Chapter 6
Image Processing and CGP

Lukas Sekanina, Simon L. Harding, Wolfgang Banzhaf and Taras Kowaliw

6.1 Introduction

Computerized image processing has been studied intensively for many years [32]. Because of the inherent complexity of the problem, stochastic optimization algorithms, including evolutionary algorithms, have been applied to improve existing image processing methods and develop new algorithms [4].

Section 6.2 deals with the automatic design of low-level image filters, ones comparable in quality to conventional filters. Moreover, when the evolved filters are constructed as single-purpose circuits in a field-programmable gate array (FPGA), the implementation cost is usually lower than the cost of conventional solutions. We will describe the basic approach to filter evolution and its extensions, such as the use of a bank of evolved filters, and filtering using extended kernels.

Evolutionary design of more advanced image operators such as dilation/erosion filters is presented in Sect. 6.3. As these filters are composed of relatively complex elementary functions (sine, square root etc.), they are primarily intended for advanced image processing software tools.

An image classification task will be described in Sect. 6.4, where CGP graphs are used to define transformations in the case of a medical image problem. We will show that CGP image transformations may be used to significantly improve classification accuracy with respect to a collection of predefined image features.

6.2 Evolutionary Design of Image Filters for FPGAs

Image preprocessing (which includes image filtering, edge detection, histogram equalization, brightness and contrast adjustment, and other operations on images) is the first stage of many image processing applications. The quality of preprocessing significantly influences the accuracy, reliability, robustness and performance of

J.F. Miller (ed.), *Cartesian Genetic Programming*, Natural Computing Series,
DOI 10.1007/978-3-642-17310-3_6,

subsequent image processing steps such as segmentation, classification and recognition. In order to perform the required preprocessing (such as image filtering, edge detection) a problem-specific filter has to be created. Traditionally, engineers use a library of predefined filters and operators and manually tune promising variants of these filters for a given application. In the process of tuning, various properties of the filters might be optimized, in particular their coefficients and structure [3, 7, 24]. There are other important parameters to be optimized, such as the area, delay and power consumption, when the goal is to implement the filter as a digital circuit.

Historically, linear filters have been the most popular filters for image processing. Their popularity is due to the existence of robust mathematical models which can be used for their analysis and design. However, there exist many areas in which nonlinear filters provide significantly better results [6]. The advantage of nonlinear filters lies in their ability to preserve edges and suppress noise without loss of detail. As there is no suitable general theory for the design of nonlinear operators, evolutionary design techniques have been utilized to accomplish this task in recent years.

The first utilization of CGP for image filter design was due to Sekanina [26]. It was shown later that CGP can automatically design image filters that are competitive with filters designed conventionally in terms of filtering quality and the cost of implementation on a chip. In this section, we will briefly survey how CGP can be used to evolve human-competitive image filters and edge detectors. In order to compare the results produced by CGP, we will briefly describe the most popular conventional methods that are utilized to suppress selected types of nonlinear noise. We will take into account greyscale images only; however, the concept can be naturally extended to colour images.

6.2.1 Sliding-Window Function

Software and hardware implementations of image filters operate mostly in the spatial domain. As spatial filters operate with pixel values in the neighbourhood of the centre pixel (this neighbourhood is called the window or kernel), it is necessary to implement a local neighbourhood function (sometimes referred to as sliding-window function). This function is applied separately to all pixel locations. This function is also invariable for all locations (i.e. spatially invariant). Figure 6.1 shows the concept of the sliding window function and the utilization of CGP for the evolutionary design of such a function.

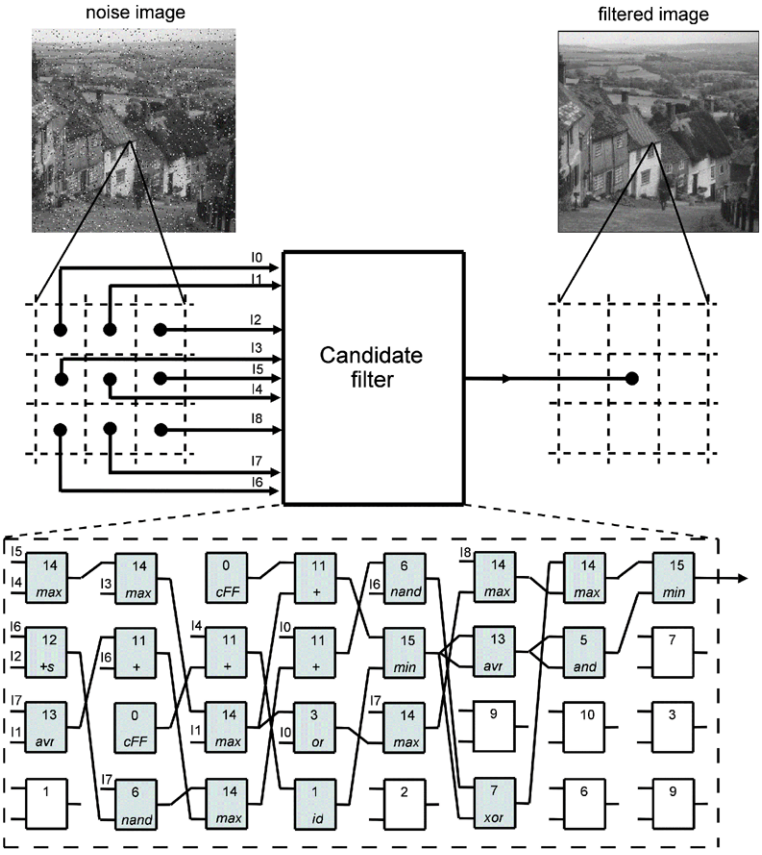


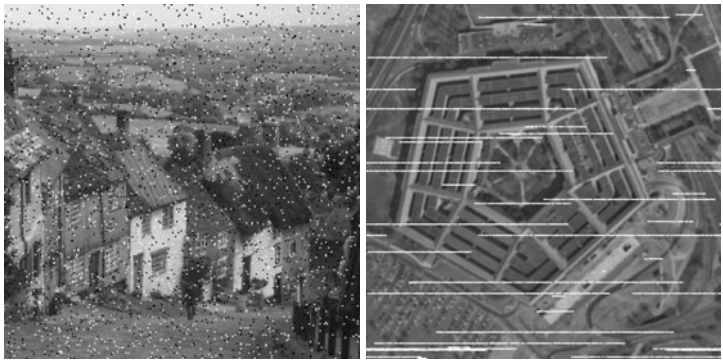
Fig. 6.1 Candidate local neighborhood function represented in CGP.

6.2.2 Types of Noise

Impulse noise is a frequently encountered type of noise. In most cases, impulse noise is caused by malfunctioning pixels in camera sensors, faulty memory locations in hardware or errors in data transmission. We distinguish two common types of impulse noise: the salt-and-pepper noise (also referred to as intensity spikes or speckle) and random-valued shot noise. For images corrupted by salt-and-pepper noise (see Fig 6.2a), the noisy pixels can take only the maximum or minimum value. In the case of the random-valued shot noise, the noisy pixels have an arbitrary value. Impulse noise cab be characterized by one parameter the noise intensity p, which gives the ratio of the number of corrupted pixels to the total number of pixels.

Impulse burst noise typically occurs in remote sensing images such as satellite images (see Fig. 6.2b). The main reason for the occurrence of bursts is interfer-

ence in the frequency-modulated carrying signal caused by signals from other data sources. This interference can occur several times during the transmission of a single image and corrupt several image pixels in one or more neighbouring rows. Impulse burst noise is a specific kind of noise which is difficult to filter out even if a non-linear filter is used. We will show in Sect. 6.2.8.3 that median filters are capable of suppressing impulse burst noise but, at the same time, they usually damage image detail too heavily.



(a) **(b)**

Fig. 6.2 Types of noise: (a) salt-and-pepper noise (5%), (b) impulse burst noise (5%).

6.2.3 Conventional Filters

Traditionally, impulse noise is suppressed by a median filter, the most popular non-linear filter. In this case the output of the local neighbourhood function is calculated from the median value of the kernel. However, the standard median filter gives poor performance for images corrupted by impulse noise of high intensity. A simple median filter utilizing a 3×3- or 5×5-pixel window is sufficient only when the noise intensity is less than approximately 10–20%. Various approaches have been proposed, to overcome this deficiency, including switching median filters [43], weighted median filters [2], weighted order-statistic filters [22] and adaptive median filters [15] (see e.g. [25] for a detailed survey of these methods). In addition to median filters, some other algorithms exist (e.g. [23]). Almost all filters of this type have already been implemented in hardware (see [40] for a survey of hardware implementations).

The adaptive median filter provides significantly better results than do standard median filters, especially for images corrupted by a high noise intensity [15]. The adaptive median filter operates with a kernel of $S_{max} \times S_{max}$ pixels. Let S_{xy} denote the processed (i.e. central) pixel. The kernel is processed by a set of sorting net-

works with $3 \times 3, 5 \times 5, \ldots, S_{max} \times S_{max}$ inputs. Each sorting network provides the minimum, maximum and median value of its window. The output value generated by the adaptive median filter is calculated on the basis of comparison of the outputs of the sorting networks as described in [15]. Figure 6.3 compares the results of applying some conventional filters to suppress 40% salt-and-pepper noise. Reasonable results can be obtained with an adaptive median filter; however, the implementation cost is too high for FPGAs (see Table 6.3).

