

Learning Connectedness and Convexity of Binary Images from Their Projections

Mihály Gara,* Tamás Sámuel Tasi† and Péter Balázs‡

Abstract

In this paper we investigate the retrieval of geometrical information (especially, convexity and connectedness) of binary images from their projections which can be useful in binary tomography to facilitate the task of reconstruction. Supposing that the projections are the features of the images, we study how decision trees, neural networks, and nearest neighbor learning algorithms perform in classifying binary images with different connectedness and convexity properties.

Keywords: Discrete tomography; machine learning; classification; *hv*-convex and connected discrete set

1 Introduction

Reconstruction Tomography is an imaging procedure for obtaining the two-dimensional (2D) cross-sections of three-dimensional objects from their projections. Image reconstruction is mainly used in medical imaging, but it also has applications in non-destructive testing, crystallography, data security, image processing, and so on. For reconstructing a simple 2D slice, usually several hundreds of its projections are needed [15]. *Discrete Tomography* [13, 14] investigates how the prior knowledge that the image to reconstruct should only contain a few grey-values known beforehand can be exploited to guarantee accurate reconstructions from just a few (say, less than 10) projections, too. In *Binary Tomography* (BT) the task is to reconstruct a binary image (also called discrete set) from a small number of projections. Applications of BT arise from several fields of science, too, including non-destructive testing [9], angiography [22], and electron microscopy [16].

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

†Faculty of Natural Sciences and Informatics, University of Szeged, Aradi vértanúk tere 1., H-6720 Szeged, Hungary, email: Tasi.Tamas.Samuel@stud.u-szeged.hu

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

Due to the small number of available projections in BT the reconstruction is usually an underdetermined and/or an NP-hard problem. Despite this, there is a hope of avoiding intractability and of reducing the number of possible solutions if some prior knowledge can be incorporated into the reconstruction process. The most commonly used properties that the discrete set to be reconstructed has to satisfy to facilitate the reconstruction are of geometrical nature, like connectedness or convexity. Depending on the geometrical properties of the image many direct reconstruction algorithms have been developed [3, 7, 10, 11, 12, 18, 19, 20]. However, these are restricted to work just for a given class of images defined by those geometrical features. Although, prior information about the geometry of the image proved to be useful, unfortunately, no method is known to gain such knowledge solely from the projections of the image. This means that – unless geometrical information is not explicitly given – one has to perform all the candidate reconstruction algorithms until a proper solution is found. Beside the direct methods, binary reconstruction can also be solved by several global optimization techniques (see e.g. [24] and the references given there) but even in this case, additional a priori geometrical information is important to set the parameters of the global search procedure properly, and to improve the speed and quality of the reconstruction.

In this paper we study the retrieval of geometrical (especially convexity and connectedness) properties of binary images from their projections themselves. We study decision trees, neural networks, and nearest neighbor learning methods how they perform in classifying binary images with different connectedness and convexity properties. The paper is structured as follows. The necessary definitions and some results of binary tomography related to our work are given in Section 2. In Section 3 we investigate several learning methods how they can be applied to obtain geometrical properties of binary images from tomographic projections. In Section 4 we present experimental results. Finally, in Section 5 we summarize our experiences.

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*, *vertical*, *diagonal*, and *antidiagonal* projections of a discrete set F are defined by the vectors $\mathcal{H}(F) = (h_1, \dots, h_m)$, $\mathcal{V}(F) = (v_1, \dots, v_n)$, $\mathcal{D}(F) = (d_1, \dots, d_{m+n-1})$, and $\mathcal{A}(F) = (a_1, \dots, a_{m+n-1})$, respectively, where $h_i = \sum_{j=1}^n f_{ij}$ ($i = 1, \dots, m$), $v_j = \sum_{i=1}^m f_{ij}$ ($j = 1, \dots, n$), $d_k = \sum_{i+(n-j)=k} f_{ij}$, and $a_k = \sum_{i+j=k+1} f_{ij}$ ($k = 1, \dots, m+n-1$). For example, for the discrete set F in Fig. 1 $\mathcal{H}(F) = (2, 3, 3, 2, 3, 2)$, $\mathcal{V}(F) = (1, 3, 4, 2, 1, 2, 2)$, $\mathcal{D}(F) = (0, 0, 0, 0, 2, 5, 5, 3, 0, 0, 0, 0)$, and $\mathcal{A}(F) = (0, 2, 2, 2, 1, 2, 1, 0, 1, 1, 2, 1)$.

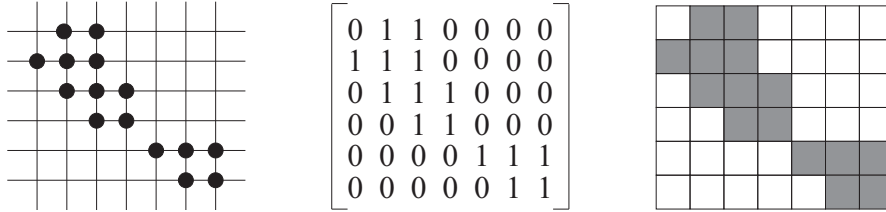


Figure 1: An hv -convex 8- but not 4-connected discrete set represented by its elements (left) and by a binary matrix (center), and the corresponding binary image (right).

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 $|p_1 - q_1| + |p_2 - q_2| = 1$ ($\max(|p_1 - q_1|, |p_2 - q_2|) = 1$), 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 $P_0 = P, \dots, P_k = Q$ in the discrete set F such that P_l is 4-adjacent (8 -adjacent) to P_{l-1} , respectively, for each $l = 1, \dots, k$. 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.

