






Convolutional Genetic Programming

Lino Rodríguez-Coayahuitl^(✉) , Alicia Morales-Reyes ,
and Hugo Jair Escalante 

Instituto Nacional de Astrofísica, Óptica y Electrónica,
Sta. Ma. Tonantzintla, 72840 Puebla, Mexico
{linobi,a.morales,hugojair}@inaoep.mx

Abstract. In recent years Convolutional Neural Networks (CNN) have come to dominate many machine learning tasks, specially those related to image analysis, such as object recognition. Herein we explore the possibility 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 bridging the gap between deep learning and evolutionary computation.

Keywords: Deep Genetic Programming ·
Evolutionary machine learning · Genetic Programming ·
Image filtering · Deep Learning

1 Introduction

Convolutional Neural Networks (CNN) are a type of connectionist machine learning (ML) algorithms particularly adept at image processing tasks [9]. This is thanks to a clever architectural design that allows them to scale well to high dimensionality problems. In recent years, CNN and Deep Neural Networks (DNN) in general have achieved record performance in typical ML tasks such as classification and regression, outclassing both systems handcrafted by human experts of the problem's domain and ML systems based on techniques other than CNN [8]. DNN have achieved this performance thanks to an ever increasing number of stacked convolutional layers [7, 13].

Herein we explore the possibility to implement the fundamental architecture of CNN through a different algorithmic paradigm, Genetic Programming (GP) [6]. GP is an evolutionary algorithm typically used for ML tasks. In GP a population of solutions (often encoding models) is evolved by using mutation and crossover operators. GP is known to be suitable for modeling highly complex functions, hence we think it is appealing to mimic tasks approached by CNN.

The motivation to follow CNNs' architectural design through GP is twofold: first, we wish to explore the idea of replacing neurons in CNNs 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 [3], and we suspect it might benefit from CNNs’ architectural design.

In order to test the proposed approach, we tackle the problem of image denoising. The purpose of image denoising is to recover a clean image from a contaminated one. The contamination model may be of different kinds. In this work we attempt to clean images from additive Gaussian noise, which is a fairly standard problem targeted by CNN models.

The main contributions of this work are as follows:

- We introduce a novel GP-based method for image denoising filters that operates at pixel level.
- We propose a multi-layer convolutional GP architecture.
- We propose different training/evolution mechanisms to suit the proposed multi-layer convolutional GP architecture.
- We compare the performance of the evolved 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

2.1 Genetic Programming

GP is an evolutionary algorithm that iteratively modifies 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 solutions are also evaluated and the best performing replace the worst from current ones, leading to a new generation of individuals. This process repeats until a stop criterion is met. Canonical individuals in GP are syntax trees that represent a mathematical function or simple computer programs [6, 10]. Internal nodes in these trees are basic functions called primitives, while leaf nodes are constants or 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. Figure 1 shows an example of a tree structure that represents the function $f(x, y) = (2.2 - (\frac{x}{11})) + (7 * \cos(y))$ [2].

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, the *curse of dimensionality* [1], and the large number of parameters that need to be tuned within algorithms to work properly when faced with such high dimensionality problems.

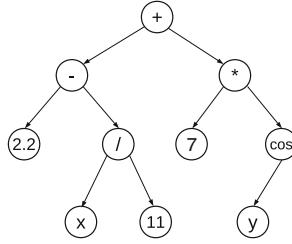


Fig. 1. Typical tree structures used in GP to represent candidate solutions.

In the case of GP, high dimensionality issues arise due the nature canonical individual representation itself. Notice how trees depth has to increase in order to accommodate more input features at 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). Figure 2 shows a noisy image contaminated with AWGN, as well as a clean version of it we wish to recover through image denoising.

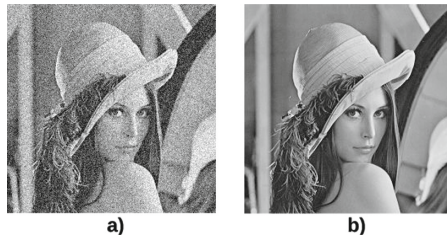


Fig. 2. (a) Image contaminated with AWGN; (b) clean image.

