Evolving Approximate Image Filters

Simon Colton and Pedro Torres

Computational Creativity Group
Department of Computing, Imperial College London
sgc,ptorres@doc.ic.ac.uk, http://www.doc.ic.ac.uk/ccg

Abstract. Image filtering involves taking a digital image and producing a new image from it. In software packages such as Adobe's Photoshop, image filters are used to produce artistic versions of original images. Such software usually includes hundreds of different image filtering algorithms, each with many fine-tuneable parameters. While this freedom of exploration may be liberating to artists and designers, it can be daunting for less experienced users. Photoshop provides image filter browsing technology, but does not yet enable the construction of a filter which produces a reasonable approximation of a given filtered image from a given original image. We investigate here whether it is possible to automatically evolve an image filter to approximate a target filter, given only an original image and a filtered version of the original. We describe a tree based representation for filters, the fitness functions and search techniques we employed, and we present the results of experimentation with various search setups. We demonstrate the feasibility of evolving image filters and suggest new ways to improve the process.

1 Introduction

Image filtering is the process whereby a digital image is taken as input and a new image is produced, usually via a mathematical transformation of the bitmap information in the original image. Such image filtering finds application in numerous areas, including machine vision, medical imaging [1], graphic design and the visual arts. In the latter two cases, sophisticated software packages such as Adobe's Photoshop are employed to produce artistic versions of images, usually via a chain of image filter applications. Such software often includes hundreds of different image filtering algorithms, with each having many parameters to explore the space of filters available. While this freedom of exploration may be liberating to artists and designers, the plethora of opportunities for less experienced users may be daunting. While Photoshop provides image filter browsing technology, it does not yet enable image filter search, i.e,. retrieving a filter given an example of an image produced by it, nor can it construct a filter from scratch to approximate a target image filter. We investigate here whether it is possible to automatically evolve an image filter to approximate such a target filter, given only an original image and a filtered version of that original.

We look here at the general question of evolving a filter to approximate a given target filter. As described in section 2, we have developed a fairly expressive tree-based representation of image filters, and we have built a library of

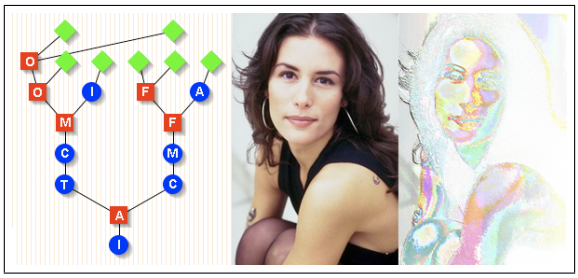

Fig. 1. Example image filter tree consisting of transforms in blue circles: A=Add Colour, C=Convolution, I=Inverse, M=Median and T=Threshold; and compositors in red squares: A=And, F=Fade, M=Min and O=Or. Image inputs are in green diamonds. An example original image and the filtered version are shown.

1000 image filters in this representation. As described in section 3, filters represented in this way are amenable to crossover and mutation, and hence admit an evolutionary search. As described in section 4, we experiment with two ways in which to seed an initial population, namely by retrieving filters from the library, as per [9], and by random generation. We also experiment with four fitness functions, and discuss the visual differences between the types of filters that are evolved using them. Working with 75 image filters, 46 of which are from Photoshop, and 29 from our library, we show that, while it is difficult, automatically evolving suitable approximations to filters is feasible, but there is much room for improvement. We conclude by describing some of the planned improvements.

2 Representing Image Filters

We represent image filters as a tree of fundamental (unary) image **transforms** such as inverse, lookup, threshold, colour addition, median, etc., and (binary) image **compositors** such as add, and, divide, max, min, multiply, or, subtract, xor, etc. As an example, the *Median* transform takes two parameters, the first of which determines the way in which the median RGB values of each pixel is calculated, and the second of which determines the extent of the neighbourhood that is taken into account when the median is calculated. As an example compositor, the *And* compositor takes two images and calculates the `and` logical value of the pair of bitmap RGB values at each coordinate in the image. An example tree is provided in figure 1, where the overall filter uses seven transform steps and six compositor steps, and the original image is input to the tree seven times.