We also define a hierarchy of discrete sets which was introduced in [4]. For $0 \leq p \leq 100$ let \mathcal{HV}_p denote the class of binary matrices that we get from an arbitrary hv -convex polyomino F by changing at most p percent of the elements of the matrix representing F . The elements to be modified are chosen from a uniform random distribution. In other words, a discrete set F of size $m \times n$ is in \mathcal{HV}_p ($0 \leq p \leq 100$) if one can obtain an hv -convex polyomino by inverting at most $pmn/100$ positions of the matrix representing F . In this way \mathcal{HV}_0 corresponds to the class of hv -convex polyominoes and for an arbitrary $p > 0$ a discrete set of the class \mathcal{HV}_p differs from an hv -convex polyomino in at most p percent of the positions. Of course, it can happen that the resulted set is also an hv -convex polyomino (possibly different from the original one). Thus, the classes \mathcal{HV}_p can be regarded as the generalizations of the class of hv -convex polyominoes (see Fig. 2).

In the reconstruction task a class \mathcal{G} of discrete sets is defined, and the vectors H and V (or additionally also D and A) are given. The goal is construct a discrete set $F \in \mathcal{G}$ such that $\mathcal{H}(F) = H$, $\mathcal{V}(F) = V$ (or additionally $\mathcal{D}(F) = D$ and $\mathcal{A}(F) = A$) are satisfied. Due to the small number of projections this task is usually extremely underdetermined and for certain classes NP-hard as well. Therefore every information available before

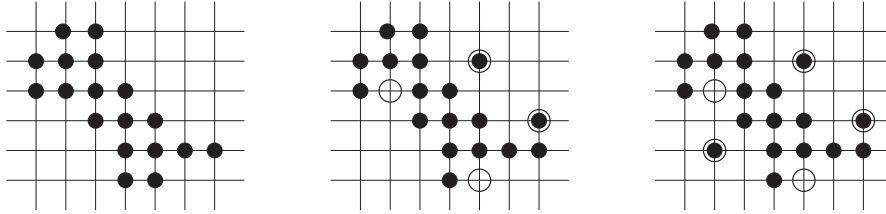


Figure 2: A discrete set of the class \mathcal{HV}_{10} (center) and one of \mathcal{HV}_{15} (right) derived from an hv -convex polyomino (left). Altered positions are marked by circles.

the reconstruction may help in increasing accuracy and speeding up the process. In the last 10-15 years several reconstruction methods using some prior knowledge of the image to be reconstructed have been developed for certain classes of discrete sets. Some of them performs the reconstruction directly while others by reformulating the problem to a global optimization task. 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 to solve this problem.

3 Methods for Learning Geometrical Properties from Projections

Feature selection methods, like decision trees [26] or floating search approaches [25], and in general, machine learning techniques, like instance-based learners or neural networks [21] are effective tools to detect important attributes of a given object, or to classify objects of the same type with the aid of their attributes. In addition, they are usually also robust to noise. In discrete image reconstruction problems one can think of the projection vectors of the binary images as the attributes of the discrete set. In this scenario – in order to facilitate the reconstruction – our task is to determine structural or geometrical properties of the discrete set from the attributes (i.e., from the projection components). The idea of applying learning methods in discrete tomography is not new in the sense that in [8] the authors designed neural networks to reconstruct discrete images from their projections. Although the methods presented there are quite promising they suffer from several drawbacks (restricted size of images, large number of hidden neurons, huge computing time, inexact reconstruction). In opposite, our

aim is not to reconstruct the image but to obtain robust characterization of its properties which then can be used to improve the speed and quality of the reconstruction.

3.1 Decision Trees

Decision trees are generally used for classification purposes. Internal nodes of the tree represent conditional expressions of the attributes while the leaves of the tree give classification results as labels for the investigated classes. The evaluation of an input sample starts at the root of the tree, and based on the result of the current expression of the node, we move down on the branch which belongs to the results obtained. If we reach a leaf the procedure returns with a classification label for that particular sample.

For building decision trees a labeled training dataset is required. During this process the training samples are analyzed by comparing the corresponding attributes of the input samples to create the expressions in the inner nodes. The more information we gain from a particular attribute about the class of the input pattern, the closer that attribute will be tested to the root of the tree, which means that the attributes that separate the dataset better get closer to the root. This is usually implemented with a greedy algorithm that works with an entropy-like measure to calculate the information gain for each attribute of the sample. Although the attributes can have various values the test of these usually have two outcomes – every inner node has two descendants – therefore a given attribute could be tested in several locations and in several different expressions in the same decision tree. A test for an attribute could be based on whether the value is in a given interval or belongs to a given set.

Unfortunately we cannot expect decision trees to provide correct classifications without any error, since generalization is impossible in some cases, unless the decision tree overfits our training samples, but then it will not be able to correctly classify unseen patterns. To avoid overfitting some branches of the tree built during the train phase are pruned. For more details on decision trees and their applications in binary tomography see, e.g., [4, 21].

3.2 Neural Networks

A neural network consists of units called *neurons* connected to each other by direct links [23]. A link from unit j to unit i transmits the activation value a_j of unit j to unit i . Each link between two neurons also has a weight ($W_{j,i}$) attached to it, which determines the strength and the sign of that particular connection. Note that every outgoing connection has the same activation value for a given unit, but the weights associated with the links may differ. A neuron calculates the weighted sum of its input values: $in_i = \sum_{j=0}^n W_{j,i} a_j$

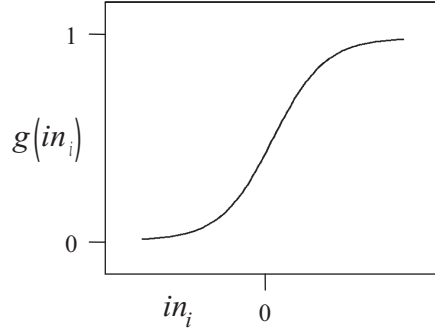


Figure 3: The most common choice for the activation function, the sigmoid function.

then applies an *activation function* to this sum to produce its output value: $a_i = g(in_i) = g\left(\sum_{j=0}^n W_{j,i}a_j\right)$. A common choice for this function g is the *sigmoid* $1/(1 + e^{-x})$ shown in Fig. 3. It has the advantage that it is differentiable, which is crucial for the learning algorithm presented shortly. The position of the actual threshold for unit i can be adjusted with the use of the *bias weight* of the unit ($W_{0,i}$). This is normally connected to a fixed input $a_0 = -1$.