Fig. 6.3 Images obtained by using conventional filters. (a) Original image. (b) Noisy image with 40% salt-and-pepper noise (peak signal-to-noise ratio (PSNR) 9.364 dB). (c) Filtered by median filter with a kernel size 3×3 (PSNR 18.293 dB). (d) Filtered by median filter with a kernel size 5×5 (PSNR 24.102 dB). (e) Filtered by adaptive median filter with a kernel size of up to 5×5 (PSNR 26.906 dB). (f) Filtered by adaptive median filter with a kernel size up to 7×7 (PSNR 27.315 dB).

In addition to various median-based filters, specific filters have been developed for impulse burst noise, such as training-based optimized soft morphological filters and variational approaches [6, 23, 19, 18]. Unfortunately, they are not suitable for hardware implementation.

6.2.4 Edge Detectors

Edge detection can be considered as image filtering too. The Sobel operator, which performs a 2D spatial-gradient measurement on an image, is one of the most popular edge detectors [32]. It utilizes two convolution kernels, which are defined as

$$p = \frac{1}{8}\begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix}, \qquad q = \frac{1}{8}\begin{pmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{pmatrix},$$

By applying these two kernels, we can calculate the new pixel value as

$$NewPixelValue = c + |p| + |q|,$$

where c is a suitable constant, (e.g. $c = 128$).

6.2.5 Basic Approach to Filter Evolution

Every image filter operating with a 3×3-pixel kernel can be considered as a function (a digital circuit in the case of a hardware implementation) of nine eight-bit inputs and a single eight-bit output, which processes greyscale (eight-bits/pixel) images. As Fig. 6.1 shows, every pixel value of the filtered image is calculated using the corresponding pixel and its eight neighbours in the processed image.

A candidate filter was represented using $n_c \times n_r$ nodes, where the typical grid size was $n_c = 8$, $n_r = 4$. The settings of the other CGP parameters were $n_i = 9$, $n_o = 1$, $l = 1$, $n_a = 2$ and $\lambda = 8$. Each node represented a two-input function which receives two eight-bit values and produces an eight-bit output. An eight-bit node output was utilized to ensure a straightforward connectivity of the nodes in the hardware implementation. Table 6.1 lists the functions that were confirmed as useful for this task [27]. We note that these functions are also suitable for hardware implementation (i.e. there are no complex functions, such as multiplication or division).

Table 6.1 List of functions implemented in each programmable node

Code	Function	Description	Code	Function	Description
0	255	Constant	8	$x \gg 1$	Right shift by 1
1	x	Identity	9	$x \gg 2$	Right shift by 2
2	$255 - x$	Inversion	10	$swap(x,y)$	Swap nibbles
3	$x \vee y$	Bitwise OR	11	$x + y$	+ (addition)
4	$\bar{x} \vee y$	Bitwise $\bar{x}$ OR y	12	$x +^S y$	+ with saturation
5	$x \wedge y$	Bitwise AND	13	$(x+y) \gg 1$	Average
6	$\overline{x \wedge y}$	Bitwise NAND	14	$max(x,y)$	Maximum
7	$x \oplus y$	Bitwise XOR	15	$min(x,y)$	Minimum

CGP uses a single genetic operator – mutation – which modified 5% of the chromosome. Slany and Sekanina have analyzed various genetic operators for this task; however, the mutation-only search was confirmed to be the most successful [31]. The initial population was randomly generated. The evolution was usually terminated when a predefined number of generations was exhausted.

In order to evolve an image filter capable of removing a given type of noise, an original uncorrupted (training) image was needed to determine the fitness values of the candidate filters. The goal of CGP was to minimize the difference between the original image and the filtered image. The generality of the evolved filters (i.e. whether the filters could operate sufficiently well also for other images containing the same type of noise) was tested by means of a test (validation) set. The quality of filtering had to be expressed numerically. The peak signal-to-noise ratio (PSNR) is usually used in image processing for this purposes. The PSNR is defined was

$$\mathrm{PSNR} = 10\log_{10}\frac{255^2}{(1/MN)\sum_{i,j}(v(i,j)-w(i,j))^2}, \tag{6.1}$$

where $N \times M$ is the size of the image, v denotes the filtered image and w denotes the original image. Note that the higher the PSNR value the better the filter. However, computing the PSNR value is expensive, especially for hardware accelerators (see Sect. 7.3. Hence in most cases the fitness function was based on calculating the mean difference per pixel (MDPP)

$$MDPP = \frac{1}{MN}\sum_{i=1}^{M}\sum_{j=1}^{N}|v(i,j)-w(i,j)|, \tag{6.2}$$

and the goal of evolution was to minimize the MDPP value.

6.2.6 Bank of Evolved Filters

In order to create a salt-and-pepper noise filter which could provide similar filtering quality to the adaptive median filter and which would also suitable for hardware implementation, a bank of evolved filters (or, simply, a filter bank) was proposed in [36, 39]. A filter bank combines several simple image filters that have been designed by the basic approach using 3×3-pixel kernel. As Fig. 6.4 shows, the procedure has three steps: (1) reduction of the dynamic range of the noise, (2) processing using a bank of n filters and (3) deterministic selection of the result.

Step 1. It has been shown that evolved filters have problems with a high dynamic range of corrupted pixels (0–255). A straightforward solution to this problem is to create a component which inverts all pixels with the value 255, i.e. all spikes are transformed to have a uniform value. This operation is performed by the first stage of the filter bank.

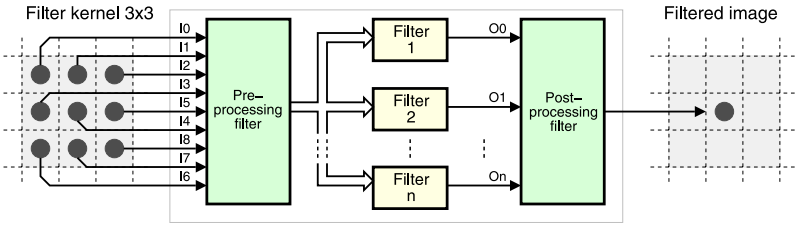


Fig. 6.4 Removal of salt-and-pepper noise using a filter bank.

Step 2. The Preprocessed image then enters a bank composed of n filters that operate in parallel. Because the evolutionary design of salt-and-pepper noise filters can be repeated many times, it is possible to obtain various different but similarly performing implementations of the filter. The bank of filters is then made up from the best-performing filters that have been evolved. In the present work, all these filters were designed by CGP using the same type of noise and training image and with the same aim – to remove 40% salt-and-pepper noise using 3×3-pixel kernel.

Step 3. Finally, the outputs from banks $1, \ldots, n$ have to be combined together. One possible solution is to use the n-input median function. Another type of aggregation function was proposed in [30].

6.2.7 Extended Kernel

Because of the nature of impulse burst noise, it is necessary to utilize a kernel larger than 3×3 pixels. Unfortunately, the image filter design method described in Sect. 6.2.5 is not scalable to larger filtering windows (e.g. if the number of inputs is increased to 25). In order to simultaneously support larger filtering windows and leave the problem reasonably difficult for evolution, CGP was extended by means of a selector that determined the pixels of a filtering window that would be used in evolved nine-input filters [35] (see Fig. 6.5). The selector was encoded as a string consisting of S bits, joined to the chromosome. If the bit of the selector string corresponding to a given pixel of the filter window had a logic value of 1, then the pixel was selected; otherwise, the pixel was not used as an input to the filtering logic. A special mutation operator was also developed for mutation of the selector [35].

6.2.8 Experimental Results

The applicability and performance of the CGP-based design method has been investigated in several papers dealing with evolution of filters (for Gaussian noise [26], impulse noise of lower intensity [29], salt-and-pepper noise of high intensity [36],

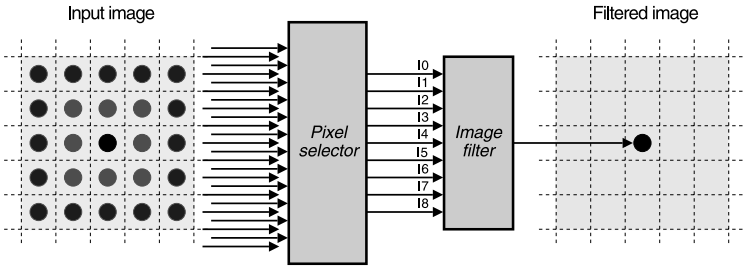


Fig. 6.5 The concept of filtering using a 5×5 filter kernel followed by a selector.

and impulse burst noise [35]), edge detectors [27] and edge detectors combined with impulse noise removal [38]. Extensive testing was carried out to analyze the optimal choice of the population size, mutation rate, size of the CGP array, size of the training image [28], set of node functions [27], pseudo-random number generator [38] and search algorithm [37]. We will briefly summarize the most important results for the basic method for a bank of evolved filters and for filtering with an extended kernel.

6.2.8.1 Filter Evolution Using a 3×3-Pixel Kernel

Table 6.2 summarizes the results that were obtained for the best filter for Gaussian noise, salt-and-pepper noise (5% intensity), random-valued shot noise and edge detection according to [28]. In these experiments, 500 independent runs (50,000 generations in each run) were performed for all noise types using a 256×256 pixel training image (the Lena image), a population size of 4, a mutation rate of 10 bits/chromosome and 4×8 programmable nodes. The third column of Table 6.2 indicates the generation in which the best filter was discovered. The last column gives the results for conventional filters. The MDPP is the mean difference per pixel (see Eqn. 6.2).

