Available at
www.ComputerScienceWeb.com
POWERED BY SCIENCE DIRECT®

Applied Soft Computing 3 (2003) 123–137

Applied Soft Computing

www.elsevier.com/locate/asoc

Competitive fuzzy edge detection

Lily Rui Liang, Carl G. Looney*

Computer Science and Engineering Department, 171 University of Nevada, Reno, NV 89557, USA

Received 14 February 2002; received in revised form 3 January 2003; accepted 23 January 2003

Abstract

Our fuzzy classifier detects classes of image pixels corresponding to gray level variation in the various directions. It uses an extended Epanechnikov function as a fuzzy set membership function (FSMF) for each class where the class assigned to each pixel is the one with the greatest fuzzy truth of membership. This classification is done first, after which a competition is run as a second step to thin the edges. Like the Canny edge detector, the edge sensitivity of our competitive fuzzy edge detector (CFED) can be set from low to high by the user. The performance of our algorithm is somewhat similar to that of the Canny algorithm but ours is significantly faster. For both, the proper level of sensitivity must be chosen by the user for the best results because the tradeoff is more edges with more noise versus fewer edges and less noise. However, the settings are less sensitive and more intuitive for our algorithm. We make comparisons on good and degraded images.

Keywords: Fuzzy classifier; Edge detection; Competitive edge selection

1. Introduction

Edge pixels are defined as locations in an image where there is a significant variation in gray level (or intensity level of color) pixels [5] in a fixed direction across a few pixels. Edge pixels form curved or straight boundaries. Edges are one of the most important visual clues for interpreting images [6]. Edge detection is by far the most common approach for detecting meaningful discontinuities in the gray level. The process of edge detection reduces an image to its edge details that appear as the outlines of image objects that are often used in subsequent image analysis operations for feature detection and object recognition. While this is usually white lines on black backgrounds, we prefer to avoid the excessive use of printer toner and the non-uniform shading on black printed regions by making black line drawings on white backgrounds.

There are many different methods for edge detection [5,6], such as Sobel filtering, Prewitt filtering, Laplacian of Gaussian filtering, moment-based operators, the Shen and Castan operator and the Canny and Deriche operator, but some common problems of these methods are a large volume of computation, sensitivity to noise, anisotropy and thick lines. Russo [13,14], and also Russo and Ramponi [15], designed fuzzy rules for edge detection. Such rules can smooth while sharpening edges, but require a rather large rule set compared to simpler fuzzy methods [8]. Neural networks can be trained to detect edges [10] and radial basis functional link nets [9] are especially powerful for this, but here we develop a special fuzzy classifier for edges that does not require training.

A *fuzzy classifier* is a system that accepts inputs that are either: (i) feature vectors; or (ii) vectors of

* Corresponding author. Tel.: +1-775-784-4313; fax: +1-775-784-1877.
E-mail addresses: liang@cs.unr.edu (L.R. Liang), looney@cs.unr.edu (C.G. Looney).

1568-4946/$ – see front matter
doi:10.1016/S1568-4946(03)00008-5

fuzzy truths for the features to belong to various *fuzzy set membership functions* (FSMFs). It outputs fuzzy truths for the memberships of the input vector in the various classes. The class assigned to an input feature vector is the one with the maximum fuzzy truth given by the FSMFs. We usually require the maximum to exceed the second greatest fuzzy truth by a certain amount to yield a unique class membership (otherwise we can only say that the input feature vector belongs to each class with a particular fuzzy truth). Different types of fuzzy classifiers are used in [1,12,18,20] for other purposes.

An earlier fuzzy classifier [11,21] created *extended ellipsoidal Epanechnikov* functions as the fuzzy set membership functions centered on the class prototypes. Such classifiers were inspired by *probabilistic neural networks* [2,3,16,17] but avoid the higher extraneous error that is due to large mixtures of Gaussians in Parzen windows [19]. Our non-competitive fuzzy classifier [7] does not implement an edge thinner and has five classes. Its advantages are easy modeling, efficient computation, low sensitivity to noise and isotropy, but its disadvantage is that its lines are thick as are the lines obtained by most edge detectors such as by thresholding with XView for Unix/Linux and LView Pro (for Windows) or by applying the Sobel or other edge operators. We develop the competitive edge modification here that thins edges, and we also employ other enhancements.

Our new *competitive fuzzy edge detector* (CFED) not only detects edge pixels in the first step, but applies competitive rules as a second step for the purpose of thinning the ridges around local maxima in difference magnitude. It detects a portion of a ridge or embankment that can be rather broad in the case of diffuse edges, which in some methods results in a thick band of pixels in the edge map [5]. A third step despeckles by removing single and double pixel noise specks.

2. Methodology

2.1. The feature vector for a pixel

Fig. 1 shows the 3×3 neighborhood of pixels about the center pixel p_5 as well as the four directions in which edges may appear. The *bi-directional* summed magnitude differences in gray level between p_5 and its neighbors are designated by d_1, d_2, d_3 and d_4 for Directions 1, 2, 3 and 4, respectively, are shown in Fig. 1 and are calculated by

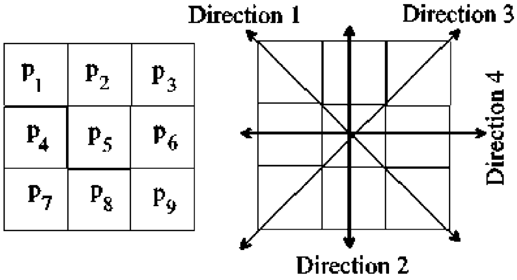


Fig. 1. Pixels and directions in a 3×3 nbhd.

$$d_1 = |p_1 - p_5| + |p_9 - p_5| \quad \text{(Direction 1)} \tag{1a}$$

$$d_2 = |p_2 - p_5| + |p_8 - p_5| \quad \text{(Direction 2)} \tag{1b}$$

$$d_3 = |p_3 - p_5| + |p_7 - p_5| \quad \text{(Direction 3)} \tag{2a}$$

$$d_4 = |p_4 - p_5| + |p_6 - p_5| \quad \text{(Direction 4)} \tag{2b}$$

For each pixel in an input image that is not on the outer boundary of the image, we compute a four-dimensional *feature vector* $\boldsymbol{x} = (d_1, d_2, d_3, d_4)$ of gray level summed magnitude differences in four directions on its 3×3 neighborhood. The magnitudes make each difference d_j bi-directional.

2.2. Pixel edge classes

Our new fuzzy classifier differentiates pixels into four *edge classes*, a *background class* and a *speckle edge class* (a speckle is a noisy pixel). Four typical neighborhood *situations* are used for each edge class: each directional edge neighborhood shown in Fig. 2, its rotation by 180° and the exchange of darker and lighter pixels in each of these two cases.

Each set of four situations for a class has a single feature vector of summed magnitudes of differences as far as the low and high values are concerned. The background class is for any pixel whose neighborhood has low magnitude differences in the four directions. A speckle edge class is used for pixels on whose neighborhood the change magnitudes in all directions are high (this class is shown in Fig. 3c and d as examples. Given a pixel, any neighborhood has a situation that determines a feature vector such as $\boldsymbol{x} = (3, 35,$

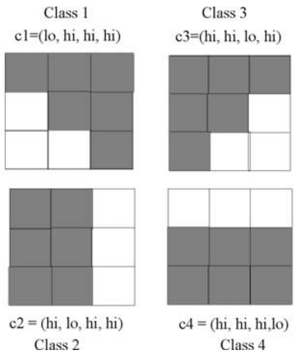


Fig. 2. Edge classes.

