Creating Edge Detectors by Evolutionary Reinforcement Learning

Nils T Siebel, Sven Grünewald, Gerald Sommer

Abstract— **In this article we present results from experiments where a edge detector was learned from scratch by EANT2, a method for evolutionary reinforcement learning. The detector is constructed as a neural network that takes as input the pixel values from a given image region—the same way that standard edge detectors do. However, it does not have any per-image parameters. A comparison between the evolved neural networks and two standard algorithms, the Sobel and Canny edge detectors, shows very good results.**

I. Introduction

EDGES characterise boundaries of image regions and are therefore of great importance for image understanding. Methods for edge detection, like most image processing techniques, have a long signal processing tradition. Text books generally take a very practically oriented approach to edge detection, and many edge detectors have parameters that need to be fine-tuned to an image in order to achieve good detection results.

Our approach is to learn an edge detector from scratch by EANT2, a method for evolutionary reinforcement learning. The evolved detector is a neural network that takes as input the pixel values from a given image region—the same way that most edge detectors, like Canny's and Sobel's methods, do. However, the network does not have any per-image parameters, following our paradigm that learning methods should be general and require only a minimum of input from the user.

The remainder of the article is as follows. Section II describes edge detection in general and related work. Details on our learning algorithm can be found in Section III. The test setup and main results are located in Section IV, followed by conclusions in Section V.

II. Preliminaries and Related Work

In this section we will discuss related work concerning the two main areas that this work is meant to bring together: edge detection and methods for learning neural networks by evolutionary algorithms. Before we do so, however, we will briefly discuss the nature and definition of edges in images.

A. Edges

Edge detection is a fundamental tool for image understanding. Object boundaries that are visible in the image usually define edges, i.e. borders of image regions that have a difference in brightness and/or colour. (Higher level image analyses may also be able to distinguish edges that occur within an object's image region and outside of it but these considerations are not within the scope of this article.) What is considered a "correct" edge and what not is usually dependent on the context in which an edge detector is used. Intended applications and the nature of images (source, quality, visual appearance of objects in it) influence this decision. The definition of what constitutes an edge is not straightforward, and this makes it difficult to evaluate and compare different edge detectors. Please consider the 11×9 pixel image depicted in Figure 1 where each square corresponds to a pixel. Where is the edge around the dark grey object? The actual border around the object is between the dark and light pixels. However, here we are considering edge detection methods that should tell us, for each pixel of the image, whether this pixel contains an edge or not.

In order to determine the probability of an edge at a given pixel position it is necessary to look at neighbouring pixels. Commonly this is done by considering a square image region centred around the pixel under consideration. This region may for instance be of size 3×3 or 5×5 pixels. In Figure 1 we have marked such a 3×3 region by a red outline. The pixel in question is marked with an "X".

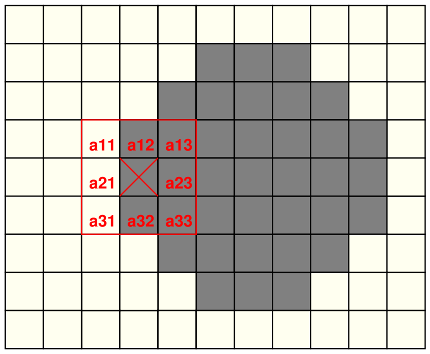


Fig. 1. Classification of edge pixels by considering a 3×3 region

It is clear that if an edge can only be located at a pixel position (not in between pixels) then the centre pixel "X" definitely contains one. When looking at the 3×3 region one does not have access to the information whether this pixel is located inside or outside the dark object. With this in mind if follows that the pixel marked "a21" is also an edge pixel if we examine again a 3×3 square centred around it. If it were not marked an edge as well our method might favour dark objects on a light background over light objects on a dark background (or, right edge pixels over left ones, etc.) which is usually not desirable. Therefore edge detectors that only consider such a 3×3 region would have to mark both the "X" pixel and "a21" as edge pixels, resulting in an edge outline around the object that is 2 pixels wide.

The authors are with the Cognitive Systems Group, Institute of Computer Science, Christian-Albrechts-University of Kiel, Germany (e-mail: {nts,svg,gs}@ks.informatik.uni-kiel.de).

B. Edge Detection Methods

Mathematics tells us that changes such as those in pixel values along a line in the image can be detected by examining the derivative. This can be approximated by taking into account differences of pixel values in a region like the 3×3 "mask" introduced above. In order to reduce the influence of image noise and similar factors the image and/or the response from the detector may also be smoothed, e.g. by convolution with a Gaussian mask. The following common methods for edge detection use approximations of the 1st and 2nd derivative [1]:

- 1st derivative: Canny, Sobel and Prewitt edge detectors
- 2nd derivative: Marr-Hildreth algorithm (Laplacian of the Gaussian a.k.a. Mexican hat image operator)

Methods that are approximating a 2nd derivative by looking at a small square region are very sensitive to noise, so we will be focusing on 1st order methods here.

The **Sobel edge detector** was first presented by Irwin Sobel in 1968 [2]. It is a discrete differentiation operator that computes an approximation of the gradient of image intensities. It operates on the image using 2 square masks like the 3×3 mask above and combines the 2 results (gradient magnitudes in x and y direction) to form an estimate of the absolute gradient magnitude at each image point. Then usually a threshold is applied to classify edge pixels. The Sobel edge detector is not very complex and hence very fast.