Table 6.2 Parameters of the best filters evolved using the Lena image

Problem	Best MDPP	Generations	Avg. MDPP	Std. dev.	Avg. generations	Conventional
Salt-and-pepper 5%	0.31	49808	0.95	0.51	29388	2.95 (median)
Random shot 5%	1.11	40616	1.65	0.31	26781	2.98 (median)
Gaussian noise	6.36	27179	6.73	0.24	31032	6.43 (mean)
Edge detection	1.16	49608	1.73	0.41	35074	–

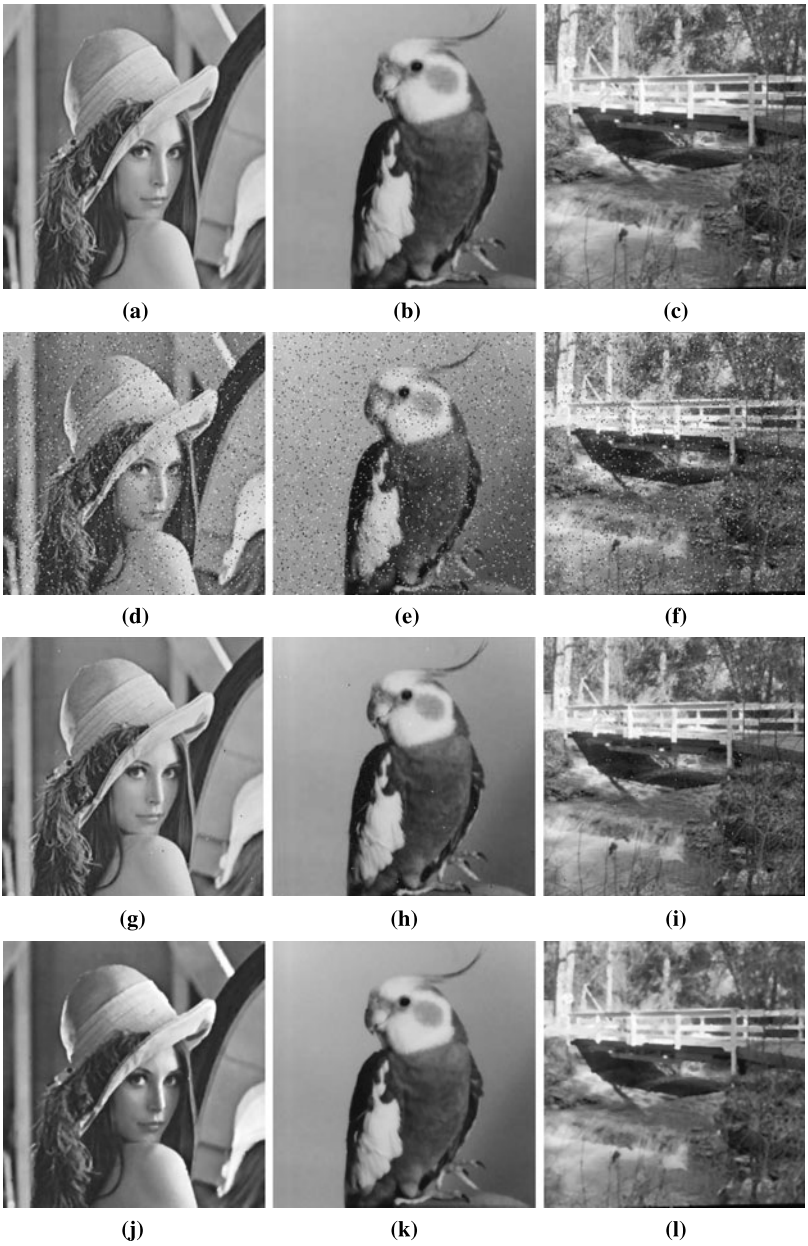


Fig. 6.6 Original images (a, b, c), images with salt-and-pepper noise (d, e, f), images filtered using an evolved filter (g, h, i) and images filtered using a median filter (j, k, l). The evolved filter was trained via the Lena image (a, d).

The evolved filters were applied to other images to check whether the discovered filtering algorithms were general enough. The images in Fig. 6.6 demonstrate that the visual quality of the filtered images was sufficient and that the evolved filters worked correctly for the class of images considered. We can see from Fig. 6.6 that the median filter does not preserve details in the images and tends, in fact, to smudge the images. On the other hand, the evolved filters are good at preserving details. The performance and cost of this type of filter will be compared further with those of other filters in Sect. 6.2.8.2.

The following C program represents an implementation of the best-evolved edge detector, created according to Fig. 6.7. Note that `kernel` denotes the nine inputs of the image filter. Figure 6.8 compares the effect of the Sobel edge detector and an evolved edge detector on one of the validation images.

```
uint8 filter(uint8 kernel[9]) {
  uint i14,i17,i19,i22,i27,i29;

  i14 = min((kernel[1] + kernel[7]) >> 1, kernel[7]);
  i17 = i14 ^ kernel[7];
  i19 = min(i14 + (255 - kernel[1]), 255);
  i22 = 255 - i19;
  i27 = min(i22, (i17 + i19) >> 1);
  i29 = min(i22 + i27, 255);
  return (i27 + i29) & 0xff;
}
```

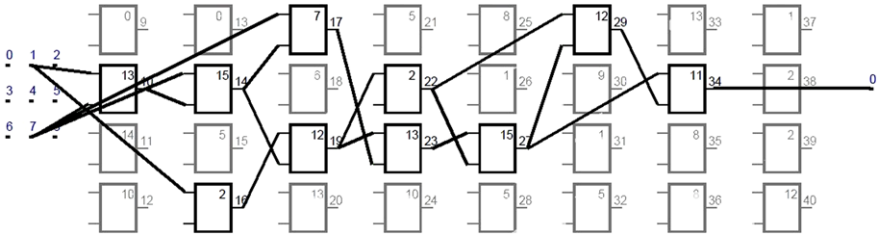


Fig. 6.7 The best edge detector from [37]. The node functions are numbered according to Table 6.1.

(a) input **(b)** Sobel operator **(c)** evolved filter

Fig. 6.8 Input image (a), and application of the Sobel edge detector (b) versus an evolved edge detector (c).

6.2.8.2 Bank of Evolved Filters

In order to construct a bank of filters, various filter implementations were evolved using the method described in Sect. 6.2.5. A 128×128-pixel training image, partially corrupted by 40% salt-and-pepper noise, was used in the fitness function. CGP operated with an eight-member population and 5% mutation. A single run was terminated after 200,000 generations. Three of the evolved filters were utilized in the bank filter and applied to suppress the 40% salt-and-pepper noise. The difference between the output of the filter bank and that of the adaptive median filter is almost invisible in Fig. 6.9.

Fig. 6.9 Comparison of images filtered using an adaptive median filter with a kernel size of up to 7×7 (a, b, c) and using a three-bank filter (d, e, f).

An FPGA platform was used for comparison of the implementation cost of median filters, adaptive median filters, evolved simple filters and filter banks [36]. The FPGA implementation was obtained in such a way that the CGP chromosome was transformed to VHDL code, which was then synthesized for a particular FPGA. Registers were automatically inserted in suitable places in order to allow the filter to be pipelined. The results of the synthesis (including the number of configurable logic blocks (CLBs)) are given for the relatively large Virtex II Pro XC2vp50-7 FPGA, which contains 23,616 slices (configurable elements of the FPGA), in Table 6.3. We can see that the filter bank requires a considerably smaller area on the chip than do the adaptive median filters, whose implementation was based on area-demanding sorting networks. One of the evolved filter banks is now protected using a utility model (a form of patent) by the Czech Industrial Property Office [30].

Table 6.3 Results of synthesis for various filters

Filter	Kernel size	CLBs	Max. frequency
Median filter	3×3	268	305 MHz
Median filter	5×5	1506	305 MHz
Median-column	5×1	69	310 MHz
Adaptive median	5×5	2024	303 MHz
Adaptive median	7×7	6567	298 MHz
FIB3	3×3	72	281 MHz
FIB5	5×5	108	242 MHz
single evolved	3×3	200	318 MHz
Three-bank filter	3×3	843	305 MHz

Figure 6.10 compares the results obtained from evolved filters (a single filter, and a three-bank filter) with conventional solutions (median filters (MFs) and adaptive median filters (AMFs)) using 25 images corrupted by salt-and-pepper noise of various intensities. We can observe that a single filter evolved using a 3×3-pixel kernel represents a better solution than the median filter if the noise intensity is low. For noise of high intensity, the bank of evolved filters is a better solution than the adaptive median filters. Nevertheless, neither the filter bank nor the adaptive median filters achieve the quality of the 'best SW algorithm' [5]. But that solution is not suitable for hardware implementation, as it does not utilize the concept of a local filtering window.

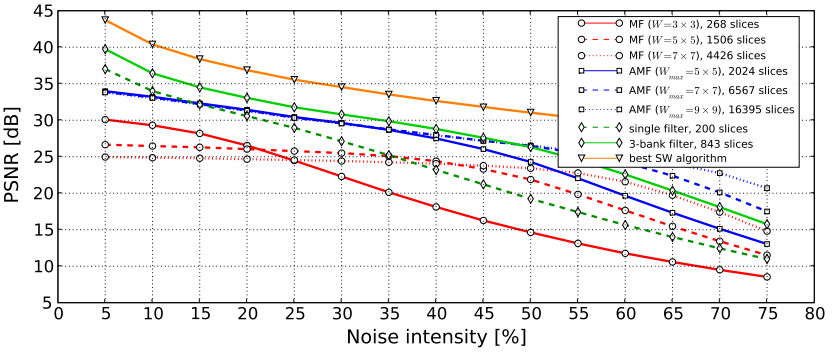


Fig. 6.10 Comparison of standard image filters (MFs, AMFs) and evolved filters (a single filter, and a three-bank filter) for salt-and-pepper noise of various intensities. The filtering quality is given as the mean PSNR value calculated for 25 images. The implementation cost is given in FPGA slices.