26, 41), of magnitudes of differences in each of the four directions shown in Fig. 1. We construct six prototype vectors $\boldsymbol{c}_0, \ldots, \boldsymbol{c}_5$ to be the respective centers of the six classes (four edge, one background and one speckle edge classes). These centers, or prototypes, for the respective classes have component values 'lo' and 'hi' that represent low and high summed magnitude differences in the directions indicated. The parameters lo and hi are to be set by the user and depend on the image region contrasts and the sensitivity desired. These class centers for the situations, some of which are displayed in Fig. 2, are listed in Table 1. All other combinations are mapped to white.

In practice, the values of lo and hi are defined by the user for each particular image to achieve a desirable result. For example, lo could be assigned to a gray level difference of 5 and hi could be set to a value from 30 to 40. These low and high values determine the prototypes $\boldsymbol{c}_0, \ldots, \boldsymbol{c}_5$ that are the center points of the six fuzzy set membership functions for the six classes. Other neighborhoods such as those shown in Fig. 3

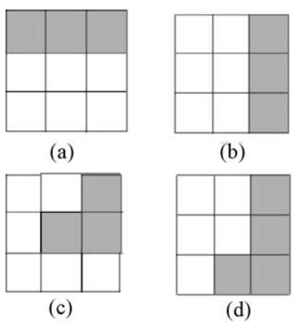


Fig. 3. Other nbhds.

Table 1
The classes and their prototype vectors

Class 0 (background)	$\boldsymbol{c}_0$ = (lo, lo, lo, lo)
Class 1 (edge)	$\boldsymbol{c}_1$ = (lo, hi, hi, hi)
Class 2 (edge)	$\boldsymbol{c}_2$ = (hi, lo, hi, hi)
Class 3 (edge)	$\boldsymbol{c}_3$ = (hi, hi, lo, hi)
Class 4 (edge)	$\boldsymbol{c}_4$ = (hi, hi, hi, lo)
Class 5 (speckle edge)	$\boldsymbol{c}_5$ = (hi, hi, hi, hi)

may indicate that the center pixel is an edge pixel or is next to an edge pixel. In Parts (c) and (d), the pixel p_5 is in the speckle edge class, but is initially mapped to an edge. Parts (a) and (b) display neighborhoods of regular edges.

The FSMFs are dome shaped symmetrical functions (upside-down cups) determined by a center vector $\boldsymbol{c}_i$ and a width parameter w that also must be set by the user. They are defined in the next section. Every image interior pixel must be recognized as belonging to a class centered on one of these prototypes or else it is mapped to the background (white) because it is a purer form of speckle than edge speckle.

2.3. The fuzzy classifier architecture

Fig. 4 shows the original *fuzzy classifier* [11,21] architecture (with two outputs here for two classes) that we used as a starting model. Each node in the hidden (middle) layer represents an extended Epanechnikov

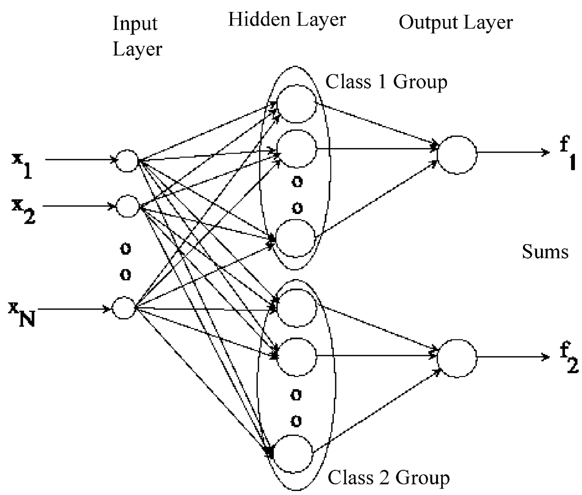


Fig. 4. Fuzzy classifier diagram.

fuzzy set membership function centered on a prototype $\boldsymbol{c}_j$ (see Eqs. (3a)–(3e)), where each class has one or more prototypes. We have also successfully used Gaussians as the FSMFs but they take more time to evaluate by the computer and are positive everywhere. The input feature vector $\boldsymbol{x} = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N)$ activates certain fuzzy set membership functions of which at least one will go relatively high provided the domain regions where the FSMFs are positive over the feature domain.

In the original fuzzy classifier of Fig. 4, the output layer contains a single node for each class to sum the values passed to it by the hidden layer nodes in its class group. When one of the FSMFs at a node in the hidden layer goes high, the summing node in the output layer for that class also goes high. The output layer node with the maximum value determines the class.

This fuzzy classifier led to our first, non-competitive fuzzy edge detector [7], which performs edge/non-edge classification but results in thick edges. Our new CFED is made up of: (i) the *fuzzy classifier*, modified to detect the six classes, and (ii) a set of *competitive rules* that implement a competition between neighboring edge pixels across the edge width for designation as an edge (see [3,4] for other competitions).

Our new edge detector uses a single node for each class in the hidden (middle) layer and so we do not need the output layer to sum the output values from the hidden nodes. Thus, we have only two layers in the CFED network, in contrast to Fig. 4. The CFED feeds the input feature vectors directly to six output nodes, which have extended Epanechnikov FSMFs that classify a pixel as one of four types of edges, a non-edge or a speckle edge. One of these FSMSs will be a maximum.

We next apply a competitive rule to each edge pixel according to its assigned class. Only the pixels that are first classified as edge pixels and then win in the edge competition, or are speckle edges, are mapped to black pixels in the new output image. All other pixels are mapped to white. This creates a thinner black line drawing on a white background. On the four-dimensional feature space we define the fuzzy set membership functions for the six classes as extended Epanechnikov [11] functions by Eqs. (3a)–(3f) for any input feature vector $\boldsymbol{x}$.

Class 0 (background)

$$\mu_0(x) = \max\left\{0, \frac{1 - ||\boldsymbol{x} - \boldsymbol{c}_0||^2}{w^2}\right\} \tag{3a}$$

Class 1 (edge) $$\mu_1(x) = \max\left\{0, \frac{1 - ||\boldsymbol{x} - \boldsymbol{c}_1||^2}{w^2}\right\} \tag{3b}$$

Class 2 (edge) $$\mu_2(x) = \max\left\{0, \frac{1 - ||\boldsymbol{x} - \boldsymbol{c}_2||^2}{w^2}\right\} \tag{3c}$$

Class 3 (edge) $$\mu_3(x) = \max\left\{0, \frac{1 - ||\boldsymbol{x} - \boldsymbol{c}_3||^2}{w^2}\right\} \tag{3d}$$

Class 4 (edge) $$\mu_4(x) = \max\left\{0, \frac{1 - ||\boldsymbol{x} - \boldsymbol{c}_4||^2}{w^2}\right\} \tag{3e}$$

Class 5 (speckle edge)

$$\mu_5(x) = \max\left\{0, \frac{1 - ||\boldsymbol{x} - \boldsymbol{c}_5||^2}{w^2}\right\} \tag{3f}$$

The width (or spread) parameter w is the radius of these FSMFs and w must be large enough so the *support* (the region where a function is non-zero) of each FSMF combines to cover the cube $[0, 255]^4$ domain (the diagonal distance of this cube is $\{4 \times 256^2\}^{1/2} = 512$. Thus, the quality of the edge detection, as measured by the fuzzy truth of its memberships in the fuzzy classes, depends on the parameters lo, hi, and w (and on the image contrast and the purpose of the edges). We can use a value of, say, 200–256 for w.