The **Canny edge detector** is probably the most commonly used method. It was published by John Canny in 1986 along with a thorough examination of the underlying theory [3]. It is a fairly complex algorithm that works by first filtering the image with a Gaussian smoothing filter. Then it calculates the image intensity gradient (e.g. using Sobel-like masks) and looks for peaks in the gradient image using non-maximum suppression. In the last stage the resulting detector responses are filtered using two thresholds: First a high threshold is applied with the intention of filtering out spurious responses from image noise. Then the resulting strong edges are traced and responses from the detector that are above a second, lower threshold are added to the result. The Canny algorithm thus contains adjustable parameters which can affect its computation time and effectiveness, the parameters for the Gaussian smoothing and the two thresholds. More elaborate versions of the algorithm contain additional features like edge thinning, surround inhibition etc.—and thereby, more parameters that often need to be tuned on a per-image basis for optimal results. Together with these features, however, the Canny algorithm is one of the best performing edge detectors.

It should be noted that both in the pre- and postprocessing steps the Canny algorithm does use information outside the square image area where the gradient calculation takes place. Also, due to its thinning step it usually calculates an edge outline that is one 1 pixel wide, not 2.

C. Evolutionary Ways of Creating Neural Networks

Until recently, only small neural networks have been evolved by evolutionary algorithms [4]. According to Yao, a main reason is the difficulty of evaluating the exact fitness of a newly found structure: In order to fully evaluate a *structure* one needs to find the optimal (or, some near-optimal) *parameters* for it. However, the search for good parameters for a given structure has a high computational complexity unless the problem is very simple *(ibid.)*.

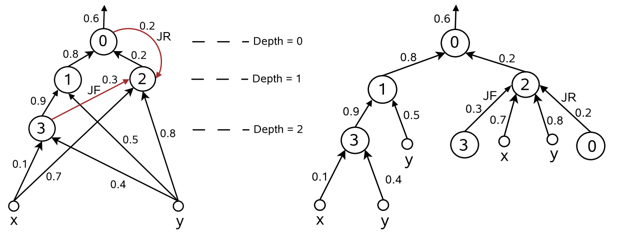
Most recent approaches evolve the structure and parameters of the neural networks simultaneously. Examples are EPNet [5], GNARL [6] and NEAT [7]. EPNet uses a modified backpropagation algorithm for parameter optimisation (a local method). Mutation operators for searching the space of neural structures are addition and deletion of neurons and connections (no crossover is used). EPNet has a tendency to remove connections/nodes rather than to add new ones. This is done to counteract "bloat" (i.e. ever growing networks with only little fitness improvement; called "survival of the fattest" in [8]). GNARL also does not uses crossover during structural mutation. However, it uses an evolutionary algorithm for parameter optimisation. Both parametrical and structural mutation use a "temperature" measure to determine whether large or small random modifications should be applied—a concept known from simulated annealing [9]. In order to calculate the current temperature, the algorithm needs some knowledge about the "ideal solution" to the problem, e.g. the best fitness expected to be reached.

NEAT, unlike EPNet and GNARL, uses a crossover operator that allows to produce valid offspring from two given neural networks. It works by first aligning similar or equal subnetworks and then exchanging differing parts. Like GNARL, NEAT uses evolutionary algorithms for both parametrical and structural mutation. However, the probabilities and standard deviations used for random mutation are constant over time. NEAT also incorporates the concept of speciation, i.e. separated sub-populations that aim at cultivating and preserving diversity in the population [8].

D. Using Neural Networks for Edge Detection

Pinho has applied artificial neural networks to edge detection as early as 1993 [10]. He used a network of fixed topology, with one fully connected hidden layer consisting of 9 neurons with a tanh-like activation function and thresholded outputs. The 9 network inputs are pixel values from a 3×3 mask as in Figure 1. Rotational symmetry, as expected from the detector network, was achieved by having the neurons share weights with their symmetric counterparts, which reduced the number of parameters from 100 to 34. Training was done on synthetic example data using the backpropagation algorithm. Their results are reported to be very good when compared to the Sobel method, and less sensitive to noise.

More recently, Becerikli *et al.* trained a neural network by data generated by a Laplacian edge detector [11]. Again a network of fixed topology was used, with 9 inputs from a 3×3 mask, one hidden layer of 12 neurons and trained using backpropagation. They conclude that in their experiments the approach with neural networks exhibits less sensitivity to noise but is otherwise comparable to the Laplacian approach.



(a) Original neural network (b) Network in tree format

N 0	N 1	N 3	I x	I y	I y	N 2	JF 3	I x	I y	JR 0
W=0.6	W=0.8	W=0.9	W=0.1	W=0.4	W=0.5	W=0.2	W=0.3	W=0.7	W=0.8	W=0.2

(c) Corresponding Linear Genome

Fig. 2. An example of encoding a neural network using a linear genome

In our approach we wish to avoid all pre-designing of the solution, whether it be setting the neural structure manually or re-formulating the problem by using information about an anticipated property of the network (like symmetry). Instead, we use evolutionary reinforcement learning to develop neural networks that solve the edge detection task. It uses evolutionary optimisation methods for the structure and the weights since these global optimisation methods are known to yield convergence to good solutions even in high-dimensional spaces and with numerically "difficult" (e.g. ill-posed) functions.

