In Proceedings of Joint 13th International Conference on Artificial Neural Networks and 10th International Conference on Neural Information Processing (ICANN/ICONIP 2003) – Istanbul, Turkey, pp. 306–309, June 2003.

Learning Iterative Binarization using Hierarchical Recurrent Networks

Sven Behnke
International Computer Science Institute
1947 Center St., Berkeley, CA, 94704, USA
behnke@icsi.berkeley.edu, www.icsi.berkeley.edu/∼behnke

***Abstract*— In this paper the binarization of matrix codes is investigated as an application of supervised learning of image processing tasks using a recurrent version of the Neural Abstraction Pyramid.**

The desired network output is computed using an adaptive thresholding method for undegraded images. The network is trained to iteratively produce the same output even when the contrast is lowered and typical noise is added to the input. The network discovers the structure of the codes and uses it for binarization. This makes the recognition of degraded matrix codes possible for which adaptive thresholding fails.

I. Introduction

Iterative image processing techniques have proven to be useful for many tasks, including registration [8] and image restoration [7]. However, they mainly used flat image models with relatively few parameters, offering limited adaptability.

Neural networks are a popular tool for image processing tasks [6]. They are used for feature extraction and pattern recognition and to transform one image into another. When training a network for such a task, e.g. for binarization, the most costly part of the process is usually to obtain the desired network outputs. The reason for this is that human experts are needed to label the images.

To avoid such costs, I use here an existing algorithm that suffices to process the not so difficult examples. In addition, I make the same examples more difficult by degrading image quality with typical noise. I train a hierarchical neural network with local recurrent connectivity to reproduce the outputs not only for the undegraded images, but for the degraded ones as well. This is an efficient way to extend the domain of the network beyond the one of the existing algorithm.

A similar approach to binarization has been recently proposed by Wolf and Doermann [11]. They used flat Markov random fields with cliques of 4×4 pixels to model the distribution of text images and employed simulated annealing for inference. Image degradation was done by JPEG compression.

The paper is organized as follows: The next section introduces the data set, the algorithm for generating desired outputs, as well as image degradation. Section III describes the Neural Abstraction Pyramid architecture and the supervised learning algorithm used. Finally, Section IV presents some experimental results.

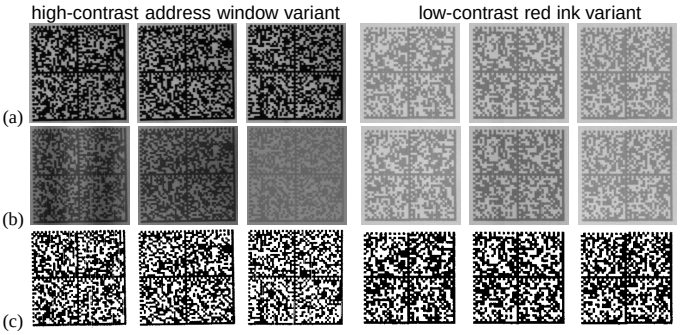

Fig. 1. Canada Post Data Matrix codes: (a) original images; (b) degraded images; (c) output of adaptive thresholding for degraded images.

Fig. 2. Problems for binarization: (a) non-uniform lighting (with estimated background); (b) vertical bright and dark lines; (c) high noise.

II. Matrix Code Data Set

Two-dimensional codes are an extension to one-dimensional bar codes. This paper deals with the binarization of Data Matrix [1] codes. The code matrix contains dark and light square data modules. It has a finder pattern of solid and dotted lines. Data Matrix is designed with a fixed level of error correction. Recognition is possible if less than one quarter of the bits have been destroyed.

I used a database provided by Siemens ElectroCom. Gray-scale images of size 216×216 show Data Matrix codes as used by Canada Post. The matrix encodes the meter value, the date of sending, the sender, and the addressee. There are two code variants, shown in Fig. 1(a): 515 images contain a high-contrast code visible through a letter's address window and 694 images show a low-contrast code that is printed with red ink in the upper right corner of a letter.

Reading of the code requires to localize the symbol, to binarize it, to locate finder patterns, to read the bits, to correct for errors, and to validate the result. Here, the focus is on the binarization step only. Because of noise, inhomogeneous lighting, printing errors, and low-image contrast this problem is challenging.