For easy visualization, Fig. 5 provides a three-dimensional portrayal of only two features, rather than four, versus fuzzy truth. The extended Epanechnikov functions [11] are shown here with small diameters for clarity. In practice, they overlap so that each input feature vector falls into one or more of the fuzzy set membership functions. Such functions are dome shaped.

2.4. The competitive rules

Before an edge pixel is mapped to either white or black in the output image, a competition with its neighbor edge pixels is done. Once a pixel is classified as an edge class, it competes with the two edge pixels on

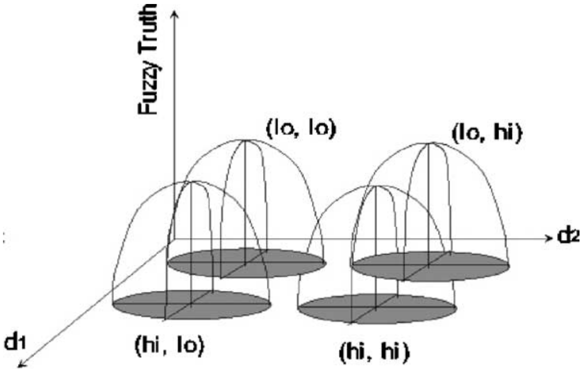


Fig. 5. A three-dimensional view of the FSMFs.

either side of it across the edge width. For these three pixels, only the one with the largest difference magnitude is saved as a black edge (the others are saved as white background). Thus, the edges are thinned. The rules for this competition are given below:

IF ***x*** is Class 0 (background) THEN change pixel to white.

IF ***x*** is Class 1 (edge) THEN
compete d_3 with neighbor pixels in Direction 3
IF it wins THEN change it to black (edge) ELSE change to white.

IF ***x*** is Class 2 (edge) THEN
compete d_4 with neighbor pixels in Direction 4
IF it wins THEN change it to black (edge) ELSE change to white.

IF ***x*** is Class 3 (edge) THEN
compete d_1 with neighbor pixels in Direction 1
IF it wins THEN change it to black (edge) ELSE change to white.

IF ***x*** is Class 4 (edge) THEN
compete d_2 with neighbor pixels in Direction 2
IF it wins THEN change it to black (edge) ELSE change to white.

IF ***x*** is Class 5 (speckle edge) THEN change pixel to black (edge).

3. The algorithm

Because the speckle edge maps to a black edge pixel and is not always an edge, we implement a *despeckler* that removes isolated single and isolated double edge pixels from the edges after the fuzzy classification and edge competition have been done. We also implement a *smoother* before any edge detection is done, but we make this selectable by the user because it does not need to be used except when noise is above a certain level. Such smoothing uses a 3×3 mask of all 1's except the center, which is 2 and the multiplier is 1/10 (a mild smoother).

The CFED algorithm operates on grayscale PGM images via three passes through the image. It first makes a pass across all interior pixels (not on the image boundary) and classifies each pixel as belonging to Class 0, 1, 2, 3, 4, or 5. The classification is done by putting the feature vector $\boldsymbol{x} = (d_1, d_2, d_3, d_4)$ for each pixel through each of the six extended Epanechnikov functions [11] to obtain their fuzzy truths of membership in one of the corresponding six classes. The largest fuzzy truth determines the class membership. The second and third passes thin edges and despeckle. The high level pseudo code follows.

3.1. Fuzzy classification

Step 1: set parameters lo, hi and w; open image file;
select smooth/no smooth,
if smoother selected then smooth image.

Step 2: for each pixel in the image,
compute and save the directional summed magnitudes of differences,
construct the feature vector $\boldsymbol{x}$,
compute the six fuzzy truth values: $\mu(\boldsymbol{x})$, $i = 0, \ldots, 5$,
determine maximum fuzzy truth and record pixel class for pixel.

After all pixels have been classified, then a second pass is made where the class of each pixel is examined. If it is an edge pixel (in Classes $1, \ldots, 4$), then the direction is determined to use for examining its adjacent edge pixels. Of three pixels in that direction across the 3×3 neighborhood, the one with the maximum sum of magnitude difference is selected as the edge pixel, which is changed to black in the output image. Despeckling is done next on the third pass.

3.2. Edge strength competition

Step 1: for each pixel in the image,
  if edge class then apply appropriate rule and record pixel value,
  if background class then write white pixel,
  if speckle edge class then write black pixel.

3.3. Despeckling

Step 1: for each pixel in the image,
  if isolated single or double speckle then change to white.

4. Experimental results

4.1. Edge results

All of our results were obtained by using a 3×3 neighborhood centered in turn on each interior pixel. The center parameters, lo, hi and the width parameter w must be provided to achieve good results by positioning and spreading the extended Epanechnikov fuzzy set membership functions. In practice, different people may look for different details in the same image, so those parameters should be input by the user to obtain the desired type of edge [22]. For example, we could put lo $= 4$, hi $= 48$ and $w = 240$. A smaller hi value yields more sensitivity to edges (and displays more noise), whereas a larger lo value maps more of the weak edges to the background. Any value of w greater than 200 appears to yield similar results.

To obtain results for the Canny edge detector we used Matlab 6 (Release 12). The results are white edges on black background, so we take the negative for comparison with the CFED. The Matlab command for Canny edge detection is

$$\text{Output\_Image} = \text{edge}(\text{Input\_Image}, \text{`canny'}, T, \sigma)$$

where "Output_Image" and "Input_Image" are the respective output edge image and input image. The Canny parameters that must be input by the user are the 'upper threshold' T (upper edge sensitivity) and 'sigma' σ (the Gaussian parameter).

The standardized parameter T ranges from 0 up to 1.0 in Matlab 6, which also uses the Canny 'lower threshold' t for finer edges and sets it to $t = 0.4T$ by default (we did not get better results by setting t independently). The default σ value is 1.0. We found that a smaller threshold T gives more detail (and noise). Making σ smaller also gives more detail but without the noise. T is the most sensitive. Appendix A displays tradeoffs for the Canny edge detector of T and F on *building.tif*. Appendix B shows some tradeoffs for CFED parameters lo, hi and w on the same image. Figs. A-6 and A-7 have the most detail without too much noise for the Canny method. Fig. B-1 has the most detail (sensitivity) and essentially no noise with the CFED, for which no smoothing was done.

The threshold T can be interpreted as the minimum probability for an edge to be an actual edge. The lower threshold t can be interpreted as the maximum probability for neighbor pixels to not be part of the edge. The Canny operator first smooths the image by a Gaussian convolution. It next uses σ to make a first pass that assigns a probability to each possible edge pixel. Then a simple two-dimensional derivative operator is applied to the smoothed image to highlight regions with high first spatial derivatives. Edges cause ridges in the gradient magnitude image, on which the algorithm tracks the top points (greater than T) and sets them to be edges. It sets all pixels not on top of the ridge to background. When the ridge falls below the lower threshold t, the tracking stops.