III. Learning Neural Networks with EANT2

A. *The Algorithm*

EANT2, "Evolutionary Acquisition of Neural Topologies Version 2", is an evolutionary reinforcement learning system that realises neural network learning with evolutionary algorithms both for the structural and the parametrical part. It is based on the previous method EANT [12] but uses different algorithms for structural mutation and parameter optimisation [13]. EANT2 represents neural networks and their parameters in a compact genetic encoding, the "linear genome". It encodes the topology of the network implicitly by the order of its elements (genes). The following basic gene types exist: neurons, network inputs, biases and forward connections. There are also "irregular" connections between neural genes which we call "jumper connections". Jumper genes can encode either forward or recurrent connections. Figure 2 shows an example encoding of a neural network using a linear genome. The figures show (a) the neural network to be encoded. It has one forward and one recurrent jumper connection; (b) the neural network interpreted as a tree structure; and (c) the linear genome encoding the neural network. In the linear genome, `N` stands for a neuron, `I` for an input to the neural network, `JF` for a forward jumper connection, and `JR` for a recurrent jumper connection. The numbers beside `N` represent the global identification numbers of the neurons, `x` and `y` are the inputs coded by input genes.

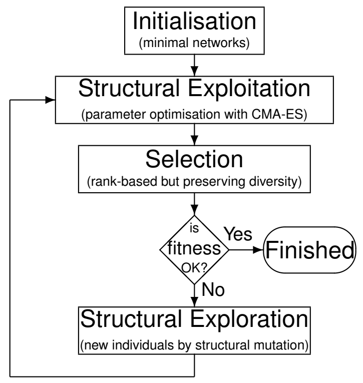

Initialisation
(minimal networks)
Structural Exploitation
(parameter optimisation with CMA-ES)
Selection
(rank-based but preserving diversity)
is fitness OK?
Yes
Finished
No
Structural Exploration
(new individuals by structural mutation)

Fig. 3. The EANT2 algorithm. Please note that CMA-ES has its own optimisation loop which creates in EANT2 a nested loop.

A linear genome can be interpreted as a tree based program if one considers all the inputs to the network and all jumper connections as terminals.

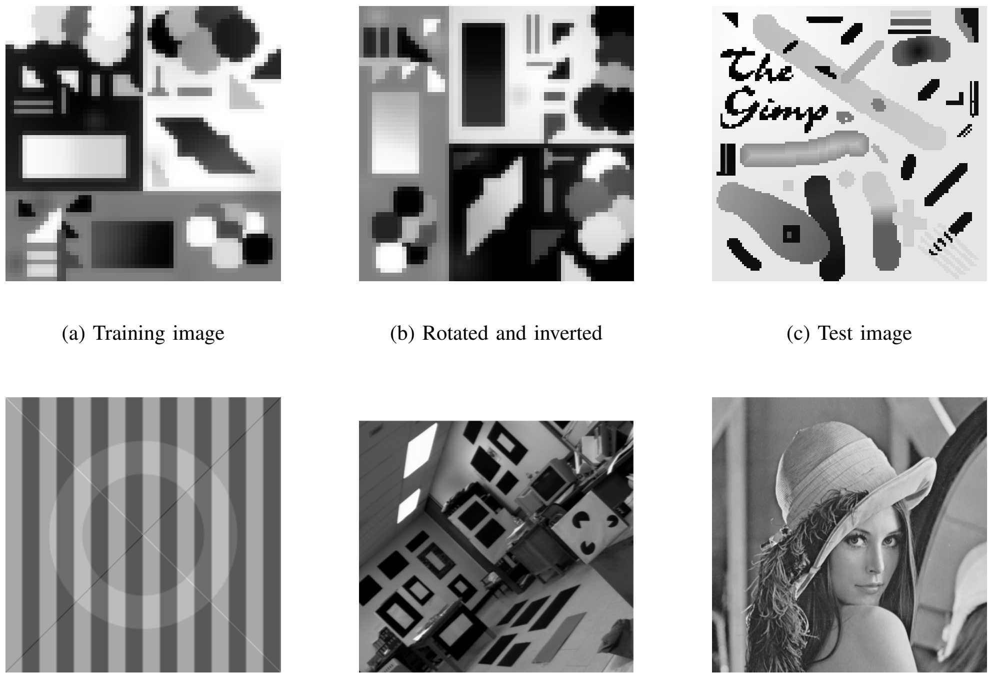
Linear genomes can be evaluated, without decoding, similar to the way mathematical expressions in postfix notation are evaluated. For example, a neuron gene is followed by its input genes. In order to evaluate it, one can traverse the linear genome from back to front, pushing inputs onto a stack. When encountering a neuron gene one pops as many genes from the stack as there are inputs to the neuron, using their values as input values. The resulting evaluated neuron is again pushed onto the stack, enabling this subnetwork to be used as an input to another neuron. Connection ("jumper") genes make it possible for neuron outputs to be used as input to more than one neuron, see `JF3` in the example above.

The steps of our algorithm, shown in Figure 3, are explained in detail below.