To produce the desired output for training the binarization network, first the background intensity is estimated for each

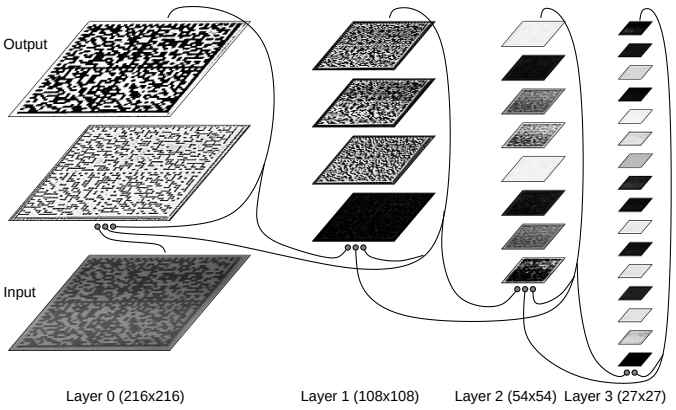

Fig. 3. Architecture of the network used for learning the binarization of Data Matrix codes. The resolution of the layers decreases as the number of feature arrays increases.

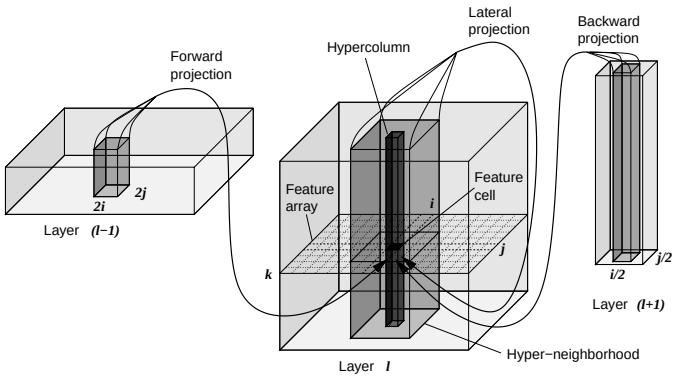

Fig. 4. A feature cell with its projections.

location of the code image and used to correct for the non-uniform lighting (see Fig. 2(a)). Next, the intensity histogram is computed and smoothed, until only one local minimum is left. Finally, the contrast of the image is stretched linearly around the threshold. As can be see in Fig. 1(c), most pixels saturate to black and white. Only some pixels at module borders have intermediate intensities, indicating uncertainty.

Closer inspection of the contrast stretched low-contrast images reveals some problems, illustrated in Fig. 2(b,c). Most problematic outputs are either due to printing errors (vertical dark or bright lines) or due to noise caused by the paper of the envelope. To address the problematic vertical lines, a horizontal low-pass filter is applied to the low-contrast images prior to thresholding.

To make the codes more difficult to read, the images are degraded in a way that induces typical binarization problems. Degradation is done by adding vertical dark and bright lines, adding a smoothly varying background level, lowering contrast, and adding pixel noise. Degradation for the low-contrast code variant is less severe than for the high-contrast variant. Fig. 1(b) shows some degraded images. Details of adaptive thresholding and the degradation can be found in [3].

III. Network Architecture and Training

If one wants to develop a binarization method that outperforms adaptive thresholding, one has to utilize the structure present in the data. More specifically, one can expect a method to perform well that recognizes the Data Matrix modules and assigns white or black to an entire module, and not to single pixels. Of course, one could develop manually an algorithm that works in this way, but I will demonstrate that it is possible to solve the problem without the need to think about an application-specific algorithm. The approach followed is to use a general-purpose tool, the Neural Abstraction Pyramid [5], [3], and to adapt it to the specific task by learning from input-output examples. This architecture has been applied successfully to image reconstruction [2] and face localization [4].

The architecture of the binarization network is sketched in Figure 3. It has four layers with an increasing number of feature arrays and a decreasing resolution. Layer 0 contains the input image and two additional feature arrays of size 216×216. The number of feature arrays doubles, while their resolution is halved when going to the next layer, until Layer 3 is reached, where 16 feature arrays of size 27×27 are present. A two pixel wide border surrounds the feature arrays. The activities of the border cells are copied from feature cells using wrap-around.

Figure 4 magnifies one layer l of the pyramid. All feature cells k that share the same location (i, j) within a layer form a hypercolumn. A hypercolumn describes all aspects of the corresponding image window in a distributed representation. Neighboring hypercolumns define a hyper-neighborhood.