Fig. 6 shows the original image *building.tif*. The results of the Canny edge detector on this image are shown for both high and low sensitivity, respectively, in Figs. 7 and 8 with σ fixed at 0.5. Decreasing σ would

Fig. 6. The original building image.

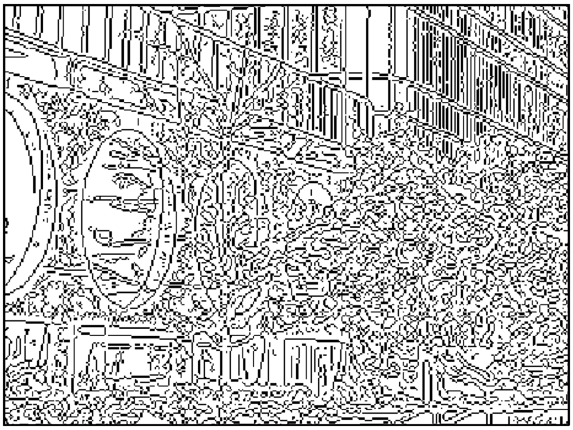


Fig. 7. Canny edges, high sensitivity ($T = 0.02$, $\sigma = 0.5$).

show more detail without increasing the noise significantly. The results for our competitive fuzzy edge detection for the higher sensitivity to edges are presented in Fig. 9.

Figs. 10 and 11 show the best results that we obtained with the respective CFED and Canny methods on several parameter settings (in our judgement, which is a subjective decision). There is no ground truth against which to make a measure of success here so the human eye and subjectivity determine the winning results. Recall that the CFED uses a despeckler afterwards and the Canny uses a smoother beforehand. The T value of 0.04 is large enough so as to not reveal too many weak edges as noise, but the σ value of 0.6 was small enough to reveal many details.

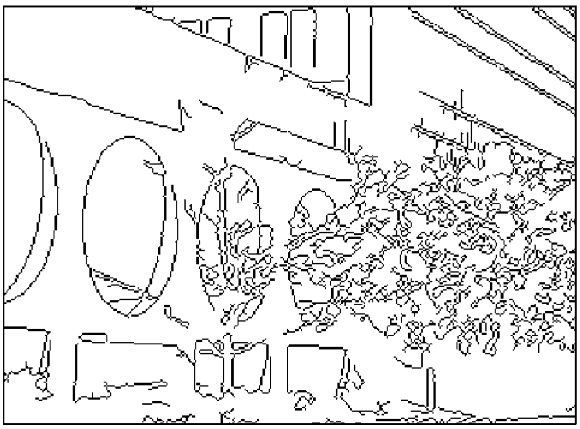


Fig. 8. Canny edges, low sensitivity ($T = 0.2$, $\sigma = 0.5$).

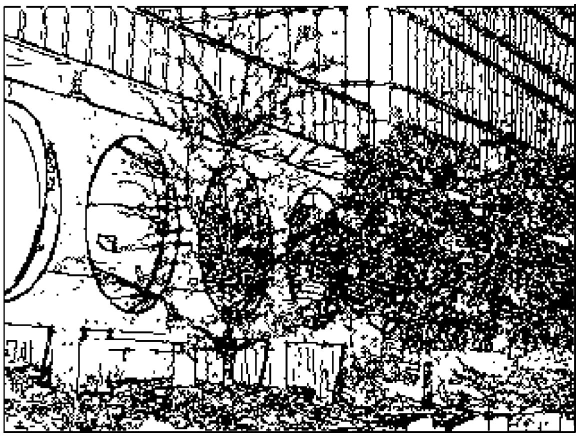


Fig. 9. CFED edges, higher sensitivity (lo = 0, hi = 20, $w = 256$).

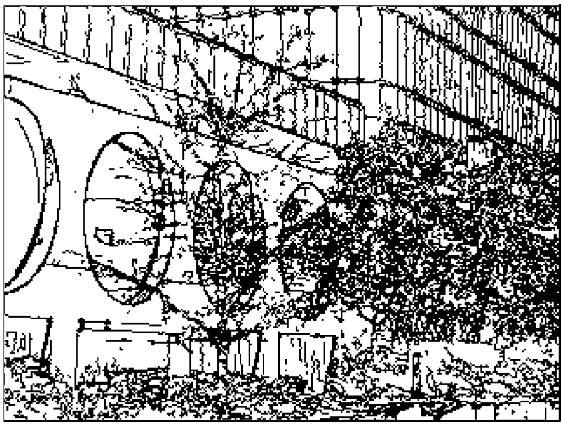


Fig. 10. CFED edges, best sensitivity (lo = 0, hi = 25, $w = 256$).

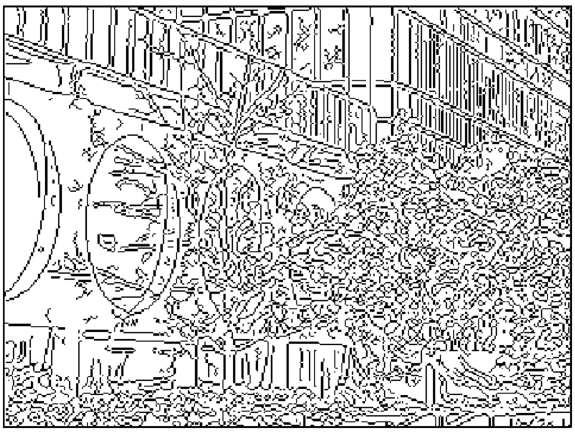


Fig. 11. Best Canny result from many trials ($T = 0.04$, $\sigma = 0.6$).

Fig. 12. XView edges (threshold = 20).

We tried histogram equalization before applying both the Canny edge detector and our CFED, but it made the noisy edges worse. Lowering the contrast actually helped with the edges that were not too weak. But contrast must be traded off with edge sensitivity. We found the Canny edge detector more difficult to use because of the interaction of T and σ.

Fig. 12 displays the result of thresholding to black and white with the popular Unix/Linux based shareware tool XView. Fig. 13 shows the result of using our original non-competitive fuzzy edge detector [7]. These have low noise levels, moderate detail and thick edges.

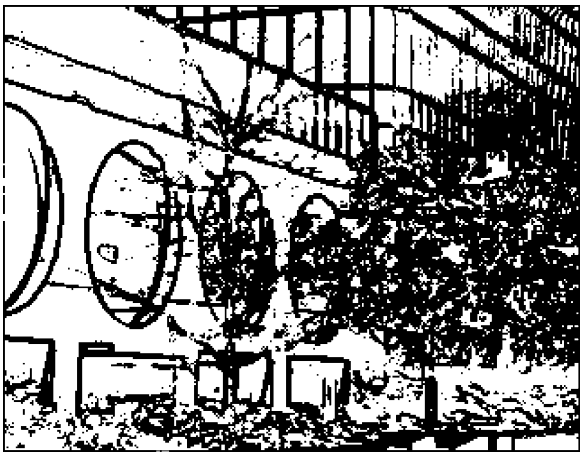


Fig. 13. Fuzzy edge detection only.