Initialisation: EANT2 usually starts with minimal initial structures. An minimal network has no hidden layers or recurrent connections, only 1 neuron per output, connected to some or all inputs. EANT2 gradually develops these simple initial structures further using the structural and parametrical evolutionary algorithms discussed below. On a larger scale new neural structures are added to a current generation of networks. We call this "structural exploration". On a smaller scale the current structures are optimised by changing their parameters: "structural exploitation".

Structural Exploitation: At this stage the structures in the current EANT2 population are exploited by optimising their parameters. Parametrical mutation is realised using CMA-ES ("Covariance Matrix Adaptation Evolution Strategy") [14]. CMA-ES is a variant of Evolution Strategies that avoids random adaptation of strategy parameters. Instead, the search area spanned by the mutation strategy parameters, expressed here by a covariance matrix, is adapted at each step depending on the parameter and fitness values of current population members. CMA-ES uses sophisticated methods to avoid problems like premature convergence and is known for fast convergence to good solutions even with multi-modal and non-separable functions in high-dimensional spaces *(ibid.)*.

Selection: The selection operator determines which pop-



(a) Training image (b) Rotated and inverted (c) Test image

(d) “Ledge” image (e) Natural image “Lab” (f) Natural image “Lena”

Fig. 4. Original images, training (top left) and 5 testing images

ulation members are carried on from one generation to the next. Our selection in the outer, structural exploration loop is rank-based and “greedy”, preferring individuals that have a larger fitness. In order to maintain diversity in the population, it also compares individuals by structure, ignoring their parameters. The operator makes sure that not more than 1 copy of an individual and not more than 2 similar individuals are kept in the population. “Similar” in this case means that a structure was derived from an another one by only changing connections, not adding neurons.

Structural Exploration: In this step new structures are generated and added to the population. This is achieved by applying the following structural mutation operators to the existing structures: Adding or removing a random subnetwork, adding or removing a random connection and adding a random bias. New hidden neurons are connected to approx. 50 % of inputs; the exact percentage and selection of inputs are random.

B. Comparison with Other Methods

EANT2 is closely related to the methods described in the related work section above. One main difference is the clear separation of structural exploration and structural exploitation. By this we try to make sure a new structural element is tested (“exploited”) as much as possible before a decision is made to discard it or keep it, or before other structural modifications are applied. Another main difference is the use of CMA-ES in the parameter optimisation. Further differences of EANT2 to other recent methods, e.g. NEAT, are a small number of user-defined algorithm parameters (the method should be as general as possible) and the explicit way of preserving diversity in the population (unlike speciation).

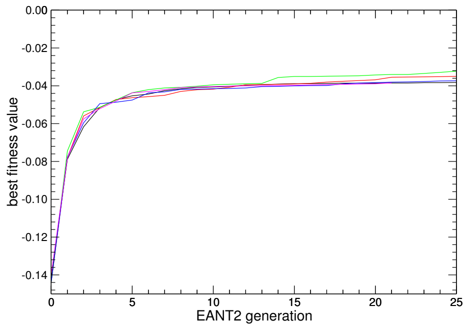
In the past we have compared EANT2 with NEAT by applying both algorithms to the same problem [15]. The test environment was a visual servoing task run in a simulation. Both methods were to develop neural networks to control a robot in 3 degrees of freedom in order to align its gripper to an object. The robot movement was determined based on 10 image measurements, so the networks had 10 inputs and 3 outputs. The results showed that NEAT had more problems than EANT2 finding good parameters for given networks. EANT2 was at a clear advantage in this comparison. More details can be found in [15].

IV. Main Results

Our goal was to develop an edge detector with EANT2 that takes the pixel values of a 3×3 or 5×5 image region as an input and give the probability of an edge at the centre pixel as an output. Since our activation function is a tanh function with the range -1 to 1 we used the absolute value of the output neuron’s value as network output. No information about the task or desired properties of the solution (e.g. invariance to rotation and edge-preserving changes to grey levels) were explicitly given to the algorithm. The algorithm needed to derive these properties from the training data.

A. Training and Testing Setup

In order to develop and evaluate edge detectors we have created a synthetic training image, shown in Figure 4(a). It



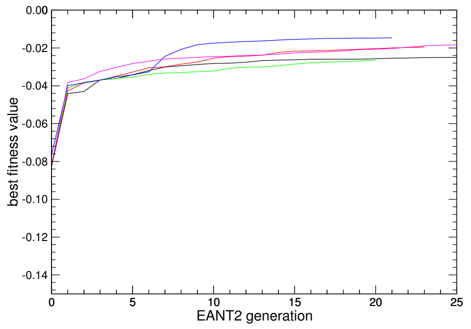


Fig. 5. Results from 5 EANT2 runs each, mask size 3×3 (left) and 5×5 (right).

has 3969 pixels out of which 1878 (47 %) are edge pixels with different grey levels and edge orientations, with an added slight variation of brightness across the image. In order to make it more natural-looking, a Gaussian smoothing with $\sigma = 0.6$ pixels, similar to the effect one would have in a camera, was applied to the image.

As test images we used the following (see Figure 4):

1) The training image, rotated and inverted so as to test for invariance to rotation and grey level changes.
2) A test image that is more complex and difficult, containing stronger non-edge grey level changes.
3) The "Ledge" image by Hans du Buf[1].
4) The well known and widely used "laboratory" image.
5) The even better known and more widely used "Lena".