6.2.8.3 Impulse Burst Noise

An extended version of CGP was utilized for the evolution of impulse burst noise filters with a 5×5-pixel filtering window [35]. In fact, CGP was used to evolve nine-input filters, whose inputs were identified using a selector. The CGP parameters were initialized as follows: $n_c \times n_r = 6\times6$; population size = 8; mutation rate = 5%; number of generations = 50,000; and number of independent runs = 150. The initial population was generated randomly.

In order to evaluate the quality of filtering for the selected approaches, the mean value of PSNR was calculated for 15 test images with selected noise levels (Fig. 6.11). We compared evolved filters (FIB3 and FIB5), standard median filters, adaptive median filters and the three-bank filter.

Figure 6.12 shows one of the filters (denoted by FIB5) evolved with a 5×5 filtering window. Table 6.3 gives the implementation cost of the evolved filters for various existing approaches. The FIB5 filter occupies 108 slices and can operate at 242 MHz. We can observe that the evolved filters are relatively small but perform well.

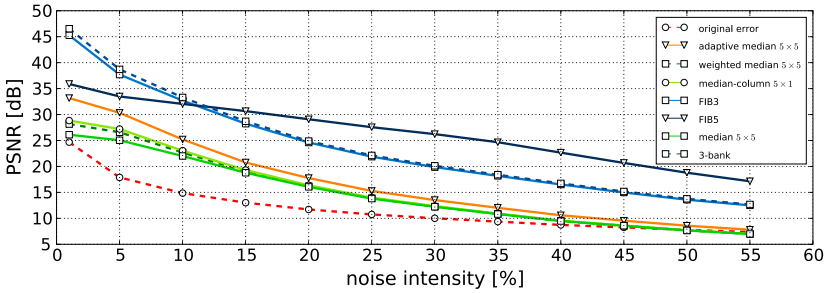


Fig. 6.11 Mean PSNR calculated using 15 images for different levels of impulse burst noise intensity.

Figure 6.11 shows the filtering capabilities of various filters for an image corrupted by 5% impulse burst noise. The results demonstrate that the evolved filter FIB5 provides the best results in most cases, especially for higher noise intensities. It is interesting that the best filter discovered by CGP (see Fig. 6.12) utilizes only the pixels of the central column of the filter window. These pixels are clearly the most important ones for suppressing this type of noise.

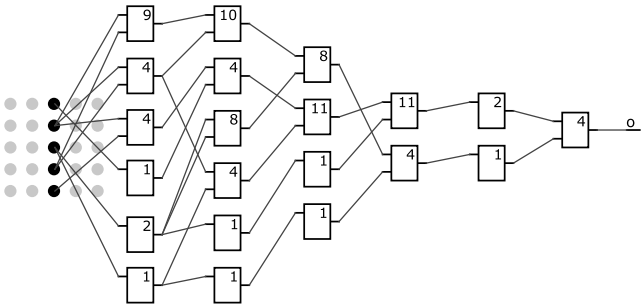


Fig. 6.12 The structure of the best evolved filter, FIB5 for impulse burst noise. The node functions are encoded as follows: (1), identity; (2), inversion; (4), minimum; (8), addition with saturation; (9), average; (10), if ($x > 127$) then y else x; and (11), the absolute value of the difference.

(a) Original image **(b)** Corrupted image **(c)** FIB3 **(d)** FIB5

(e) Adaptive median 5×5 **(f)** Three-bank **(g)** Median 5×5 **(h)** Median-column

Fig. 6.13 One of the validation images corrupted by 5% impulse burst noise and filtered by various filters.

The experiments were conducted on a cluster consisting of 100 PCs (Pentium IV, 2.4 GHz, 1 GB RAM) using the Sun Grid Engine (SGE) which enables up to 100 independent experiments to be run in parallel. The evolution time of a single run was approximately 6 hours, until the CGP algorithm reached 50,000 generations.

6.2.9 Summary

The experimental results clearly demonstrate that CGP can automatically produce image filters that are competitive with conventional filters in terms of filtering quality. We observed that the images filtered by evolved filters exhibited more detail (and thus a higher visual quality) than images filtered by conventional filters (e.g. median filters). The implementation cost in an FPGA is also encouraging.

We will see in Sect. 7.3.2 that image filter design can be significantly accelerated when CGP is implemented using suitable hardware.

6.3 Evolving Advanced Image Filters

Using CGP, it is possible to evolve many types of advanced image filters. In previous work by Harding and Banzhaf [10, 11, 9, 12], the use of CGP to evolve a number of different common image filters was discussed. The task was framed as that of reverse-engineering some image filters such as the Sobel edge detection filter and the or dilation/erosion filter. By reverse engineering, we mean finding a mapping between an image and the output of a filter applied to it. The technique found may not be the same as that used by the process, but it produces similar results.

The approach used was similar to that described previously in this chapter, where kernel-based programs were evolved. An evolved CGP program takes a pixel and its neighbourhood, performs a calculation and returns a new value for the centre pixel. This set-up is illustrated in Fig. 6.14. The program is applied to each pixel in the image.

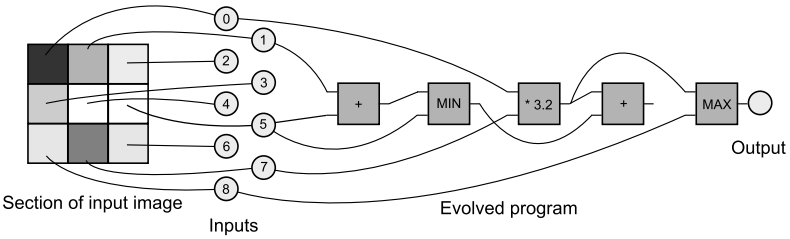


Fig. 6.14 In this example, the evolved program has nine inputs – these correspond to a section of an image. The output of the program determines the new colour of the centre pixel. Note that one node has no connections to its output. This means that this node is redundant, and will not be used during the computation.

Running an evolved program on a large number of pixels is a computationally intensive exercise. In Sect. 8.6, a methodology for accelerating this using graphics processing units (GPUs) is discussed. Here, we limit the discussion to the basic approach and to the evolved filters obtained.

Fig. 6.15 The training and validation image set. All images were presented simultaneously to the GPU. The first column of images was used to compute the validation fitness, and the remaining 12 for the training fitness. Each image is 256 $\times$ 256 pixels, with the entire set containing 1024 $\times$ 1024 pixels.

6.3.1 Fitness Function

The fitness function used was in many ways typical of those used in image processing. The quality of an evolved filter was the average error of the pixel values between the generated image and a target image.

With the acceleration in evaluation offered by a GPU, we were able to test individuals on a number of different images. Figure 6.15 shows the 16 different images used in the fitness evaluation. These were largely taken from the USC-SIPI image repository. To improve the quality of the results, 12 images were used for fitness evaluation, and four for a validation score. This allowed increased confidence that the evolved filters would generalize well.

Each image used was an eight-bit greyscale image, measuring 256×256 pixels.

The target images (i.e. the images after processing by a filter) were generated using the open source image-processing program GIMP [8].

The GPU implementation of the fitness function is discussed later, in Sect. 8.6.

6.3.2 Changes to the Standard CGP Genotype

The genotype and processing of the CGP genotype had a number of differences compared with the classical CGP implementation.

First, relative addresses were used instead of directly accessing connections. This approach is discussed in detail in Chap. 4.

In addition to the function type and connection information, each node also encoded a parameter as a floating-point number. This parameter was used by the evolved programs to encode a constant. The function set shown in Table 6.4 indicates where and how this parameter was used.

Another feature of the fitness function used here was that it allowed the image to be passed through the evolved filter a number of times. In addition to the encoded program, the genotype also contained an 'iteration counter' that specified how many times the filter should be applied to each pixel.

6.3.3 Evolutionary Algorithm, Parameters and Function Set

The algorithm used here was a simple evolutionary algorithm. The population had a size of 25. The mutation rate was set to be 5%, i.e. each gene in the genotype was mutated with probability 0.05. Crossover was not used. The iteration counter was also mutated with a 5% probability. The counter was mutated by adding a random number in the range -2 to 2. The counter was bounded between 1 and 5. Selection was done using a tournament selection of size 3. The five best individuals were promoted to the next generation without modification. The CGP graph was initial-

Table 6.4 CGP function set, defined on two inputs x and y and on the parameter c of the node

Function	Description
ITERATION	Returns the current iteration index
ADD	Adds the two inputs
SUB	Subtracts the second input from the first
MULT	Multiplies the two inputs
DIV	Divides the first input by the second, returning 1 if y is very close to zero
ADD CONST	Adds a constant (the node's parameter) to the first input
MULT CONST	Multiplies the first input by a constant (the node's parameter)
SUB CONST	Subtracts a constant (the node's parameter) from the first input
DIV CONST	Divides the first input by a constant (the node's parameter)
SQRT	Returns the square root of the first input
POW	Raises the first input to the power of the second input, returning 1 if the value is undefined.
SQUARE	Squares the first input
COS	Returns the cosine of the first input
SIN	Returns the sine of the first input
NOP	No operation – returns the first input
CONST	Returns a constant (the node's parameter)
ABS	Returns the absolute value of the first input
MIN	Returns the smaller of the two inputs
MAX	Returns the larger of the two inputs
CEIL	Rounds up the first input
FLOOR	Rounds down the first input
FRAC	Returns the fractional part of the number, $x - \lfloor x \rfloor$
LOG2	Log (base 2) of the first input
RECIPRICAL	Returns $1/\text{first input}$
RSQRT	Returns $1/\sqrt{\text{first input}}$

ized to contain 100 nodes (it is important to note that not all nodes were used in the generated program). Evolution was allowed to run for 50,000 evaluations.