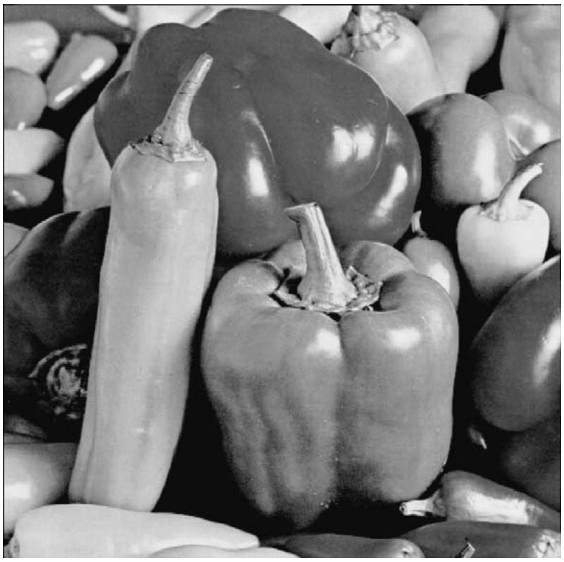


Fig. 14. The original peppers image.

Fig. 14 is the well-known *peppers* image. The result of the Canny edge detector with a good sensitivity setting is shown in Fig. 15, while a perhaps better Canny result (less noise) is shown in Fig. 16. Fig. 17 is the result of our CFED with low-moderate sensitivity (and despeckling). We note that the Canny processed

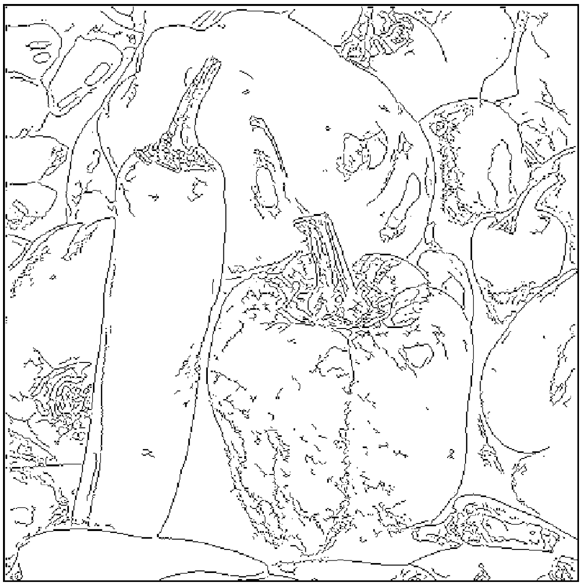


Fig. 15. Canny edge detected peppers ($T = 0.04$, $\sigma = 0.5$).

Fig. 16. Canny detected peppers ($T = 0.1$, $\sigma = 1.0$).

Fig. 18. XView edges (threshold = 20).

images cannot be improved by despeckling because the noise is coarse and connected by tracking. Fig. 18 displays the result of using the program XView to threshold to black and white with the threshold gray level set to 20.

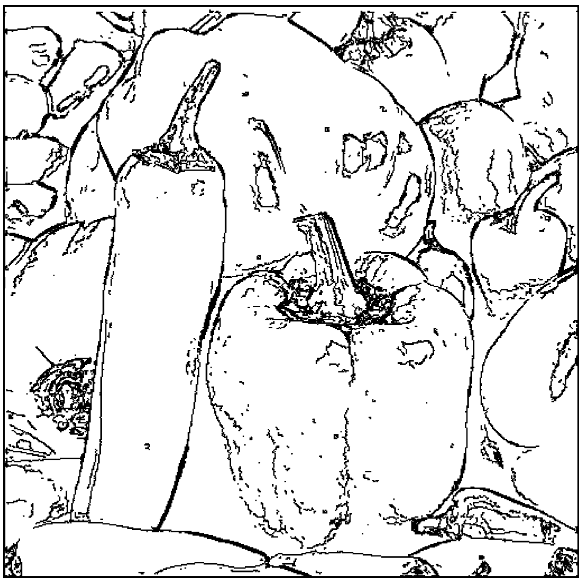


Fig. 17. Competitive fuzzy edges of peppers (lo = 0, hi = 20, $w = 256$).

Fig. 19 is the building image that has been strongly corrupted with uniform noise that has large average value (from 0 to 255, truncated to remain in this range) to test the Canny and CFED algorithms on noisy images. Figs. 20–24 show the results of Canny with different parameter settings, while Figs. 25–27 show those from the CFED with different parameter settings. We used the smoother with the CFED only on these runs with the uniform noise (Canny always smooths before edge detecting).

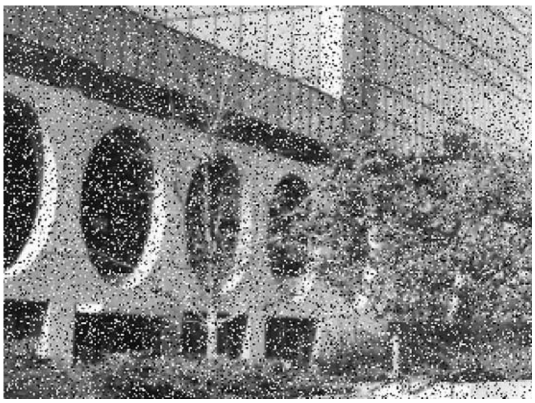


Fig. 19. Noise degraded building image.

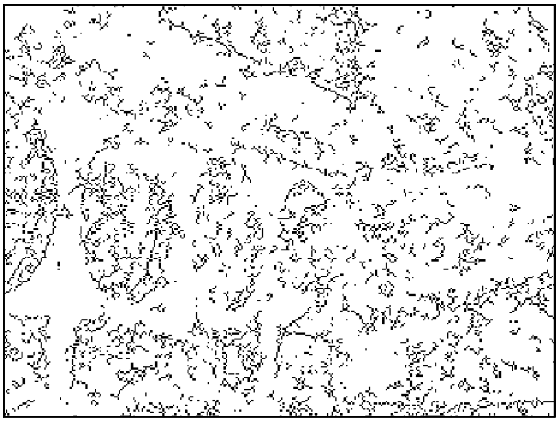


Fig. 20. Canny ($T = 0.3$, $\sigma = 1.0$).

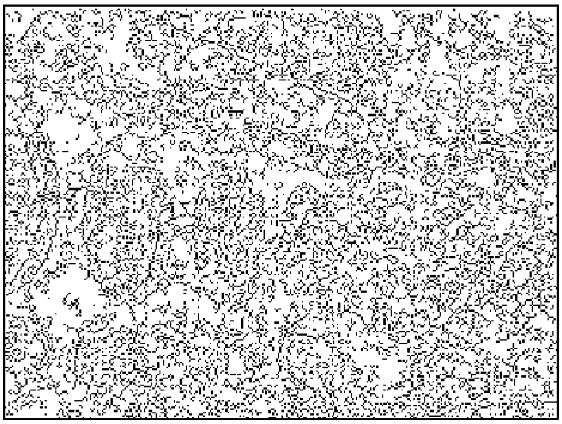


Fig. 21. Canny ($T = 0.2$, $\sigma = 0.8$).