There is a local recurrent connectivity. Three types of projections are used:

- **Forward** projections originate in the hyper-neighborhood at the corresponding position in the next lower layer and are used for feature extraction.
- **Lateral** projections stay within a layer. They mediate competition and cooperation within a hyper-neighborhood and make the features consistent.
- **Backward** projections come from the corresponding hyper-neighborhood of the next higher layer. They expand abstract features to less abstract ones.

The feature cells in the Neural Abstraction Pyramid contain simple processing elements that make a single value, the activity, available to other cells. The activity of a cell represents the strength of the associated feature at a certain position. It is accessed via weighted links. The update of the activities proceeds layer by layer in a bottom-up manner. The activity $a^t_{ijkl} \in \mathbb{R}$ of a feature cell $(ijkl)$ at time t is computed as follows:

$$a^t_{ijkl} = \sigma(\sum_{p=1}^{P_{kl}} b^{tp}_{ijkl} \; + \; w^0_{kl}); \qquad b^{tp}_{ijkl} = \sum_{q=1}^{Q^p_{kl}} w^{pq}_{kl} \; a^{t^*}_{i^*j^*k^*l^*}. \tag{1}$$

The cell computes a sum of its projection potentials $b^{tp}_{ijkl} \in \mathbb{R}$ and a bias value w^0_{kl} that it is passed through the sigmoidal transfer function $\sigma = 1/(1+e^{-x})$. The use of such a nonlinear function is crucial for the stability of the network dynamics. When the activity of a cell is driven towards saturation, the effective gain of the transfer function is reduced considerably. This avoids the explosion of activity in the network. Furthermore, the nonlinearity is needed to make decisions. Each

projection computes a weighted sum of activities $a^{t^*}_{i^*j^*k^*l^*}$ with the weighting factors described by $w^{pq}_{kl} \in \mathbb{R}$.

Forward projections come from 4×4 windows of all feature arrays in the layer below. Lateral projections originate from the 5×5 hyper-neighborhood in the same layer and backward projections access a single cell of all feature arrays in the layer above. See [3] for a more formal description.

Weights are described by templates that are shared by all cells of a feature array. For backward projections there are four different template versions, corresponding to the four positions that are mapped to a position in the next higher layer. The network has a total of 11,910 different weights. Most of them are located in the top layer, since the weights in the lower layers are shared more often than the ones in the higher layers.

One important idea of the architecture is that each layer maintains a complete image representation in an array of hypercolumns. The degree of abstraction of these representations increases with height. At the bottom of the pyramid, features correspond to local measurements of a signal, the image intensity. Subsymbolic representations, like the responses of edge detectors or the activities of complex cells are present in the middle layers of the network. When moving upwards, the feature cells respond to image windows of increasing size, represent features of greater complexity, and are less variant to image deformations.

The Neural Abstraction Pyramid has been designed for the iterative interpretation of images. The refinement of initial image interpretations through local recurrent interactions of simple processing elements that are arranged in a hierarchy is the central idea of the architecture. Such a refinement is needed to resolve ambiguities. The interpretation of ambiguous stimuli is postponed until reliably detected features are available as context. Horizontal and vertical feedback loops allow contextual influences between neighboring image locations and between adjacent layers, respectively. Information flow is asymmetric: reliable features bias the unreliable ones. Iterative image interpretation has the features of an anytime algorithm. Usable partial interpretations are available very early. They are completed as the processing proceeds.

The 515 high / 694 low - contrast images were partitioned randomly into 334/467 training images and 181/227 test examples. For each example, one degraded version was added to the sets. The undegraded Data Matrix images as well as their degraded versions are presented to the network without any preprocessing. One of the feature arrays in the bottom layer is used as network output. The target values that are used as desired output for the supervised training are computed using the adaptive thresholding method for the undegraded images. The network is trained to iteratively produce them not only for the original images, but for the degraded versions of these images as well.

Two networks are randomly initialized and trained for ten iterations with a linearly increasing weight on the squared error using backpropagation through time [10] and RPROP [9]. Such a combination has been found to allow for fast and stable supervised learning in hierarchical recurrent networks [3].

