Extracting Geometrical Features of Discrete Images from Their Projections Using Neural Networks*

Tamás Sámuel Tasi

Department of Image Processing and Computer Graphics
University of Szeged
Árpád tér 2., H-6720, Szeged, Hungary
ttasi@inf.u-szeged.hu

Abstract. Machine learning can be used in *Discrete Tomography* as a preprocessing step in order to choose the proper reconstruction algorithm or – with the aid of the knowledge acquired – to improve its accuracy. In this paper we investigate for a well-known machine learning method, namely the artificial neural network, how it's able to obtain certain geometrical features from discrete images and classify those images accordingly. As an example, we present results when the task is to distinguish between images of different classes, to determine the number of different intensity levels present in the discrete image, and to estimate the perimeter of simple and more complex discrete sets from their horizontal and vertical projections. The information extracted this way can be useful to simplify the problem of reconstructing the discrete set from its projections, which task is in focus of Discrete Tomography.
Keywords Discrete Tomography, Machine Learning, Artificial Neural Network, Geometrical Features, Gray-Level Estimation, Perimeter Estimation.

1 Introduction

The aim of *Computerized Tomography* (CT) is to reveal information about the inner structure of given objects without damaging or destroying them. Methods of CT (like filtered backprojection, algebraic reconstruction techniques) usually require several hundreds of projections to obtain an accurate enough reconstruction of the studied object [19, 23]. Since the projections are usually produced by X-ray, gamma-ray, or neutron imaging, the acquisition of them can be quite expensive, time-consuming or even might (partially or fully) damage the examined object. Therefore, in many applications it is impossible to apply reconstruction methods of CT effectively. In those cases there is still a hope to get a satisfactory reconstruction results by using *Discrete Tomography* (DT) [20, 21].

In DT we assume that the object to be reconstructed is composed of a few materials known beforehand. By utilizing this extra information it is often possible to get accurate reconstructions even from a small number of projections

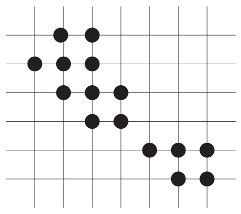
* The present paper summarizes the contents of previously published results [16,17,25].

(say, 2-10). Applications of DT therefore is crucial in the field of industrial non-destructive testing [10] and electronmicroscopy [6]. Very recently, with a technique of DT, the authors of [27] were able to investigate the 3D structure of crystalline nanoparticles on the atomic scale.

Unfortunately, the reconstruction task in DT is usually extremely undetermined, i.e., there can be many different solutions with the same projections. A common way to reduce the number of possible solutions is to assume that the image to be reconstructed has some additional (geometrical or more complex structural) features. There are several reconstruction algorithms in DT working in different classes of discrete images defined by certain geometrical or topological properties. For example various kinds of convexity and connectedness are examined in [2–4, 11–13]. In most cases these geometrical properties of discrete images are expected to be given explicitly, but in reality, this information might not be available prior to reconstruction. However, few efforts have been made to study how these features can be extracted before the reconstruction solely from the projections, if they are not explicitly given [16, 25, 17]. In this paper, a summary of the latter results is given, to prove that it is possible to retrieve different structural properties from certain classes of discrete images by using the projection data only.

Artificial Intelligence (AI) has an extremely broad range of tools for data mining. Surprisingly, up to now, in discrete tomography only a few of them have been used, and only for certain purposes. Methods of AI were mostly used in the reconstruction process itself, like in [7], where *Neural Networks* and in [5, 26], where *Genetic Algorithms* were successfully applied. However, reconstruction tasks performed this way usually require extensive computational time, hence the benefits of using these complex tools are not displayed in these cases. We investigate for the former neural networks how they perform in classifying images that differ in certain geometrical features, like connectedness, convexity, number of gray-intensity levels, and the perimeter. Recently, some authors have considered the problem of reconstructing discrete sets with minimal or predefined perimeter, and related problems [9, 14, 15, 22]. However, all those reconstruction methods assume that the perimeter is given a priori. This paper also shows how prior information of the perimeter can be extracted by using neural networks, if it is not given beforehand.

The structure of the paper is the following. First, the necessary definitions and notations are introduced in Section 2. Section 3 gives an overview of machine learning, the theory behind neural networks and the implementation strategies that were chosen. In Section 4 we present neural networks that are able to separate 8-, but not 4-connected *hv*-convex discrete sets from *hv*-convex polyominoes. In Section 5 we investigate an important problem of discrete tomography, namely, the identification of the number of gray-intensity values that can be present in the image. In Section 6 we present results of estimating the perimeter from the horizontal and vertical projections, first for simpler, then for more complex discrete sets. Finally, Section 7 is for the conclusion.



$$\begin{bmatrix} 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 \end{bmatrix}$$

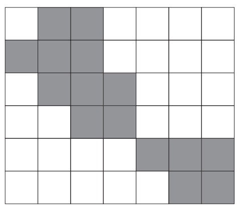


Fig. 1. An *hv*-convex 8- but not 4-connected discrete set represented by its elements (left) and by a binary matrix (center), and the corresponding binary image (right).

