An Alternative Proof of the Universality of the CNN-UM and its Practical Applications

Giovanni Egidio Pazienza
and Xavier Vilasís-Cardona
Enginyeria i Arquitectura La Salle
Universitat Ramon Llull
Barcelona, Spain
Email: {gpazienza,xvilasis}@salle.url.edu

Riccardo Poli,
University of Essex
Department of Computing and Electronic Systems
Colchester, United Kingdom
Email: rpoli@essex.ac.uk

***Abstract*— In this paper we give a proof of the universality of the Cellular Neural Network - Universal Machine (CNN-UM) alternative to those presented so far. On the one hand, this allows to find a general structure for CNN-UM programs; on the other hand, it helps to formally demonstrate that machine learning techniques can be used to find CNN-UM programs automatically. Finally, we report on two experiments in which our system is able to propose new efficient solutions.**

I. Introduction

One the most interesting aspects of Cellular Neural Networks is their capability of performing universal computation in a Turing sense. Soon after the introduction of CNNs, several authors started adding some features to the Chua-Yang model with single-layer and with space-invariant weights in order to make it universal. As a consequence, new kinds of CNNs were proposed, characterized by exotic nonlinearities [1] [2], multiple layers [3], or time-varying templates [4]. The best-known result in this field is the Cellular Neural Network - Universal Machine (CNN-UM) [5], which can run the *Game of Life*, a cellular automaton equivalent to a universal computer [6].

The proof of the universality of the CNN-UM through the *Game of Life* is powerful from a theoretical point of view, but it provides no practical information about the structure of CNN-UM programs. In other words, it states that given a certain input, it is possible to obtain the desired output by using a *certain* combination of CNN templates in a *proper* configuration. However, except for specific cases (e.g., the Boolean functions [7]), the definition of the algorithm is left to the expertise of the designer.

In this paper, we present an alternative proof of the universality of the CNN-UM which does not make use of the *Game of Life*. This result, found thanks to an analogy between the CNN-UM and a paradigm equivalent to an arbitrary Turing machine, has two remarkable consequences. Firstly, it defines a general form for all CNN-UM programs; secondly, it allows the exploration of the search space through a technique called Genetic Programming [8], [9]. The latter idea is not totally new, but here, for the first time, we formally demonstrated the properties of closure and sufficiency for Genetic Programming when applied to CNNs.

Furthermore, we present two examples in which our system gives better results, in terms of complexity and robustness of the solution, than those presented in the original papers.

The paper is structured as follows: firstly, we discuss on the universality of the CNN-UM and we give an alternative proof for it; thanks to this result, we then show how CNN-UM programs can be automatically found; thirdly, we study in details two new cases; and finally we discuss the results.

II. On the Turing completeness of the system

A. Equivalence to an arbitrary Turing Machine

In [10] the universality of a paradigm called the GP+IM machine was addressed without making use of the *Game of Life*, but instead proving that it can duplicate the functionality of an arbitrary Turing Machine. This result, apparently irrelevant for our purposes, becomes important when we find that the GP+IM machine is strictly related to the CNN-UM; then, studying the first, we can find unknown properties of the second. To clarify the reason for such a similarity, it is necessary to first provide some details of the GP+IM machine.

The acronym GP+IM stands for Genetic Programming (GP) + Indexed Memory (IM): the first is a machine learning technique that will be described in detail in Sec. III-A, whereas the second is a simple addition to the basic GP paradigm which allows reading from and writing to some memory locations. According to the notation introduced in [10], a GP+IM machine can be written as

Repeat
Evaluate $<$ GP+IM function $>$
Until
$<$ Some specific state of the memory $>$

A GP+IM function is a mapping from inputs to outputs: it cannot have any iterative or recursive process, but it can contain only instructions and IF-THEN-ELSE statements.

The population of computer programs for the GP can be represented using a class of graphs called Directed Acyclic Graphs (DAGs) [11]. In [12] it was proved that also UMF diagrams [13] — which formally describe CNN-UM programs — can be represented using DAGs. Therefore, we can conclude that GP trees and CNN-UM programs have a common

representation. The only difference is that the CNN-UM has a number of local memories which are absent from the GP. However, by adding an extra memory to the simple GP we obtain the GP+IM machine, whose programs are therefore equivalent to those of a CNN-UM.

On the one hand, this results gives an alternative proof of the universality of the CNN-UM; on the other hand, it has important practical implications, as shown in the next section.

