100	0.145	0.112	30.30
64	10	7	200	0.078	0.077	63.15
64	10	50	50	0.142	0.097	14.80
64	10	50	100	0.118	0.086	30.25
64	10	50	200	0.077	0.077	63.35

All experiments were repeated three times, and averages were taken over these three runs. Because all images are in fact located within a circle (the imaging area), we do not consider errors outside this area; therefore, mean error values are computed with respect to pixels within the imaging area. Average absolute errors are reported on independent test sets consisting of 1,000 images, both for gray level reconstruction and for rounded reconstruction (the 0–1 error). Table 1 shows results for the full-image network for two image sizes: 32×32 and 64×64. Table 2 gives results for the single-pixel network for three image sizes: 32×32, 64×64 and 128×128. The parameters are as follows: three sizes of the hidden layer: 50, 100 and 200 hidden nodes; and two different sets of projection angles,

Table 2. Results from experiments for the single-pixel network; 32×32, 64×64 and 128×128 images; training runtime in hours; (*) contains run with 0–1 error equal to 0

image width	angles	image class	hidden nodes	gray error	0–1 error	run-time
				average over 3 runs		
32	4	7	50	0.038	0.026	3.05
32	4	7	100	0.038	0.025	4.95
32	4	7	200	0.037	0.025	9.65
32	4	50	50	0.054	0.034	3.10
32	4	50	100	0.054	0.034	4.95
32	4	50	200	0.054	0.034	9.55
32	10	7	50	0.004	0.003	6.95
32	10	7	100	0.004	0.002(*)	12.20
32	10	7	200	0.004	0.003	21.65
32	10	50	50	0.016	0.011	6.95
32	10	50	100	0.015	0.011	12.10
32	10	50	200	0.015	0.011	21.50
64	4	7	50	0.046	0.029	3.70
64	4	7	100	0.044	0.032	5.80
64	4	7	200	0.044	0.029	11.10
64	4	50	50	0.116	0.083	3.70
64	4	50	100	0.114	0.083	5.80
64	4	50	200	0.118	0.082	11.15
64	10	7	50	0.006	0.005	8.95
64	10	7	100	0.006	0.005	14.75
64	10	7	200	0.006	0.005	25.20
64	10	50	50	0.051	0.034	9.00
64	10	50	100	0.052	0.033	14.65
64	10	50	200	0.051	0.033	25.20
128	4	7	50	0.042	0.028	4.75
128	4	7	100	0.041	0.029	7.15
128	4	7	200	0.039	0.028	13.35
128	4	50	50	0.130	0.094	4.70
128	4	50	100	0.129	0.092	7.10
128	4	50	200	0.129	0.100	13.40
128	10	7	50	0.005	0.003	12.20
128	10	7	100	0.005	0.002	18.30
128	10	7	200	0.005	0.003	30.50
128	10	50	50	0.057	0.038	12.20
128	10	50	100	0.056	0.039	18.15
128	10	50	200	0.055	0.040	30.45

consisting of 4 and 10 projections, equally spaced in the interval $[0°, 180°)$. In all experiments 200 epochs, each consisting of 50,000 examples (full-pixel network) or 2,300,000 examples (single-pixel network), were used for training. The number of examples per epoch was chosen so that the training of the two networks took the same amount of time for the case of 32×32 images from the 7-class using 4 projections and 100 hidden nodes. The reason for using more examples to train

the single-pixel network is that each example only contains information on a single pixel in that case. When using a real measured dataset, a new training example can be constructed from each pixel in the dataset. Experiments were run on a single processor AMD Athlon 2.2 GHz PC. Runtimes in the tables refer to the *training times* of the networks; once trained, reconstruction of a new image is almost instantaneously. The learning rate α started at 0.5, and was multiplied by 0.99 after each epoch.

Experiments with the full-image network show that the network is capable of reconstructing small images with acceptable quality, e.g., 32×32 images. This requires a reasonably large number of hidden nodes, e.g., 200, giving a huge number of connections. For larger images however, quality drops down, while computing time increases heavily (therefore, no experiments on 128×128 images were performed). The results suggest that a further increase in the number of hidden nodes might improve reconstruction quality. Figure 6 shows some examples from a run of the full-image network for 10 projections on the 7-class for 64×64 images, using 100 hidden nodes. The average absolute pixel error for this particular run (using 1,000,000 training examples) was 0.133 (gray level reconstruction) and 0.102 (rounded reconstruction).