The neural networks we used in our experiments come under the category of *feed-forward networks*, meaning that the information flows in a single direction, so there can be no directed cycles in the system (see Fig. 4). Such a network represents a function of its current input (with the weights of the network acting as parameters to this function) and it has no internal state other than the inner weights of the connections. Units are usually arranged in *layers*, which is vital to form a network with well organized connection setup between the neurons. In a *multilayer neural network* each unit receives input only from units in the preceding layer and sends output only to units in the following layer. Multilayer networks with one plus layer (called *hidden layer*) containing sufficient amount of hidden units are able to represent any continuous function of its inputs, and with two hidden layers even discontinuous functions can be represented.

For a classification task we need the network to learn with the aid of the presented training samples, this is achieved by comparing the desired output in the sample to the actual output, measuring the error (the classical measure is the sum of squared errors), then adjusting the weights in the network to minimize this error. Accordingly this means an optimization search in weight space. For multilayer networks *back-propagation learning* carries this idea into effect.

The weights corresponding to the links in direct connection with the output units can be updated easily: $W_{j,i} = W_{j,i} + \alpha a_j \Delta_i$, where $\Delta_i =$

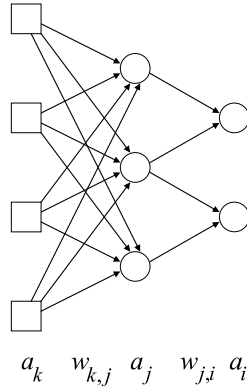


Figure 4: A simple multilayer feed-forward neural network with one hidden layer ($k = 1, \dots, 4$; $j = 5, \dots, 7$; $i = 8, 9$).

$Err_i g'(in_i)$ and Err_i is the error of the i -th output unit. The α in the formula is the *learning rate*, it determines the rate of the change made to each weight after evaluating the output of a given training example. For the weights of the connections in between the input and the hidden layer ($W_{k,j}$) the idea is that a hidden node a_j is "responsible" for some fraction of the error of a_i , and that fraction is determined by the strength of that connection. We used a technique called the *momentum* to speed up the learning process of the network. When updating a weight of a certain link it takes the direction of the previous step into consideration. It basically tries to keep the direction of the previously occurred update and even if the next weight update is in the opposite direction, it makes sure that the first step in the new direction is a small step. The final weight update formulas are the following: $W_{j,i} = W_{j,i} + \Delta W_{j,i}(t)$, where $\Delta W_{j,i}(t) = \alpha a_j \Delta_i + \beta \Delta W_{j,i}(t-1)$ and $W_{k,j} = W_{k,j} + \Delta W_{k,j}(t)$, where $\Delta W_{k,j}(t) = \alpha a_k \Delta_j + \beta \Delta W_{k,j}(t-1)$. The momentum constant β is normally set to 0.9, or some other high value between 0 and 1, while the learning rate α is usually set to a low value value in the range of 0 to 1, like 0.001 for instance.

Before we can start training our network we have to initialize it first: the output value of the neurons, the weight changes and the error gradients are set to zero. Each weight in the network is set to a random value in a specific interval e.g. $(-0.5, 0.5)$, since usually we have no idea about the optimal values. Choosing the right number of neurons in the hidden layer could prove to be a difficult task. By initializing too much usually a problem called *overfitting* arises: the network tends to memorize each given sample instead of generalizing, so it will not be able to handle unseen data correctly. On the other hand if we don't allocate as much hidden neurons as needed the network will lack performance.

To measure the networks performance on unseen input patterns, it is

common to split the data set into three parts. The *training set* is used to train the network, which means updating the weights after every training sample. A run through all the training samples is called an *epoch*, normally it takes several epochs for the network to learn the target function. The *generalization set* contains patterns that will be used to measure the accuracy of the network on unseen data after every epoch. Finally, patterns in the *validation set* will be run through the network after one of the terminating conditions for the training process is satisfied. The network's ability to handle unseen input samples is measured on this set and it is called the *validation set accuracy* (VSA). The classic split between these three sets is 60% – 20% – 20%, respectively. For the largest portion, the training dataset, advanced data partitioning methods exist, like the *growing dataset* approach, when the network is trained on a growing subset of the training set which increases with a given fixed percentage after every epoch until it contains all the samples in the training set.

As mentioned earlier, conditions need to be defined when to terminate the training of the network. 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 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.

3.3 Nearest Neighbor Approaches

Nearest neighbor approaches are statistical classifying methods. They classify the instances of the test dataset by measuring their distance from the instances in the training dataset. The distance metric used is usually the simple Euclidean distance calculated in the n -dimensional vector space (also called feature space). The Euclidean distance is $d(X, Y) = \sqrt{\sum_{i=1}^n (x_i - y_i)^2}$ for two instances $X = (x_1, x_2, \dots, x_n)$ and $Y = (y_1, y_2, \dots, y_n)$ having n attributes.

When the distances between the test instance under study and all the training examples are known, the nearest neighbor method provides a decision about the class label of that single instance. For the simple nearest neighbor procedure this label will be the same as of the nearest instance. Nevertheless, there are much more sophisticated nearest neighbor algorithms. k -nearest neighbor methods (knn) establish the class label of the test instance with the aid of the k nearest instances, in most cases by a majority vote (here k is a positive integer, and in the case $k = 1$ we get the basic nearest neighbor algorithm). When $k > 1$, k -nearest neighbor meth-

ods are usually more efficient than the simple nearest neighbor algorithm because they are more robust against noisy training data. But with great k values misclassification may also occur, e.g. if the distance between the classes to be separated is small.