As described in subsection 3.2 below, image filters can be generated randomly. Each time an image filter generated in this random way produced an interesting visual effect on one or more original images, it was added to a library of filters, until we had built up 1000 such filters, categorised into 30 sets of filters, roughly according to how the filtered images look, e.g., there are categories for filters which are blurred, grainy, monotone, etc. The time taken to apply a filter is roughly proportional to the size of the input image multiplied by the size of the tree. Over the entire library of filters, the average number of nodes in a tree is

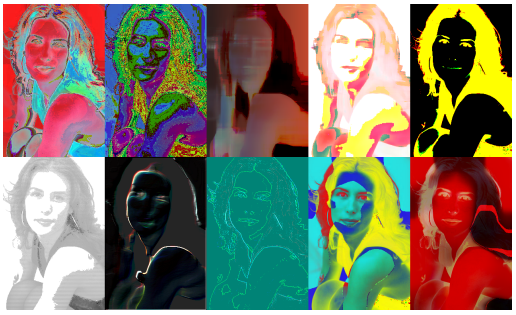

Fig. 2. Ten filters applied to the original image from figure 1

13.62, and the average time[1] to apply a filter to an image of dimension 256 by 384 pixels (the size of images in the standard Corel library) is 410 milliseconds. To give an indication of the visual variety achievable by the filters in the library, in figure 2, we provide 10 filtered versions of the original image from figure 1.

3 Evolving Filters

We are addressing the following problem: given an image I and a target filtered version of I, which we denote T_I, construct a filter F such that the filtered image, F_I, gained from applying F to I approximates T_I, to a good standard, as evaluated by a given fitness function. In order to describe these techniques, we first discuss the fitness functions we used, followed by how we randomly generate, crossover and mutate filter trees.

3.1 Fitness Functions

As with most evolutionary search applications, correctly measuring the fitness of individuals is key to success. In our case, the fitness function will be used to order individual image filters in terms of how close their output is to the target filter's output when applied to a given original image. We envisage (at least) the following two reasons to evolve image filters. In the first scenario, a user has found an example of a filtered image and wants to construct a filter to approximate it as closely as possible. In the second scenario, a user has an original image and has found a filtered version of it that pleases them, but would like to see similar variations on the theme.

To model the former case, we use the following fitness function:

$$1 - \left(\frac{\sum_{1 \leq x \leq w} \sum_{1 \leq y \leq h} (dist(T_I(x,y), F_I(x,y))}{w * h} \right)$$

where w and h are the width and height of the images being produced respectively, and $dist(T_I(x,y), F_I(x,y))$ is the Euclidean distance in RGB-space between the pixel at point (x,y) in the target filtered image and the corresponding

[1] On a MacOS X machine running at 2.6Ghz.

pixel in the image output by the filter being evaluated. Hence, if a filter achieves a score of 1, it perfectly reproduces the target filtered image at the bitmap level.

In the latter case, recreating exact pixel values is less important than gaining an overall visual style in the evolved filters. For this reason, we use two common features of an image, namely its colour distribution, and how grainy it is (i.e., the level of contrast in the image). We use the following weighted sum of these two features as a fitness function:

$$w_g * (1 - |g(T_I) - g(F_I)|) + w_c * (1 - \mathit{diff}(\mathit{hist}(T_I), \mathit{hist}(F_I)))$$

where g(X) approximates the contrast[2] present in an image by measuring the average Euclidean distance in RGB-space of a pixel from its neighbours in image X and $\mathit{diff}(\mathit{hist}(T_I), \mathit{hist}(F_I))$ denotes the difference between the colour histograms of the target and evolved filtered images. This is measured by calculating the average of the difference over all the bins in a 4 by 4 by 4 colour histogram for both images. As we see in section 4, we experimented with three different pairs of weights $\langle w_g, w_c \rangle$ for the weighted sum.

