Convolutional Genetic Programming

Lino Rodriguez-Coayahuitl[1[0000−1111−2222−3333]], Alicia Morales-Reyes[1[1111−2222−3333−4444]], and Hugo J. Escalante[1[2222−−3333−4444−5555]]

National Institute of Astrophysics, Optics and Electronics, Sta. Ma. Tonantzintla PUE 72840, Mexico
{linobi,a.morales,hjair}@inaoep.mx

Abstract. In recent years, Deep Neural Networks (DNN), a special type of Convolutional Neural Networks (CNN), have come to dominate many machine learning related tasks, specially those related to image processing, such as object recognition, detection, etc. Here we explore the posibility of developing image denoising filters by stacking multiple Genetic Programming (GP) syntax trees, in a similar fashion to how CNNs are designed. We test the evolved filters performance in removing additive gaussian noise. Results show that GP is able to generate a diverse set of feature maps at the 'hidden' layers of the proposed architecture. Although more research is required to validate the suitability of GP for image denoising, our work set the basis for the briding the gap between deep learning and evolutionary computation.

Keywords: Deep genetic programming · Evolutionary Machine Learning · Genetic Programming · image filtering.

1 Introduction

Convolutional Neural Networks (CNN) are a type of connectionist machine learning (ML) algorithms particularly adept at image processing tasks [13, 12]. This is thanks to a clever architectural design that allows them to scale well to high dimensionality problems.

In recent years, a special type of CNN known as Deep Neural Networks (DNN) have achieved record performance in typical ML tasks such as classification, regression or prediction, outclassing both systems handcrafted by human experts of the problem's domain and ML systems based on techniques other than CNN [11]. DNN have achieved this performance thanks to an ever increasing number of stacked convolutional layers [10, 18, 6].

In this work we explore the possibility to implement the fundamental architecture of CNN through a different algorithmic paradigm, Genetic Programming (GP) [9]. GP is an evolutionary algorithm typically used for ML tasks.

Our motivation to import the architectural design of CNN into GP is twofold: first, we wish to explore the idea of replacing neurons in CNN with GP syntax trees, as we believe they have the same, or even higher, computational power

than that of CNN's neurons; and secondly due to the fact that GP does not scale well to high dimensionality problems [5], and we suspect it might benefit from the same architectural design than that of CNN.

In order to test our proposed approach, we use as target problem the task of image denoising. The purpose of image denoising is to recover a clean image from contaminated original. The contamination model may be of different kinds. In this work we attempt to clean images from additive gaussian noise.

The original contributions of this work are:

- We generate GP-based image denoising filters that operate at individual pixel level.
- We propose the multi-layer convolutional GP architecture.
- We propose different training/evolution mechanisms that target the multi-layer convolutional GP architecture.
- We compare the performance of the proposed GP filters to that of recent DNN.

The implicit relevance of this work lies in the fact that for the first time, to the best of the authors' knowledge, we establish in a quantitative manner, the performance gap between evolutionary algorithms/GP and Deep Learning. Many other works related to this subject have avoided such direct comparison.

2 Background

In this section we briefly introduce GP and define the target problem.

2.1 Genetic Programming

In the context of ML, GP consists in a population of candidate solutions to the problem at hand. These candidate solutions are called individuals. Each individual's performance is tested against a training dataset; the best individuals are selected to reproduce through the use of genetic operations, i.e. generate slightly modified versions of themselves; these new candidate solutions are also evaluated and the best performing of them replace the worst performing of the original ones, leading to a new generation of individuals, from which the process repeats until a stop criterion is met.

Canonical individuals in GP are syntax trees that represent a mathematical function or simple computer programs [9, 14]. Internal nodes in these trees are basic functions called primitives, while leaf nodes are constants or the feature variables from the instance being processed. In this way, data flows from bottom nodes to the top root node where the final output is generated. Primitives are usually unary, binary or ternary mathematical operations such as addition or substraction between two inputs (binary), or square root or sine and cosine functions (unary). Fig. 1 shows an example of a tree structure that represents the function $f(x, y) = (2.2 - (\frac{x}{11})) + (7 * \cos(y))$ [3].

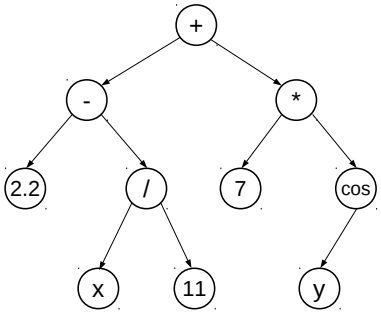

Fig. 1. Tree structures like this are typically used in GP to represent individual candidate solutions.