2 Preliminaries

The finite subsets of the two-dimensional integer lattice are called *discrete sets.* A discrete set is defined up to a translation and it can be represented by a binary matrix or a binary image as well (see Fig. 1). The *horizontal* and the *vertical* projections of a discrete set F are defined by the vectors $\mathcal{H}(F) = (h_1, \ldots, h_m)$, $\mathcal{V}(F) = (v_1, \ldots, v_n)$, respectively, where $h_i = \sum_{j=1}^{n} f_{ij}$ $(i = 1, \ldots, m)$, and $v_j = \sum_{i=1}^{m} f_{ij}$ $(j = 1, \ldots, n)$. For example, for the discrete set F in Fig. 1 $\mathcal{H}(F) = (2, 3, 3, 2, 3, 2)$, $\mathcal{V}(F) = (1, 3, 4, 2, 1, 2, 2)$.

Two positions $P = (p_1, p_2)$ and $Q = (q_1, q_2)$ in a discrete set are said to be *4-adjacent* (*8-adjacent*) if $|p_1 - q_1| + |p_2 - q_2| = 1$ ($|p_1 - q_1| + |p_2 - q_2| \leq 2$), respectively. The discrete set F is called *4-connected* (*8-connected*) if for any two positions $P, Q \in F$ there exists a sequence of distinct positions $P_0 = P, \ldots, P_k = Q$ in the discrete set F such that P_l is 4-adjacent (8-adjacent) to P_{l-1}, respectively, for each $l = 1, \ldots, k$. The 4-connected discrete sets are also called *polyominoes.* From the above definitions it follows that every 4-connected discrete set is 8-connected as well, but the counterpart is not always true. The discrete set F is *hv-convex* if all the rows and columns of F are 4-connected. Figure 1 shows an *hv*-convex 8-connected but not 4-connected discrete set.

The perimeter of a discrete set F of size $m \times n$ is defined by

$$\sum_{(i,j) \in \{0,\ldots,m\} \times \{0,\ldots,n\}} |f_{ij} - f_{i,j+1}| + |f_{ij} - f_{i+1,j}| \ , \qquad (1)$$

with the convention that $f_{ij} = 0$ whenever $i = 0$, $j = 0$, $i = m + 1$ or $j = n + 1$. For example, the perimeter of the discrete set of Fig. 1 is equal to 26. Note that the perimeter of an *hv*-convex discrete set is always determined by its size (and then, of course, by its projections, too). Namely, every *hv*-convex discrete set of size $m \times n$ has a perimeter of $2(m + n)$. Moreover, it is also easy to see, that it is the minimal possible perimeter value among discrete sets of size $m \times n$.

In the reconstruction task, a class $\mathcal{G}$ of discrete sets is defined, and the vectors H and V are given. The goal is construct a discrete set $F \in \mathcal{G}$ such that $\mathcal{H}(F) = H$, $\mathcal{V}(F) = V$ are satisfied. Throughout the presented experiments in this paper, we used only the horizontal and vertical projections to try to retrieve the mentioned features of the test images. Due to the small number of projections the reconstruction task is usually extremely underdetermined and for certain classes, NP-hard as well. For example Fig. 2 shows two different discrete sets with the same horizontal and vertical projections. Therefore every information available before the reconstruction could provide help in increasing accuracy, and speeding up the process. In the last 15-20 years several reconstruction methods have been developed for certain classes of discrete sets, all of them using some prior knowledge of the image to be reconstructed. Some of them performs the reconstruction directly, while others reformulate the task to a global optimization problem. However, it is always supposed that the prior information is given explicitly. Even in this case, it is a hard work to fine-tune the parameters of the global optimizer (e.g. simulated annealing or genetic algorithms) to obtain good reconstructions. Up to now, no methods are known to choose the appropriate reconstruction algorithm (or to set its parameters) automatically, if the prior information is not known in advance. In the following we take an attempt to use learning methods, like neural networks, to solve this problem.

For the task of estimating the number of distinct gray-level intensities in a discrete image, obviously only binary images would not have been enough to test our method, and determine that our approach performes well. Hence, in addition to the various types of binary images generated for each problem defined, specifically for Section 5, several series of discrete images have been created. These images have a defined number of disks, with fix positions and fix sizes, and only differ in the gray-intensities of the disks. Such an image can be seen in Fig. 3. Note that formulation the horizontal and vertical projections do not differ from that of the binary case, the only difference is that f_{ij} can not only be 0 or 1, but it can be any of the finite number of intensities in between those two extremal values.

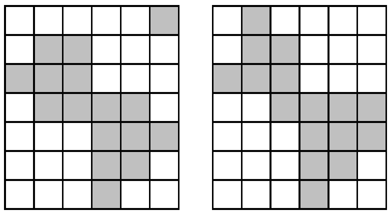


Fig. 2. Discrete sets having the same horizontal and vertical projections, but different perimeters.