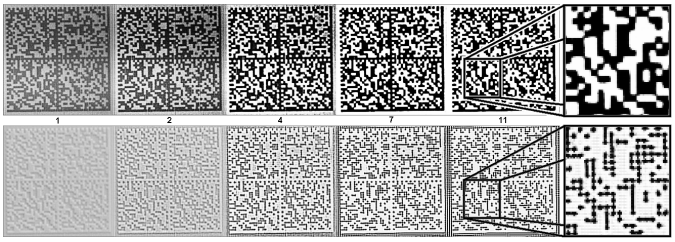

Fig. 5. Recall of network trained for binarization of Data Matrix codes. The development of the feature activities at Layer 0 is shown for one of the degraded test examples.

IV. Experimental Results

After training, the networks are able to iteratively solve the binarization task. The lower layers represent the cell structure of the code, while the higher layers represent the background level and the local black-and-white ratio.

The network performs an iterative refinement of an initial solution, with most changes occurring in the first few iterations. The activities of the two Layer 0 feature arrays are displayed in Fig. 5. In the first iterations, the non-uniform background level causes the upper part of the code to have higher activity than the lower part. This inhomogeneity is removed during refinement. Furthermore, the output is driven from intermediate gray values that signal uncertainty towards black and white, as typical for the desired output.

In the hidden feature array a representation of the code structure emerges. Bright areas of the input image are broken into a discrete number of modules. For each bright module an activity blob rises and remains stable. Adjacent blobs are connected either vertically or horizontally, depending on the prominent local orientation of the corresponding bright area. If such a local orientation cannot be determined, e.g. in bright areas that have a larger width as well as a larger height, the blobs form a loosely connected matrix. The blobs inhibit the output feature cells. Hence, the network has learned that the output of a code module must be coherent. This suppresses thin vertical lines and pixel noise.

To understand the emergence of the blobs, one can analyze the contributions made by individual projections. Weak input projections detect contrast at the upper and right border of a bright area. The contributions of lateral projections shape the blobs through a center-center excitation and a center-surround inhibition. Here, the typical blob distance of about four pixels is enforced. Finally, the backward projections excite or inhibit entire areas, not discrete blobs. Thus, at Layer 1 a coarser representation of black and white areas must exist.

Figure 6(a) displays the average squared output changes over time. In the first iterations, the output changes considerably. The changes decrease quickly until iteration 10 and remain low afterwards. Thus, the network dynamics is stable even when iterating twice as long as it has been trained for. In the figure one can also observe that the average error decreases rapidly during the first iterations. It reaches a minimum at about iteration 8 and increases again slowly. Hence, the network's attractors are not identical to the desired outputs.

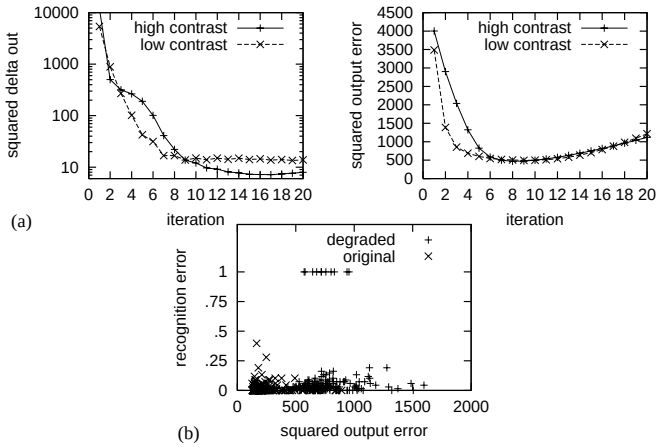

Fig. 6. Recall of iterative binarization: (a) performance over time; (b) recognition error vs. squared output error for high-contrast examples.

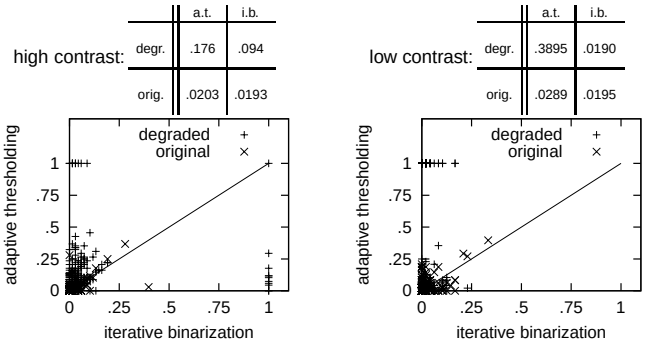

Fig. 7. Data Matrix test set recognition error distributions and averages.