Problems with high dimensionality inputs, such as in the case of image processing tasks, are challenging for ML algorithms for several reasons, such as time complexity issues (time required to train such algorithms), the *curse of dimensionality* [2], and the large number of parameters that need to be tuned inside the algorithms for them to work properly when faced with such high dimensionality problems.

In the case of GP, the high dimensionality issues arise due the nature canonical individual representation itself. Notice how the depth of the trees has to increase in order to accomodate more input features at the leaf nodes. Larger trees means an exponentially growing search space of candidate solutions, that eventually becomes intractable.

2.2 Image Denoising

The problem of image denoising is defined as follows: extract a clean image $\mathbf{x}$ from a noisy observation $\mathbf{y}$ such that $\mathbf{y} = \mathbf{x} + \mathbf{v}$, where $\mathbf{v}$ is a contamination process; a typical example is when $\mathbf{v}$ follows a Gaussian distribution with some given σ, which case is known as Additive Gaussian Noise (AWGN).

3 Related Work

GP has been succesfully used in the past to synthetize image filers. Examples of these type of works can be found in [21, 8]. However, these works rely on a modified version of the canonical GP individual such that primitive functions may include already specialized image filters or at least well known image processing functions. This property is undesirable if we wish to build ML systems that relied as little as possible on domain human expert's knowledge, i.e. highly automated learning systems. A more agnostic approach has been proposed in [7], where terminals of the syntax trees consist in simple statistics taken over regions of pixels.

It is relevant to contrast such specialized GP approaches with recent developments in the area of Deep Learning (DL). DNN are a special type of CNN composed of several stacked convolutional layers. Each convolutional layer is made of linear approximators coupled with a non-linear transformation. There is really nothing specialized regarding image processing in the architecture of DNN other than the use of convolution to efficiently process images. DnCNN [22] is a recent DNN designed to tackle image denoising; its flexibility is such that, by just switching the dataset with which is trained, the same network can learn to remove vastly different types of noises such as Gaussian noises with different or unknown levels of deviation, deblocking artifacts, and can even perform super resolution. DnCNN is competitive with fully and partially handcrafted image filters designed by human experts, thus positioning DNN as very powerful learning systems.

In more general terms, high dimensionality issues have been long acknowledged in the GP community [5]. Standard approaches to tackle such issues generally involve grouping input features in one way or another, process each cluster separately, and then attempt to assemble a joint global solution [20, 15]. Rodriguez-Coayahuitl et al. [15] proposed a GP autoencoder that generated a compact representation of an input image and could decode the original image from the compact representation. The proposed autoencoder relied on the canonical GP individual representation. In order to use the proposed GP autoencoder on images, it was required to partition the input images in small groups of neighboring pixels that are processed independently in isolated GPs. Even though this approach allowed GP to process a large enough input such as images, it is still not the most efficient approach, since the isolated GPs did not share information with neighboring GP processes and such many independent GP required vast amounts of memory and processing power.

On the hand, in this work we draw inspiration from CNN and propose a single sliding GP window that swipes an input image for processing, instead of many multiple independent GP processes.

4 Proposed Method

Our approach to evolve image denoising filters through GP is to leverage from the CNN architecture, where we replace neurons with GP syntax trees. Initially we propose to evolve a single sintax tree that acts as image filter by sliding over the noisy input image and cleaning pixel by pixel. Thereafter, we propose to stack multiple layers of these GP filters. We explain the theoretical advanges of stacking filters in this manner further below in this section.

4.1 Single Layer Convolutional GP Filter

We propose to use a standard GP individual representation, i.e. a syntax tree, to act as an image filter. This filter operates over a small window region of $d \times d$ pixels (where d is an odd number), receiving as input the pixels within

such region, and returning as output a single value that is the level of noise of the central pixel in the operating window; in order to filter a whole image, the window is slided over an the entire image, generating a residual image the same size of input image that we desire to clean of noise. Fig. 2a shows a depiction of the proposed sliding GP filter. This residual image represents the (estimated) level of noise of each pixel that composes the input image. In order to retrieve an approximation of the clean image, we simply substract the residual image from the noisy input image.