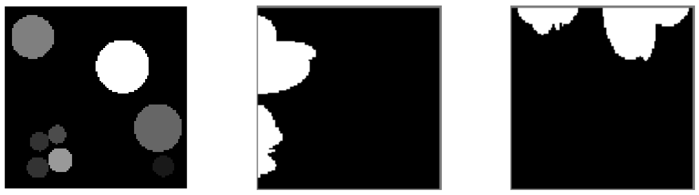


Fig. 3. A discrete image and its horizontal and vertical projections (from left to right, respectively).

3 Machine Learning

3.1 Basics of Neural Network Theory

A multilayer neural network (NN, in short) consists of units called neurons arranged in separate layers. Neurons pass information to each other via directed links. Each of these directed connections, i.e. from neuron j to neuron i, transmits an activation value a_j of the sending neuron with a weight $W_{j,i}$ attached to it. In a *feed-forward* neural network, a neuron that belongs to a certain layer can receive input only from neurons in the preceding layer. After computing the weighted sum of its inputs $in_i = \sum W_{j,i}a_j$, the neuron passes this value to an activation function g of its own, which calculates the output $a_i = g(in_i)$. The result is then forwarded to neurons in the next layer. Basically one could view a feed-forward multilayered network architecture as a directed, weighted graph, where the information flows in the same direction from layer to layer. A simple, three layer architecture is shown in Fig. 4. Such a system with one hidden layer in the middle holding a sufficient number u of units is able to represent any continuous function of its input, thus is chosen for our experiments.

The training of the network is done with the aid of a *training set* of samples. Each sample and the desired output as a pair serves as a single input pattern for the network. A run through all the training samples is called an *epoch*, normally it takes several epochs for the network to learn the target function. By propagating the input through the network, then comparing the network's output with the target value corresponding to the sample, the error is measured. Weights of the connections responsible for the error are slightly adjusted with each visited sample. Updating the weights of links between the hidden and the output layer is rather straightforward: $W_{j,i} = W_{j,i} + \alpha a_j \Delta_i$, where $\Delta_i = Err_i g'(in_i)$ and Err_i is the error of output unit i. The *learning rate* α determines the degree of the modification to be made to a weight after evaluating the error on a single training sample. For updating the weights of links in between the input and the hidden layer ($W_{k,j}$), the well-known *backpropagation method* is used [24]. The aim is to minimize the error on the entire training set, therefore the training is a search through the weight space to find a setting with a minimal sum of squared errors on the training data. The assumption is that the obtained setting

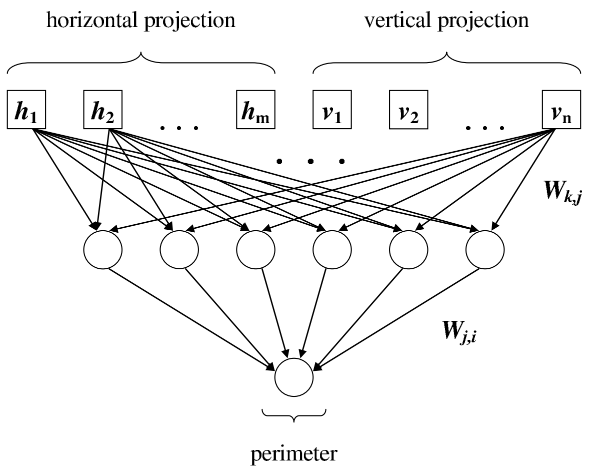


Fig. 4. A basic feed-forward neural network with one hidden layer. In case the discrete set is of size $m \times n$ the horizontal and vertical projecions form a vector of length $m+n$, hence the input layer itself contains $m + n$ units. The hidden layer consists of u units, while the output layer is only one neuron. $W_{k,j}$ and $W_{j,i}$ denote the weights for each connection between the input and the hidden layer, and between the hidden and the output layer, respectively. The range of the indices are: $k = 1, \ldots, m+n$; $j = 1, \ldots, u$; $i = 1$.

will provide the best performance on an unseen set of samples as well, given that both originate from the same distribution.

One of the unseen set of samples is the *generalization set*, that contains patterns that will be used to measure the accuracy of the network on unseen data after every epoch. Patterns in the *validation set* will be run through the network after one of the terminating conditions for the training process is satisfied, so this is basically a final check of the networks ability how it handles unseen input patterns. The classic split between these three sets is $60\% - 20\% - 20\%$, respectively. For the largest portion, the training dataset, advanced data partitioning methods exist which enable further partitioning of this subset into smaller parts. The *growing dataset* approach trains the network with a growing subset of the training set which would increase with a given fixed percentage after every epoch until it contains all the samples in the training set.