Following the rationale in Section II-A we created "ground truth" locations of edges with an edge width of 2 pixels. While for the synthetic images it was easy to define ground truth, no definite ground truth exists for the 2 natural images.

As a fitness function f for evaluating a neural network N we used the negative mean deviation of the neural network output to the binary ground truth:

$$f(N) = \frac{-\sum_{(x,y)} \left| G(x,y) - |o(x,y)| \right|}{(w-b)(h-b)},$$

where w and h are the width and height of the image, $o(x,y)$ the output of the neural network and $G(x,y)$ the ground truth at image position (x,y). b is the size of the border region in which no square neighbourhood can be defined, i.e. $b = 1$ for a 3×3 and $b = 2$ for a 5×5 mask size. Our definition of $f(N)$ means that it takes on negative values with $f(N) = 0$ for an ideal network. Pixel values range from 0 to 1 (=white).

B. Development of 3x3 and 5x5 Networks

The results from 5 runs each of EANT2 with mask sizes 3×3 and 5×5 are shown in Figure 5. Plotted is the development of the fitness function dependent on the EANT2 generation, i.e. the number of structural mutations.

With both mask sizes the performance increases significantly in the first few generations with steady but smaller improvements as time goes on. After generation 20 the mean resulting fitnesses are -0.0374 and -0.0216, which shows that the network with 25 inputs is at a clear advantage. This was expected considering that the larger mask enables a better analysis of the context a pixel under examination appears in. The mean network sizes are 51.6 and 97, respectively.

TABLE I
COMPARISON OF RESULTS: DETECTION CORRECTNESS

Method	Training	Rot.&Inv.	Test	"Ledge"
EANT2, 3x3	97.71 %	97.32 %	**95.03 %**	**94.75 %**
EANT2, 5x5	**99.19 %**	**98.31 %**	91.82 %	89.52 %
Sobel, 3x3	96.53 %	96.53 %	93.53 %	94.22 %
Sobel, 5x5	82.17 %	82.17 %	85.97 %	81.43 %

C. Comparison and Evaluation of Results

For comparison, we applied the resulting neural networks from generation 20 as well as the Canny and Sobel edge detectors to the training image and 5 test images mentioned above. In order to make a fair comparison to the neural approaches we tuned the parameters of the Canny and Sobel detectors to the training image and kept them fixed at these values for the test images. We first show the results for the comparison with the Sobel detector, where ground truth (for 2-pixel edges) are available. Results with Canny will be discussed below.

In addition to the visual evaluation we have quantified the results in table I. We used the absolute value of the network outputs and thresholded them with the value 0.5 to get a Boolean classification result edge/no edge, and misclassifications were counted. For the Canny edge detector there is no ground truth with edge widths of 1 pixel (see also Section II-A). A comparison with the 2-pixel ground truth would unfairly yield classification rates of less than 50 %, therefore an additional comparison to the visual one (below) cannot be made.

Figure 6 shows the resulting edge images on the original training image and the rotated and inverted version. The results all look very good and similar, except for the Sobel detector with the 5×5 mask. The detector was so sensitive, generating false positives, that during training the threshold had to be set as low as 0.11 to achieve the best results in the

[1]see http://w3.ualg.pt/~dubuf/pubdat/ledge/ledge.html

training image. Even then the detection rate is only at 82.2 % and parts of the contours with lower contrast are missing. Detection rates of EANT2 and the 3×3 Sobel range from 97 % to 99 % for both images. The 5×5 EANT2 network, like the 5×5 Sobel, seems to be more susceptible to low frequency noise like the slight "bump" in the middle left, creating more false positive responses than the 3×3 network. However, with this increased sensitivity comes a smaller number of false negatives which leads to the better detection rate.

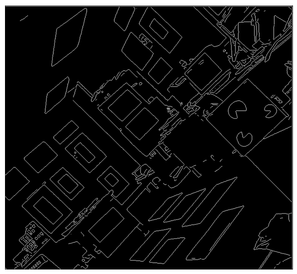
The invariance of all 4 algorithms to rotation and grey level changes is very good. For the Sobel detector this does not come as a surprise since these properties have been explicitly incorporated. EANT2, on the other hand, learned edges in all directions with different brightnesses independently by examples but gives very similar responses.

In Figure 7 we see the results with the test and "Ledge" images. For the test image the results from the 3×3 network and 3×3 Sobel are very similar, while again the 5×5 network with its higher sensitivity shows fewer false negatives, but this time more false positives at positions with strong low frequency transitions. This leads to the lower detection rate of 91.8 %, with the 3×3 network and Sobel at 95.0 % and 93.5 %, 5×5 Sobel at 86.0 % with many false negatives. The "Ledge" image has very low contrast edges, making it difficult. Visually, EANT2's 5×5 network shows the best results, clearly showing the circles' contours almost completely. The 3×3 network does not have most circle edges but still achieves a result is 94.8 % since there are few false positives, with Sobel following at 94.2 % and the 5×5 network at 89.5 %. The 5×5 Sobel achieves 81.4 %.