The leaf nodes of the GP individual should be the individual pixels in the region being processed, or constanst values within some range. The primitives can be any function that can operate at this individual pixel level. This is done in this way to avoid the use of any image filtering expert's knownledge.

4.2 Multi-layer Convolutional GP

Additionally we also propose to stack multiple of these sliding GP filters, both in parallel and in series, since DNN are actually designed this way. That is, instead of using a single GP syntax tree that filters the image, we can slide multiple, different, GP syntax trees that generate as output *feature maps*, which are intermediate transformations of the input that may be useful for generating the desired output. All these feature maps form a volume of codified information that is further processed by another GP sliding tree that generates the final output, i.e. the residual image. Fig. 2b shows a GP filter architeture composed of two stacked filter in series, while Fig. 2c depicts an architecture with multiple GP filters both in series and in parallel.

Stacking these convolutional filters in series carries the advantage of increasing the *field of view*. This means that if we use two sliding filter with windows of 3×3 in series, when we reconstruct the central pixel at the output of the second filter, we are actually using information of a 5×5 window size around it (this is as along as the first filter did manage to codify information at feature map it outputs). Notice however how when using this approach in such example scenario, we are using only 18 features as inputs (9 for the first filter and 9 for the second) whereas if we attempt to directly process filters of 5×5 window size, then such filter would need to process 25 input features. This is one of the reasons on why Convnet and DNN scale well to high dimensionality problems and image processing tasks.

On the other hand, stacking filters in parallel per layer allows to generate more than one feature map at each layer. Each feature map might codify different information useful for the next layer of processing.

The canonical form of GP contemplates individuals that are composed of a single syntax tree. In our proposed method, in the case of multiple stacked filters, we would need to evolve more than a single GP tree. Although there do exists GP individual representations based on forests (multiple trees), in this type of representations the trees are loosely dependent on each other, whereas in the multilayer architecture we are proposing here, the filter trees series rely completely on the output generated by the previous trees in the structure.

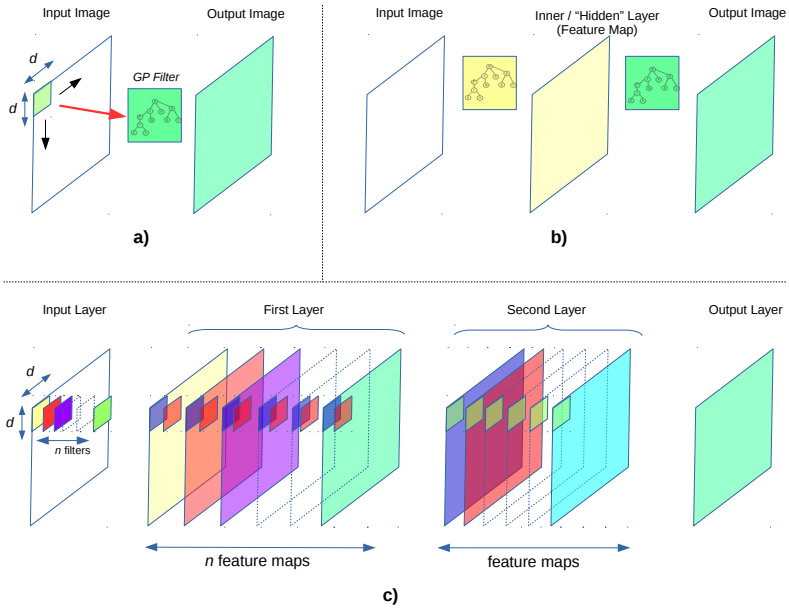

Fig. 2. Multilayer GP architecture. a) Single layer, single filter; b) Two layer, one filter per layer; c) Three layer, first layer and second with n filters, third layer with only 1, output, filter.

4.3 Evolving Multiple Layers of Convolutional GP filters

In order to train this complex architecture, we propose three different approaches: (**straightforward**) define the GP individual as the entire set of trees across all layers, evolve individuals by applying genetic operations layer-wise; (**sequential**) evolve the multi-layer structure sequentially, i.e. evolve the first layer for fixed number of generations; once this first evolution is finished, the second layer of filters are evolved, which take as input a cleaner version of the noisy image generated by the first layer, and so on; (**ensamble**) the third approach is based on the idea that the multiple feature maps at the penultimate layer might actually act as ensamble learner, with the last layer only performing the mean function, so in this architecuture we enforce this behavior by taking as output the mean over the feature maps of the last layer. Fig. 3 illustrates these three variants.

