Department of Computer Science,
University of Otago

Technical Report OUCS-2011-02

Performance of a convolutional classifier network on the MNIST handwritten digit database

Authors:

Hayden Walles, Anthony Robins and Alistair Knott
Department of Computer Science, University of Otago, New Zealand

---

Department of Computer Science,
University of Otago, PO Box 56, Dunedin, Otago, New Zealand

http://www.cs.otago.ac.nz/research/techreports.php

Performance of a convolutional classifier network on the MNIST handwritten digit database

Hayden Walles, Anthony Robins and Alistair Knott
Department of Computer Science, University of Otago

October 25, 2011

Abstract

This report describes an experiment in which the convolutional neural network of Walles *et al.* (2008) is trained and tested on the MNIST database of handwritten digits (LeCun et al, 1999).

1 Introduction

In this report, we describe an experiment using the visual object classifier described by Walles *et al.* (2008). The classifier is a convolutional neural network (CNN) similar to that described in Walles et al (2008), with a few refinements described here.

In Walles et al (2008) we showed that the convolutional neural network is capable of 'group classification' of simple shapes. We trained the network on simple shapes of various types (squares, ells, triangles, crosses and arrows) appearing individually at random positions on the retina. After this training, the network was able to successfully classify shapes appearing at arbitrary retinal locations—but more interestingly, it was also able to classify homogeneous *groups* of shapes (i.e. groups containing only shapes of one particular type). When presented with a 'heterogeneous' group, containing objects of more than one type, tended not to offer a classification at all. The network was therefore 'cardinality blind': it responded in the same way to a single X and to a group of Xs.

The training stimuli used in Walles et al (2008) were very simple. In the current paper we test the network's ability to classify more complex naturalistic shapes, both individually and in groups. The shapes we use are handwritten digits taken from the MNIST dataset (LeCun et al, 1999).

2 The classifier

The classifier used was a CNN taking as input a set of simple visual feature maps and activating as output a set of output category units via a series of layered plies which alternately combine visual

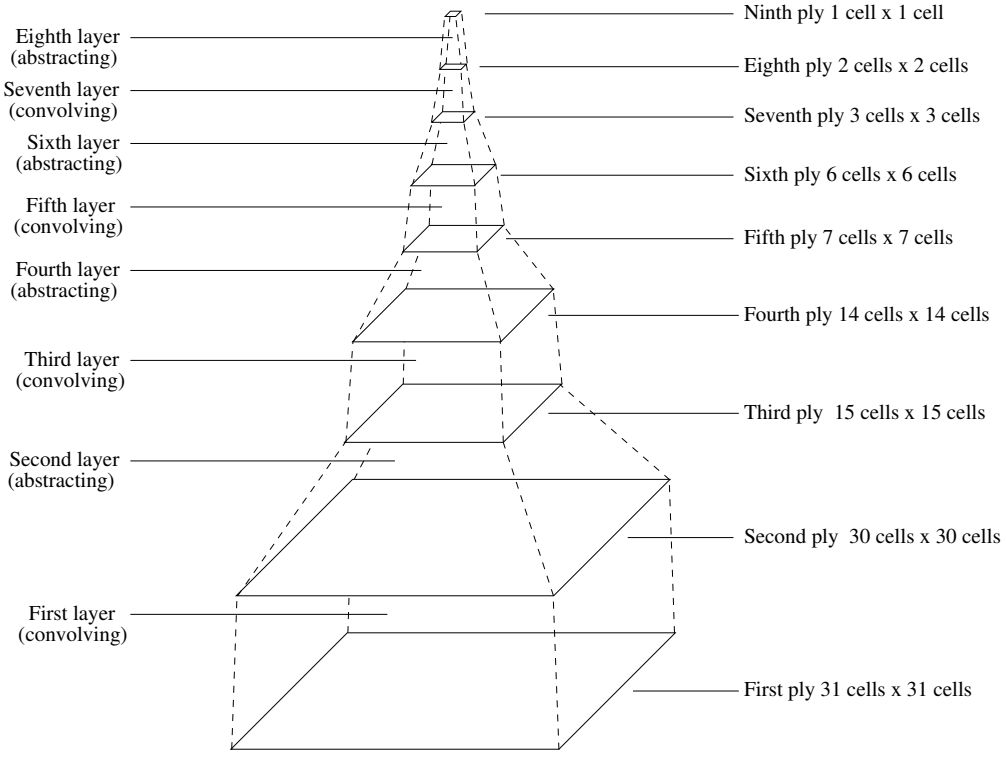

Figure 1 The general structure of our convolutional neural network. Plies of cells are connected by layers of weights. Each cell contains one unit for every feature represented by that ply.