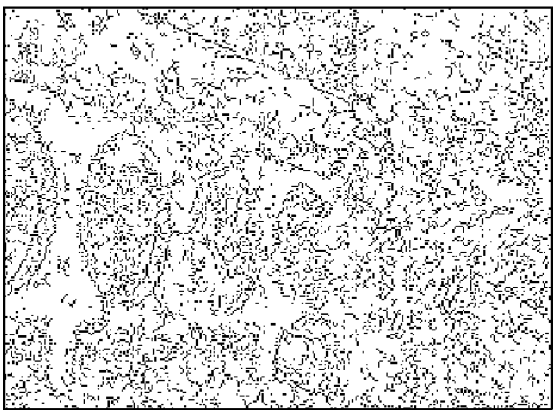


Fig. 22. Canny ($T = 0.2$, $\sigma = 1.4$).

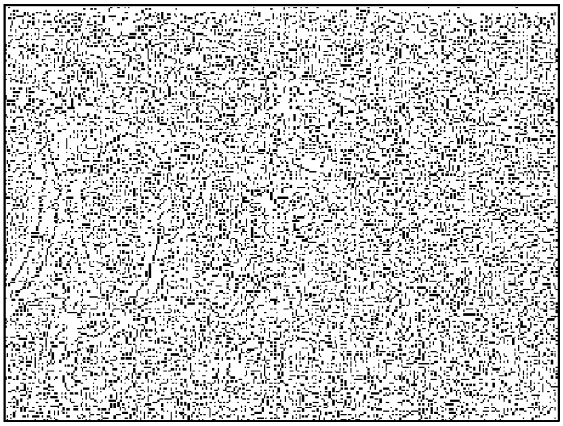


Fig. 23. Canny ($T = 0.1$, $\sigma = 1.8$).

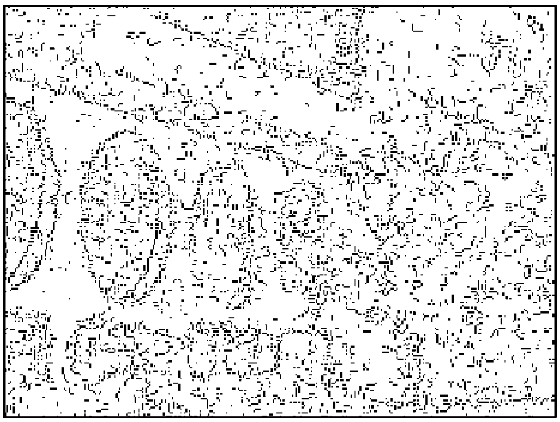


Fig. 24. Canny ($T = 0.2$, $\sigma = 2.0$).

Fig. 25. CFED (lo $= 0$, hi $= 20$, $w = 200$).

Fig. 26. CFED (lo = 0, hi = 30, $w = 200$).

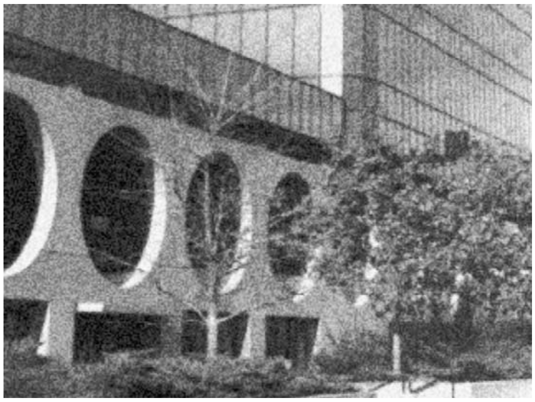


Fig. 28. Building with Gaussian noise.

Fig. 28 is the building image degraded with Gaussian noise that has smaller mean-square error. Figs. 29–32 show the Canny results for various good parameters. Figs. 33–35 show some CFED results. The Canny results were significantly worse than those of the CFED on the uniform noise, but only slightly worse on the Gaussian noise.

4.2. Speed results

Our compiled C program for the basic CFED algorithm (no smoothing nor despeckling) on the building image (240 × 320) took about 0.49 s to run on a Sun Sparc 64 bit processor running at 266 MHz when the algorithm time included reading in the image file from the hard drive and then writing it out. The Canny algorithm running from the Matlab 6, Release 12 command line took 2.3 s on the same machine, but it had already read in the image file and did not write it out. On the *peppers* image, which is 512 × 512, our method required 4–8.2 s for the Canny algorithm.

When we eliminated the reading from the timing, but left in the writing to a memory buffer, which Matlab does, the basic CFED program took 0.1 s for the *building* image and 0.338 s for the *peppers* image. Thus, the basic CFED computes in about

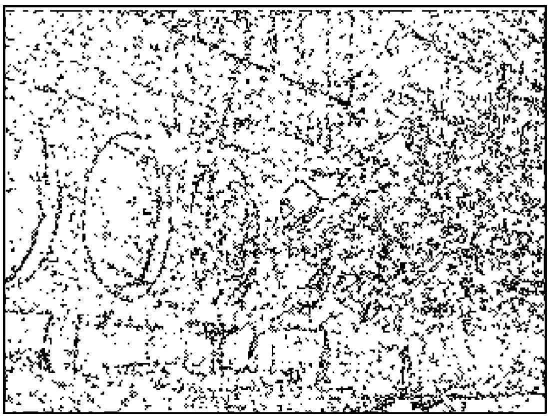


Fig. 27. CFED (lo = 20, hi = 50, $w = 256$).

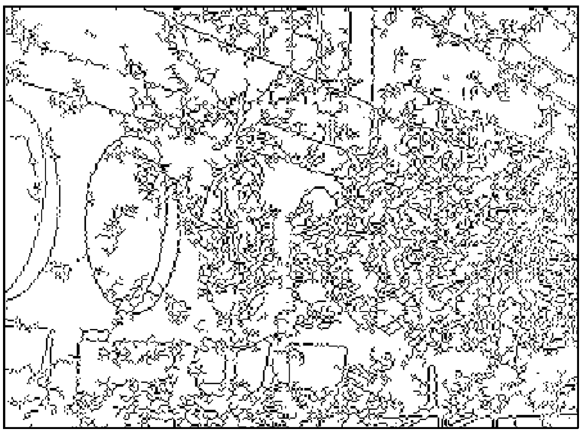


Fig. 29. Canny ($T = 0.2$, $\sigma = 0.4$).

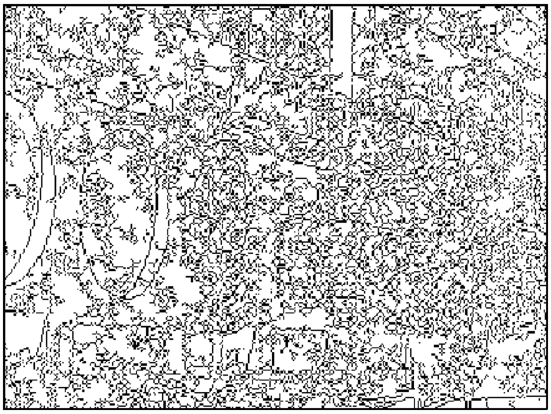


Fig. 30. Canny ($T = 0.2$, $\sigma = 0.1$).