3.2 Random Generation of Filter Trees

Filter trees can be randomly generated by starting with an input image node (a diamond-shaped node in figure 1), and iteratively choosing to replace an input image node with a transform node or a compositor node. If the node is replaced by a transform, then a single input image node is added to the tree above it, and similarly, if a node is replaced by a compositor, a pair of input nodes are added to the tree above it (as input to it). Each time a new node is generated, random parameters for that node are generated, with the parameters being within a suitable range of values. There are various ways in which we can control the random generation, including limiting the number of transforms/compositors/nodes and the longest branch in the tree, and controlling the likelihood of each particular transform/compositor being chosen as a node in the tree.

3.3 Crossover and Mutation

Representing image filters as trees enables both the crossing over of branches into offspring from each of two parents, and the mutation of trees, thus enabling an evolutionary approach to filter construction. To perform crossover, we start with two parent trees (called the *top* and *bottom* parent), and we choose a pair of nodes: N_t on the top parent and N_b on the bottom parent. These are chosen randomly so that the size of the tree above and including N_t added to the size of the tree gained from removing the tree above and including N_b from the bottom parent is between a user-specified minimum and maximum. After 50 failed attempts to choose such a pair of nodes, the two trees are deemed

[2] We acknowledge that there are other methods for measuring the contrast in an image, in particular by taking the second order derivative of its colour histogram.

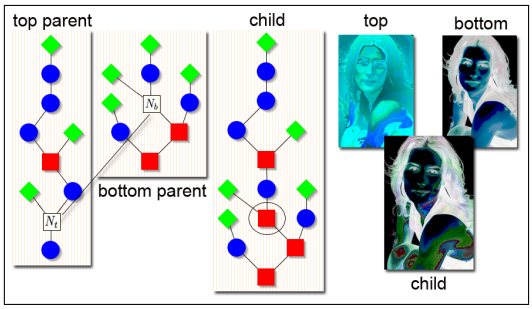

Fig. 3. An example of the crossover operation. We see that the child image inherits the colours from both parents and the texture from the top parent.

incompatible, and a new couple is chosen. If they are compatible, however, the tree above and including N_t is substituted for the tree above and including N_b to produce the offspring. An example crossover operation is shown in figure 3.

We have implemented three techniques for mutating the offspring trees. Firstly, each node in a tree will be parameterised by a set of values, and the *low-mutate* mutation method looks at each parameter of each node and, with a probability of P_l, generates a new set of parameters for the node. Secondly, the *mid-mutate* mutation method looks at each node in the tree, and with a probability of P_m, chooses to replace that node by a suitable alternative (i.e., transforms are replaced by transforms and compositors are replaced by compositors). Finally, the *high-mutate* mutation method looks at each node in the tree and with a probability of P_h replaces the tree above and including the node with a randomly generated sub-tree (as per the random generation method described above, and subject to the same restrictions as for entire filter trees).

4 Experiments and Results

Our ambition here is modest: we wish to show that evolutionary search for image filters converges on solutions which have some visual similarity with the target filter. We have only experimented with a very small subset of the parameters used in automating the evolution of image filters, and we discuss further experiments that we plan to undertake in section 6.