features from the ply below into more complex features and abstract over the spatial location of visual features. The CNN was mostly as described in Walles et al (2008), except that the number of features used in each ply was different and there was some additional input preprocessing.

2.1 Classifier structure

Figure 1 illustrates the overall structure of our CNN.

The *units* of the network are arranged in a series of *plies*, with units in each ply connected to units in the one above by a *layer* of *weights*. Our network used had nine plies and eight layers. Units within each ply are clustered into *cells*, which are arranged retinotopically. Every cell in a particular ply contains the same number of units, one for each feature that the ply represents. Each unit in a cell represents the strength of its associated feature at the cell's location, and so each cell in a ply represents in parallel the presence of a set of features at the corresponding location in the input field. The successive plies of our network (going from input to output, measured in terms of cells) 31×31, 30×30, 15×15, 14×14, 7×7, 6×6, 3×3, 2×2 and 1×1.

The features in the first (input) ply were divided into two groups. One feature was provided for high-frequency input which represented luminance directly. Four features represented low-frequency input, each of which was obtained by convolving the input with one of the two filter matrices below or their 90° rotations. These centre-surround bar filters are intended to extract

low-frequency orientation information and remove high-frequency noise. They were arrived at by balancing the totals of the centre (positive) and surround (negative) elements and normalising so that each of these totals has magnitude one.

$$\frac{1}{180}\begin{bmatrix} -4 & -4 & 5 & 5 & 5 & 5 & -4 & -4 & -4 \\ -4 & -4 & 5 & 5 & 5 & 5 & -4 & -4 & -4 \\ -4 & -4 & 5 & 5 & 5 & 5 & -4 & -4 & -4 \\ -4 & -4 & 5 & 5 & 5 & 5 & -4 & -4 & -4 \\ -4 & -4 & 5 & 5 & 5 & 5 & -4 & -4 & -4 \\ -4 & -4 & 5 & 5 & 5 & 5 & -4 & -4 & -4 \\ -4 & -4 & 5 & 5 & 5 & 5 & -4 & -4 & -4 \\ -4 & -4 & 5 & 5 & 5 & 5 & -4 & -4 & -4 \\ -4 & -4 & 5 & 5 & 5 & 5 & -4 & -4 & -4 \end{bmatrix}$$

$$\frac{1}{1568}\begin{bmatrix} 49 & 49 & -32 & -32 & -32 & -32 & -32 & -32 & -32 \\ 49 & 49 & 49 & -32 & -32 & -32 & -32 & -32 & -32 \\ 49 & 49 & 49 & 49 & -32 & -32 & -32 & -32 & -32 \\ -32 & 49 & 49 & 49 & 49 & -32 & -32 & -32 & -32 \\ -32 & -32 & 49 & 49 & 49 & 49 & -32 & -32 & -32 \\ -32 & -32 & -32 & 49 & 49 & 49 & 49 & -32 & -32 \\ -32 & -32 & -32 & -32 & 49 & 49 & 49 & 49 & -32 \\ -32 & -32 & -32 & -32 & -32 & 49 & 49 & 49 & 49 \\ -32 & -32 & -32 & -32 & -32 & -32 & 49 & 49 & 49 \end{bmatrix}$$

We would like to have used similar filters for the high-frequency input, too, but this gives poor results because the high-frequency stimuli are so small. This is again a practical limitation forced on us by the size of the model's retina.

Units receive input from a small square region of the ply beneath, the *integration window*, meaning they are connected locally and can only make use of local features. we used a window measuring 2×2 cells in all layers. All units in a cell have the same window, which can thus also be called the cell's window. The region of the retina that contributes to a unit's input is its *receptive field*. In addition the weights for corresponding units in different cells of a ply are constrained to be the same, effectively sharing the weights. This means that the response to activity inside a cell's window will be the same irrespective of where in a ply the cell is located.