A technique called *momentum* is also used to speed up the training process. An update of a certain weight of a connection is based on the previous update made on the same connection. It is favorable that the direction (+/-) of the previous modification is preserved, since the network is more likely to generalize better this way. In case the current weight update specified by the output error is in the opposite direction as the previous, the momentum constant β also makes sure that a step in a new direction is a small, initial one. The resulting formulas for weight updates are the following:

$$\begin{aligned} W_{j,i} &= W_{j,i} + \varDelta W_{j,i}(t) \\ \varDelta W_{j,i}(t) &= \alpha a_j \varDelta_i + \beta \varDelta W_{j,i}(t-1), \end{aligned} \tag{2}$$

and

$$\begin{aligned} W_{k,j} &= W_{k,j} + \varDelta W_{k,j}(t) \\ \varDelta W_{k,j}(t) &= \alpha a_k \varDelta_j + \beta \varDelta W_{k,j}(t-1). \end{aligned} \tag{3}$$

As mentioned earlier, conditions need to be defined when to terminate the training of the network. The training can be stopped once a maximum number of epochs is reached, this option is useful in cases when the network fails to converge to a stable state. Accuracy can be measured too, both on the training set (*training set accuracy* - TSA) and the generalization set (*generalization set accuracy* - GSA). The former gives the number of correctly classified training patterns divided by the the total number of training patterns, while the latter gives the number of correctly classified patterns in the generalization set divided by the total number of patterns in the generalization set.

In summary, the common structure of the NN's used in our experiments are the following. The network holds only one hidden layer to estimate the perimeter of certain classes of discrete sets. Neurons in the input layer receive the horizontal and vertical projection values, so their number is fixed by the size of discrete sets under investigation. The output layer consists of a single output node which gives the final estimated value we are looking for, like which class does the image belong to, what is the estimated perimeter, and so on (see again Fig. 4). Determining the number of hidden neurons to allocate is a rather difficult task. Having too few causes lack of performance, on the other hand having too many causes overfitting on the training data. For each and every one of the studied problems, the optimized parameters of the networks are presented in a tabular form.

3.2 Implementation of NN's

Two different implementations have been chosen for our simple feed-forward multilayer neural network, that possesses one hidden layer and utilizes back-propagation learning, as described before. Initially, we decided to use the implementation of [1]. This is a fairly easy to understand C++ implementation of such networks, and it avoids the unnecessary object oriented design in this case. Rather than modeling each neuron as an object and modeling the network as a container for these objects, it uses several one- and two dimensional arrays to store the activation values, the weights, the weight changes and the error gradients for all three layers, providing possibly the most efficient way to represent a simple multilayer network. This form of implementation proved to be succesful during the experiments shown in Section 4.

Later on, we decided to take advantage of other existing solutions as well, such as the open-source WEKA framework [18]. WEKA offers several machine learning tools implemented in JAVA, encapsulated in a user friendly GUI that enables users to set various parameters. Among those tools is the class named *Multilayer Perceptron*. This is a realization of a multilayer feed-forward network with back-propagation learning, aided by the momentum technique.

We describe a few properties of both of the implementations here in more detail:

– The activation function, by default, for every neuron is the *sigmoid function* $P(t) = 1/(1 + e^{-t})$.
– Throughout our experiments we used networks with a single hidden layer to connect the input and output layers, only changing the number of hidden units in that particular layer when needed.
– The initial weights for all links between neurons in adjacent, separate layers were randomly set to uniformly distributed values in the interval of [-0.5, -0.5] and [-0.05, +0.05], respectively, for the two implementations.

Table 1. The parameters of the neural network and the precision of the classification for the datasets of *hv*-convex polyominoes and *hv*-convex 8- but not 4-connected discrete sets. In all test cases $\beta = 0.9$ and the number of train and test samples were 1800 and 600, respectively. For the size of the squared matrices only one dimension is given.

Size	Epoch	Nr. Hidden	α	VSA	Error
10	50000	30	10^{-4}	78.6667	21.3333
50	50000	120	$10^{-3} \to \cdots \to 10^{-6}$	85.5556	14.4444
100	10000	200	$10^{-3} \to \cdots \to 10^{-7}$	88.8889	11.1111
150	7500	250	$10^{-3} \to \cdots \to 10^{-7}$	92.6667	7.3333
200	5000	300	$10^{-3} \to \cdots \to 10^{-7}$	94.9444	5.0556

4 Classification of *hv*-Convex Polyominoes and *hv*-Convex 8-, but not 4-Connected Discrete Sets

Interestingly the reconstruction of *hv*-convex 8- but not 4-connected discrete sets can be performed faster than the reconstruction of *hv*-convex polyominoes [2]. For this to happen we need a method to decide which of the earlier mentioned two classes does the discrete set belong to. The evident approach so far was to apply the faster algorithm for reconstructing *hv*-convex 8- but not 4-connected sets, if the outcome is failure then the discrete set is a *hv*-convex polyomino and the other algorithm has to be used for reconstruction. If we could answer this question prior to using any reconstruction methods, that information would help to choose the appropriate reconstruction process.

We generated 8-, but not 4-connected discrete sets by generating 8-connected sets [3] and discarding the 8-connected sets which were also 4-connected from the bunch. We used 1800 training and 600 test samples for all sizes of matrices. All of our datasets contained 50-50% of the investigated classes. We calculated the error as the percentage of incorrectly classified samples over the number of total samples. This is easily obtained from the validation set accuracy as 100−VSA.