4.1 Reducing Image Size

As previously mentioned, the time taken to perform a filter is proportional to both the image size and the size of the filter. Hence, since in initial studies we saw no discernable difference in the quality of the filters produced using reduced images, we experimented with reducing the size of the original (unfiltered) image in order to increase the efficiency of the evolutionary process. Reducing the size of the image to be filtered is not detrimental to an evolutionary search if the ordering of individuals in terms of the fitness function does not change too much. To determine an appropriate scaling, we used the first fitness function described

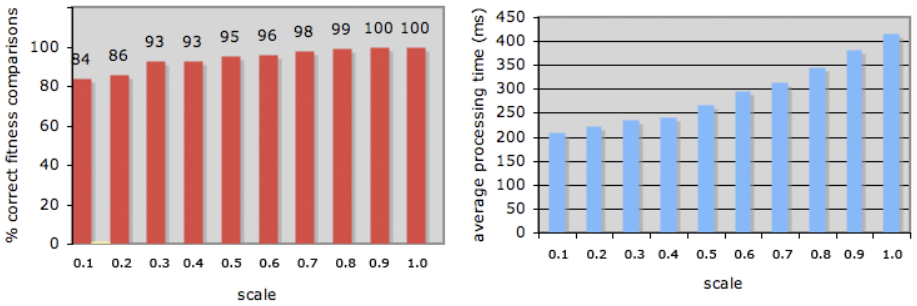

Fig. 4. First graph: percentage of correct fitness comparisons at various scales (from 250 tests). Second graph: average time taken to perform a single image filter and evaluation operation.

above, namely the average Euclidean distance of pixels in RGB-space. Scaling the original image at scales of $0.1, 0.2, \ldots, 1.0$, in each case, we tested 250 sets of three filters f_1, f_2 and f_3 randomly chosen from our library of 1000 filters. For each triple, using f_1 as the target filter, we measured the fitness of f_2 in terms of its approximation of f_1 and the fitness of f_3 in the same terms. We repeated this for the scaled images, and recorded a hit if the same filter (from f_1 and f_2) was determined to be fittest in the scaled case as in the unscaled case. We also recorded the average time taken to perform a single filtering operation and fitness evaluation. The percentage hit rates and the average times are shown in figure 4. Note that these tests were performed with an original image of size 256 by 384 pixels on a Mac OS X 2.6Ghz. Looking for a balance between gain in efficiency and fidelity of fitness comparison, we chose to reduce our original image to 0.3 times its size in all subsequent experiments. This gives a 43.5% increase in efficiency for only a 7% loss in the fidelity of fitness comparisons.

4.2 Search Setups

We have barely scratched the surface of possibilities for the evolutionary search for image filters – indeed, our search setups are rather simplistic at the moment. Common to all the search setups are the following aspects: we used a population size of 100 and evolved for at most 100 generations. Moreover, the search was set to terminate if there was no improvement in the fittest individual or the average fitness over the population for five generations in a row. Note that in all but a handful of the 375 sessions we ran, the search terminated before the full 100 generations were exhausted. We used a simple selection method: the top 25% of filters were put into an intermediate population, from which pairs were chosen randomly and a single offspring produced if possible. Compensating for discarding 75% of each population, to increase diversity, new filters were only allowed into the next population if the image they produced differed in at least 5% of its pixels with the images produced by all the filters which were already added. We found that in most cases, this measure was enough to avoid premature convergence of the search. Offspring filters were required to have between 3 and

20 nodes in their trees, and we imposed a mutation rate of 0.1 for each node in the offspring tree, using the mid-mutate technique only. We experimented with five search setups, which differed as follows:

• *Setup 1:* Random generation of the initial population, with a random tree size limit of 20 nodes. Fitness function: Euclidean RGB-distance.
• *Setup 2:* As setup 1, except the initial population is generated by searching through our library of 1000 filters using methods described in [9] to find the closest matching filters. Naturally, when the target filters were themselves from the library, we did not allow them to be retrieved into the initial population.
• *Setups 3, 4 and 5:* As setup 1, but with weighted histogram/contrast fitness described in §3.1 with weights $\langle 1, 0\rangle$, $\langle 0.75, 0.25\rangle$, $\langle 0.25, 0.75\rangle$, respectively.