5 Experimental Results and Analysis

In this section we present and discuss experimental results of the different variants of the proposed method.

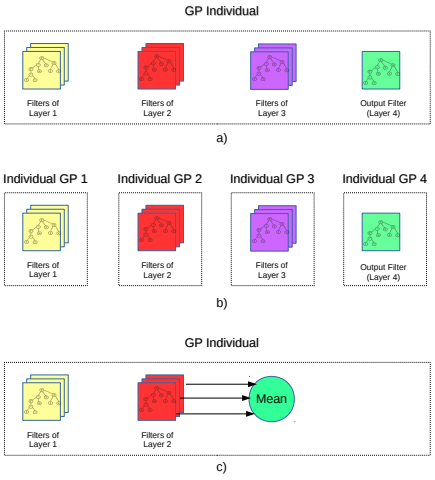

Fig. 3. Different possible GP individual representations for multilayer GP filters.

5.1 Training and Testing Datasets

We generated the training data following the works of [22, 17, 4]. From the Berkeley Segmentadion Dataset [16] we extracted 19,200 unique 40×40 image patches for training purposes. For testing, we use the same classic image processing set used in [22, 21, 8], composed of well-known pictures such as "Lena" and "Boats". A total of 12 (seven 256×256 and five 512×512) pictures were used for testing.

We contaminated both the training patches and the testing images by adding them noise masks generated with a Gaussian distribution of $\sigma = 25$. All training and testing was performed on grayscale images.

5.2 Evolutionary Algorithm Setup

For all experiments we used a multi-population, island based, model [19]. We used a population of 500 individuals splitted across 5 islands each with 100 individuals. We used an heterogeneous and asynchronous [19] model where each island had different crossover/mutation probabilities, and every 10 generations send their top 10 performing individuals to another, randomly selected, island (migration). The crossover/mutation probabilities were set as follow for each island: $[0.9/0.1, 0.7/0.3, 0.5/0.5, 0.3/0.7, 0.1/0.9]$. The set of primitives used consist on binary arithmetic operators, $[+, -, \times, \div]$, binary functions *max*, *min*, *mean*, and unary functions x^2, x^3, and Rectifier Linear Units (ReLUs).

We utilized an on-line form of learning defined in [15]. We partitioned the entire training dataset into mini-batches of 60 samples, and use one mini-batch per evolutionary cycle for evaluating both individuals and offspring generated. We used a steady state population replacement policy.

Fitness function We used the minimization of the mean square error (MSE) between the predicted noise level and the actual noise level to drive the evolution of all systems proposed.

5.3 Results

We tested two Single Layer Convolutional GP, one consisting in a sliding window of 3×3 pixels, and another with a window of 5×5 pixels.

We tested three different Multi-layer Convolutional GP, each under one of the three different proposed methods for evolving multi-layer GPs. All Multi-layer architectures consisted in only 2 layers (2 layers + mean, in the case of the ensamble method). Both the straightforward and the sequential architectures were composed of 3 filters in the first layer, and 1 filter in the second layer (3 filters in both layers for the ensamble method). All filters were 3×3 windows.

Table 1 shows the results obtained by the different tested approaches. We include in Table 1 the values of the unfiltered noisy images (to understand how much the proposed approaches actually denoise the images), as well as the performance of the DnCNN network [22], to fully appreciate how far GP is from modern DNN. These results were obtained on the same testing dataset for all approaches (including the DnCNN), and using the same training dataset (also applies for DnCNN). All the GP approaches were given the same computational time[1]. Therefore these results are based on a comparison as fair as possible.

Table 1. Average performance of all Convolutional GP architectures tested. Values expressed in decibels. Higher is better.

Noisy Image	**Single GP, 3x3**	**Single GP, 5x5**	**Strfwd-GP (2 Layers)**	**Sequential GP (2 Layers)**	**Ensamble (2L + Mean)**	**DnCnn**
20.32	25.96	25.07	25.22	25.93	23.60	30.43

Fig. 4 shows the performance of a 2-Layer, Sequentially evolved variant GP, on ten training patches. We found no visually appreciable difference between this output and the one from a single layer GP.

5.4 Additional Results

We also performed experiment using 10 filters per layer for the Multi-layer GP architectures. Although we found them to be consistently inferior in performance to the 3 filters per layer reported above, we found that these GP variants generated interesting paterns in the hidden layer. Fig. 5 shows the feature maps generated by the ten filters for ten different training patches. Some of the feature maps appear to be signaling borders or other points of interest.