3 Related Work

GP has been successfully used in the past to synthesize image filters. Examples of those works can be found in [5, 16]. However, they 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

is undesirable if we wish to build ML systems that relies as little as possible on domain human expert’s knowledge, i.e. highly automated learning systems. A more agnostic approach has been proposed in [4], where terminals of the syntax trees consist of simple statistics taken over pixels regions.

It is relevant to contrast such specialized GP approaches with recent developments in the area of Deep Learning (DL). DNN are artificial neural networks composed by several stacked processing layers. CNN are a type of DNN where these processing layers perform convolutional operations. 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 [17] 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.

In more general terms, high dimensionality issues have been long acknowledged in the GP community [3]. 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 [11, 15]. In [11], authors 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 isolated GPs did not share information with neighboring GP processes and such many independent GP required vast amounts of memory and processing power.

Our work draws 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 A Convolutional GP for Image Denoising

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 syntax tree that acts as image filter by sliding over a noisy input image and cleaning it pixel by pixel. Thereafter, we propose to stack multiple layers of these GP filters. We explain the theoretical advantages 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, with d an odd number, receiving as input 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 slid over the entire image, generating a residual image with the same size of the input image that we want to clean of noise. Figure 3a shows a depiction of the proposed 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 subtract the residual image from the noisy input.

The leaf nodes of the GP individual should be the individual pixels in the region being processed, or constants 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 knowledge.

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 several *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 tree that generates the final output, i.e. the residual image. Figure 3b shows a GP filter architecture composed of two stacked filter in series, while Fig. 3c 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 *receptive field*. 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). 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.

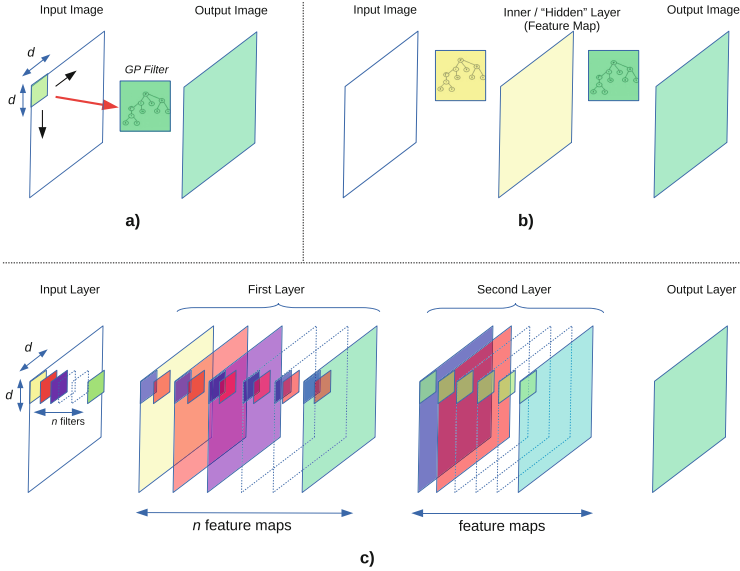


Fig. 3. Multilayer GP architecture. (a) Single layer, single filter; (b) Two layer, one filter per layer; (c) Three layer, first and second layers with n filters, third layer with only 1, output, filter.

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 ensemble learner, with the last layer only performing the mean function, so in this architecture we enforce this behavior by taking as output the mean over the feature maps of the last layer. Figure 4 illustrates these three variants.

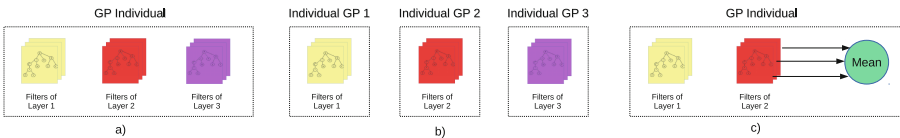


Fig. 4. Different possible GP individual representations for multilayer GP filters.