We experimented with 75 image filters, namely the 46 filters which come with the standard Photoshop distribution, using the default settings, and 29 from our library of 1000 filters, chosen to cover a wide range of visual styles. The fitness of best individual seen in any population, averaged over all filters (both from our library and from Photoshop), for search setups 1, 2, 3, 4 and 5 was 0.92, 0.92, 0.94, 0.94 and 0.97, respectively. For all the search setups, it was possible to evolve filters with fitness scores greater than 0.9 on average, which is encouraging. However, this belies the fact that filters can be evolved to maximise the fitness functions without necessarily producing images which look like the target filter – see the next subsection for examples. We also note that choosing to generate filters randomly, or choosing them from our library produces very little difference in terms of the fitness of the best evolved filter. On inspection of the data, we found that – in general – the initial populations for setup 2 score worse than for setup 1, but the subsequent populations catch up fairly quickly. We also see that it is easier to evolve filters which maximise the histogram/contrast fitness function than the RGB-distance fitness function. The production of a new population takes around 73 seconds (on 2.6Ghz machine). Over the 375 sessions, the shortest/average/longest sessions produced 11/31/100 populations, taking 13 minutes, 37 minutes and approximately 2 hours respectively.

4.3 Illustrative Examples

In figure 5, we present 18 illustrative examples of the best (in terms of the appropriate fitness function) evolved filters, with 9 where our library provided the target filter and 9 where it was provided by Photoshop. We have tended to show the better results, but we include the approximations of the *embossed7* filter as an example where none of the search setups produced a visually similar or appealing filter. This example indicates a phenomena we saw fairly often, namely filters evolving to produce single colour filters (which minimises the RGB-distance on average when the target filter produces images dominated by one colour). Another phenomena we saw fairly often was the evolving of filters which produce rather greyscale images when the target filter actually produces quite colourful images. Again, this can be explained, as greyscale images have pixels which are not at extremes in the RGB-space, hence they tend to maximise fitness.

Fig. 5. Illustrative examples of the best filters evolved for each search setup. First column: target filters from our library. Second column: target filters from Photoshop.

Such a greyscale image was produced with search setup 2 for the *modern12* filter and to a lesser extend with setup 3 for *colourful14*. We describe some ways in which to combat this phenomena in section 6. Note, however, that when using the RGB-distance fitness function, colourful filters can be evolved for colourful target filters, as evidenced in searches 1 and 2 for the *psychodelic13* filter.

Some highlights (judged entirely subjectively by ourselves) include: (a) the near perfect re-creation of the *bitone3, bright2, colourful14* and *psychodelic13* filters (and also the *grey1* and *vivid1* target filters, not shown in figure 5) (b) the fairly good approximation of some Photoshop filters, including the *charcoal, dark strokes, diffuse glow, film grain, glowing edges, neon glow, patchwork* and *stamp* target filters. Also, many filters found using the histogram/contrast fitness function were not exact matches, but had a similar visual style to their targets, e.g., the filter developed using setup 5 for *fade4* captures its style well; the filter generated by setup 2 to approximate *painterly28* has a particularly painterly

style, and while the approximation to the Photoshop *chrome* filter produced by setup 5 is a bad approximation, it does capture some of its essence. On the whole, the evolution performs better for our library filters than for the Photoshop filters. This is largely due to the Photoshop filters being rather subtle, i.e., they alter the image very little. In these cases, an identity filter scores highly for all the fitness functions, and so the search tends to discover ways in which to construct trees which do nothing. However, there are some exceptions: e.g.,, the *accented edges* filter is very subtle, but setups 1 and 4 found good approximations.

5 Related Work

The NEvAr system [3] is an evolutionary art tool which uses a similar tree-based representation for image filters but is different in that the system described here (1) explicitly uses an initial existing image as a seed for evolution, (2) uses tree nodes which correspond to high-level transformations of the input images, (3) uses a target image as a fitness function. A general approach for evolving image filters tree was introduced by Sims [7] and although complex image transformation were allowed at node level, it still differs from the work present here on aspects (1) and (3) above. In [4], which is conceptually close to the approach described here, the authors use Genetic Programming to evolve programs that give colour to greyscale images based on existing coloured training images. Particular applications of image filter evolution include evolving noise removal filters for graphics processor units using Cartesian genetic programming [2] and an implicit context representation [8], and evolving simple filters such as the salt and pepper filter for Field Programmable Gate Arrays [6].