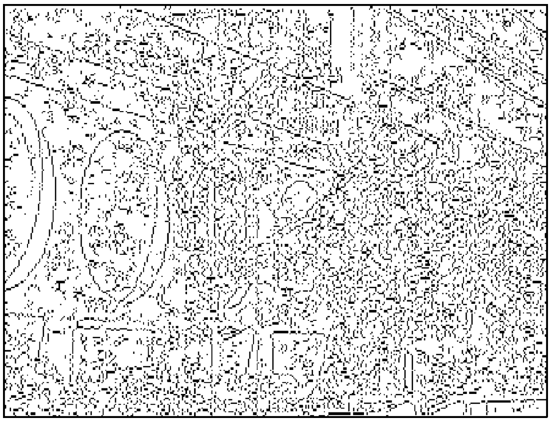


Fig. 31. Canny ($T = 0.1$, $\sigma = 1.0$).

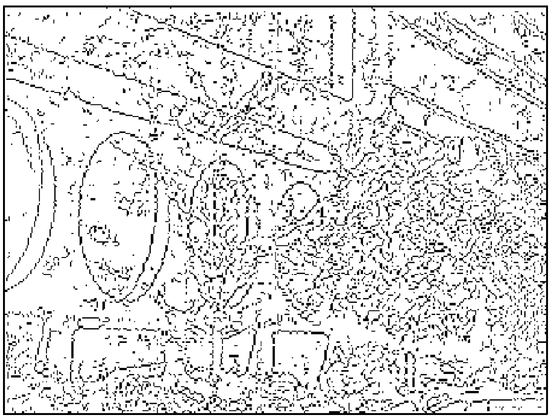


Fig. 32. Canny ($T = 0.1$, $\sigma = 1.4$).

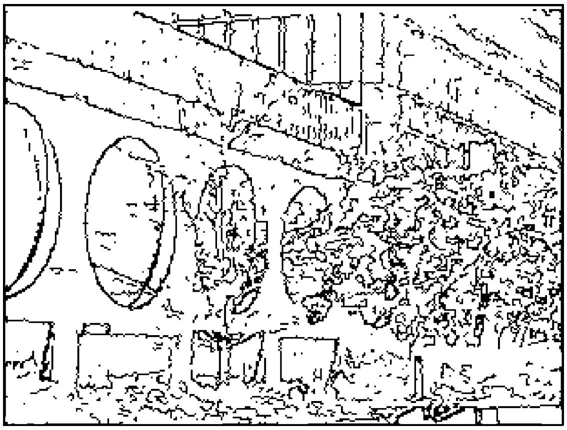


Fig. 33. CFED (lo $= 0$, hi $= 40$, $w = 256$).

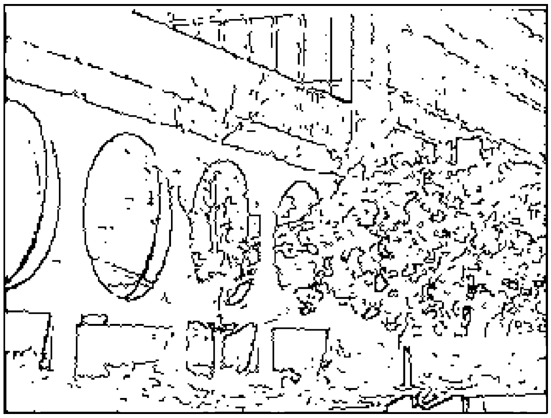


Fig. 34. CFED (lo $= 0$, hi $= 50$, $w = 256$).

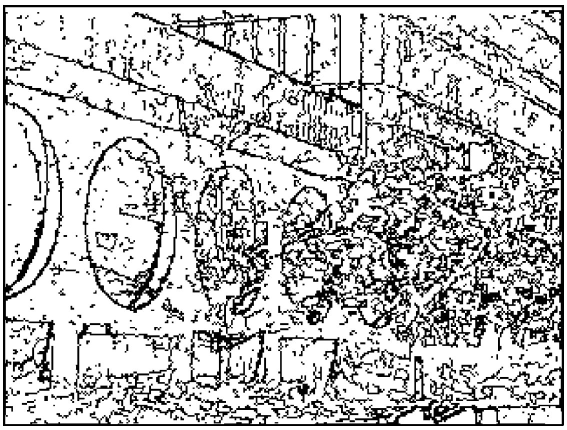


Fig. 35. CFED (lo $= 0$, hi $= 30$, $w = 256$).

4.2–4.5% of that of Canny and so is more than 20 times faster than the Matlab Canny. We rarely use smoothing, but adding in the despeckler requires much less time than the basic CFED algorithm, so the result is that the CFED with despeckling is still more than 10 times faster than the Canny edge detector.

5. Conclusions

We have put the neighborhood summed magnitudes of differences on a 3×3 neighborhood of each pixel into a feature vector and fed it into a new type of fuzzy classifier to classify a pixel as a type of edge or background. Using the competitive rules on the pure edge types, the results are a line drawing of moderately thin black lines on a white background, whereas some methods yield thick lines and/or more noise (e.g. thresholding, the Prewitt and Sobel operators). A competition is applied to consecutive edges across the edge width to thin the lines to yield good line drawings of edges.

The benefits of using our CFED model in edge detection are:

- it yields moderately thin black lines even when the edge in the input image is diffuse;
- it is fast with only six simple fuzzy set membership functions (the extended Epanechnikov functions reduce the computation further);
- the method works well even when the intuitive parameters are adjusted somewhat coarsely;
- the process is isotropic in that lines of all directions are detected equally well.

In making runs with the Canny and CFED software, we found that it is easy to select CFED parameters that yield good results whereas the Canny edge detector may require many more runs using different combinations of parameters (although a very good tradeoff can be found after a sufficient numbers of runs). The CFED does not perform contour tracking as does the Canny edge detector, which at times tends to connect lines into a closed contour that should be separated. Our algorithm computes much less time than that required for the Canny algorithm, which can be significant for large images. Finally, the CFED does as well or better on images degraded with noise.

Future work will use dynamic parameters of lo and hi that adjust over the image to be more or less sensitive where needed, depending on contrast and differences. It will also allow different values of w for the different FSMFs. We will also explore the use of weak edges in lighter, but still dark, shades of gray to add more information into the edge map.

Acknowledgements

Supported by US Army Research Office Grant DAAD19-99-1-0089.

Appendix A. Runs on *building.tif* with Matlab 6 Canny edge detector

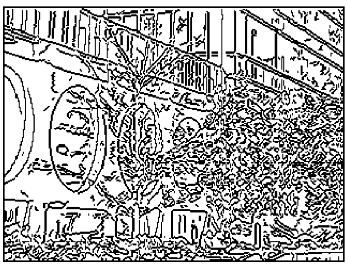


Fig. A-1. (*T = 0.04, σ = 1.0*).

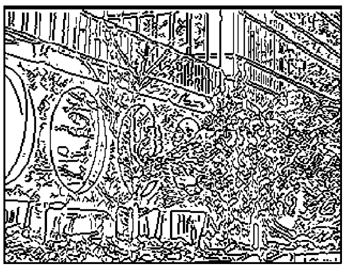


Fig. A-2. (*T = 0.02, σ = 1.0*).

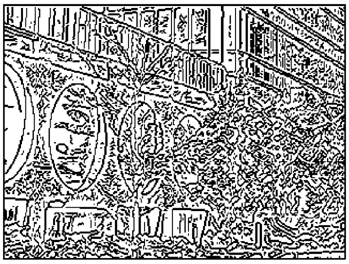


Fig. A-3. (*$T = 0.01$, $\sigma = 1.0$*).