Successive plies divide the visual field more and more coarsely, so contain fewer cells than their predecessors, each of which has a wider receptive field than those in earlier plies. However later plies generally represent more features than earlier plies, and therefore contain more units per cell.

The function and structure of the weight layers alternates throughout the network between convolution and abstraction. Convolving layers compute combinations of features in the previous ply with little change in the number of cells between plies, while abstracting layers reduce the number of cells of the input ply without interaction between different features.

In convolving layers, an output unit receives input from every unit within its 2×2 integration window. A unit receiving input from a ply representing n features will have $4n + 1$ inputs

(including a bias).

Weights in abstracting layers are simpler. Input and output plies contain the same number of features and there is no interaction between features. A unit receiving input from its 2×2 window will have 5 inputs (including a bias). The window of a cell in the output ply precisely abuts but does not overlap with the windows of neighbouring cells. The effect is that the integration windows of cells in the output ply tile the input ply. Weights are shared even further within abstracting layers, with all weights for a feature constrained to be identical. This means that each abstracting layer really has only two variable parameters per feature: one weight shared among all the inputs units, and the bias.

Apart from the varying structure of the layers, unit activation is computed in the same way throughout the network. For a unit with n inputs $p_1 \ldots p_n$ (excluding the bias) and $n+1$ weights (including the bias) $w_1 \ldots w_{n+1}$ the unit's *activation*, a weighted sum, σ is computed:

$$\sigma = \sum_{i=1}^{n} p_i w_i + w_{n+1}$$

which for an abstracting unit can be simplified further to:

$$\sigma = w_{ply} \sum_{i=1}^{n} p_i + w_{bias}$$

because of weight sharing.

The output of the unit is then computed via the logistic function:

$$f = \frac{1}{1+e^{-\sigma}}$$

This is conventional for feed-forward networks.

Going from the input ply to the output ply the number of features in each ply were 5, 25, 25, 32, 32, 32, 32, 7 and 7.

Although inputs to the system as a whole measure 128×128 pixels, inputs to the classifier always measure 31x31 pixels as in our original design. This is a practical limitation of the classifier to allow training in reasonable time and the disparity is resolved by always centring the attended region in the classifier's input for classification purposes. This ensures that the bounding rectangle of the attended region is centred in the classifier's input.

2.2 Training algorithm

There are two components to the training – the algorithm used and the details of the training examples used. The algorithm is discussed in this section.

The network was trained using the RPROP algorithm (Riedmiller, 1994). This is a variation of the BACKPROP algorithm (Rumelhart et al, 1986). This algorithm works to minimise the squared error of each output unit:

$$\varepsilon = (d-f)^2$$

where d is the desired output and f the actual output.

During a single cycle (or batch or epoch) of training, each example is presented to the network, the discrepancy between the actual and desired output is registered in the units of the final ply and this is propagated backwards through the network. Once all of the examples have been presented, the error information stored throughout the network is used to update the weights.

Each time a pattern is presented, the value $\delta_{N,i}$ for unit i in the last ply of an N ply network is computed using

$$\delta_{N,i} = f_{N,i}(1 - f_{N,i})(d_{N,i} - f_{N,i})$$

where $d_{N,i}$ is the desired output of the unit and $f_{N,i}$ is the actual output of the unit. For units in earlier layers, the value $\delta_{l,i}$ for unit i in the lth ply which connects to n units in the next ply is computed recursively by

$$\delta_{l,i} = f_{l,i}(1 - f_{l,i}) \sum_{k=1}^{n} \delta_{l+1,k} w_{l,i}^{l+1,k}$$

where $w_{l,i}^{l+1,k}$ is the weight connecting the ith unit of ply l and the kth unit of ply $l+1$. The units summed over are precisely those that the unit in question connects to in the subsequent ply. The size and layout of this set differs depending on the location of the unit and what kind of layer (convolving or abstracting) connects the plies.