5 Experimental Results and Analysis

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

5.1 Training and Testing Datasets

We generated the training data following the work of [17]. From the Berkeley Segmentation Dataset [12] we extracted 19,200 unique 40×40 image patches for training purposes. For testing, we use the same classic image processing set used in [5, 16, 17], 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 training patches and 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 [14]. We used a population of 500 individuals split across 5 islands each with 100 individuals. We used an heterogeneous and asynchronous [14] 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). 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 used an on-line form of learning defined in [11]. 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. As 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, for the ensemble method). Both straightforward and sequential architectures were composed of 3 filters at the first layer, and 1 filter in the second layer (3 filters in both layers for the ensemble method). All filters were 3×3 windows. Table 1 shows results obtained by different tested approaches. We include in Table 1 unfiltered noisy images values

(to understand how much the proposed approaches actually denoise images), as well as DnCNN network performance [17], to fully appreciate how far GP is from modern DNN. These results were obtained on the same testing dataset for all approaches (including DnCNN), and using the same training dataset (also applies for DnCNN). All GP approaches were given the same computational time¹. Therefore these results are based on a comparison as fair as possible. For completion, Fig. 5 shows the performance of a 2-Layer, sequentially evolved GP, on ten training patches. We found no visually appreciable difference between this output and the one from a single layer GP.

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

Noisy Image	Single GP, 3×3	Single GP, 5×5	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

5.4 Additional Results and Discussion

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 patterns in the hidden layer. Figure 6 shows feature maps generated by ten filters for ten different training patches. Some feature maps appear to be signaling borders or other points of interest.

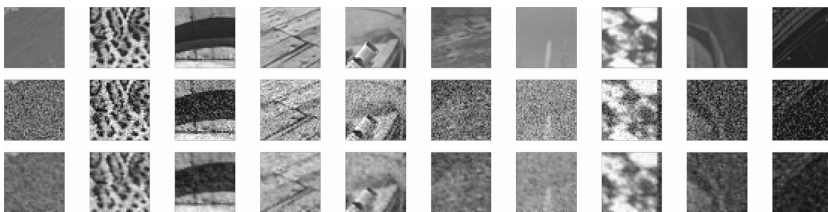


Fig. 5. Visual results of the output generated by a 2-Layer convolutional, sequentially evolved, GP. From top to bottom: original images, noisy samples, filtered images.

Results shows that GP can successfully synthesize 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

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

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 perform 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.