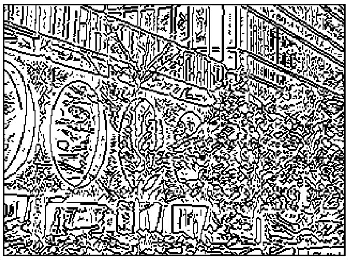


Fig. A-4. (*$T = 0.005$, $\sigma = 1.0$*).

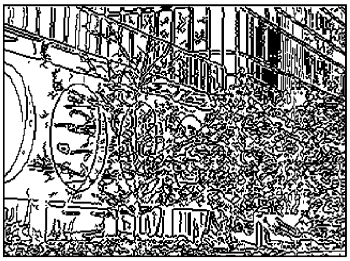


Fig. A-5. (*$T = 0.04$, $\sigma = 0.6$*).

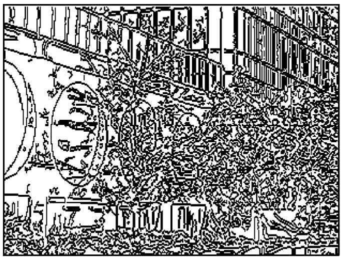


Fig. A-6. (*$T = 0.04$, $\sigma = 0.3$*).

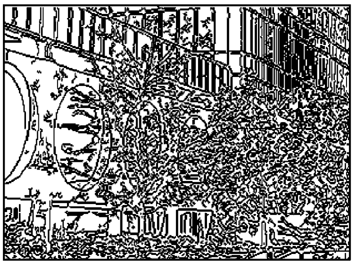


Fig. A-7. (*$T = 0.04$, $\sigma = 0.15$*).

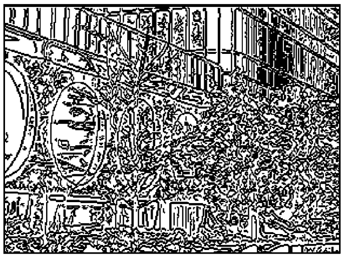


Fig. A-8. (*$T = 0.02$, $\sigma = 0.5$*).

Appendix B. Runs on *building.tif* with CFED

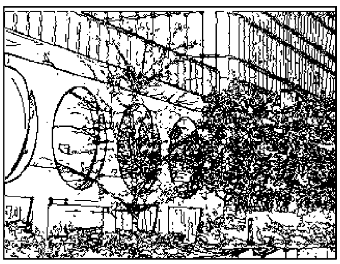


Fig. B-1. (*lo = 0, hi = 25, w = 256*).

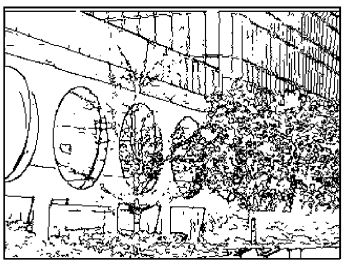
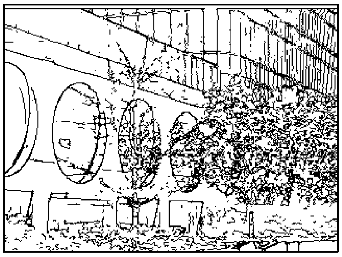


Figure B-2. (*lo = 0, hi = 45, w = 256*).



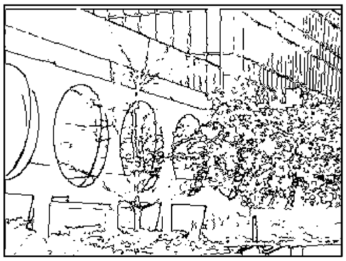


Fig. B-3. (*lo = 0, hi = 45, w = 1000*). Figure B-4. (*lo = 15, hi = 45, w = 256*).

References

[1] S. Abe, R. Thawonmas, A fuzzy classifier with ellipsoidal regions, IEEE Trans. Fuzzy Syst. 5 (2) (1997) 358–368.

[2] C. Anagnostopoulos, J. Anagnostopoulos, D. Vergados, E. Kayafas, V. Loumos, G. Stassinopoulos, A neural network and fuzzy logic system for face detection on RGB images, in: Proceedings of the ISCA International Conference on Computers and Their Applications, Seattle, March 2001, pp. 233–236.

[3] K. Chen, H. Chi, A method of combining multiple probabilistic classifiers through soft competition on different feature sets, Neurocomputing 20 (1998) 227–252.

[4] F.L. Chung, T. Lee, Fuzzy competitive learning, Neural Netw. 7 (3) (1994) 539–551.

[5] N. Efford, Digital Image Processing, Addison-Wesley, Reading, MA, 2000, pp.164–173.

[6] E. Gose, R. Johnsonbaug, S. Jost, Pattern Recognition and Image Analysis, Prentice-Hall, Englewood Cliffs, NJ, 1996, p. 298.

[7] L.R. Liang, E. Basallo, C.G. Looney, Image edge detection with fuzzy classifier, in: Proceedings of the ISCA 14th International Conference, Las Vegas, 2001, pp. 279–283.

[8] C.G. Looney, Nonlinear rule-based convolution for refocusing, Real Time Imaging 6 (2000) 29–37.

[9] C.G. Looney, Radial basis functional link nets and fuzzy reasoning, Neurocomputing 48 (1–4) (2002) 489–509.

[10] C.G. Looney, Pattern Recognition Using Neural Networks, Oxford University Press, New York, 1997.

[11] C.G. Looney, A Fuzzy Classifier Network with Ellipsoidal Epanechnikovs, Technical Report, Computer Science Department, University of Nevada, Reno, NV, 2001.

[12] H. Maturino-Lozoya, D. Munoz-Rodriguez, F. Jaimes-Romero, H. Tawfik, Handoff algorithms based on fuzzy classifiers, IEEE Trans. Vehicular Technol. 49 (6) (2000) 2286–2294.

[13] F. Russo, A new class of fuzzy operators for image processing, Proc. IEEE Int. Conf. Neural Netw. 1 (1993) 815–820.

[14] F. Russo, A user-friendly research tool for image processing with fuzzy rules, in: Proceedings of the First IEEE International Conference on Fuzzy Systems, San Diego, 1992, pp. 561–568.

[15] F. Russo, G. Ramponi, Fuzzy operator for sharpening of noisy images, IEE Electron. Lett. 28 (1992) 1715–1717.

[16] D.F. Specht, Probabilistic neural networks for classification, mapping or associative memory, Proc. IEEE Int. Conf. Neural Netw. 1 (1988) 525–532.

[17] D.F. Specht, Probabilistic neural networks, Neural Networks 1 (3) (1990) 109–118.

[18] M. Valenzuela-Rendon, Reinforcement learning in the fuzzy classifier system, Expert Syst. Appl. 14 (1–2) (1998) 237–247.

[19] D.K. Wedding II, K.J. Cios, Certainty factors versus Parzen windows as reliability measures in RBF networks, Neurocomputing 19 (1998) 151–165.

[20] C.-C. Wong, C.-C. Chen, S.-L. Yeh, K-means-based fuzzy classifier design, IEEE Int. Conf. Fuzzy Syst. 1 (2000) 48–52.

[21] http://www.cs.unr.edu/∼looney/cs791j/unit4.

[22] http://prettyview.com/edge/.