Table 6.4 shows the available functions. These functions operate upon single-precision floating-point numbers. The function set was determined by mapping all the appropriate GPU programming API calls in the Microsoft Accelerator toolkit to functions.

6.3.4 Results

The results for evolving each filter are summarized in Table 6.5. In the following, the best validation result is shown, alongside the output of the target filter obtained from GIMP. Examples of the evolved programs are included to illustrate the types of operations that evolution found to replicate the target filters. Owing to space constraints, it is not possible to include such an analysis for every filter type. For the

implementation of the original filters, there is extensive coverage in the literature and also in the source code for GIMP.

Results for the GPU acceleration can be found in Chap. 8.

Table 6.5 Results for each evolved filter. 'Best error' is the lowest error seen when testing against the validation images. 'Avg Validation Error' is the average of the best validation error. 'Avg. validation evaluations' is the average number of evaluations required to find the best validation error. 'Avg. training error' is the average of the lowest error found on the training images. 'Avg. training evaluations' is the average number of evaluations required to find the best training fitness. Each experiment was repeated for 20 trials

Filter	Best error	Avg. validation error	Avg. validation evaluations	Avg. training error	Avg. training evaluations
Dilate	0.57	0.71	2,422	0.67	3,919
Dilate2	5.84	6.51	11,361	6.10	39,603
Emboss	3.86	8.33	15,517	7.41	34,878
Erode	0.56	0.78	3,892	0.73	4,066
Erode2	5.70	6.72	26,747	6.64	40,559
Motion	2.08	2.32	29,464	2.24	43,722
Neon	1.32	2.69	15,836	2.41	35,146
Sobel	8.41	22.26	26,385	20.12	45,744
Sobel2	1.70	3.82	19,979	3.55	39,155
Unsharp	5.85	5.91	301	5.61	37,102

6.3.4.1 Dilation and Erosion

Figure 6.16 shows the result of evolving the Dilate filter. In Dilate2 (Fig. 6.17), the filter is applied twice. Figure 6.18 shows the result of evolving the Erode filter. In Erode2 (Fig. 6.19), the filter is applied twice.

It is possible to analyse the evolved programs to determine how the filters work. For the erosion filter, the best evolved program contained eight operations and required five iterations to run. The evolved expression is

$$OUTPUT = MAX(LOG_2(I_8), MIN(I_8 + (MIN(I_3, I_7) - MAX(I_7, I_8)), FLOOR(I_1))),$$

where I_1 to I_9 are the input pixels, with I_1 being in the top left of the kernel and I_9 in the lower right.

The best dilation program contained four instructions, and again required five iterations. The evolved program is

$$OUTPUT = MAX(MAX(I_9, MAX(I_1, I_5)), MAX(I_3, I_7)).$$

In contrast, the best evolved programs for applying erosion and dilation twice both contained 17 instructions (and again required five iterations). It is unclear why these programs should need to be so much more complicated.

6.3.4.2 Emboss, Sobel and Neon

The Emboss, Sobel and Neon filters are various types of edge detectors. It was chosen to evolve these three different types, as the outputs are very different. Emboss is a directional filter, whereas Sobel and Neon are not. It was found that all three types of filters could be accurately evolved.

Figure 6.20 shows the result of evolving the Emboss filter. When visually compared, the evolved Emboss filter is very similar to that used by GIMP (for this and the other sample images here, the most representative and visually useful subimage has been used). The evolved program for this filter contains 20 nodes:

$$\begin{aligned} Output = {} & ABS(MIN(-0.3857, \\ & POW(SQRT(I_2/((I_8 + RSQRT(I_5)) - 0.863)), I_8)) + \\ & CEIL(MIN(((I_3 - I_9) + (I_1 - I_7)) \\ & -129.65, FRAC(I_5)))). \end{aligned} \tag{6.3}$$

Figure 6.21 shows the result of evolving the Neon filter.

GIMP has two versions of the Sobel filter. Figure 6.22 shows the result of evolving the normalized Sobel filter. In the case of Sobel2 (Fig. 6.23), the target was the output of the standard Sobel filter. The standard Sobel filter achieved poor error rates; however, the visual comparison is very good. It would appear that the evolved output is scaled differently, and hence the pixel intensities are different. If the two images are normalized, the error is reduced. However, the fitness function does not normalize automatically, but leaves this task to evolution.

The evolved Sobel filter is quite complicated:

$$\begin{aligned} A &= I_7 - I_3 \\ B &= I_9 - MAX(I_1, LOG_2(A)) \\ OUTPUT &= 2.0 * (MIN(MAX(ABS(B) + FRAC(1) + ABS(2.0 * A), \\ &\quad MAX(FLOOR(LOG_2(A)), 2.0 * B)), \\ &\quad (CEIL(FRAC(I_5)) * -0.760) + 127.24)). \end{aligned}$$

The best evolved program for Sobel2 is considerably shorter:

$$\begin{aligned} Output = {} & MAX(ABS((I_9 - I_1) * 0.590), \\ & POW(I_3 - I7, SQRT(0.774))/-2.245). \end{aligned}$$

Again, both programs required five iterations. This suggests there is some bias in the algorithm towards increasing the number of iterations to the maximum allowed.

6.3.4.3 Motion Blur

Figure 6.24 shows the result of evolving the Motion filter. The output of the evolved filter did not match the desired target very accurately. But although there is a degree of blurring, it is not as pronounced as in the target image. Motion blur is a relatively subtle effect, and as the target and input images are quite similar, it is likely that evolution will become trapped in a local minimum.

6.3.4.4 Unsharp

Figure 6.25 shows the result of evolving the Unsharp filter. Unsharp was the most difficult filter to evolve. It is suspected that this is due to the Gaussian blur that needs to be applied as part of the procedure. It is difficult to see how, with the current function set, such an operation could be evolved. In future work, this issue will need to be rectified.

Fig. 6.16 Dilate: evolved filter, GIMP filter and difference image.

Fig. 6.17 Dilate2: evolved filter, GIMP filter and difference image.

Fig. 6.18 Erode: evolved filter, GIMP filter and difference image.

Fig. 6.19 Erode2: evolved filter, GIMP filter and difference image.

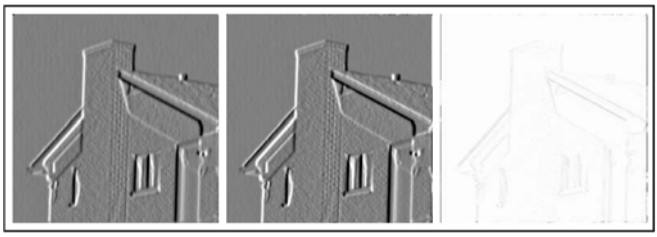


Fig. 6.20 Emboss: evolved filter, GIMP filter and difference image.

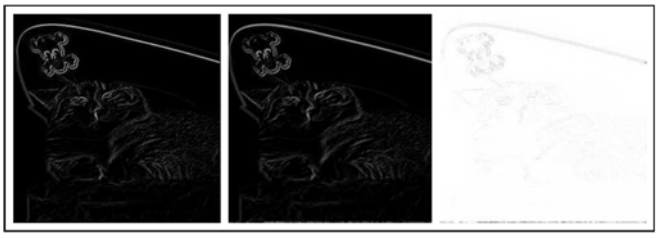


Fig. 6.21 Neon blur: evolved filter, GIMP filter and difference image.

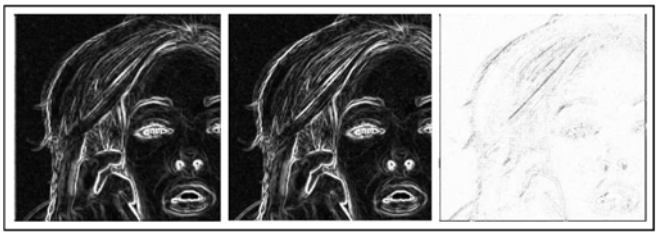


Fig. 6.22 Sobel: evolved filter, GIMP filter and difference image.

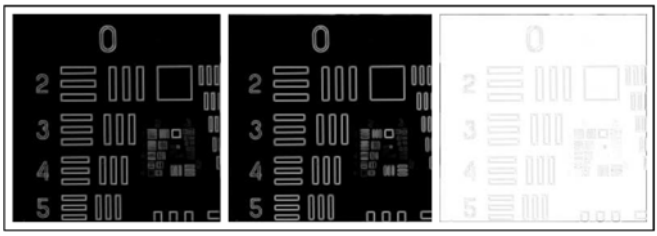


Fig. 6.23 Sobel2: evolved filter, GIMP filter and difference image.

Fig. 6.24 Motion blur filter: evolved filter, GIMP filter and difference image.

