Edge Pixel Classification Using Automatic Programming

Kristin Larsen, Lars Vidar Magnusson and Roland Olsson

Østfold University College

Abstract

We have considered edge detection as a classification problem, and we have applied two popular machine learning techniques to the problem and compared their best results to that of automatic programming. We show that ADATE, our system for automatic programming, is capable of producing solutions that are as good as, or better than, the best solutions generated by two other machine learning techniques.

The results demonstrates the ability of the ADATE system to create powerful heuristics to solve image analysis problems.

1 Introduction

Edges in digital images are often used to find the boundaries of any objects of interest in an image. This is why edge detection one of the first steps in many image analysis applications. Problems in which the task is to classify examples into one of a discrete set of possible categories, are often referred to as classification problems. In this paper we have considered edge detection as a classification problem. We have applied popular machine learning techniques to the problem in order to compare their best solutions to the best solutions generated by ADATE, our system for automatic programming.

We show that automatic programming performs as well or better than the competing machine learning techniques on this problem, and that the performance can be attributed to the ability of the automatic programming system to generate powerful feature extraction that investigate complex relationships between the input attributes. The solution created is unlikely to be useful for real-world applications, but the result provide further evidence that ADATE can be used to create or improve solutions to image analysis problems.

2 Background

The following sections provide relevant background information about traditional edge detection and the machine learning techniques we have applied to the related problem of edge classification.

This paper was presented at the NIK-2014 conference; see http://www.nik.no/.

Edge detection

Edge detection is a complicated problem in that there exists no objective truth. Which is one of the reasons why it is impossible to design an edge detector that will find all the true edges in an image and nothing more. It has been shown that edge detectors only give ambiguous local information about the presence of object boundaries [1].

Despite this fact, there exists a variety of edge detecting algorithms. The simplest algorithms are typically filter-based, meaning that they convolve a small mask with an image in order to locate the edges. Points that lie on an edge can be located by detecting local maxima or minima of the first derivative, or detecting the zero-crossing of the second derivative. The Prewitt operator [2] introduced by Judith Prewitt in 1970 and the Sobel Operator [3] by Irwin Sobel from 1973 are both examples of algorithms that detect edges by convolving an image with local derivative filters.

There are also more complex algorithms, such as the Canny edge detector [4] introduced by John Canny in 1986. This algorithm start out using a filter to locate sharp discontinuities in the image. These discontinuities are then processed further through what has been termed non-maximum suppression and then hysteresis thresholding. There are other algorithms that attempt to find the edges through a series of steps. The Global Probable Boundary (gPb) algorithm from 2011 is, like Canny, a three step algorithm [5]. All the steps are more complex than the ones in Canny, but the purpose of them are similar. First, find indications of edges in the image, then process the edge pixels found to clean up the initial results. In this instance, this is done by by incorporating global information.

In this paper we have looked at the edge detection problem from a classification perspective. We only consider the immediate neighbors to any given pixel, meaning that our approach is comparable to that of the simple filter approach, and perhaps also to the first step of multi step algorithms. No comparison with any traditional edge detection has been done, since the purpose of the experiment is to compare automatic programming to other machine learning techniques.

Machine Learning Techniques

Decision trees classify instances by sorting them through a tree structure of nodes and branches, where each node specifies a test of some attribute of the instance, and each branch corresponds to one of the possible values for the attribute. The leaf node gives the class of the instance [6]. C5.0 is a machine learning algorithm based on decision trees. The decision trees are built from a list of possible attributes and set of training cases, and the trees can be used to classify subsequent sets of test-cases [7]. The trees can also be converted to rules, by creating one rule for each path from the root node to a leaf node [6]. Decision Trees have successfully been applied to a wide range of tasks, from diagnosing medical cases to assessing credit risk of loan applicants [6].