The biggest disadvantages of these algorithms are that they require large amount of memory and huge computational cost. One has to store all the training examples in the memory, and during the classification of the instance investigated, its distance to all known ones has to be computed. Though, more effective index-based methods exist (see, e.g., [17] for an overview), this is a problem and it is one of the main drawbacks nearest neighbor methods suffer from.

The *k-means algorithm* is also a distance-based classifier. This clustering procedure calculates the class centers iteratively, and does not require a training dataset. Each new point in the feature space is marked with the label of the nearest class center. After this, the class centers are recalculated which can be done in several different ways. The simplest one computes the vector representing a given class center as the average of the feature vectors of that class. The serious disadvantage of this method is that it gives different results depending on the order of the instances.

We also combined the knn and the k-means methods to get a very fast classifier which is independent from the order of the instances and does not require a lot of memory. This modified method consists of an offline and an online part. The offline – training – part determines the centers of the classes from the given training examples, while the second – testing – part only computes the distances between the class centers and the testing data. It is easy to see that this approach is much faster than the knn methods. Certainly, this algorithm does not provide as good results as the knn algorithms in most cases, especially when the classes are not well-separable. For some other modified versions of the k-means algorithm see [1] and the references given there.

4 Experimental Results

We conducted several experiments to study the performance of the learning approaches presented above. Discrete sets of size $m \times n$ were represented by an $(m + n)$ -dimensional feature vector $(h_1, \dots, h_m, v_1, \dots, v_n)$ formed by their horizontal and vertical projections. During the classification these feature vectors have been used as the input patterns for all studied learning algorithms. Once the training has completed the parameters have been appropriately set, e.g. the weights in a neural network have become final or the decision rules in a decision tree have been fully generated. Figure 5 shows an example for a decision tree built to classify hv -convex polyominoes and random binary matrices. When it comes to neural networks Fig. 6 illus-

trates the process of evaluating an input feature vector which is presented to the input nodes of the network in the top layer. These nodes send their activation values through the links to the units in the following layer, where the weights of all connections are already determined. At the end of the process the classification result appears on the output neuron.

In our experiments we used the well-known C4.5 decision tree [26]. Regarding the other main method, a simple feed-forward multilayer neural network with one hidden layer and back-propagation learning was applied. We decided to use the implementation of [2]. 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. For implementing the nearest neighbor approaches a self-made program has been developed in C programming language. For both the knn and the modified k-means algorithms the squared Euclidean distance metric has been used. The knn approaches have been tested on values $k = 1, 3, 5, 7$. Not surprisingly, for $k = 3$ and $k = 5$ the classification accuracy fell between the results obtained for $k = 1$ and $k = 7$, in all test cases studied. Therefore, in the followings we only deal with these latter values of k . In case of the modified k-means algorithm for defining the class centers one feature vector for each class have been created. Each element of these vectors was the average of the corresponding values in the training samples of the same class.

For the generation of the train and test data sets we used two algorithms. The first one generated random matrices by simply putting 0s and 1s into each matrix position with $1/2$ probability (if the resulted matrix was hv -convex and 4-connected then we omitted it). The other one was a uniform generator of [6] for generating hv -convex 4-connected or 8-connected sets. Elements of the class \mathcal{HV}_p for a given $p > 0$ were generated by generating an hv -convex polyomino using a uniform random distribution and then by applying the procedure discussed in the definition of this class.

In our experiments on decision trees – and also nearest neighbor approaches – we used 1800 training and 600 test samples for all sizes of matrices. For the classification of hv -convex polyominoes and random matrices (Section 4.1) and almost hv -convex polyominoes (Section 4.2) we found that the neural network has the best performance if we train it on 2880 examples (to keep the ratio of the train and test examples constantly 3:1 we used 960 samples for testing). In our latest experiment (Section 4.3) we used the same size and ratio for the train and test datasets for neural networks as for the decision trees. While we generated 10 separate datasets for each studied problem and we calculated the final results as the average of the

```

h10 <= 2 :
| | v1 <= 3 :
| | | h1 <= 4 : hv-convex
| | | h1 > 4 :
| | | | v10 <= 2 : hv-convex
| | | | v10 > 2 :
| | | | | h4 <= 6 : random
| | | | | h4 > 6 : hv-convex
| | v1 > 3 :
| | | v10 <= 2 : hv-convex
| | | v10 > 2 :
| | | | h5 <= 7 : random
| | | | h5 > 7 : hv-convex
h10 > 2 :
| | v10 <= 2 :
| | | h1 <= 3 :
| | | | v1 <= 2 : hv-convex
| | | | v1 > 2 :
| | | | | v5 <= 4 : random
| | | | | v5 > 4 :
| | | | | | v3 > 4 : hv-convex
| | | | | | v3 <= 4 :
| | | | | | | h1 <= 2 : hv-convex
| | | | | | | h1 > 2 : random
| | | | h1 > 3 :
| | | | | h6 > 7 : hv-convex
| | | | | h6 <= 7 :
| | | | | | v9 <= 2 : hv-convex
| | | | | | v9 > 2 :
| | | | | | | v1 <= 1 : hv-convex
| | | | | | | v1 > 1 : random
| | v10 > 2 :
| | | v7 <= 8 :
| | | | v1 <= 1 :
| | | | | h1 <= 2 : hv-convex
| | | | | h1 > 2 :
| | | | | | h10 <= 4 : hv-convex
| | | | | | h10 > 4 : random
| | | | | v1 > 1 :
| | | | | | h1 > 2 : random
| | | | | | h1 <= 2 :
| | | | | | | h6 > 7 : hv-convex
| | | | | | | h6 <= 7 :
| | | | | | | | h2 <= 2 : hv-convex
| | | | | | | | h2 > 2 :
| | | | | | | | | v10 > 3 : random
| | | | | | | | | v10 <= 3 :
| | | | | | | | | | h7 <= 5 : random
| | | | | | | | | | h7 > 5 : hv-convex
| | | v7 > 8 :
| | | | v6 <= 5 : random
| | | | v6 > 5 : hv-convex

```

Figure 5: A decision tree for classifying *hv*-convex polyominoes and random discrete sets of size 10×10 .