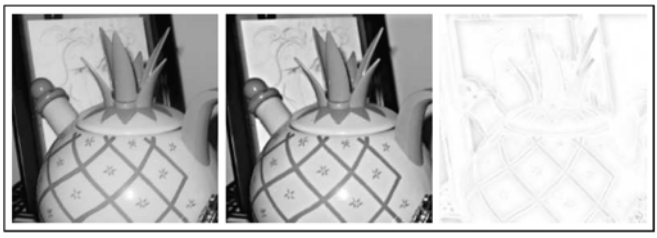


Fig. 6.25 Unsharp blur filter: evolved filter, GIMP filter and difference image.

6.4 The Automated Design of Features for Image Classification

In this section, we discuss the application of CGP to the automated discovery of features in an image classification task. We attempt to evolve transformations on the space of images, in the hope that particular transforms which emphasize distinguishing characteristics may be found. A set of moments describing the transformed image are extracted, and used for classification.

We have applied our work to the recognition of cells in a medical image database. Specifically, we attempted to recognize nuclear inclusions which indicate the presence (or absence) of OPMD, a form of muscular dystrophy. Recognition of these inclusions is a difficult task, usually requiring expert training of a human operator.

6.4.1 Motivation

Image classification (recognition) typically uses a set of features to reduce the dimensionality of the pattern space. A great deal of effort is spent on the design, selection and weighting of features. Often, practitioners begin with a set of 'standard' (generally applicable) features, and then use some machine-learning technique to find the most appropriate choices; these features often involve statistical moments, or entropy- or histogram-based measures (as in [17]). Domain-specific measures can be used as well, such as the cell-nucleus-specific measures used in the construction of the Wisconsin Breast Cancer Database [33].

The advantage of domain-specific features is, of course, potential increased relevance to the problem domain at hand, ideally increasing the efficacy of an image-processing system. The downside is the attention that the assembly of domain-specific features requires, where human operators – possibly domain experts – are often required for research, exploration and optimization.

The capacity to extract database-specific features automatically is enticing: one can conceivably have both application without expert intervention and the increased performance associated with database specificity. There are several existing techniques for the automated discovery of learned features. Common techniques include linear discriminant techniques and principal-component-analysis-based techniques, and there are also others [13]. There also exist several related genetic-programming-based applications in image processing, such as the automated detection of points of interest in an image [42], and the classification of image textures [21].

Our technique has followed similar motivation, but with some key differences: (a) we make nearly no assumptions regarding the distribution of class data; (b) our technique is specifically oriented towards image recognition, thus emphasizing local neighbourhoods of pixels in two dimensions; and (c) the results of our technique are easily analysed, both visually and analytically.

6.4.2 The Model

We treat an image space as a collection of pixels $p = (p_x, p_y) \in I$, and an image as a collection of intensities on that space, $f(p) \in [0,1]$, or simply $f(I)$.

We have seen how CGP kernel functions can be used to reverse-engineer image-processing filters. Here we use this notion, that of a sliding window function, and apply it to the design of transforms on the space of images. Since we know that this space of image transforms contains *some* useful image filters, we assume that more specialized or effective filters can be designed to highlight portions of an image database. Ultimately, we are searching for kernel functions that will help us in the classification of images.

Here we outline the definition of an evolvable feature extractor from a CGP graph. The CGP graph is used to define a transformation on the space of images. In this case, however, no explicit image target is given; instead, we hope to create a transform useful for accentuating the important features of the images. Next, these transformed images are converted to a set of numerical values. Finally, these numerical values are used to train or test a classifier. An overview of the entire process is illustrated in Fig. 6.26.

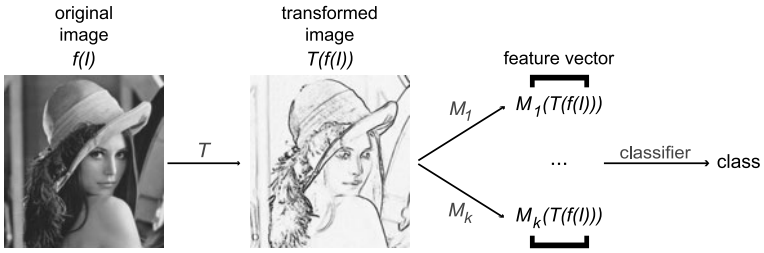


Fig. 6.26 An overview of the feature extraction procedure.

6.4.2.1 CGP Graphs as Image Transforms

We again use a simple form of CGP. The graph consists of a single row of connection nodes of length $n_c = L_n$. These are augmented by a large collection of inputs, labelled $I_0, \ldots, I_{n_i-1}$. Levels-back is chosen so that any connection node may connect to any previous node or input, and addressing is relative. There is a single output, automatically defined to be the final node in the representation. We use the function set {Max, Min, Add, Sub, Mult, Div, Const, Abs, Square, Pow}, i.e. a simple listing of functions known to beuseful in common convolution functions.

As before, the CGP graphs operate on a square of pixel intensities, $\bar{p} = \{p_0, p_1, ..., p_{m^2}\}$, surrounding a central pixel p. Given a CGP graph T with $n_i = m^2$, we can apply the graph to $\bar{p}$ as follows:

$$T(f(\bar{p})) = T(f(p_0), f(p_1), \ldots, f(p_{n^2})). \tag{6.4}$$

Note that if $f(q)$ does not exist for some pixel q (beyond the edges of the image, say), then we return the value $f(q) = -1$. Furthermore, if $T(f(\bar{p})) \notin [0,1]$, we replace it by the closest boundary value, either 0 or 1.

Given some image $f(I)$, we can define a new image $T(f(I))$ as follows. Let I' be an image space of the same dimensions as I. For each pixel $p' \in I'$, let $f(p') = T(f(\bar{p}))$. Hence, every CGP graph T can be viewed as a transform on the space of images.

6.4.2.2 Feature Extraction

Thus far, we have discussed the generation of an image transform from a CGP graph, allowing us a means of mapping from the space of images to itself. However, our data still exists in a very high-dimensional space, unsuitable for a classifier. To perform the dimensionality reduction, we have to rely on some common statistical moments, chosen for simplicity and speed. The choice of a simple statistical measure is a purposeful attempt to constrain the results of the evolutionary process. That is, the use of GP is generally known to be capable of low bias and high variance [16], and a well-known but biasing step is likely to help us avoid overfitting.

There are many forms of statistical moments that could be used in this context. Local, orthogonal and multi-scale moments – such as wavelet-basis and Zernike polynomials, are common choices, but we are not considered desirable for the given application. We instead selected a collection of simple geometrically-based moments, some of which were transform-, rotation- and scale-invariant. These properties are desirable for cell images, since the exact location, zoom level and orientation of the cell image relative to the image boundaries is superfluous. We thus chose a collection of 16 moments for the reduction of images to numerical vectors, drawn from the first few geometric moments, central moments and Hu's moments [14]. For details, see [20].

6.4.2.3 A Cell Classification Problem

We worked with a database of images of cells, CellsDB, originally collected by the Centre hospitalier de l'Université de Montréal (CHUM), where the causes and associated symptoms of oculopharyngeal muscular dystrophy (OPMD) have been studied extensively [1]. Intranuclear inclusions (INIs) have been detected via both pathological studies and electron microscopy. These INIs are tubular, about 8.5 nm in external diameter and 3 nm in internal diameter and up to 0.25 μm in length, and converge to form tangles or palisades [34]. Detection of these inclusions is expected to lead to the detection of OPMD. CellsDB was collected and prepared by Tarundeep Dhot at the Centre for the Study of Brain Diseases at CHUM, and is pre-segmented and divided into two categories associated with the presence or

absence of inclusions indicating OPMD: 'healthy' and 'sick'. An example of some cell images may be seen in Fig. 6.27.

Detecting OPMD-indicating INIs is a difficult task, requiring training for human classification. It is dependent upon the relative pixel intensities in a neighbourhood, and it is thus difficult to distinguish between noise and other intranuclear patterns. A further difficulty is that cell images come in a variety of scales: for this reason, we chose to mix images taken at 10×, 20×, and 40× zoom. We broke the database into two sets: 186 healthy and 200 sick cell images for training, and 200 healthy and 200 sick cell images for validation.

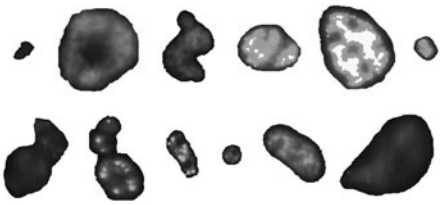


Fig. 6.27 Examples of healthy (top row) and sick (bottom row) cell images.

We chose to investigate the classifiability of this database using a collection of predefined moments, drawn from previous work in image classification. We implemented a collection of general features used in images processing: *image entropy*, *fractal dimension*, and our 16 statistical moments. We also selected a collection of cell-specific features, designed to operate on the nuclear boundary and interior of a cell image: *threshholded area*, *variance*, *mean radius*, *radius variance*, *perimeter* and *compactness*. These 24 features were calculated for the validation set of images, and evaluated using a collection of classifiers with 10-fold cross-validation. The results are summarized in Table 6.6. The best expected performance was an *SS* value of 0.580 – composed of a true-positive rate of 0.774 and a true-negative rate of 0.750 – under a 5-nearest neighbour (5-NN) classifier.

6.4.2.4 Evaluation of a Transform

A common and fast choice for the evaluation of a set of feature values is the use of a measure of inter-class and intra-class spread. Here, however, we used a classifier directly. Although slower, this effectively eliminated assumptions regarding the underlying distribution of data points, thus allowing us to consider a wider range of potential feature extractors.