During this task we tried the growing dataset method, which turned out to be successful. Initially we took a subset of 360 training samples, and we increased this subset with a fixed percentage after every completed epoch. This way we managed to keep the values of TSA and GSA relatively closer to each other than in other tasks before (see [16] for further details). This feature, and the fact that this problem suited neural networks well, resulted in low error percentages in the validation set, as seen in Table 1 and Fig. 5.

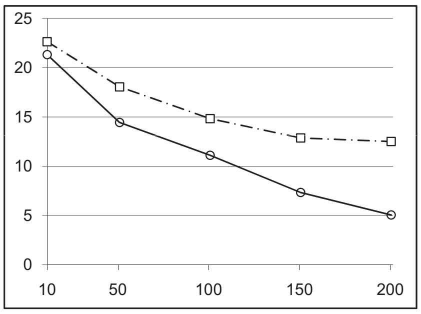


Fig. 5. Average error of classification of *hv*-convex polyominoes and *hv*-convex 8-, but not 4-connected discrete sets in percentage (vertical axis) as it depends on the size of the matrix (horizontal axis, only one dimension of the size of the squared matrices is given) by using decision trees (□) and neural networks (○).

5 Determining the Number of Distinct Intensity Levels in Discrete Images

Discrete tomography utilizes the strong assumption that the image to be reconstructed contains just a few gray-intensity values that are known beforehand. Determining the intensity levels is seemingly one of the most difficult problems in discrete tomography. In [10] the authors suggested to reconstruct the discrete image with many intensity levels, and then to perform a second reconstruction with the gray-intensity values defined by the peaks of the histogram of the image obtained in the previous reconstruction. In [8] a semi-automatic method was proposed to select the intensity values. However, up to now, no general method is known to solve this task. In this section we investigate a closely related problem. We study how machine learning can be used for determining the number of intensity values present in the discrete image, at least for a restricted class of images. For neural networks we use the aforementioned Multilayer Perceptron of the WEKA toolbox [18].

5.1 Generated Datasets

In the experiments we used the horizontal and vertical projections, thus the attributes of each learning instance were the values of those two projections. In the following we will call a set of disks with fixed size and position as a *configuration.* That is, instances of the same configuration differ only in the gray-intensity values used in the image. We performed the classification with 10 different configuration for neural networks. Each configuration contained 8 randomly generated disjoint disks with fix positions and equal – at least 5 unit long – radius for that particular configuration (for an example see again Fig. 3). For classification purposes the generated training and testing datasets contained 3600 and 1200 images, respectively, for every configuration. Beside the background intensity (that was 0 in every case), futher intensities of the disks were randomly chosen from a given intensity list. The lists that were used contained equidistant points in $[0, 1]$ defining the grayscale values.

5.2 Experimental Results

In our experiments we used two types of error measurement. The first one is the common, strict method to calculate classification errors: each misclassification is treated as an error. In this case only the diagonal elements of the classification matrix belong to the correctly classified cases (dark gray elements of Table 2). Hereunder we call this method the *normal* error measurement. The second one is a more permissive type of measure. In this case if the difference between the output of the classifier and the exact number of distinct intensities is not greater than 1, the result is accepted. For example for a given image with 4 different intensity values, outputs 3, 4, and 5 are all treated as correct classifications (none of the gray elements of Table 2 are misclassifications).

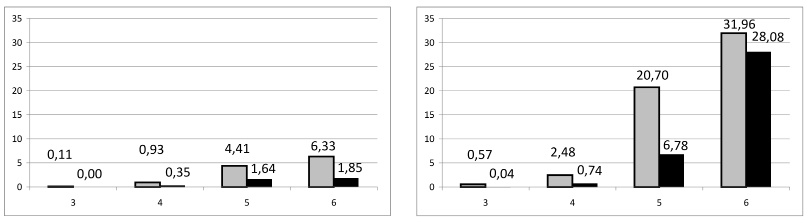
Table 2 shows the average of the 10 acquired classification matrices for neural networks on classifying images with 1-6 distinct intensity levels. The dark gray elements represent the good classifications in normal measurement, while during the permissive measurement mode every case that corresponds to a gray-shaded element in a row is accepted as a correct classification.

We also investigated the robustness of the presented methods by performing the same experiments but this time with noisy projection data. In these tests we used additive noise with uniform distribution with a noise ratio of 5%. Comprehensive results are shown in 6. With NN's we attempted to distinguish 3 to 6 different intensity levels at once (as seen in Fig. 6). The main reason behind the reduction in the latter case was the extremely long training time. Finding the proper weights of neural networks proved to be cumbersome. Nevertheless, practical applications of discrete tomography usually involve no more than 4 or 5 intensity levels.

Table 2. The average of 10 classification matrices of all configurations for neural networks for 1-6 equidistant intensity values. The numbers in brackets in the last column represent the exact number of intensities actually in the image, while in the first row the estimated number of intensities by the machine learning method is shown. Matrix entries are given in percentage (rounded to two digits) of the test cases for each number of intensities.