As the δ values are computed, error derivatives are also computed. After the presentation of a single pattern the partial derivative of each unit's error, ε with respect to each of its incoming weights is

$$\frac{\partial \varepsilon}{\partial w_{l,i}^{l+1,k}} = -2\delta_{l+1,k} f_{l,i}$$

which is an error gradient of the unit. It is associated with the weight $w_{l,i}^{l+1,k}$ and where weights are shared the different derivatives are summed.

The partial derivatives are accumulated over all of the training examples in the batch. At the end of the batch weight changes can be made.

RPROP requires that apart from its actual weight value, each weight has two values associated with it between batches. The first is the Δ value, which determines the size of weight changes. The other is the immediately previous value of the partial error derivative. The old partial derivative allows the algorithm to detect when the sign of the partial derivative has changed, indicating it has jumped over a local minimum. The Δ value is adjusted up or down to accelerate learning when big changes are possible, and slow it down when fine control is required.

For a given weight let $m^{(t)}$ indicate the value m at batch t. Then

$$\Delta^{(t)} = \begin{cases} \eta^{+}\Delta^{(t-1)} & \text{, if } \frac{\partial \varepsilon}{\partial w}^{(t-1)} \frac{\partial \varepsilon}{\partial w}^{(t)} > 0 \\ \eta^{-}\Delta^{(t-1)} & \text{, if } \frac{\partial \varepsilon}{\partial w}^{(t-1)} \frac{\partial \varepsilon}{\partial w}^{(t)} < 0 \\ \Delta^{(t-1)} & \text{otherwise} \end{cases}$$

where $0 < \eta^- < 1 < \eta^+$. Now the weight update $\Delta w^{(t)}$is

$$\Delta w^{(t)} = \begin{cases} -\Delta^{(t)} & \text{, if } \frac{\partial \varepsilon}{\partial w}^{(t)} > 0 \\ +\Delta^{(t)} & \text{, if } \frac{\partial \varepsilon}{\partial w}^{(t)} < 0 \\ 0 & \text{otherwise} \end{cases}$$

based on the Δ values computed earlier. There are two additional points to note. The Δ values are constrained to never exceed a constant value Δ_{max} and never go below a constant value Δ_{min}. This is performed by simple thresholding during the computation of the values. And in the event that the sign of $\frac{\partial \varepsilon}{\partial w}$ changes, there is no adaptation of the weight on the next batch, which is achieved by setting the recorded value of $\frac{\partial \varepsilon}{\partial w}$ to zero for the next batch. The partial derivatives are initialised to zero before training begins, and the Δ values to a value Δ_0.

The values of the parameters Δ_{min}, η^-, $\eta^+$and Δ_0 in our implementation were 1×10^{-6}, 0.5, 1.2 and 0.1 respectively. Δ_{max}had the value 0.08 except in the first two layers (at the input end) where it had the value 0.5.

3 Classification of Natural Images

3.1 Methods

The classifier was trained on handwritten digits from the MNIST database (described in Le Cun et al 1999, available for download at `http://yann.lecun.com/exdb/mnist/index.html`). The CNN structure and training regime was similar to that described for the artificial shapes in section §2, with differences noted where necessary.

The MNIST database consists of sixty thousand handwritten digit images in a training set and ten thousand images in a test set. All images are normalised to fill a 28×28 pixel area. Because our network is set up to classify figures around half that size, we scaled each image to 14×14 pixels by averaging 2×2 pixel blocks. This also allowed us to present several digits to the network simultaneously.

Since handwritten digits are more complex than simple shapes, we increased the number of features used in each ply of the network—except the first ply, which just contained one feature representing luminance, similar to the high-frequency inputs of the shape classifier. Larger numbers of features gave better performance, but also increased training times. The network described in this section used 1, 12, 12, 28, 28, 65, 65, 10 and 10 features per map location in successive plies, which allowed good performance on benchmark tests while keeping training times within manageable bounds.

Two kinds of training were performed: training with images placed in the centre of the retina for easier comparison to existing classifiers, and training with images placed at random locations.