Fig. 6. From left to right: 64×64 original, gray level reconstruction and rounded reconstruction using a full-image network, with absolute total errors 413.7 and 291, respectively

As another example, we show results for 64×64 images within the 50-class, using 10 projections, and 100 hidden nodes, see Figure 7. The figure shows best and worst reconstruction from a random set of 25 images from the 50-class.

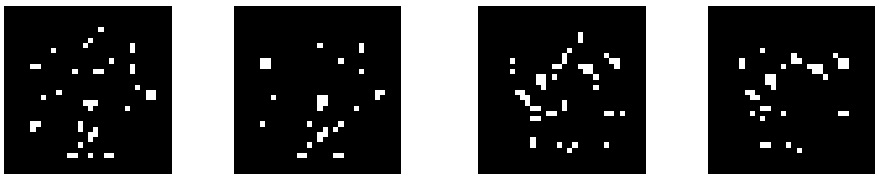

Fig. 7. From left to right: two pairs of 64×64 original and rounded reconstruction using a full-image network, with absolute total errors 235 and 310, respectively

Clearly, the results for the 50-class are not satisfactory. As we can see in Table 2, single-pixel networks give much better reconstructions. We now consider the single-pixel network exclusively.

For all test cases in Table 2 the number of hidden nodes has hardly any effect on the quality of the reconstructions. This suggests that a relatively small number of hidden nodes suffices. Structural Risk Minimization might be used to find a suitable size for the hidden layer. The 50-class is clearly more difficult to learn than the 7-class.

Figure 8 compares the single-pixel network and filtered backprojection (FBP), which is the fastest algorithm available for continuous tomography and the only one that can be used for real-time reconstruction. The FBP reconstructions were computed with the MATLAB Imaging Toolbox, using the Ram-Lak filter multiplied by a Hann window. The single-pixel reconstructions are clearly better. Other discrete tomography algorithms might be capable of producing more accurate reconstructions, perhaps even using fewer projections. However, to our knowledge these algorithms are far too slow for real-time reconstruction.

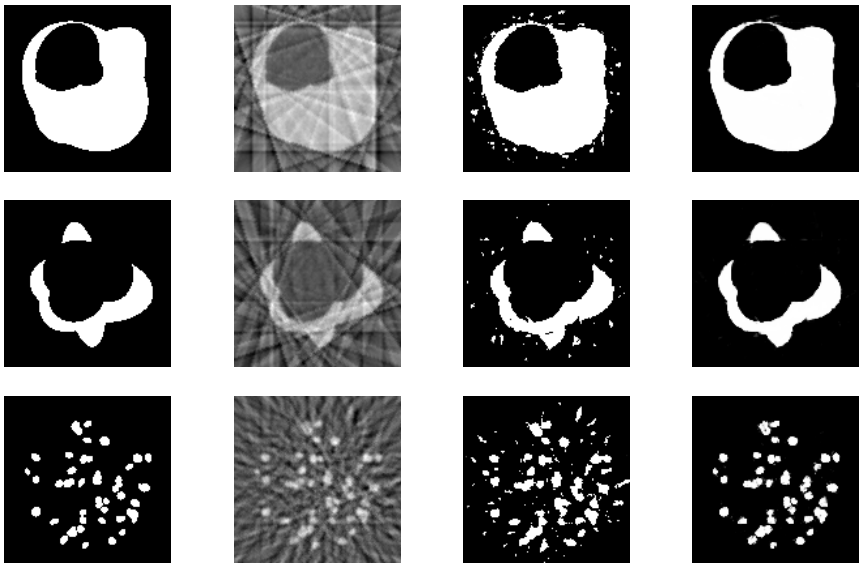

Fig. 8. From top to bottom: three 128×128 images and their reconstructions using filtered backprojection, rounded filtered backprojection and the single-pixel network, respectively; top and middle image from the 7-class, with 10 projections; bottom image from the 50-class, with 18 projections. Absolute 0–1 errors for the single-pixel network reconstructions are 101, 84 and 153, respectively

A natural question to ask is whether the network is capable of reconstructing images outside the class it was trained on. Figure 9 shows the reconstruction results of two such images, which are clearly not in any of the training classes. Though not perfect, the results are surprisingly good.

Though results from [3] suggest that local nodes might give some improvement for the full-pixel network (putting more complexity into the network), this was only shown for relatively small images. In the current paper we have chosen

Fig. 9. From left to right: two pairs of a 128×128 image and its reconstruction; original images are *not* from the 7-class for which the single-pixel network was trained. Absolute 0–1 errors are 189 and 291, respectively

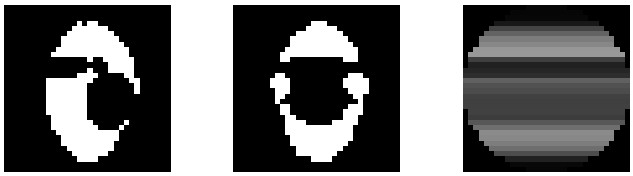

Fig. 10. From left to right: 32×32 original and reconstruction using a full-image network and a single-pixel network, respectively; only one projection

for global nodes only, to allow for better comparison between full-image and single-pixel networks.