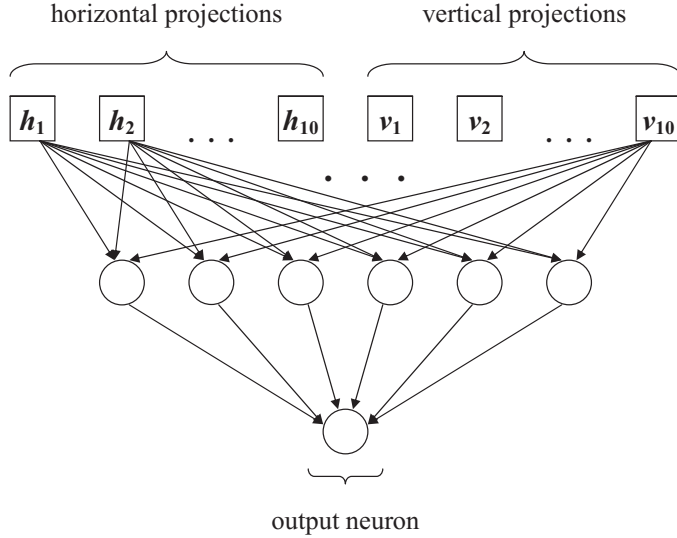


Figure 6: The neural network structure used for classifying matrices of size 10×10 .

classification errors on these datasets for the decision trees and the nearest neighbor approaches, we measured the ability of our neural networks on only one dataset for each examined problem. 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. In case of the neural networks this is easily obtained from the validation set accuracy as $100 - \text{VSA}$.

4.1 Classification of hv -Convex Polyominoes and Random Matrices

We started with a classification which was expected to be easy to learn: we tried to classify random matrices and hv -convex polyominoes. Figure 7 shows our experimental results. It clearly demonstrates that all learning methods performed the classification with an error less than 1% if the size of the discrete set was at least 30×30 . This means that – as it was expected – the class of hv -convex polyominoes and the class of random discrete sets can be very effectively separated using only the projection values by all classifiers studied. The explanation of this becomes obvious if we take a closer look at the expected horizontal and vertical projections of the sets classified. In the case of a randomly generated general discrete set these are $(n/2, \dots, n/2)$ and $(m/2, \dots, m/2)$, respectively. At the same time by simply calculating the averages we found that, e.g., in the case of hv -convex

polyominoes of size 10×10 these vectors are about $(2, 4, 5, 6, 7, 7, 6, 5, 4, 2)$ for both the horizontal and the vertical projections. Figure 5 – which represents a decision tree of this classification task for matrices of size 10×10 – also justifies the explanation above by showing that the test of the variables h_1 , v_1 , h_{10} , and v_{10} are close to the root of the tree, which is usually followed by the test of variables h_5 , v_5 , h_6 , and v_6 showing that these projection components have the most important role in the decision.

Table 1 presents our results with neural networks. The column of VSA shows the accuracy measured on the validation set, and the last column contains the error percentages, the latter values are displayed in Fig. 7. The learning rate (α) and the momentum (β) were set to a fix value of 0.001 and 0.9, respectively, in all test cases. This was due to the fact that we did not need to change α to get decent results and actually in no case it proved to be useful to change β from 0.9 to any other value so we kept it as 0.9 throughout all our experiments with neural networks in this paper. We experienced strange behavior when not allocating enough hidden neurons, e.g. for matrices of size 100×100 the network with 6 hidden units crashed down in performance in most cases with 0% of correctly classified patterns, however with 8 hidden units the VSA turned out to be 100% all the time. This behavior was not only the consequence of the difference in the number of units in the hidden layer but the consequence of the randomly initialized weights as well.

The results of the test on the nearest neighbor methods are also presented in Fig. 7. It shows that the error of the 1-nn algorithm is less than 1% for all sizes of matrices, which means that 99% of the instances have a close neighbor from their own class. Comparing the charts it’s clear-cut that all three methods performed well on this task, and for smaller matrices the knn algorithm has given slightly better results.

4.2 Almost hv -Convex Polyominoes

In this section we attempted to separate hv -convex polyominoes and \mathcal{HV}_4 discrete sets, due to the fact that the previous task came out relatively easy for all three methods. Our previous research [4] in this area made us believe that the \mathcal{HV}_4 class is a convenient choice for comparing these classifying methods. It is to be noted that even for a fixed p the modified positions increase with the size of the matrices.

Figure 8 shows the experimental results. This classification task turned out to be a much harder one for neural networks, actually decision trees proved to be the better solution in every aspect for this problem as they give relatively good results for bigger matrices, although for the smaller ones the classification still seems to be a hard task.