When looking at the natural images in Figure 8 again the results from Sobel and the 3×3 network look very similar and good. Sobel is least distracted by noise here. The 5×5 network shows good results on the lab image but has too many responses in the Lena image, again being very sensitive. The 5×5 Sobel again has many false negatives.

Figures 9 and 10 show the results from the Canny edge detector (using a 3×3 mask), again with parameters tuned to the training image. The results generally look very good but it is clear that tuning the parameters to each image would necessary to improve the results. This is most obvious in the Lena and Lab images. Canny detects many high-frequency edge outlines, which becomes most apparent in Lena's hat. It has probably detected most of the edges in the Lab image but also has false positive responses due to noise in background scene of Lena. Canny's post-processing can, however, still improve the detection results for the Ledge image (compare Figure 9(d) to Sobel's unrefined result in Figure 7(g)). In Figures 9(a) and 9(b) it can be seen that the thin edges show problems like a jagged contour and rounded edges of the rectangles, which might create problems for higher-level image processing algorithms working with these outlines.

Overall, all edge detectors have shown good results on our images, with some more sensitive and therefore prone to yield false positives while others show very clear and correct edges. It is difficult to create an edge detector that works well without parameter adjustment on such a wide range of images. The neural networks created by EANT2 have shown the best classification rates for all images, with the 3×3 network showing the most constant rates of 95–98 %. Considering that features like the desired invariance to rotation and edge-preserving grey-level changes were not explicitly programmed into the solution as with the theoretically founded and proven Sobel and Canny methods we are very happy about their good performance.


(a) Lab

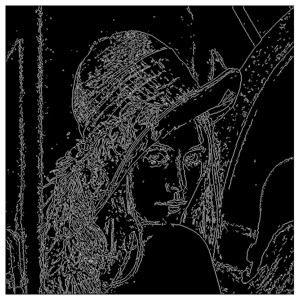

(b) Lena

Fig. 10. Edge responses of the Canny detector (parameters fixed, from the training image)

V. Conclusions

Our aim was to create neural networks by evolutionary reinforcement learning that could perform edge detection using a square image mask of size 3×3 or 5×5, giving the pixel values from these regions as 9 or 25 separate inputs to the networks. The development was done on a synthetic training image using our method EANT2 [15]. We compared the 3×3 and 5×5 networks on 5 test images to the standard edge detection methods by Sobel and Canny.

The evaluation of the 4 edge detectors has shown that the neural networks developed by EANT2 perform as good or better than the Sobel detector. Comparison with Canny's method is difficult due to incompatible results. However, a visual comparison shows that its pre- and postprocessing like hysteresis thresholding and edge thinning improves results beyond the capabilities of the neural networks that only operate on masks in a one-step algorithm. However, in some cases Canny's results showed problems like jagged lines and rounded edges that were not present in the other methods.

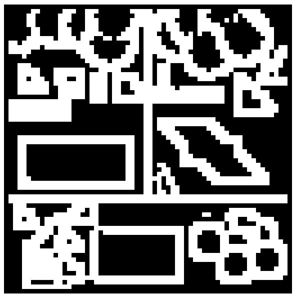
To conclude, our experiments have shown that EANT2 can generate neural networks that can compete with standard edge detectors on a range of test images.

References

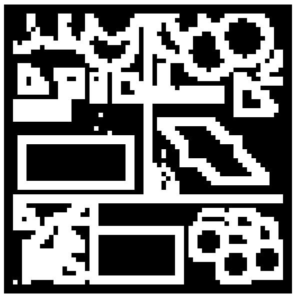
[1] D. Ziou and S. Tabbone, "Edge detection techniques – an overview," *International Journal of Pattern Recognition and Image Analysis*, vol. 8, no. 4, pp. 537–559, 1988.

[2] I. E. Sobel and J. A. Feldman, "A 3x3 isotropic gradient operator for image processing," 1968, unpublished, presented as a talk within the Stanford Artificial Intelligence Project.

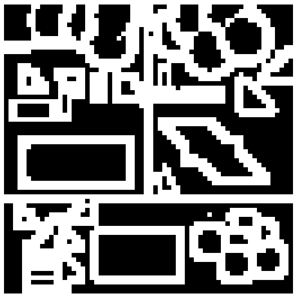
[3] J. F. Canny, "A computational approach to edge detection," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 8, no. 6, pp. 679–698, November 1986.