6 Conclusions and Future Work

Using a tree-based representation of image filters, we have investigated how to evolve approximations of a target filter, by referring to an image produced by the target filter when calculating the fitness of individuals in a population. We have experimented with five search setups which use four fitness functions, and we have shown that in some cases we can evolve filters which very closely approximate the target, and in other cases, we can evolve filters which have a similar visual style to the target. On inspection, 17 of the 46 Photoshop filters (37%) were approximated in an appropriate way by at least one of the search setups, and 21 of the 29 filters from our library (72%) were adequately approximated. While this is encouraging, there is certainly room for improvement.

In future work, we hope to find more effective search setups, by experimenting with more sophisticated selection techniques, as well as parameters such as the population size, mutation rate, mutation style, and different methods for tree-based crossover [5]. We also plan to experiment with different fitness functions, to address some of the limitations identified here. In particular, searches can converge on identity filters, filters which produce single-colour images and

greyscale images (for colourful target filters). To address this, we plan to experiment with fitness functions which look at the mapping from the original to the filtered image that the filter produces. In particular, we will use a sliding scale of importance for pixels: those which are not altered at all by the target filter will be ignored by the fitness function, while those which are altered a great deal will be used prominently. We will also consider different numbers of generations of consecutive identical fitness as a terminating criterion for evolution. We hope that such a fitness function will also enable the evolution of filters which approximate the more subtle Photoshop filters, where most of the pixels remain the same. Finally, given the lack of perceptual uniformity in RGB space, we plan to carry out the same set of experiments for different colour spaces, e. g. HSV.

By substituting the transforms and compositors in our filter trees by more sophisticated operations, we believe that evolving filters could be a very useful tool in an environment such as Adobe Photoshop: inexperienced users could ask the software to produce a filter using only an example image, and experienced users could ask the software for variations on a theme. We believe that future graphic design software should be able to act as such a creative collaborator, and we hope to see AI methods being used increasingly in this field.

Acknowledgements

We would like to thank Stefan Rüger and João Magalhães for their valuable suggestions regarding image analysis and the anonymous referees for their useful comments. This work is supported by EPSRC grant EP/F067127.

References

1. C Behrenbruch, S Petroudi, S Bond, J Declerck, F Leong, and J Brady. Image filtering techniques for medical image post-processing: an overview. *British Journal of Radiology*, 77(2).
2. S Harding. Evolution of image filters on graphics processor units using Cartesian genetic programming. In *IEEE Congress on Evolutionary Computation*, 2008.
3. P Machado, A Cardoso. All the truth about NEvAr. *App. Int. 16, 101 – 118*, 2002.
4. P Machado, A Dias, N Duarte, A Cardoso. Giving Colour to Images. In *Proceedings of the AISB'02, Symposium on Artificial Intelligence and Creativity in Arts and Science, London*, 2002.
5. R Poli, W Langdon. Schema Theory for Genetic Programming with One-Point Crossover and Point Mutation. Evolutionary Computation 6(3): 23 1-252, 1998.
6. L Sekanina and T Martínek. Evolving image operators directly in hardware. In *Proc. of Genetic and Evol. Computation for Image Processing and Analysis*, 2007.
7. K Sims. Artificial evolution for computer graphics. *Proceedings of SIGGRAPH '91.*
8. S Smith, S Leggett, and A Tyrrell. An implicit context representation for evolving image processing filters. In *Proceedings of the EvoWorkshop on Applications of Evolutionary Computing*, 2005.
9. P Torres, S Colton, and S Rüger. Experiments in example-based image filter retrieval. In *Proceedings of the Cross-Media Workshop*, 2008.