Artificial Neural Networks (ANNs) are inspired by biological learning systems, like the human brain, which are constructed by complex webs of interconnected neurons. Each neuron takes a number of inputs, which may be outputs from other neurons, and produces a single output, which may be input to yet other neurons [6]. Artificial Neural Networks provide a robust, practical method for many different tasks. Some claim it is the only solution needed for solving any type of problem. With the introduction of the back-propagation learning algorithm, neural networks with one or more hidden layers can in theory be trained to solve any regression or discrimination task. Since the mid-nineties, neural networks have been applied to image processing applications. It has been used for image segmentation and object recognition, as well as low level tasks such as noise

reduction and image enhancement [8].

Automatic Programming

Automatic programming is a relative new machine learning technique that supports the automatic generation of algorithms based on specifications provided by the user. Automatic Design of Algorithms Through Evolution (ADATE) [9] is a system for automatic programming that creates functional programs using evolutionary principles. It can be used to develop new programs, or evolve and improving existing ones [10, 11, 12].

Image segmentation is closely related to edge detection. It divides the image into multiple parts, and is typically used to identify objects or other relevant information in digital images. It has been shown that ADATE is capable of drastically improving the segmentation quality of a popular and highly efficient graph-based segmentation algorithm—while retaining the computational efficiency of the original algorithm [10].

To evolve a solution to a problem, the ADATE system needs a *specification file* that defines data types and auxiliary functions, a number of training and validation input examples, and an *evaluation function* that is used to grade and select potential solutions during evolution. Additionally, the specification file may contain an initial program from which evolution will start. Of course, it is possible to start the evolution from any given program, for example to search for improvements for the best known program for a given problem.

The programs are constructed using a limited number of so-called *atomic program transformation*. The most important ones are as follows.

- **R** (**R**eplacement) - A part of an existing program is replaced by a newly synthesized expression. Due to the extremely high number of expressions that can be synthesized, **R** transformations are combinatorially expensive.
- **REQ** (**R**eplacement preserving **Eq**uality) - An **R** transformation that does not make the program worse according to the given evaluation function. REQ transformations are quite useful due to their ability to explore plateaus in the search landscape.
- **ABSTR** (**Abstr**action) - Like **REQ** transformations, these neutral transformations exist to aid the system in exploring plateaus. In contrast to the general **REQ** transformation, **ABSTR** transformations have the very specific task of introducing new functions in the program by factoring out a piece of code and replacing it with a function call. This gives the system the important ability of inventing needed help functions on the fly, something which has proven to be an extremely useful feature.

Atomic program transformations are composed to *compound program transformations* using a number of different heuristics to avoid common cases of infinite recursion, unnecessary transformations etc. For example, after an **ABSTR** transformation, the newly introduced function should be used in some way by a following **R** or a **REQ** transformation. More details on the atomic program transformations and the heuristics employed to combine them can be found in [9].

Each time a new program is created by a compound transformation, it is considered for insertion into the so-called kingdom. As in all evolutionary systems, individuals with good evaluation values are preferred, but in ADATE, the syntactic complexities of the individuals also play an important role. According to what is commonly known as Occam's Razor, simple theories should usually be preferred over more complex ones,

as the simpler theories tend to be more general. This principle is utilized by ADATE to reduce the amount of overfitting, in that small programs are preferred, and if a large program is to be allowed in the kingdom, it has to be better than all programs smaller than it [9]. In other words, a new program will only be allowed to be inserted into the kingdom if all other programs in the kingdom are either larger or worse than it. Each time a program is added to the kingdom, all programs in the kingdom both larger and worse than than the new one are removed.

Having described the most important components of the ADATE system, we conclude this section by giving a brief overview of the overall evolution occuring in a run of the system.

1. Initiate the kingdom with the single program given as the start program in the specification file (either an empty program or some program that is to be optimized). In addition to the actual programs, the system also maintains an integer value C_P for each program P, called the *cost limit* of the program. For new programs this value is set to the initial value 1000.