We wished to award fitness to any particular collection of feature values based on its ability to distinguish between healthy and sick cells. To do so, we converted the database to a set of features, and then attempted to classify the cells using some given classifier and evaluation technique. The classifier returned a false-positive rate, FPR, for both classes, which we combined into a measure of *sensitivity speci-*

ficity:

$$SS = (1 - FPR(healthy))(1 - FPR(sick)) \tag{6.5}$$

$SS \in [0, 1]$ iwould be maximized for perfect recognition. All classifiers and evaluation techniques were implemented via the Weka machine-learning system, version 3.5.7 [41].

For the training runs, a set of 16 feature values were computed for the transformed image, consisting of the 16 chosen moments. These were evaluated under a 50–50 train–test split.

For the validation runs, a set of 16 features were computed for the transformed image, and added to the original 24 predefined features. These 40 values were evaluated using a 1-NN classifier with 10-fold cross-validation.

The reason that a smaller set of features was used in the definition of the training evaluation was as follows: randomization of the database prior to classification is a useful feature to help prevent overfitting, but leads to a stochastic fitness function. Beginning with a set of features which is already adept at classification – as the predefined features are – makes the variance due to stochasticity greater than the fitness gains in early evolution, hence hindering the selection operator.

Table 6.6 shows the expected classification accuracies of the feature sets for completely randomly generated transforms. There is little difference in mean performance between the predefined features alone and the inclusion of a randomly generated transform, but a difference can be seen in the overall variance, suggesting that some new useful information is occasionally added.

Table 6.6 Comparison of classification on a standard database of unevolved features ($\emptyset$) with a database augmented by a randomly generated transform. All figures are the mean (with the standard deviation in parentheses) of *SS* over 40 runs

Transform	Decision tree	Ridor	1-NN	5-NN
$\emptyset$	0.495 (0.020)	0.461 (0.024)	0.517 (0.012)	0.580 (0.016)
Random transform	0.521 (0.080)	0.503 (0.059)	0.553 (0.070)	0.611 (0.067)

6.4.3 Transform Evolution

We conducted an informal parameter search to select a good running point for our evolutionary algorithm. Contrary to results for previous applications, in this context experimentation demonstrated to us that crossover was a useful genetic operator: we therefore included a single-point crossover in our work. We selected a population size of 200, a rate of mutation of 0.02, a probability of crossover of 0.6, single-member elitism, a graph size of $L_n = 100$ and a kernel size of 6×6. Each

evolutionary algorithm was run for 50 generations, using a 1-NN classifier; 40 runs were undertaken in total.

Evolution was quite successful at increasing the fitness (training *SS*), which increased from a mean best fitness of 0.523 in the first generation to a mean best fitness of 0.620. In a few cases, evolution optimized training *SS* at the expense of validation *SS*, but a good general increase in the latter was seen in most runs. The mean best validation *SS* for the final generation was 0.676 (s.d. 0.052), and was maximized at a value of 0.766, a 32% improvement over the expected performance of the best classifier found for predefined features alone.

6.4.3.1 Best Discovered Transform

In this subsection, we explore the best evolved transform in more detail. The best individual of the final (50th) generation of the run with the highest validation fitness was selected. This best individual had a sensitivity specificity of 0.766, encompassing true-positive rates of 0.878 for healthy cells and 0.873 for sick cells.

The same evolved attribute values, when evaluated using a J48 decision tree instead of a 1-NN, gave a sensitivity-specificity of 0.801, encompassing true-positive rates of 0.912 for healthy cells and 0.878 for sick cells, or a 38% improvement over the expected performance of the best classifier found for predefined features alone. The performance for the best discovered transform, relative to the best expected performance for the unevolved features alone, is summarized in Table 6.7.

Table 6.7 Comparison of classification results for unevolved versus best evolved features

Classifier	**Transform**	*SS*	$TPR(H)$	$TPR(S)$
5-NN	$\emptyset$	0.580	0.774	0.750
1-NN	Best transform	0.766	0.878	0.873
Decision tree	Best transform	0.801	0.912	0.878

The best transform, once neutral code is removed, may be written as the following neighbourhood function:

$$\text{Output}(I_0,\ldots,I_{35}) = \text{Min}(I_{11} - I_{15}, \text{Pow}(I_9 - \text{Max}(I_6, I_{25}), I_8)).$$

The same transform is illustrated in CGP graph form in Fig. 6.28. This function mostly returns black (0), except when both $I_{11} - I_{15}$ and $(I_9 - \max\{I_6, I_{25}\})^{I_8}$ are (relatively) high. The first subsection ensures that the variance of the right-hand part of the neighbourhood is high (excluding inclusions that are too large). The second subsection ensures that I_9 is high, while both of I_6 and I_{25} are low or just I_8 is low. This ensures that that the left part of the image is dark, while there is some lightness in the right half, hence excluding light spots that extend to the right of the neighbourhood. Hence, we detect the left half of inclusions of the proper size and

variance. Note that this function works for several different microscope zoom levels simultaneously.

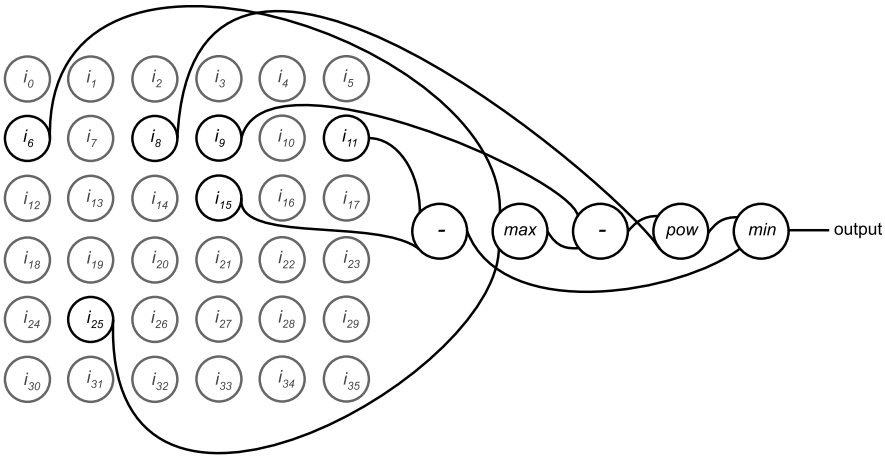


Fig. 6.28 A CGP graph view of the best discovered individual.

Although non-OPMD-indicating inclusions are still highlighted using this function, the OPMD-indicating inclusions are highlighted with more intensity, hence allowing greater recognition of the cells than with the original image features alone. Examples of the output are shown in Fig. 6.29.

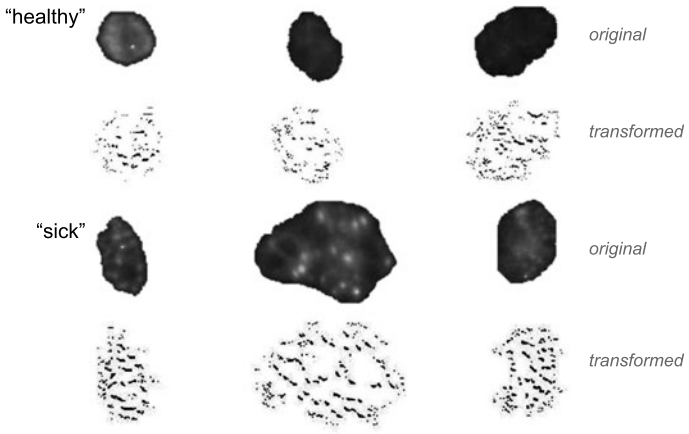


Fig. 6.29 Examples of the action of the best discovered transform on cell images. The transformed images have been subjected to contrast stretching and colour inversion to make the distinction between black and dark grey more visible.

Feature selection was run on the database of the original moments and the evolved moments, using an information gain attribute evaluator. The evaluator selected 30 of the features as significant, discarding the other 10. The top 10 ranked attributes, by information gain, were all moments of the evolved transformed image.

6.4.4 Future Directions

The use of a single image transform in the above work was based on our expectation of the pattern to be detected – we aimed to discover one type of nuclear INI, and hence we chose a single transform. Evolving several transforms simultaneously is also a possibility. Figure 6.30 shows a visualization of some transforms that were evolved when we selected for four transforms rather than one using the CellsDB database. In this case, a transform recovering the area of the interior of the cell nucleus is visible, and also partial edge detectors: this is an effective recovery of some of the more important information originally included in the predefined features. Clearly, some problem domains would require several transforms to distinguish the class data, and the best means of representing multiple image transforms and ensuring good cooperation between them is an open problem.

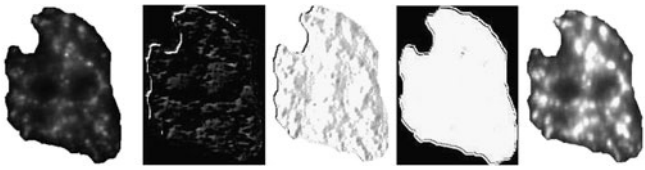


Fig. 6.30 The action of four simultaneously evolved transforms (*right*) on a cell image (*left*).

Evolutionary algorithms are known for their capacity to automatically discover parameters in learning techniques. Another valuable line of development would be the automated evolution of database-specific parameters, such as the selection of a dimension-reducing set of moments, the selection of an appropriate local pixel neighbourhood size, and the selection of an appropriate number of image transforms. These would also make the system less dependent on expert input.

6.5 Acknowledgements

Lukas Sekanina was partially supported by the Grant Agency of the Czech Republic under Contract No. P103/10/1517 and by the research programme MSM 0021630528. WB and SH gratefully acknowledge funding from Atlantic Canada's

HPC network ACENET, from the Canadian Foundation of Innovation, New Opportunities Grant No. 204503, and from NSERC under the Discovery Grant Program RGPIN 283304-07.