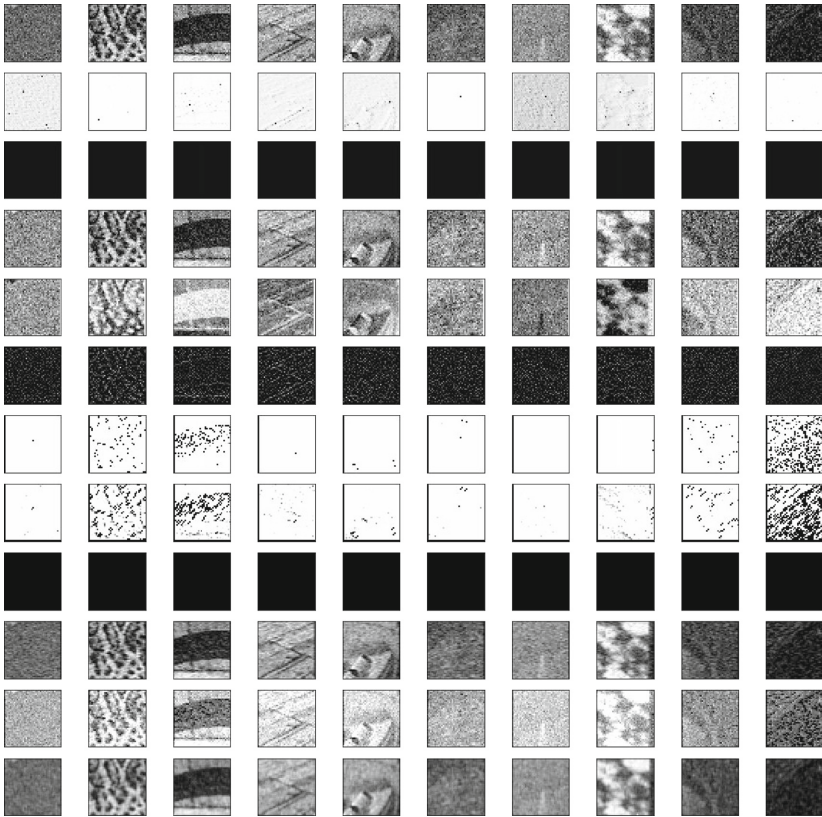


Fig. 6. Feature maps generated by a 2-layer GP in the hidden layer given 10 different input patches and 10 evolved GP filters in the hidden layer. From top to bottom: first row, 10 different noisy patches; rows 2 to 10, feature maps generated; last row, filtered final output.

6 Conclusions

We 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 synthesize 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, quantitative, 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 as a reference for future works that attempt to attack problems with GP in which DL excels at.

References

1. Alpaydin, E.: Introduction to Machine Learning. MIT Press, Cambridge (2009)
2. Axelrod, B.: Genetic programming (2007). Accessed 5 May 2017
3. Gathercole, C., Ross, P.: Tackling the boolean even N parity problem with genetic programming and limited-error fitness. *Genet. Program.* **97**, 119–127 (1997)
4. 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 (2016)
5. Khmag, A., Ramli, A.R., Al-haddad, S., Yusoff, S., Kamarudin, N.: Denoising of natural images through robust wavelet thresholding and genetic programming. *Vis. Comput.* **33**(9), 1141–1154 (2017)
6. Koza, J.R.: Genetic Programming: On the Programming of Computers by means of Natural Selection, vol. 1. MIT Press, Cambridge (1992)
7. Krizhevsky, A., Sutskever, I., Hinton, G.: ImageNet classification with deep convolutional neural networks. In: *NIPS* (2012)
8. LeCun, Y., Bengio, Y., Hinton, G.: Deep learning. *Nature* **521**(7553), 436 (2015)
9. LeCun, Y., Bengio, Y., et al.: Convolutional networks for images, speech, and time series. In: *The Handbook of Brain Theory and Neural Networks*, vol. 3361, no. 10 (1995)
10. Poli, R., Langdon, W.B., McPhee, N.F., Koza, J.R.: A field guide to genetic programming (2008). <http://www.lulu.com>
11. Rodriguez-Coayahuitl, L., Morales-Reyes, A., Escalante, H.J.: Structurally layered representation learning: towards deep learning through genetic programming. In: Castelli, M., Sekanina, L., Zhang, M., Cagnoni, S., García-Sánchez, P. (eds.) *EuroGP 2018. LNCS*, vol. 10781, pp. 271–288. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-77553-1_17
12. Roth, S., Black, M.J.: Fields of experts: a framework for learning image priors. In: *Null*, pp. 860–867. *IEEE* (2005)
13. Szegedy, C., et al.: Going deeper with convolutions. In: *Proceedings of CVPR*, pp. 1–9 (2015)
14. Tomassini, M.: Spatially Structured Evolutionary Algorithms: Artificial Evolution in Space and Time. Springer, Heidelberg (2006). <https://doi.org/10.1007/3-540-29938-6>

15. Tran, B., Xue, B., Zhang, M.: Using feature clustering for GP-based feature construction on high-dimensional data. In: McDermott, J., Castelli, M., Sekanina, L., Haasdijk, E., García-Sánchez, P. (eds.) EuroGP 2017. LNCS, vol. 10196, pp. 210–226. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-55696-3_14
16. Yan, R., Shao, L., Liu, L., Liu, Y.: Natural image denoising using evolved local adaptive filters. *Sig. Process.* **103**, 36–44 (2014)
17. Zhang, K., Zuo, W., Chen, Y., Meng, D., Zhang, L.: Beyond a gaussian denoiser: residual learning of deep CNN for image denoising. *TIP* **26**(7), 3142–3155 (2017)