For the centred training set, training images were selected from the MNIST training set as follows. We selected 5338 images at random made up of roughly equal numbers of each digit from 1 to 9, and scaled these images to 14×14 pixels as described above. Each image was placed at the centre of the 31×31 input image. These images were the training input. In addition

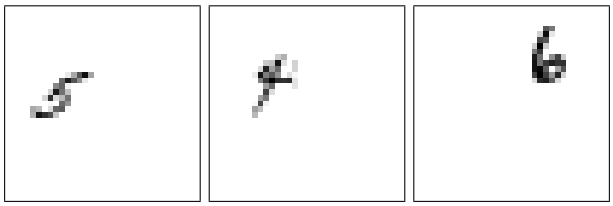

Figure 2 Example training images used in assessing the CNN's ability to handle natural images.

there were 600 noise patterns which were generated anew on each training cycle. Training was otherwise as before. After 1533 training cycles the error had reached an asymptote.

For the randomly placed training set, we selected 5384 images from the MNIST training set, again scaling each image to 14×14 pixels. This time, each scaled image was placed at a random location on the 31×31 input image. Figure 2 shows some example training patterns. These images were the training input. In addition there were 600 noise patterns which were generated anew on each training cycle.[1] Training was otherwise as before. After 2837 training cycles the error had reached an asymptote.

3.2 Results

3.2.1 A benchmark of the classification network on single-digit classification

We first evaluated the classifier on a task as close as possible to the standard MNIST digit classification task. For this task we evaluated the network trained on centred images on the 9,020 images of digits 1 – 9 from the MNIST test set presented at this same centred location. And we used an evaluation criterion which incorporates the knowledge that each test image is a single digit: we defined the output of the network to be the strongest of the possible digit output units, regardless of its absolute level. This network achieved a performance of 88% on the 9,020 images from the MNIST test set.

This performance is reasonable, but it is still considerably lower than the best-performing classifiers, which categorise unseen digits almost perfectly (see e.g. Ranzato et al, 2007; Hinton, 2007). There are several reasons for this. Most importantly, our classifier is configured to solve a more difficult problem than purpose-built classifiers: it identifies the type of a digit anywhere on a large 31×31 pixel image, rather than a digit normalised to the size of the classifier's input field. In addition, our classifier uses images which are half the resolution of the actual MNIST images. Finally, our set of training images is smaller by an order of magnitude than those used by state-of-the-art classifiers. Even though the network only received inputs in the centre of its retina, we did not want to customise its learning algorithm, so it still learned weights for all units, including those responding to the periphery. To make training times manageable, we only trained on around 1/10th of the total MNIST training images.

[1] As in Walles et al (2008), there were 11 groups of noise patterns, with densities ranging between 0.05 and 0.5. The probability of a pixel being set in a noise pattern was equal to the density of its group. If set, a pixel took a random value between zero and one, otherwise the pixel took the value zero. This provided noise of varying density.

3.2.2 Evaluation of a network trained and tested on randomly-placed digits

It is also interesting to see how our classifier performs when trained on randomly-placed digits, and tested on digits at a range of locations, either singly or in groups. We evaluated the network trained with randomly-placed digits on the 9,020 images of digits 1 – 9 from the MNIST test set, presented individually, at random locations, and scaled to 14×14 pixels as before. For this experiment, in preparation for dealing with heterogeneous groups of digits, we allowed the network to return an 'unknown category' output, as well as a digit category. If any of the system's 'digit' output units were active at above 0.5, its output was taken to be the highest of these units. If the 'error' output was highest, or if no units were active above 0.5, the network was understood as not offering a classification.

After training to asymptote, the network produced a correct classification in 66% of cases and an incorrect classification in 15% of cases. In the remaining cases no classification was offered.

Clearly, the network's performance is worse when training and testing on randomly placed images. Of course, the task is much harder: the network needs to generalise over two dimensions of space, as well as over the different possible shapes of each digit. (In addition, we are employing a stricter evaluation criterion, which does not build in knowledge that every test item is a digit.) However, these results still show that the network achieves a reasonable degree of translation invariance.