References

1. Brais, B., Bouchard, J.P., Xie, Y.G., Rochefort, D.L., Chretien, N., Tome, F.M., Lafreniere, R.G., Rommens, J.M., Uyama, E., Nohira, O.: Short GCG expansions in the PABP2 gene cause oculopharyngeal muscular dystrophy. Nature Genetics **18**, 164–167 (1998)
2. Brownrigg, D.: The weighted median filter. Commun. ACM **27**(8), 807–818 (1984)
3. Burian, A., Takala, J.: Evolved Gate Arrays for Image Restoration. In: Proc. Congress on Evolutionary Computing, pp. 1185–1192. IEEE Press (2004)
4. Cagnoni, S., Lutton, E., Olague, G.: Genetic and Evolutionary Computation for Image Processing and Analysis. EURASIP Book Series on Signal Processing and Communications, Volume 8. Hindawi Publishing Corporation (2007)
5. Chan, R.H., Ho, C.W., Nikolova, M.: Salt-and-Pepper Noise Removal by Median-type Noise Detectors and Edge-preserving Regularization. IEEE Transactions on Image Processing **14**, 1479–1485 (2005)
6. Dougherty, E.R., Astola, J.T. (eds.): Nonlinear Filters for Image Processing. SPIE/IEEE Series on Imaging Science & Engineering. SPIE/IEEE (1999)
7. Dumoulin, J., Foster, J.A., Frenzel, J.F., McGrew, S.: Special Purpose Image Convolution with Evolvable Hardware. In: Real-World Applications of Evolutionary Computing – Proc. Workshop on Evolutionary Computation in Image Analysis and Signal Processing, *LNCS*, vol. 1803, pp. 1–11. Springer (2000)
8. GNU: GNU image manipulation program (GIMP). www.gimp.org (2008). [Online; accessed 21-January-2008]
9. Harding, S.L.: Evolution of Image Filters on Graphics Processor Units Using Cartesian Genetic Programming. In: J. Wang (ed.) Proc. IEEE World Congress on Computational Intelligence. IEEE Press (2008)
10. Harding, S.L., Banzhaf, W.: Genetic Programming on GPUs for Image Processing. In: J. Lanchares, F. Fernandez, J. Risco-Martin (eds.) Proc. Workshop on Parallel and Bioinspired Algorithms, pp. 65–72. Complutense University of Madrid Press (2008)
11. Harding, S.L., Banzhaf, W.: Genetic programming on GPUs for image processing. International Journal of High Performance Systems Architecture **1**(4), 231–240 (2008)
12. Harding, S.L., Banzhaf, W.: Distributed Genetic Programming on GPUs using CUDA. In: I. Hidalgo, F. Fernandez, J. Lanchares (eds.) Proc. Workshop on Parallel Architectures and Bioinspired Algorithms, pp. 1–10 (2009)
13. Hastie, T., Tibshirani, R., Friedman, J.: The Elements of Statistical Learning: Data Mining, Inference, and Prediction, 2nd edn. Springer (2008)
14. Hu, M.K.: Visual pattern recognition by moment invariants. IRE Transactions on Information Theory **IT-8**, 179–187 (1962)
15. Hwang, H., Haddad, R.A.: Adaptive median filters: new algorithms and results. IEEE Transactions on Image Processing **4**(4), 499–502 (1995)
16. Keijzer, M., Babovic, V.: Genetic Programming, Ensemble Methods and the Bias/Variance Tradeoff – Introductory Investigations. In: Proc. European Conference on Genetic Programming, pp. 76–90. Springer (2000)
17. Kharma, N., Kowaliw, T., Clement, E., Jensen, C., Youssef, A., Yao, J.: Project CellNet: Evolving an Autonomous Pattern Recognizer. International Journal of Pattern Recognition and Artificial Intelligence **18**(6), 1039–1056 (2004)
18. Koivisto, P., Astola, J., Lukin, V., Melnik, V., Tsymbal, O.: Removing Impulse Bursts from Images by Training-Based Filtering. EURASIP Journal on Applied Signal Processing **2003**(3), 223–237 (2003)

19. Koivisto, P., Huttunen, H., Kuosmanen, P.: Training-based optimization of soft morphological filters. Journal of Electronic Imaging **5**(3), 300–322 (1996)
20. Kowaliw, T., Banzhaf, W., Kharma, N., Harding, S.: Evolving Novel Image Features Using Genetic Programming-Based Image Transforms. In: Proc. IEEE Congress on Evolutionary Computation. IEEE Press (2009)
21. Lam, B., Ciesielski, V.: Discovery of Human-Competitive Image Texture Feature Extraction Programs Using Genetic Programming. In: Proc. Genetic and Evolutionary Computation Conference, pp. 1114–1125. Springer (2004)
22. Marshall, S.: New direct design method for weighted order statistic filters. IEE proceedings. Vision, image and signal processing **151**(1), 1–8 (2004)
23. Nikolova, M.: A Variational Approach to Remove Outliers and Impulse Noise. J. Math. Imaging Vis. **20**(1–2), 99–120 (2004)
24. Porter, R.: Evolution on FPGAs for Feature Extraction. Ph.D. thesis, Queensland University of Technology, Brisbane, Australia (2001)
25. Schulte, S., Nachtegael, M., Witte, V.D., der Weken, D.V., Kerre, E.E.: Fuzzy Impulse Noise Reduction Methods for Color Images. In: Computational Intelligence, Theory and Applications International Conference 9th, Fuzzy Days in Dortmund, pp. 711–720. Springer (2006)
26. Sekanina, L.: Image filter design with evolvable hardware. In: Applications of Evolutionary Computing, *LNCS*, vol. 2279, pp. 255–266. Springer (2002)
27. Sekanina, L.: Evolvable components: From Theory to Hardware Implementations. Natural Computing. Springer (2004)
28. Sekanina, L., Martinek, T.: Evolving Image Operators Directly in Hardware. In: S. Cagnoni, E. Lutton, G. Olague (eds.) Genetic and Evolutionary Computation for Image Processing and Analysis, EURASIP Book Series on Signal Processing and Communications, Volume 8, pp. 93–112. Hindawi Publishing Corporation (2007)
29. Sekanina, L., Ruzicka, R.: Easily Testable Image Operators: The Class of Circuits Where Evolution Beats Engineers. In: Proc. NASA/DoD Conference on Evolvable Hardware, pp. 135–144. IEEE Computer Society (2003)
30. Sekanina, L., Vasicek, Z.: Nonlinear Image Filter (2009). Czech Utility model UV020017
31. Slany, K., Sekanina, L.: Fitness Landscape Analysis and Image Filter Evolution Using Functional-Level CGP. In: Proc. of European Conf. on Genetic Programming, *LNCS*, vol. 4445, pp. 311–320. Springer (2007)
32. Sonka, M., Hlavac, V., Boyle, R.: Image Processing: Analysis and Machine Vision. Thomson-Engineering (1999)
33. Street, W., Wolberg, W., Mangasarian, O.: Nuclear feature extraction for breast tumor diagnosis. In: IS&T/SPIE 1993 International Symposium on Electronic Imaging (1993)
34. Tome, F.M.S., Fradeau, M.: Nuclear changes in muscle disorders. Methods Achiev. Exp. Pathol. **12**, 261–296 (1986)
35. Vasicek, Z., Bidlo, M., Sekanina, L., Torresen, J., Glette, K., Furuholmen, M.: Evolution of Impulse Bursts Noise Filters. In: Proc. NASA/ESA Conference on Adaptive Hardware and Systems, pp. 27–34. IEEE Computer Society (2009)
36. Vasicek, Z., Sekanina, L.: An Area-Efficient Alternative to Adaptive Median Filtering in FPGAs. In: Proc. International Conference on Field Programmable Logic and Applications, pp. 216–221. IEEE Computer Society (2007)
37. Vasicek, Z., Sekanina, L.: An Evolvable Hardware System in Xilinx Virtex II Pro FPGA. International Journal of Innovative Computing and Applications **1**(1), 63–73 (2007)
38. Vasicek, Z., Sekanina, L.: Evaluation of a New Platform For Image Filter Evolution. In: Proc. NASA/ESA Conference on Adaptive Hardware and Systems, pp. 577–584. IEEE Computer Society (2007)
39. Vasicek, Z., Sekanina, L.: Reducing the Area on a Chip Using a Bank of Evolved Filters. In: Proc. International Conference on Evolvable Systems, *LNCS*, vol. 4684, pp. 222–232. Springer (2007)
40. Vasicek, Z., Sekanina, L.: Novel Hardware Implementation of Adaptive Median Filters. In: Proc. IEEE Design and Diagnostics of Electronic Circuits and Systems Workshop, pp. 110–115. IEEE Computer Society (2008)

41. Witten, H., Frank, E.: Data Mining: Practical Machine Learning Tools and Techniques, 2nd edn. Morgan Kaufmann (2005)
42. Zhang, M., Ciesielski, V.B., Andreae, P.: A Domain-Independent Window Approach to Multiclass Object Detection Using Genetic Programming. EURASIP Journal on Applied Signal Processing **2003**(8), 841–859 (2003)
43. Zhou, W., David, Z.: Progressive switching median filter for the removal of impulse noise from highly corrupted images. IEEE Trans on Circuits and Systems: Analog and Digital Signal Processing **46**(1), 78–80 (1999)