(a) Neural network

(1)	(2)	(3)	(4)	(5)	(6)	← classified as
100.00	0.00	0.00	0.00	0.00	0.00	(1)
0.00	98.90	1.00	0.10	0.00	0.00	(2)
0.20	2.30	91.85	4.20	0.85	0.60	(3)
0.00	0.75	3.10	75.90	12.35	7.90	(4)
0.00	0.00	0.70	3.90	95.40	0.00	(5)
0.00	0.00	0.00	0.00	0.00	100.00	(6)



(a) Noiseless (b) With 5% noise

Fig. 6. Classification error of the neural networks depending on the number of different intensity levels in the image without (a) and with (b) noise. For each number of intensities the plotted bars represent the average error for equidistant intensity values with normal (gray) and permissive (black) error measurement.

On the other hand, neural networks accomplished the task surprisingly well. This is probably due to the various parameters available to configure the method in an optimal way. Careful adjustments of these variables – such as learning rate, momentum, etc. – can lead to better classification results. To find the parameters close to the best possible we tested several settings for each dataset. By modifying one parameter at once, and observing its effect on the classification result on the training data, we tried to keep track of each parameters optimal direction of change. This way in some cases a clear pattern has been found how to set up the network properly. The average of used parameter setups are displayed in Table 3. α is usually decreased as the training goes on, and this learning rate decay in this implementation is achieved by dividing the learning rate after each epoch by the number of epochs completed so far. Thus, Table 3 contains the averages of the initially set learning rates. The momentum, on the other hand, did not change during training.

Table 3. Average values of the parameters of the neural network classification.

Noiseless

#intensities	Learning rate	Momentum	Training time	Hidden neurons
3	0.2	0.8	100	10.5
4	0.24	0.78	190	16
5	0.27	0.75	370	41
6	0.238	0.8275	530	55.5

5% Noise

#intensities	Learning rate	Momentum	Training time	Hidden neurons
3	0.2	0.8	100	10
4	0.3	0.8	200	20
5	0.27	0.75	740	41
6	0.2218	0.8275	133	54

We drew the conclusion from our experiments on neural networks that we had to increase the number of hidden neurons as the number of intensities increased. In the noiseless case increasing the training time (number of epochs) provided better results. On the noisy dataset longer training time gave worse results, probably because the network tended to overfit the given training samples, therefore it was not be able to generalize as well as expected.

6 Perimeter Estimation of Some Discrete Sets

6.1 Methods for Generating Test Images

For investigating how well NN's can estimate the perimeter of some simple and also more complex discrete sets, we needed methods to generate discrete sets for this purpose. For images having relatively simple structure, we decided to choose the class of h-convex discrete sets. Elements of that class were generated in the following way. In each row of the h-convex discrete set, we put exactly one sequence of consecutive 1s. If the size of the discrete set is $m \times n$, then the length of the sequence can vary between 1 and n. A sequence of length $k \in \{1, \dots, n\}$ can start at $n-k+1$ different positions of the row. Therefore, the probabilty that the generated sequence for a single row is of length k was set to $\frac{n-k+1}{n(n+1)/2}$. Once the length of the sequence was determined, we defined its starting position simply by picking a random integer from the interval $[1, n-k+1]$ using a uniform distribution. Notice that the size of the discrete set produced this way is not necessarily $m \times n$, it may be less, as the vertical projection of the discrete set can contain zero values, sometimes even as the first or/and the last coordinate. However, we found that it occured in just an insignificant number of the cases, and in the following we neglect this phenomenon. Note also that a similar method can be applied to generate v-convex discrete sets, too.

In order to obtain discrete sets with more complex structure, we combined h-convex and v-convex discrete sets, by taking the union of them. Formally, using the correspondig representing matrices, such a matrix M can be calculated as $m_{i,j} = \max(b_{i,j}, c_{i,j})$ where B and C is a randomly generated h-convex (v-convex) matrix, respectively. Figure 7 shows an h-convex and a v-convex set, and a discrete set generated from these two sets in the abovementioned way. For the sake of simplicity, in the followings we will refer to this class of discrete sets as *random discrete sets*.

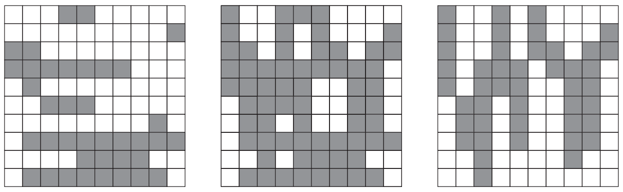


Fig. 7. Example of generating a random discrete set (middle) as a combination of an h-convex (left) and a v-convex (right) discrete set.