2. Select the program P with the lowest C_P value from the kingdom.

3. Apply C_P compound program transformations to the selected program, yielding C_P new programs.

4. Try to insert each of the created programs into the kingdom in accordance with the size-evaluation ordering described above.

5. Double the value of C_P, and repeat from step 2.

The above loop is repeated until the user terminates the system. The ADATE system has no built-in termination criteria and it is up to the user to monitor the evolving programs and halt the system whenever he considers the evolved results good enough.

3 The Experiments

The grayscale and ground truth images used to create the dataset were generated by a custom script. Each of the images contains an instance of one of the different types of edges available. Figure 1 shows some examples. Note that the edges are slightly varied in intensity and size, but that the ground truth edge is only one pixel wide and located at the center of edge. The classification dataset was made by iterating over the images, collecting values from a 5x5 neighborhood around each pixel (see Figure 2). We chose to use the 24 neighbor pixels as input attributes, and whether or not the middle pixel is an edge pixel as the target attribute. Since normally, location of the correct edge can be a matter of perception, slightly mis-localized boundaries should also be tolerated [13]. For this reason, the pixel were accepted as true if any of its closest neighbors is marked as an edge pixel, and accepted as false otherwise.

In total the dataset contains 163.840 instances with roughly 16% positive cases. Due to the size of the dataset, a fixed split into training and testing data would be too much for some of the machine learning systems to handle, since it would slow down the training process considerably. For the systems where this applies, we chose to overcome the problem by creating a suitable dataset by randomly sampling the complete dataset.

We chose to use C5.0 to assess the ability of the decision tree approach to classify the dataset. The system is capable of handling the entire dataset, so we chose to use sampling

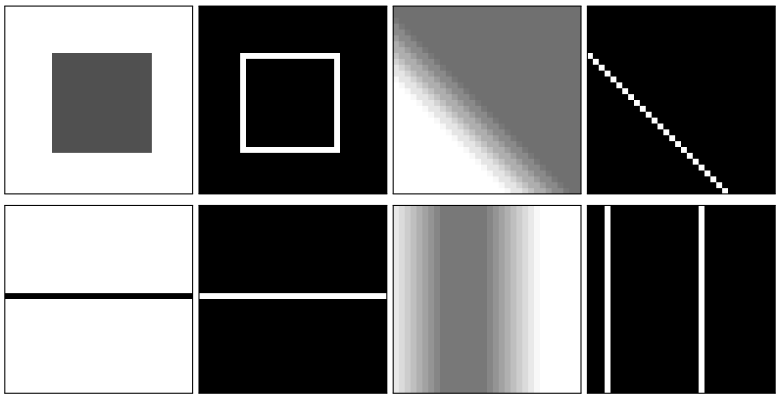

Figure 1: Step, ramp, ridge and roof edges, with ground truth

1	6	11	15	20
2	7	12	16	21
3	8		17	22
4	9	13	18	23
5	10	14	19	24

Figure 2: Order of gathering pixels from the 5x5 neighborhood

to utilize the entire dataset. 70% of the instances were used for training and 30% for testing. Due to the unbalanced nature of the dataset—with almost 5 times as many false cases as true—it is easy to get a relative low error percentage by simply classifying all cases as *false*. We would rather have the system indicating edges where there are none, than avoiding marking the real ones. We tried a number of different options in order to circumvent this issue.

To assess the performance of the neural network approach we chose to use the feedforward neural network implementation in Matlab. For this system we chose to use only 10% of the entire dataset (16.384 instances). 70% of the instances were used for training, 15% of the instances were used for testing and the remaining 15% were used for validation. The default training method for feedforward networks is the Levenberg-Marquardt backpropagation method. This method is in general the fastest training function, and it is recommended as first choice, even though it requires more memory than other algorithms [14]. More than 35 different networks were trained and tested with variations in training methods, epochs, number of layers and neurons in the hidden layers. Both the validation and test data were run manually after the training was complete.