If we take a look at the neural network configurations in Table 2, we might notice significant differences in the pattern compared to Table 1. In

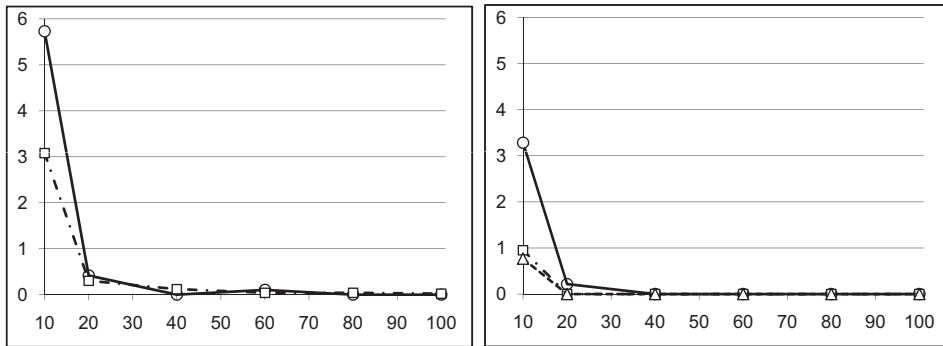


Figure 7: Average error of classification of hv -convex and random matrices in percent (vertical axis) as it depends on the size of the matrix (horizontal axis, only one dimension of the size of the squared matrices is given) by using decision trees (□) and neural networks (○) (left), and by modified k-means (○), 1-nn (□), and 7-nn (△) (right).

Table 1: The parameters of the neural network and the precision of the classification for the datasets of hv -convex polyominoes and random matrices. In all test cases $\alpha = 10^{-3}$, $\beta = 0.9$ and we run 100 epochs on the training dataset. For the size of the squared matrices only one dimension is given.

Size	Nr. Train	Nr. Test	Nr. Hidden	VSA	Error
10	2880	960	4	94.271	5.729
20	2880	960	6	99.583	0.417
40	2880	960	8	100	0.0
60	2880	960	8	99.792	0.108
80	2880	960	8	100	0.0
100	2880	960	8	100	0.0

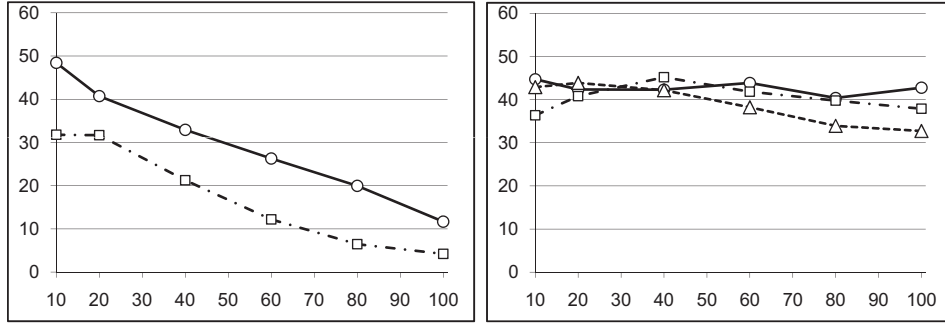


Figure 8: Average error of classification of hv -convex discrete sets and sets of \mathcal{HV}_4 in percent (vertical axis, only one dimension of the size of the squared matrices is given.) on the test data set as it depends on the size of the matrix (horizontal axis) by using decision trees (\square) and neural networks (\circ) (left), and by modified k-means (\circ), 1-nn (\square), and 7-nn (\triangle) (right).

some of the test cases we had to lower the value of α to get more accurate results. This way we attempted to "fine-tune" the network, which means smaller and smaller adjustments in the weights as the learning rate gets closer to zero. A single arrow in the field indicates a one step change in the value of α , while two arrows with three dots in between implies that the value has been changed in several steps. During the training of the networks a totally different approach had to be taken than in Section 4.1. The training process became a lot more interactive, the network itself had to be reconfigured in the parameters several times for relatively short time frames. This meant saving the weights, modifying the parameters, loading the weights back in the memory then resuming training, in this order. In practice, the whole training was a sequence of the previously defined steps. Revisions of the parameters were based on the TSA, GSA and MSE values (see Section 3.2) presented on-screen.

Nearest neighbor methods performed terrible on this problem, the errors of the classification are at almost 50%. These bad results might be caused by the base of these approaches, notably because the possible distance between a hv -convex and a \mathcal{HV}_p discrete set differs from the distance of two hv -convex discrete sets only in the modified elements – $p\%$ of the pixels which form the binary image.

4.3 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 [3]. For this to happen we need a method to decide which of

Table 2: The parameters of the neural network and the precision of the classification for the datasets of hv -convex polyominoes and \mathcal{HV}_4 discrete sets. In all test cases $\beta = 0.9$ and the number of train and test samples were 2880 and 960, respectively. For the size of the squared matrices only one dimension is given.

Size	Epoch	Nr. Hidden	α	VSA	Error
10	20000	26	10^{-3}	48.125	51.815
10	30000	30	10^{-3}	51.5625	48.4375
10	40000	40	$10^{-3} \rightarrow \dots \rightarrow 1.25 \cdot 10^{-4}$	51.1458	48.8542
20	30000	40	10^{-3}	59.2708	40.7202
20	25000	50	10^{-3}	58.3333	41.6667
40	3000	100	10^{-4}	65.3125	34.6875
40	3000	120	10^{-4}	67.0833	32.9167
60	2500	100	$10^{-4} \rightarrow 5 \cdot 10^{-5}$	73.7152	26.2848
60	2500	120	$10^{-4} \rightarrow 5 \cdot 10^{-5}$	73.2291	26.7709
80	2500	120	$10^{-4} \rightarrow 5 \cdot 10^{-5}$	80.0347	19.9653
80	2500	160	$10^{-4} \rightarrow 5 \cdot 10^{-5}$	79.9306	20.0694
100	2000	175	$5 \cdot 10^{-5} \rightarrow 10^{-5}$	88.3333	11.6667

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 [6] and discarding the 8-connected sets which were also 4-connected from the bunch.

In our first experiments on decision trees for this task each attribute of the samples were represented by a single projection component. We found that this approach did not produce as good results (see Fig. 9) as we expected, therefore additional experiments were required. First we tried to reduce the amount of the input attributes by summing the projection values. In the first experiment we summed 15-15 projection values and we used overlapping blocks as well in the following way: each composed attribute (except the first and the last) contained the last 7 projection values from the earlier and the first 8 from the following ones. The second case we summed 30-30 values with 15-15 overlaps. In case the size of the block for grouping the projection components together is greater in than the number of available projection components then we simply sum these values together. The results are presented in Fig. 10. Comparing the left hand side charts from Figs. 9 and 10 it is obvious that grouping the projection elements together provides better classification results than simply using single projections, for all test cases.