First we attempted to find the optimal settings for the perimeter estimation of h-convex discrete sets. The training set and the test set included 1500 and 300 samples, respectively. We have experimented with different splittings and ratios – 1600 training- and 300 test samples; 2000 training- and 500 test samples; etc. – but none of these changes had any significant, let alone positive effect on the network's performance. After careful experimenting the following parameters were found to be the best: learning rate α was set to 0.001, momentum β was set to 0.3, and the number u of hidden neurons used is presented in Table 4. Although $\beta = 0.3$ is an unusually low value for the momentum, it assured the most stable results. Table 4 shows that the number of hidden units somewhat increased along with the size of the sets investigated, with only a few exceptions. For estimating the perimeter of random discrete sets, the setup only differed in the required number of units in the hidden layer, this is also presented in Table 4.

Table 4. Number of hidden neurons used for the perimeter estimation of h-convex and random discrete sets, depending on the size of the sets.

Size	10	20	30	40	50	60	70	80	90	100
h-convex	20	40	40	40	50	70	80	60	70	80
random	10	30	35	45	50	45	50	55	60	60

6.2 Experimental Results

The estimation of the exact perimeter could be a very hard, in some cases even impossible task, since two different discrete sets with different perimeters can have the same horizontal and vertical projections (see, e.g., Fig. 2). Thus, our aim was to study the accuracy of the neural network in case *some degree of uncertainty* is allowed. The following uncertainty levels were introduced: from 1% to 10%, and 20% difference from the actual perimeter of the given discrete set presented in the test image.

Every test - defined by the allowed uncertainty and the size in question - was repeated five times, and the end result is the average of these separate runs. The final results for h-convex sets and random matrices were quite similar and showed the same trend. Actually, neural networks provided slightly better results for estimating the perimeter of random discrete sets, therefore results for h-convex sets were omitted from this section. Fewer hidden neurons were sufficient for this class (see Table 4) and the number of these hidden units increased more or less steadily with the size in this case, making this a noticeably easier task for the network than estimating the perimeter of h-convex sets. For example, by allowing a 10% deviation from the exact perimeter for random sets of size 100×100 the network mispredicted the perimeter of only 0.2% of all test samples. In the same test configuration, with an allowed uncertainty of 10%, the error is about 3.0% for h-convex discrete sets of size 100×100. Considering that one could obtain this information from merely two projections, it can prove to be a useful result for practical application purposes.

Table 5. Perimeter estimation of random discrete sets from size 10×10 to 100×100. The leftmost column indicates the allowed difference between the true and the estimated perimeter. The entries show the average percentage of the misclassified test data.

Uncertainty	10	20	30	40	50	60	70	80	90	100
1%	87.80	86.33	85.33	85.07	82.80	81.80	80.60	81.20	79.00	76.20
2%	77.40	73.40	70.20	68.87	67.53	62.13	61.40	61.47	59.80	56.13
3%	67.60	62.40	56.67	53.73	52.67	47.40	44.67	42.80	42.40	39.53
4%	57.07	51.07	45.33	41.93	37.20	33.80	31.20	30.80	28.47	24.27
5%	48.67	41.60	34.87	30.87	26.20	23.73	20.27	19.67	16.33	14.60
6%	40.47	31.33	26.40	21.67	16.13	14.13	13.47	11.33	9.87	7.33
7%	31.93	24.00	20.27	14.93	11.40	9.33	7.73	6.20	5.53	3.87
8%	24.60	19.13	15.13	10.33	7.47	5.33	4.40	3.53	2.53	1.73
9%	20.07	13.73	10.60	7.07	4.80	3.87	2.27	2.27	1.27	0.60
10%	15.27	10.93	7.80	5.13	2.60	2.07	1.07	1.13	0.33	0.20
20%	0.87	0.47	0.47	0.07	0.00	0.00	0.00	0.00	0.00	0.00

7 Conclusions

Results show that neural networks can be used to acquire geometrical information of a discrete (binary) image from its projections only, without actually reconstructing the image itself. We investigated here the problem of connectedness and convexity, estimating the number of intensity levels in an image, and the estimation of the perimeter solely based on the projection data. Such information is especially useful in discrete tomography. As a preprocessing step, the methods presented here can be used to supply valuable knowledge to aid advanced reconstruction algorithms that could exploit this advantage. One way to do this is to reformulate the reconstruction problem, and to attack the problem at hand as a global optimization task, this topic is still under heavy investigation. Also, improved versions of these tools might be able to identify different geometrical or topological features, or possibly even a combination of several ones. For instance, since the area of the object under investigation is given by the projections, one might be tempted to use the estimated perimeter, and try to draw conclusions about the possible shapes the object might have. Overall, in this paper we gave an up-to-date summary of our work on this topic, additional experiments are to follow.

Acknowledgments

The publication is supported by the European Union and co-funded by the European Social Fund.
Project title: ”Broadening the knowledge base and supporting the long term professional sustainability of the Research University Centre of Excellence at the University of Szeged by ensuring the rising generation of excellent scientists.”
Project number: TÁMOP-4.2.2/B-10/1-2010-0012.