The experiments with the ADATE system was based on the data and test files used in the C5.0 experiment, but we had to reduce the dataset. We decided to use only 10.000

rows for training and 10.000 for testing. We allowed the system to evolve programs using only boolean literals, relational operators and simple arithmetic. These building blocks are on their own enough for the ADATE system to generate decision trees similar to the ones created by c5.0, but, due to the power of automatic programming and the ADATE system, it is also capable of generating much more complex solutions. We also allowed the system to use the trigonometric function *tanh* since it has been used to evolve interesting abilities in the past.

4 Results

All the machine learning techniques produced solutions with good accuracy, suggesting that the problem might be easy to solve. Table 1 contains the time needed to train the best solution on each of the system, and Table 2 contains the results from running the best solutions on the training and test data. We have included the F-measure [15] in addition to the total error percentage.

Tool	Training time
ADATE	2-3 weeks
Neural Network	28 minutes
C5.0	2.3 seconds

Table 1: The amount of time needed to train the best solution for each of the systems

	Training data				Test data			
Tool	Error %	P	R	F	Error %	P	R	F
ADATE	0.11	0.99	1	1	0.33	0.98	0.99	0.99
Neural Networks	0.1	0.99	1	1	1.57	0.98	0.93	0.96
C5.0	2	0.89	1	0.94	2.2	0.89	0.99	0.94

Table 2: The performance of the best solutions created by the machine learning systems. P, R and F signify Precision, Recall and F-measure respectively.

The best result produced by C5.0 came after training with a misclassification cost of 3 preferring true. This gave a total error of 2% on the training data and 2.2% on test data. The solution performs significantly worse than both the best neural network and the best program created by ADATE. There are no signs of overfitting, since the solution performs equally well on both the training and test data. The solution was created in only 2.3 seconds, so the training time needed by C5.0 is far less than the time needed by the two other systems.

The neural network structure that performed best consisted of 4 layers with respectively 50, 25, 12 and 1 node in each layer. This structure got the best results when trained with several different training methods, but the Levenberg-Marquardt method performed significantly better than the others. The best network performed almost flawlessly on the training data, with only 0.1% wrong classifications. The solution performs slightly worse on the validation and test data, primarily due to a significantly lower *recall* value. The solution got 1.3% errors in total on the validation data, and a

```
1  boolean help( double x ) {
2    if ( x>0.157896483764 )
3      return x<0.276555798365;
4    else
5      return x<-0.125418003875;
6  }
7
8  boolean f( double t1, double t2, double t3, double t4, double t5, double t6,
9             double t7, double t8, double t9, double t10, double t11, double t12,
10            double t13, double t14, double t15, double t16, double t17, double t18,
11            double t19, double t20, double t21, double t22, double t23, double t24 ) {
12   double y1;
13   if ( t4<0.11954542403205382E-3 )
14     y1 = t14;
15   else
16     y1 = t8;
17
18   double y2;
19   if ( help( y1 ) ) {
20     boolean h1;
21     if( help( t6 ) )
22       h1 = true;
23     else
24       h1 = help( t10 );
25
26     if( h1 )
27       y2 = t9-t12;
28     else
29       y2 = t8;
30   }
31   else {
32     if ( help( t18 ) )
33       y2 = t18-t8+t2;
34     else
35       y2 = t7;
36   }
37
38   double y3;
39   boolean h2;
40   if ( help( t12 ) )
41     h2 = true;
42   else
43     h2 = help y2;
44   if ( h2 )
45     y3 = -292.561415835;
46   else
47     y3 = t16;
48
49   return help( y3 );
50 }
```

Listing 1: The best solution created by ADATE—rewritten in Java to simplify the interpretation.

F-measure of 0.96. This tells us that there are a relative large amount of true edge pixels in the test data that have been wrongly classified—indicating a slight overfitting of the solution to the training data.