We then presented the network trained on randomly-placed digits with images containing pairs of digits. Each image contained two digits: there were 10,000 'homogeneous' images (where the digits were of the same type,) and 10,000 'heterogeneous' images (where the digits were of different types). The images in each pair were scaled to 14×14, as before. One image was placed one pixel in from the top and left edges and the other was placed one pixel in from the bottom and right edges. The digits were thus presented in the top-left and bottom-right quadrants of the retina.[2]

The heterogeneous pairs were selected by choosing two different categories at random, then choosing an image from each category in the MNIST test set and placing these on the input pattern. The homogeneous pairs were selected by choosing a single category at random, choosing two images from the category in the MNIST test set, and placing these on the input pattern. In both cases selection was with replacement so that a particular image could appear in more than one test pattern. Figure 3 shows an example heterogeneous pair.

Of the 10,000 homogeneous test patterns, 75% produced a classification correctly matching the digit type used in the pattern and 4% produced a classification corresponding to another category (the remaining patterns produced no classification).

Of the 10,000 heterogeneous patterns, 64% produced a classification corresponding to one of the digits used in the pattern and 4% produced a classification corresponding to another category (the remaining patterns produced no classification).

These results suggest that the network is cardinality blind with respect to the categories—indeed if anything the network is more accurate classifying homogeneous pairs than single items—an effect which recalls the 'redundancy gain' phenomenon in humans, in which sub-

[2]Placing the digits more closely together produced poorer results, most likely because of artefactual features formed from combinations of the two digits.

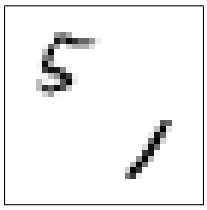

Figure 3 An example heterogeneous test pair.

jects are faster at identifying the type of a group of homogeneous items than that of a single item presented by itself (see e.g. van der Heijden, 1975; Theeuwes, 1994; Reinholz and Pollmann, 2007). At the same time, when presented with heterogeneous pairs, the network is (rightly) less willing to offer a classification than it is for heterogeneous pairs. The results are basically similar to those we obtained for the simpler shapes, as reported in Walles et al (2008): as the number of heterogeneous tokens present in a group goes up, the willingness of the CNN to produce a classification goes down.

4 Discussion

The experiments described in this report show that our classifier can be scaled up to learn complex naturalistic categories as well as simple ones. Its performance on these complex categories could of course be further improved. One option would be to redesign the lowest levels of the classifier, so that simple visual features are computed in an unsupervised manner to reflect the properties of the training images, as proposed by Hinton (2007). There is certainly evidence that convolutional networks modified in this way can achieve state-of-the-art classification performance on the MNIST training set (see Lee et al, 2009). This is an avenue for further work.

References

van der Heijden A (1975) Some evidence for a limited capacity parallel selfterminating process in simple visual search tasks. Acta Psychologica 39:21–41

Hinton G (2007) Learning multiple layers of representation. Trends in Cognitive Sciences 11(10):428–434

LeCun Y, Haffner P, Bottou L, Bengio Y (1999) Object recognition with gradient-based learning. In: Forsyth D (ed) Feature Grouping, Springer

Lee H, Grosse R, R R, Ng A (2009) Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations. In: Proceedings of the 26 th International Confer- ence on Machine Learning, Montréal, Canada, pp 609–616

Ranzato M, Poultney C, Chopra S, Le Cun Y (2007) Efficient learning of sparse representations with an energy-based model. In: Schoelkopf B, Platt J, Hoffman T (eds) Advances in Neural Information Processing Systems (Vol. 19), MIT Press, Boston, MA, pp 1137–1144

Reinholz J, Pollmann S (2007) Neural basis of redundancy effects in visual object categorization. Neuroscience Letters 412:123–128

Riedmiller M (1994) Rprop - description and implementation details. Tech. rep., Institut für Logik, Komplexität und Deduktionssyteme, University of Karlsruhe

Rumelhart D, Hinton G, Williams R (1986) Learning internal representations by error propagation. In: Parallel Distributed Processing: Explorations in the Microstructures of Cognition, vol 1: Foundations, MIT Press, Cambridge, MA, chap 8, pp 318–362

Theeuwes J (1994) The effects of location cuing on redundant-target processing. Psychological Research 57:15–19

Walles H, Knott A, Robins A (2008) A model of cardinality blindness in inferotemporal cortex. Biological Cybernetics 98(5):427–437