References

1. Anguelov, B. Basic Neural Network Tutorial : C++ Implementation and Source Code. `http://takinginitiative.wordpress.com/2008/04/23/basic-neural-network-tutorial-c-implementation-and-source-code/`
2. Balázs, P., Balogh, E., Kuba, A.: Reconstruction of 8-connected but not 4-connected hv-convex discrete sets. Disc. Appl. Math. 147, 149-168 (2005)
3. Balogh, E., Kuba, A., Dévényi, C., Del Lungo, A.: Comparison of algorithms for reconstructing hv-convex discrete sets. Lin. Algebra and its Applications 339, 23–35 (2001)
4. Barcucci, E., Del Lungo, A., Nivat, M., Pinzani, R.: Medians of polyominoes: A property for the reconstruction. Int. J. Imaging Systems and Techn. 9, 69-77 (1998)
5. Batenburg, K.J.: An evolutionary algorithm for discrete tomography. Discrete Appl. Math. 151, 36–54 (2005)
6. Batenburg, K.J., Bals, S., Sijbers, J., Kuebel, C., Midgley, P.A., Hernandez, J.C., Kaiser, U., Encina, E.R., Coronado, E.A., Van Tendeloo, G.: 3D imaging of nanomaterials by discrete tomography. Ultramicroscopy 109(6), 730–740 (2009)

7. Batenburg, K.J., Kosters, W.A.: A neural network approach to real-time discrete tomography. Lecture Notes in Comput. Sci. 4040, 389–403 (2006)
8. Batenburg, K.J., Van Aarle, W., Sijbers, J.: A semi-automatic algorithm for grey level estimation in tomography. Pattern Recognition Letters 32, 1395-1405 (2011)
9. Baudrier, É., Tajine, M., Daurat, A. Convex-set perimeter estimation from its two projections, Lect. Notes Comput. Sci. 6636, 284–297 (2011)
10. Baumann, J., Kiss, Z., Krimmel, S., Kuba, A., Nagy, A., Rodek, L., Schillinger, B., Stephan,J.: Discrete tomography methods for nondestructive testing, In: [21], pp. 303–331 (2007)
11. Brunetti, S., Daurat, A.: An algorithm reconstructing convex lattice sets. Theor. Comput. Sci. 304, 35-57 (2003)
12. Brunetti, S., Del Lungo, A., Del Ristoro, F., Kuba, A., Nivat, M.: Reconstruction of 4- and 8-connected convex discrete sets from row and column projections. Lin. Alg. Appl. 339, 37-57 (2001)
13. Chrobak, M., Dürr, C.: Reconstructing hv-convex polyominoes from orthogonal projections. Inform. Process. Lett. 69(6), 283-289 (1999)
14. B. van Dalen, The boundary and shape of binary images, Discrete Mathematics 310, 2910–2918 (2010)
15. B. van Dalen, Boundary length of reconstructions in discrete tomography, SIAM Journal of Discrete Mathematics 25, 645–659 (2011)
16. Gara, M., Tasi, T.S., Balázs, P.: Learning connectedness and convexity of binary images from their projections. Pure Math. and Appl. 20, 27–48 (2009)
17. Gara, M., Tasi, T.S., Balázs, P.: Machine Learning as a Preprocessing Phase in Discrete Tomography, Applications of Discrete Geometry and Mathematical Morphology, Lecture Notes in Computer Science Volume 7346, 109–124 (2012).
18. Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., Witten, I. H.: The WEKA Data Mining Software: An Update. SIGKDD Explorations 11(1), 10–18 (2009).
19. Herman, G.T.: Fundamentals of Computerized Tomography: Image reconstruction from projections. Springer, Heidelberg (2009)
20. Herman, G.T., Kuba, A. (eds.): Discrete Tomography: Foundations, Algorithms and Applications. Birkhäuser, Boston (1999)
21. Herman, G.T., Kuba, A. (eds.): Advances in Discrete Tomography and its Applications. Birkhäuser, Boston (2007)
22. Lukić, T., Lukity, A., Gogolák, L., Binary tomography reconstruction method with perimeter preserving regularization, Proceedings of the 8th Conference of the Hungarian Association for Image Processing and Pattern Recognition, 83–91 (2011)
23. Kak, A.C., Slaney, M.: Principles of Computerized Tomographic Imaging. IEEE Press, New York (1999)
24. Mitchell, T.M.: Machine Learning. McGraw Hill, New York (1997)
25. Tasi, T.S., Hegedus, M., Balázs, P.: Perimeter estimation of some discrete sets from horizontal and vertical projections. Proceedings of the IASTED Internation Conference, Signal Processing, Pattern Recognition and Applications 779: Computer Graphics and Imaging (SPPRA 2012), 174–182 (2012)
26. Valenti, C.: A genetic algorithm for discrete tomography reconstruction. Genet. Program Evolvable Mach. 9, 85–96 (2008)
27. Van Aert, S., Batenburg, K.J., Rossell, M.D., Erni, R., Van Tendeloo G.: Three-dimensional atomic imaging of crystalline nanoparticles. Nature 470, 374–377 (2011)