[1] DnCNN runs in less time than GP, due to being accelerated in GPU and implemented in highly optimized DL software libraries.

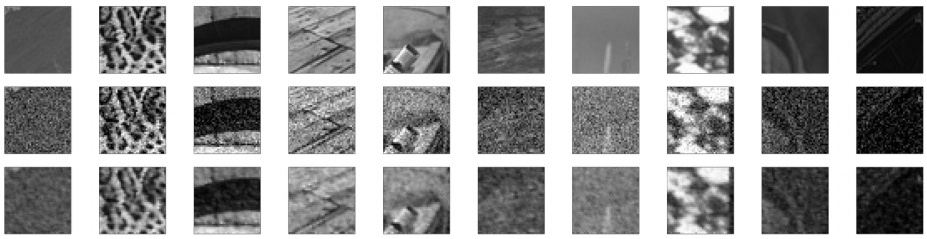

Fig. 4. Visual results of the output generated by a 2-Layer Convolutional, Sequentially evolved, GP. From top to bottom: original images, noisy samples, filtered images.

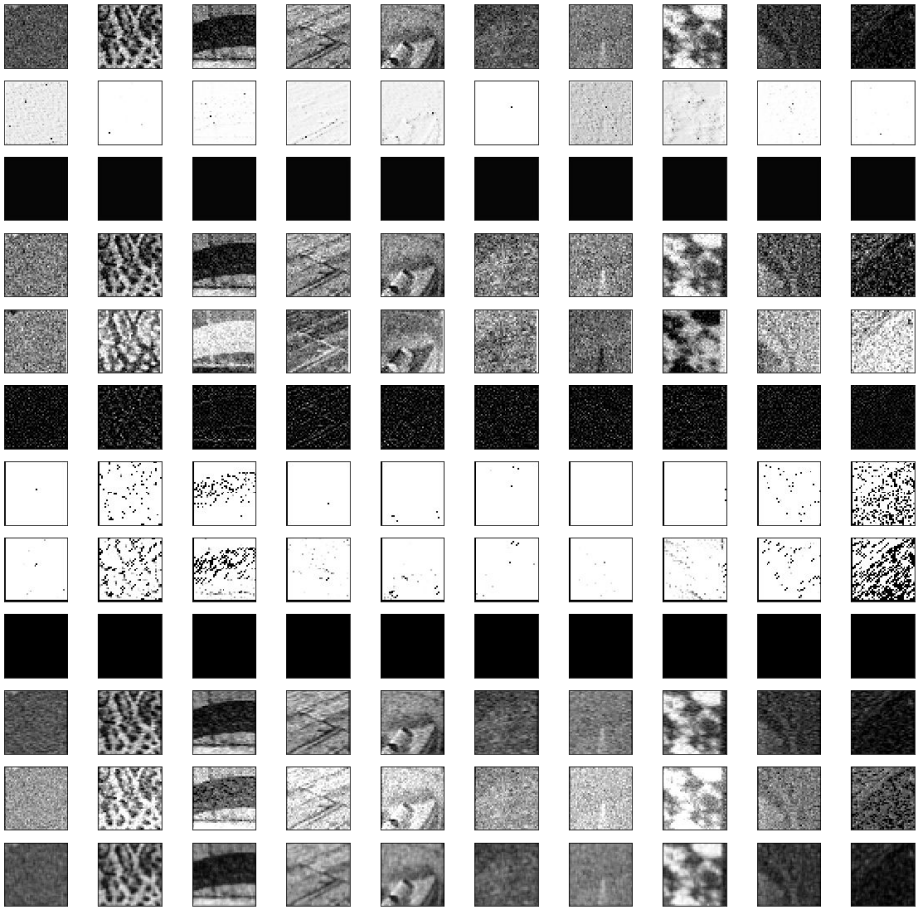

Fig. 5. Feature maps generated by a 2-layer GP in the hidden layer given 10 different input patches. From top to bottom: first row, noisy patches rows 2 to 10, feature maps; last row, filtered final output.

5.5 Discussion

Results shows that GP can succesfully synthetize image denoising filters, even though none of the proposed methods allows GP to benefit from a multi-layer convolutional architecture, thus positioning a single layer GP filter as the reference method-to-beat in future works based on GP.

Results also confirm that GP struggles with high dimensionality problems. In this case, a single layer 5×5 window GP filter does not performs any better, if not worse, than a 3×3 window one, even though the first one has more than twice context information that theoretically should allow it to perform a better filtering.