Finally we used projections from four directions for the classification of hv -convex polyominoes and hv -convex 8- but not 4-connected discrete sets. Two of these directions are the previously used horizontal and vertical, and the others are the two diagonal ones. The average error of the classification in these tests were in between the grouped and the not grouped tests for the matrices in the size from 40×40 to 190×190 . The results are shown by Fig. 10. It can be seen on the graph that smaller matrices gave fewer errors in the classification than in the earlier experiments.

After experimenting with decision trees we focused on neural networks. 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 Section 4.2. This feature, and the fact that this problem suited neural networks well, resulted in high VSA values, as seen in Table 3.

Figure 9 shows the results of our nearest neighbor tests. The 7-nn algorithm provides the best results, the classification error is less than the error of decision trees with grouped projection components, and even less than the errors measured on neural networks. Although this algorithm gives the

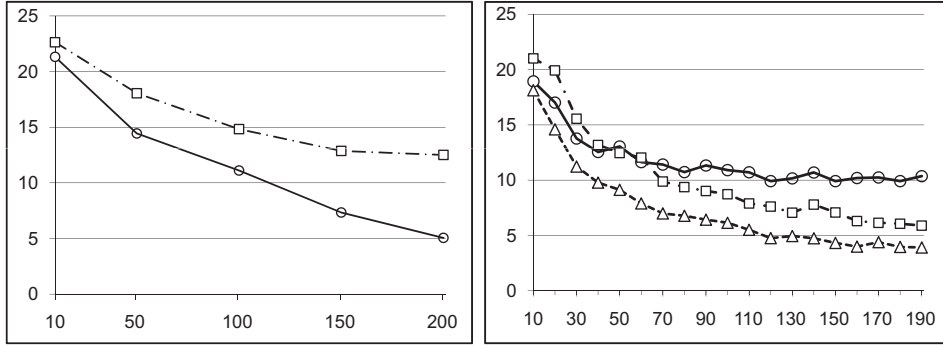


Figure 9: Average error of classification of hv -convex polyominoes and hv -convex 8- but not 4-connected hv -convex discrete sets in percent (vertical axis) as it depends on the size of the matrix (horizontal axis, only one dimension of the size of the squared matrices is given) by using decision trees (\square) and neural networks (\circ) (left), and by modified k-means (\circ), 1-nn (\square), and 7-nn (\triangle) (right).

best results it is not practical to choose this method to classify hv -convex polyominoes and hv -convex 8- but not 4-connected discrete sets because the necessary time to evaluate one sample is much longer than in the case of the other investigated methods.

5 Conclusions

This paper collects the first results on using learning methods to acquire geometrical information of discrete sets from their projections without reconstructing them. Such knowledge might be especially useful in discrete tomography. As a preprocessing step, the methods presented here, can be applied to make prior decisions on which reconstruction algorithm to choose, and how to parameterize it. Depending on the property of the set we want to determine, decision trees and/or neural networks generally seem to be a good choice for this task. Nearest neighbor approaches perform similarly good on certain tasks but in some cases their classification error is unacceptable from the viewpoint of real applications.

The learning procedure is the most time consuming for the neural networks. For simple tasks, like those of Section 4.1 it took a few seconds long, but for more complex classifications (problems of Sections 4.2 and 4.3) the convergence of the learner is much slower, yielding several hours or even more than one day of processing. Regarding decision trees and the modified k-means approach the learning part finished in less than a second, in each studied case. Recall, that knn methods do not have a learning part at all,

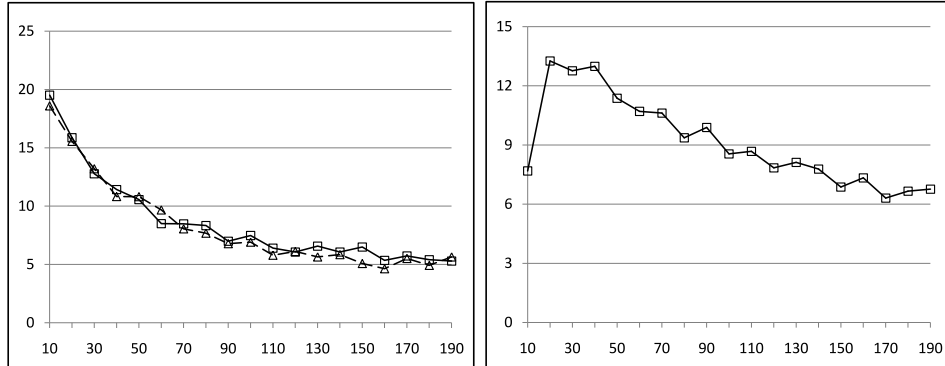


Figure 10: Average error of decision-tree classification of hv -convex polyominoes and hv -convex 8- but not 4-connected hv -convex discrete sets in percent (vertical axis) on the test data set as it depends on the size of the matrix (horizontal axis, only one dimension of the size of the squared matrices is given) when 15 (\square) and 30 (\triangle) overlapping projection components are grouped together (left), and classification by four projections (right).

Table 3: 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	$10^{-3} \rightarrow \dots \rightarrow 10^{-6}$	85.5556	14.4444
100	10000	200	$10^{-3} \rightarrow \dots \rightarrow 10^{-7}$	88.8889	11.1111
150	7500	250	$10^{-3} \rightarrow \dots \rightarrow 10^{-7}$	92.6667	7.3333
200	5000	300	$10^{-3} \rightarrow \dots \rightarrow 10^{-7}$	94.9444	5.0556

as they are simple instance-based learners. The testing was always very fast (a few milliseconds long), except in the case of knn methods where it took few seconds.