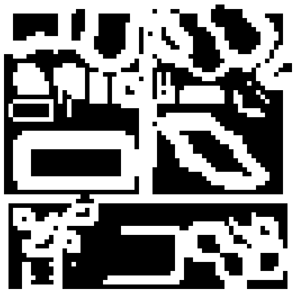
(a) EANT2, 3x3 mask



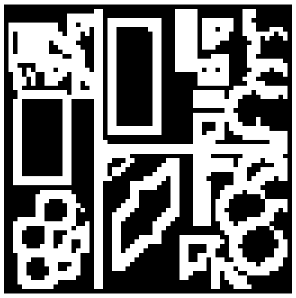
(b) EANT2, 5x5 mask



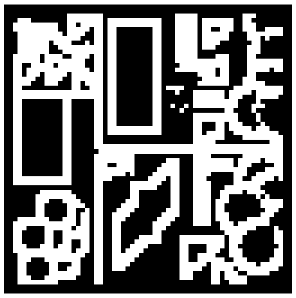
(c) Sobel, 3x3



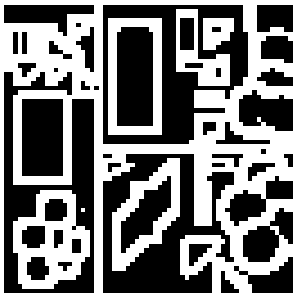
(d) Sobel, 5x5



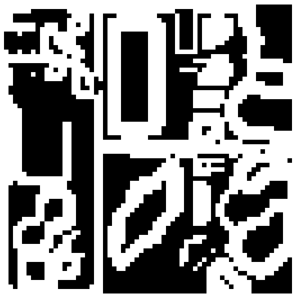
(e) EANT2, 3x3 mask



(f) EANT2, 5x5 mask

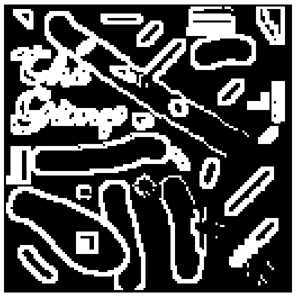


(g) Sobel, 3x3

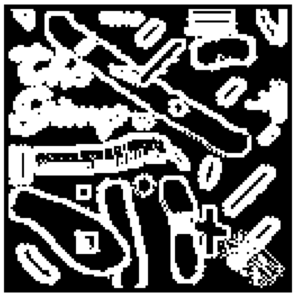


(h) Sobel, 5x5

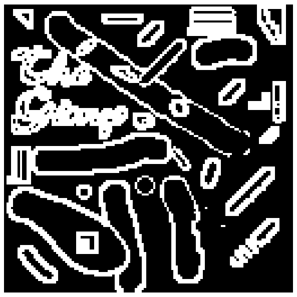
Fig. 6. Edge responses on the original and the rotated and inverted training images (parameters tuned on the original image)



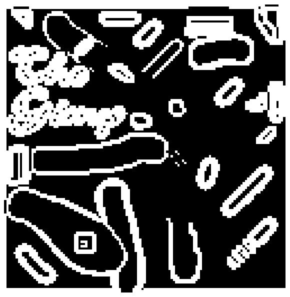
(a) EANT2, 3x3 mask



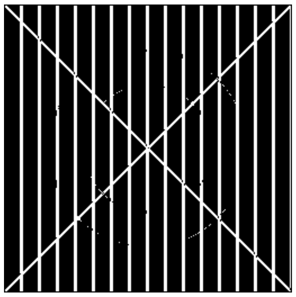
(b) EANT2, 5x5 mask



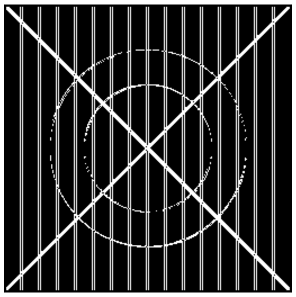
(c) Sobel, 3x3



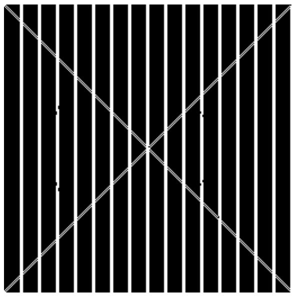
(d) Sobel, 5x5



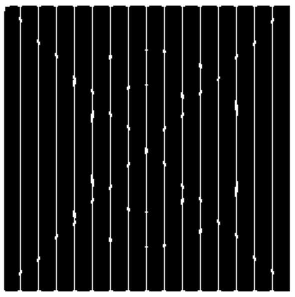
(e) EANT2, 3x3 mask



(f) EANT2, 5x5 mask

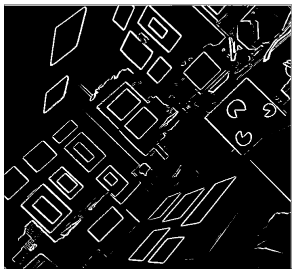


(g) Sobel, 3x3

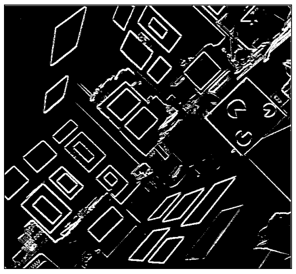


(h) Sobel, 5x5

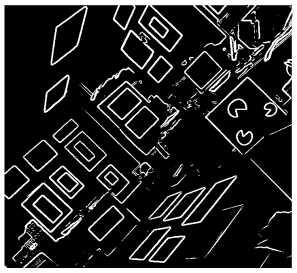
Fig. 7. Edge responses on the testing and “Ledge” images (parameters fixed, from training image)