6 Conclusions

In this work we have introduced a method to evolve image denoising filters with GP, through an architecture inspired by CNN. Our results have confirmed that:

- GP is a viable method to synthetize image denoising filters, even when processing images at individual pixel level.
- GP struggles with high dimensionality problems, since it cannot make use of input samples with as low as 25 features.
- GP cannot directly benefit from a stacked convolutional architecture. More research is necessary in this direction.

We have also draw a clear, quantitavive, performance gap between GP and DL based methods, by using the same exact training and testing datasets, and making head-to-head direct comparison with modern DNN architectures.

We believe this work should serve a reference for future works that attempt to attack problems with GP in which DL excels at.

References

1. Al-Sahaf, H., Song, A., Neshatian, K., Zhang, M.: Two-tier genetic programming: Towards raw pixel-based image classification. Expert Systems with Applications **39**(16), 12291–12301 (2012)
2. Alpaydin, E.: Introduction to machine learning. MIT press (2009)
3. Axelrod, B.: Genetic programming (2007), accessed 05/05/17
4. Chen, Y., Pock, T.: Trainable nonlinear reaction diffusion: A flexible framework for fast and effective image restoration. IEEE transactions on pattern analysis and machine intelligence **39**(6), 1256–1272 (2017)
5. Gathercole, C., Ross, P.: Tackling the boolean even n parity problem with genetic programming and limited-error fitness. Genetic programming **97**, 119–127 (1997)
6. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: Proceedings of the IEEE conference on computer vision and pattern recognition (2016)

7. Hernández-Beltrán, J.E., Díaz-Ramírez, V.H., Trujillo, L., Legrand, P.: Restoration of degraded images using genetic programming. In: Optics and Photonics for Information Processing X. vol. 9970, p. 99700K. International Society for Optics and Photonics (2016)
8. Khmag, A., Ramli, A.R., Al-haddad, S., Yusoff, S., Kamarudin, N.: Denoising of natural images through robust wavelet thresholding and genetic programming. The Visual Computer **33**(9), 1141–1154 (2017)
9. Koza, J.R.: Genetic programming: on the programming of computers by means of natural selection, vol. 1. MIT press (1992)
10. Krizhevsky, A., Sutskever, I., Hinton, G.: Imagenet classification with deep convolutional neural networks. In: Advances in neural information processing systems (2012)
11. LeCun, Y., Bengio, Y., Hinton, G.: Deep learning. nature **521**(7553), 436 (2015)
12. LeCun, Y., Bengio, Y., et al.: Convolutional networks for images, speech, and time series. The handbook of brain theory and neural networks **3361**(10), 1995 (1995)
13. LeCun, Y., Boser, B., Denker, J.S., Henderson, D., Howard, R.E., Hubbard, W., Jackel, L.D.: Backpropagation applied to handwritten zip code recognition. Neural computation **1**(4), 541–551 (1989)
14. Poli, R., Langdon, W.B., McPhee, N.F., Koza, J.R.: A field guide to genetic programming. Lulu. com (2008)
15. Rodriguez-Coayahuitl, L., Morales-Reyes, A., Escalante, H.J.: Structurally layered representation learning: Towards deep learning through genetic programming. In: European Conference on Genetic Programming. pp. 271–288. Springer (2018)
16. Roth, S., Black, M.J.: Fields of experts: A framework for learning image priors. In: null. pp. 860–867. IEEE (2005)
17. Schmidt, U., Roth, S.: Shrinkage fields for effective image restoration. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 2774–2781 (2014)
18. Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., Rabinovich, A.: Going deeper with convolutions. In: Proc. of CVPR. pp. 1–9 (2015)
19. Tomassini, M.: Spatially structured evolutionary algorithms: Artificial evolution in space and time. Springer (2006)
20. Tran, B., Xue, B., Zhang, M.: Using feature clustering for gp-based feature construction on high-dimensional data. In: European Conference on Genetic Programming. Springer (2017)
21. Yan, R., Shao, L., Liu, L., Liu, Y.: Natural image denoising using evolved local adaptive filters. Signal Processing **103**, 36–44 (2014)
22. Zhang, K., Zuo, W., Chen, Y., Meng, D., Zhang, L.: Beyond a gaussian denoiser: Residual learning of deep cnn for image denoising. IEEE Transactions on Image Processing **26**(7), 3142–3155 (2017)