Another issue with the full-image networks is that they are more inclined to learn location dependent information, in particular when there are very few projections. In that extraordinary case they behave quite differently from the single-pixel networks. As an example, in Figure 10 we show some results for the reconstruction of a 32×32 image using only projection in one horizontal direction. The single-pixel reconstruction shows the density distribution of the 0–1 intensity in the projection direction, as any two pixels on the same horizontal line cannot be distinguished by the network. The imaging area is clearly visible.

As mentioned before, the final 0–1 image is generated from the gray level reconstruction by simply rounding, giving a crisp figure, cf. Figure 6. Experiments suggest that errors often occur for pixels that have a raw reconstruction value near 0.5. It is possible to slightly improve the final reconstructed image by (for those pixels, in parallel) trying both 0 and 1 as reconstruction value, meanwhile comparing with the projections. Time restrictions clearly allow just a few pixels to be toggled simultaneously. The quality of the reconstruction may also be improved by introducing a stochastic model of the image class and computing an image (in a postprocessing step) that corresponds well with both the output of the neural netwerk and the model of the image class.

5 Conclusions

We conclude that the single-pixel network from Section 3.2 is capable of generating very good quality reconstructions of images from both classes, given

sufficiently many projection directions. Once trained, such a network can compute new reconstructions almost instantaneously, making it very suitable for real-time reconstruction. The full-image networks from Section 3.1 perform less satisfactory, but behave well if only very few projection directions are available and the images are small.

Although we use the networks for the reconstruction of *binary* images, there is no apparent reason why they could not be used for the reconstruction of images that contain a larger set of gray values, or even a continuous range. We intend to explore the possibility of using our neural network approach for continuous tomography in future research. For continuous tomography the size of the network will increase significantly, as the number of projections that is required to obtain a good reconstruction is typically much larger than for discrete tomography. This will increase the training time and the reconstruction time after training. For continuous tomography algorithms are already available that are both accurate and fast. For discrete tomography, however, the current approach seems to provide competitive algorithms, in particular for real-time reconstruction.

References

1. Batenburg, K.J.: Reconstructing Binary Images from Discrete X-rays, CWI Research Report PNA-E0418 (2004)
2. Batenburg, K.J.: An Evolutionary Algorithm for Discrete Tomography. Discrete Applied Mathematics **151** (2005) 36–54
3. Batenburg, K.J., Kosters, W.A.: Neural Networks for Discrete Tomography. Proceedings of the Seventeenth Belgium-Netherlands Conference on Artificial Intelligence (BNAIC), 21–27 (2005)
4. Bishop, C.M.: Neural Networks for Pattern Recognition. Oxford University Press (1995)
5. Helgason, S.: The Radon Transform. Birkhäuser, Boston (1980)
6. Herman, G.T., Kuba, A. (eds.): Discrete Tomography: Foundations, Algorithms and Applications. Birkhäuser, Boston (1999)
7. Hertz, J., Krogh, A., Palmer, R.G.: Introduction to the Theory of Neural Computation. Addison-Wesley (1991)
8. Kak, A.C., Slaney, M.: Principles of Computerized Tomographic Imaging. SIAM (2001)
9. Keat, N.: Real-time CT and CT Fluoroscopy. The British Journal of Radiology **74** (2001) 1088–1090
10. Kerr, J.P., Bartlett, E.B.: Neural Network Reconstruction of Single-photon Emission Computed Tomography Images. Journal of Digital Imaging **8** (1995) 116–126
11. Lampinen, J., Vehtari, A., Leinonen, K.: Application of Bayesian Neural Network in Electrical Impedance Tomography. Proceedings of the 1999 International Joint Conference on Neural Networks (1999)
12. Liao, H.Y., Herman, G.T.: A Coordinate Ascent Approach to Tomographic Reconstruction of Label Images from a Few Projections. Discrete Applied Mathematics **151** (2005) 184–197
13. Schüle, T., Schnörr, C., Weber, S., Hornegger, J.: Discrete Tomography by Convex-concave Regularization and D.C. Programming. Discrete Applied Mathematics **151** (2005) 229–243