In all of our experiments we supposed that the projection data is noiseless. However, in applications of binary tomography the projections are usually affected by noise. Thus, in our future work we intend to conduct experiments on classification from noisy projections. In addition, we plan to go on and compare the results obtained here to alternative feature selection and classification schemes as well.

With the aid of certain classification methods (e.g., decision trees) we also can investigate which are the most important projection directions or projection components in the description of certain geometrical properties. Observations of this kind might yield mathematically well-formed projection-based characterization results. With the design of reconstruction algorithms that can directly use the information gained from the attributes (projections) by applying the techniques studied, we think that it will be possible to increase the speed and accuracy, as well as robustness of the reconstruction. Our preliminary studies on this field seem to be promising [5].

Acknowledgments

This research was partially supported by the TÁMOP-4.2.2/08/1/2008-0008 program of the Hungarian National Development Agency. The work of the third author was also supported by the János Bolyai Research Scholarship of the Hungarian Academy of Sciences.

References

- [1] Al-Harbi, S.H. and Rayward-Smith, V. J. Adapting k-means for supervised clustering. *Appl. Intell.*, 24:219-226, 2006.
- [2] 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/>
- [3] Balázs, P., Balogh, E. and Kuba, A. Reconstruction of 8-connected but not 4-connected $h\nu$ -convex discrete sets. *Disc. Appl. Math.*, 147:149–168, 2005.
- [4] Balázs, P. and Gara, M. Decision trees in binary tomography for supporting the reconstruction of $h\nu$ -convex connected images. *Lecture Notes in Comput. Sci.*, 5259:433–443, 2008.

- [5] Balázs, P. and Gara, M. An evolutionary approach for object-based image reconstruction using learnt priors. *Lecture Notes in Comput. Sci.*, 5575:520–529, 2009.
- [6] Balogh, E., Kuba, A., Dévényi, Cs. and Del Lungo, A. Comparison of algorithms for reconstructing hv -convex discrete sets. *Lin. Alg. Appl.*, 339:23–35, 2001.
- [7] Barcucci, E., Del Lungo, A., Nivat, M. and Pinzani, R. Reconstructing convex polyominoes from horizontal and vertical projections. *Theor. Comput. Sci.*, 155:321–347, 1996.
- [8] Batenburg, K.J. and Kusters, W.A. A neural network approach to real-time discrete tomography. *Lecture Notes in Comput. Sci.*, 4040:389–403, 2006.
- [9] Baumann, J., Kiss, Z., Krimmel, S., Kuba, A., Nagy, A., Rodek, L., Schillinger, B. and Stephan, J. Discrete Tomography Methods for Non-destructive Testing. In [14], pp. 303–332 (2007).
- [10] Brunetti, S. and Daurat, A. An algorithm reconstructing convex lattice sets, *Theor. Comput. Sci.*, 304:35–57, 2003.
- [11] Brunetti, S., Del Lungo, A., Del Ristoro, F., Kuba, A. and Nivat, M. Reconstruction of 4- and 8-connected convex discrete sets from row and column projections. *Lin. Alg. Appl.*, 339:37–57, 2001.
- [12] Chrobak, M. and Dürr, Ch. Reconstructing hv -convex polyominoes from orthogonal projections. *Inform. Process. Lett.*, 69(6):283–289, 1999.
- [13] Herman, G.T. and Kuba, A. (eds.) *Discrete Tomography: Foundations, Algorithms and Applications*. Birkhäuser, Boston (1999).
- [14] Herman, G.T. and Kuba, A. (eds.) *Advances in Discrete Tomography and its Applications*. Birkhäuser, Boston (2007).
- [15] Kak, A.C. and Slaney, M. *Principles of Computerized Tomographic Imaging.*, IEEE Press, New York (1988).
- [16] Kisielowski, C., Schwander, P., Baumann, F.H., Seibt, M., Kim, Y. and Ourmazd, A. An approach to quantitative high-resolution electron microscopy of crystalline materials. *Ultramicroscopy*, 58:131–155, 1995.
- [17] Kim, S., Aggarwal, C.C., Yu, P.S., Effective nearest neighbor indexing with the Euclidean metric. *Proceedings of the Tenth International Conference on Information and Knowledge Management, Atlanta, Georgia, USA*, 9–16, 2001.

- [18] Kuba, A. The reconstruction of two-directionally connected binary patterns from their two orthogonal projections. *Comp. Vision, Graphics, and Image Proc.*, 27:249–265, 1984.
- [19] Kuba, A. Reconstruction in different classes of 2D discrete sets. *Lecture Notes in Comput. Sci.*, 1568:153–163, 1999.
- [20] Kuba, A. and Balogh, E. Reconstruction of convex 2D discrete sets in polynomial time. *Theor. Comput. Sci.*, 283:223–242, 2002.
- [21] Mitchell, T.M. *Machine Learning*. McGraw Hill (1997).
- [22] Onnasch, D.G.W. and Prause G.M.P. Heart chamber reconstruction from biplane angiography. In [13], pp. 385–403 (1999).
- [23] Russel, S. and Norvig, P. *Artificial Intelligence: A Modern Approach* (2nd Edition). Prentice Hall Series in Artificial Intelligence (2002).
- [24] Schüle, T., Schnörr, C., Weber, S. and Hornegger, J. Discrete tomography by convex-concave regularization and D.C. programming. *Disc. Appl. Math.*, 151:229–243, 2005.
- [25] Somol, P., Pudil, P., Nonvičová, J. and Paclík, P. Adaptive floating search methods in feature selection. *Pattern Recog. Lett.*, 20:1157–1163, 1999.
- [26] Quinlan, J.R. *C4.5: Programs for Machine Learning*. Morgan Kaufmann (1993).