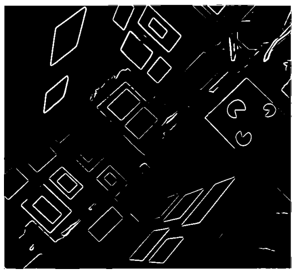
(a) EANT2, 3x3 mask



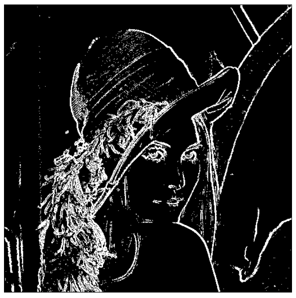
(b) EANT2, 5x5 mask



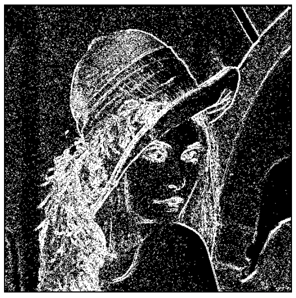
(c) Sobel, 3x3



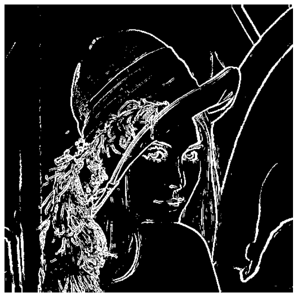
(d) Sobel, 5x5



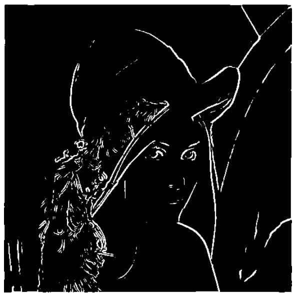
(e) EANT2, 3x3 mask



(f) EANT2, 5x5 mask

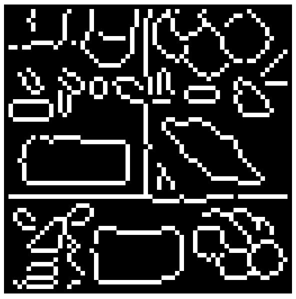


(g) Sobel

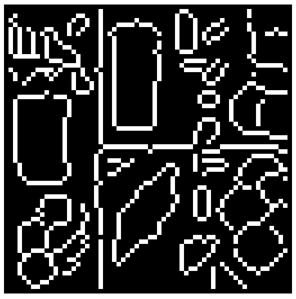


(h) Sobel

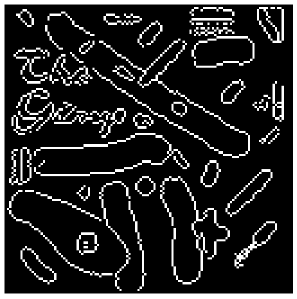
Fig. 8. Edge responses on the lab and Lena images (parameters fixed, from training image)



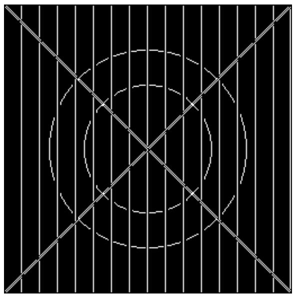
(a) training



(b) rotated and inverted



(c) testing



(d) "Ledge"

Fig. 9. Edge responses of the Canny detector (parameters fixed, from the training image)

[4] X. Yao, "Evolving artificial neural networks," *Proceedings of the IEEE*, vol. 87, no. 9, pp. 1423–1447, September 1999.

[5] X. Yao and Y. Liu, "A new evolutionary system for evolving artificial neural networks," *IEEE Transactions on Neural Networks*, vol. 8, no. 3, pp. 694–713, May 1997.

[6] P. J. Angeline, G. M. Saunders, and J. B. Pollack, "An evolutionary algorithm that constructs recurrent neural networks," *IEEE Transactions on Neural Networks*, vol. 5, no. 1, pp. 54–65, 1994.

[7] K. O. Stanley and R. Miikkulainen, "Evolving neural networks through augmenting topologies," *Evolutionary Computation*, vol. 10, no. 2, pp. 99–127, 2002.

[8] Á. E. Eiben and J. E. Smith, *Introduction to Evolutionary Computing*. Berlin, Germany: Springer Verlag, 2003.

[9] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by simulated annealing," *Science*, vol. 220, no. 4598, pp. 671–680, May 1983.

[10] A. J. Pinho, "Modeling non-linear edge detectors using artificial neural networks," in *Proceedings of the 15th Annual International Conference of the IEEE Engineering in Medicine and Biology Society, EMBS-93, San Diego, USA*, October 1093, pp. 306–307.

[11] Y. Becerikli, H. E. Demiray, M. Ayhan, and K. Aktaş, "Alternative neural network based edge detection," *Neural Information Processing–Letters and Reviews*, vol. 10, no. 8, pp. 193–199, August 2006.

[12] Y. Kassahun and G. Sommer, "Efficient reinforcement learning through evolutionary acquisition of neural topologies," in *Proceedings of the 13th European Symposium on Artificial Neural Networks (ESANN 2005)*, Bruges, Belgium, April 2005, pp. 259–266.

[13] N. T. Siebel and Y. Kassahun, "Learning neural networks for visual servoing using evolutionary methods," in *Proceedings of the 6th International Conference on Hybrid Intelligent Systems (HIS'06), Auckland, New Zealand*, December 2006, p. 6 (4 pages).

[14] N. Hansen and A. Ostermeier, "Completely derandomized self-adaptation in evolution strategies," *Evolutionary Computation*, vol. 9, no. 2, pp. 159–195, 2001.

[15] N. T. Siebel and G. Sommer, "Evolutionary reinforcement learning of artificial neural networks," *International Journal of Hybrid Intelligent Systems*, vol. 4, no. 3, pp. 171–183, October 2007.