ADATE was able to create a solution within 2-3 days with only 4,7% wrongly classified test instances. After running more than two weeks, the performance of the synthesized programs suddenly started to increase rapidly until several of them reached 100% correct classification on the training data. The best individual when considering both training and test data is listed in Java in Listing 1. This program performs similarly to the neural network solution on the training data, but it does not have same drop in performance on the test data.

The most noticeable feature in the generated program is the auxiliary function called

help on lines 1–6. This function compares a variable x to a one of two possible constants depending on the outcome of a comparison of the same variable to a third constant. All of these constants have been highly optimized—indicating that they are of high importance to the performance of the solution. This function provides a dual threshold setup that—when invoked using a difference as input—is similar to other much more complex edge detectors like Canny [4]. The function is invoked several times in the program, sometimes using one of the input variables directly, and other times using a combination of several inputs through the use of simple arithmetic.

5 Conclusions

We wanted to compare the ability of popular machine learning techniques to generate solutions that can classify a pixel as either an *edge* or *not an edge* to the ability of ADATE and automatic programming to do the same. We can conclude based on the experiments described in this paper that ADATE is capable of producing solutions that are just as good as, or even better than, the solutions created by two of most popular machine learning techniques for classification problems.

The ability of ADATE system to create powerful customized heuristics has been demonstrated once again in the domain of image analysis—providing further evidence to support the hypothesis that the system is highly suitable for solving or improving solutions to image analysis problems. This ability is likely to be equally suitable in all other domains where there are no exact solutions or if the exact solution is impossible to find in practice.

References

[1] S. Konishi, A. L. Yuille, J. Coughlan, and S. C. Zhu, "Fundamental bounds on edge detection: An information theoretic evaluation of different edge cues," in *Computer Vision and Pattern Recognition, 1999. IEEE Computer Society Conference on.*, vol. 1, IEEE, 1999.

[2] J. M. Prewitt, *Object enhancement and extraction*, vol. 75. Academic Press, New York, 1970.

[3] R. O. Duda, P. E. Hart, *et al.*, *Pattern classification and scene analysis*, vol. 3. Wiley New York, 1973.

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

[5] P. Arbelaez, M. Maire, C. Fowlkes, and J. Malik, "Contour detection and hierarchical image segmentation," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 33, no. 5, pp. 898–916, 2011.

[6] T. M. Mitchell, *Machine Learning*. McGraw-Hill, 1997.

[7] T. Bujlow, T. Riaz, and J. M. Pedersen, "A method for classification of network traffic based on c5. 0 machine learning algorithm," in *Computing, Networking and Communications (ICNC), 2012 International Conference on*, pp. 237–241, IEEE, 2012.

[8] M. Egmont-Petersen, D. de Ridder, and H. Handels, "Image processing with neural networks—a review," *Pattern recognition*, vol. 35, no. 10, pp. 2279–2301, 2002.

[9] R. Olsson, "Inductive functional programming using incremental program tranformation," *Artificial Intelligence*, vol. 74, pp. 55–81, 1995.

[10] L. V. Magnusson and R. Olsson, "Improving graph-based image segmentation using automatic programming," 2013.

[11] H. Vu and R. Olsson, "Automatic improvement of graph based image segmentation," in *Advances in Visual Computing*, pp. 578–587, Springer, 2012.

[12] A. Løkketangen and R. Olsson, "Generating meta-heuristic optimization code using adate," *Journal of Heuristics*, vol. 16, no. 6, pp. 911–930, 2010.

[13] D. R. Martin, C. C. Fowlkes, and J. Malik, "Learning to detect natural image boundaries using local brightness, color, and texture cues," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 26, no. 5, pp. 530–549, 2004.

[14] A. Ranganathan, "The levenberg-marquardt algorithm," *Tutoral on LM Algorithm*, 2004.

[15] C. J. V. Rijsbergen, *Information Retrieval*. Newton, MA, USA: Butterworth-Heinemann, 2nd ed., 1979.