This is not surprising, since the network has been trained to produce the desired output only for ten iterations. When iterated further, the dynamics evolves into stable attractors that resemble the module structure of the Data Matrix codes. If one wanted to produce a longer decrease of the average difference to the desired output, one could always train the network for more iterations. This has not been done here, since ten iterations seem to be sufficient to solve the binarization task.

For all examples, the output of adaptive thresholding as well as the one of iterative binarization has been presented to a Data Matrix recognition engine. For the low-contrast codes, the engine was queried a second time with different parameters when an example was rejected in the first run. The engine produces for each recognized example the percentage of error correction used. This value will be referred to as recognition error. It is set to one if an example could not be recognized at all. The desired outputs are not necessarily the ideal outputs, but only approximations to the best recognizable ones. This can be observed in Fig. 6(b). Thus, a deviation from the desired outputs does not necessarily indicate a decrease in output quality, as measured in recognition performance.

In Fig. 7 the recognition error of adaptive thresholding and the iterative binarization method is compared for the test sets. The performance of the two methods is not much different for the original images, since the networks have been trained to resemble the behavior of adaptive thresholding for such images. In contrast, for most degraded images the recognition error is lower when using the iterative binarization method than when adaptive thresholding is used. Here, the use of iterative binarization substantially lowers the need for error correction. This yields higher recognition rates. While 37% of the degraded low-contrast codes binarized with adaptive thresholding could not be recognized, all of these images could be recognized when binarized iteratively. This considerably extends the applicability of matrix code recognition systems to low-quality images.

V. Conclusions

It was shown that a non-trivial image processing task can be learned by an instantiation of the Neural Abstraction Pyramid architecture. An adaptive thresholding method was developed that is able to successfully binarize high-contrast images of Data Matrix codes. Its results were used as desired output for a Neural Abstraction Pyramid that was trained to iteratively produce them not only for the original images, but also for degraded versions of them.

The network learns to recognize the module structure of the Data Matrix and to use it for binarization. The performance of both methods was evaluated using a recognition system. For the high contrast code variant the trained network performs as well as adaptive thresholding for the original images, but outperforms it for degraded images. For the low-contrast red ink variant of the Data Matrix codes, the advantage of iterative binarization is more obvious. It performs better than adaptive thresholding for the original images and outperforms it dramatically for the degraded images.

Future work could apply the proposed approach to more complex image processing problems, such as the segmentation of natural scenes.

References

[1] AIM, Inc. ISS Data Matrix. BC11, ANSI/AIM, 1995.

[2] Sven Behnke. Learning iterative image reconstruction in the Neural Abstraction Pyramid. *International Journal of Computational Intelligence and Applications, Special Issue on Neural Networks at IJCAI-2001*, 1(4):427–438, 2001.

[3] Sven Behnke. Hierarchical neural networks for image interpretation. Dissertation thesis, Freie Universität Berlin, 2002.

[4] Sven Behnke. Learning face localization using hierarchical recurrent networks. In *Proceedings of ICANN 2002*, volume 2415 of *LNCS*, pages 1319–1324, 2002.

[5] Sven Behnke and Raúl Rojas. Neural Abstraction Pyramid: A hierarchical image understanding architecture. In *Proceedings IJCNN'98*, pages 820–825, 1998.

[6] Michael Egmont-Petersen, Dick de Ridder, and Heinz Handels. Image processing with neural networks - a review. *Pattern Recognition*, 35(10):2279–2301, 2002.

[7] D. Geman and S. Geman. Stochastic relaxation, Gibbs distributions, and Bayesian restoration of images. *IEEE Transactions on PAMI*, 6(6):721–741, 1984.

[8] B. D. Lucas and T. Kanade. An iterative image registration technique with an application in stereo vision. In *Proceedings of IJCAI'81*, pages 674–679, 1981.

[9] M. Riedmiller and H. Braun. A direct adaptive method for faster backpropagation learning: The RPROP algorithm. In *Proceedings of ICNN'93*, pages 586–591, 1993.

[10] Paul J. Werbos. Backpropagation through time: What it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560, 1990.

[11] C. Wolf and D. Doermann. Binarization of low quality text using a Markov random field model. In *Proceeding of ICPR 2002*, volume 2, pages 160–163, 2002.