B. Practical implications

Thanks to the analogy with the GP+IM machine, we can now state that CNN-UM programs can be represented as follows

Repeat
Evaluate < CNN-UM function >
Until
< Some specific state of the memory >

in which the difference between a CNN-UM program and a CNN-UM function is the same as between GP+IM machine and GP+IM function: the former can contain iterative and recursive processes, the latter cannot. Therefore, any algorithm for the CNN-UM can be represented as a single repeat-until loop inside which there are only a combination of CNN templates and nested IF-THEN-ELSE. This simplifies enormously the nature of the problem.

This gives a general structure which every CNN-UM program has to fit into, but we still do not know how to determine the details of the algorithm. As stated in [10],

> *[...] this notion is similar to saying that Shakespeare is expressible in some language X. That may be good to know, but that does not guaranty that any particular line from language X will be quality literature. Similarly, the language of a typewriter is expressive enough to capture Shakespeare. However, Shakespearean prose will, in practice, never arise from a monkey playing with the keys. [...]*

As we will see in the next section, there are some machine learning techniques that explore the space of CNN-UM programs, and eventually converge to a solution of the problem proposed.

III. Exploring the search space through GP

A. Generalities on Genetic Programming

Although the results of the previous section give an efficient way to represent all the possible CNN-UM programs, we still do not know how to choose properly the CNN templates performing a complex operation and in which order they must be used. For this purpose, we propose to use Genetic Programming (GP) [8], an evolutionary technique already successfully applied to the CNNs (see [12] [14]), and whose superiority to random search is documented in the literature.

In [12] a previous version of the system detailed in the next section was presented, but now we provide a more sophisticate algorithm and a better evaluation of the results.

GP repeatedly selects and evolves instances from a population of computer programs, eventually achieving a solution for a given computational task. The GP working principles can be summarised as follows: firstly, a population of initial programs is created; secondly, a fitness value is assigned to each; thirdly, programs are combined through the so-called GP operators to create a population of offspring; fourthly, the new individuals replace the current population; finally, the whole set of operations — called a generation — is repeated until a stop condition is met, and the result is retrieved from the last generation according to a certain criterion. If the various GP parameters — number of generations, population size, etc. — are set correctly, the fitness of the population improves generation by generation.

B. Fitness function and genetic operators

In a GP system the fitness function indicates how good an individual is. For instance, in a supervised image processing problem the fitness of the actual result may be quantified by measuring its resemblance with the desired output image. Other parameters can also be taken into considerations, like the number of levels and nodes of a tree: the simpler the tree, the better its fitness. These different kinds of fitness can be either combined into a single value through a weighted sum, or used to in some form of Pareto-based selection. Usually, the choice of the most appropriate fitness function strongly depends on the problem of interest, as shown in Sec. V.

As for the operators, in our experiments we employ only the three most important ones: reproduction, crossover, and mutation. Each one is applied with a certain probability that can either be fixed *a priori* or changed during the execution of the GP algorithm. The reproduction is the simplest operator, as it just copies an individual from the current generation to the following one; its main function is assuring a sort of continuity between generations. The crossover operator is applied to pairs of individuals probabilistically extracted from the population according to their fitness: a random node is selected within each tree and then the subtrees rooted at such positions are swapped, generating two new individuals that become part of the next generation. Finally, mutation provides diversity to the population, avoiding local minima. Its mechanism will be explained in more detail in Sec. IV-A.

C. Premature convergence and runs

In GP it is not infrequent for a large percentage of the population to converge to a suboptimal result. The subsequent lack of diversity makes practically impossible the creation of new individuals. This phenomenon has a counterpart in nature called *niche preemption principle* [15], which states that

> *[...] a biological niche in nature tends to become dominated by a single species, which may or may not be globally optimal [...]*

In general, the species that dominates a given niche depends on the initial conditions and the subsequent history of probabilistic events. Therefore, the negative effect of the premature

convergence can be minimized by performing multiple independent runs starting from entirely separate populations; the best-of-run individual is then designated as the result of the group of runs.

Runs can also be used to measure the amount of computational resource required by the GP solver to yield a success with a certain probability, like in Sec. V. Be $P(M,i)$ the cumulative probability of success for all the generations between 0 and i using a population of size M, and $R(z)$ the number of independent runs required to satisfy the success predicate by generation i with a probability of z = 1-ϵ, the following relation holds [8]

$$P(M,i) = 1 - 10^{(\log(\epsilon)/R(z))} \quad (1)$$

where log is the decimal logarithm. The graph of $P(M,i)$ for ϵ=0.01 is depicted in Fig. 1. For example, if for a given

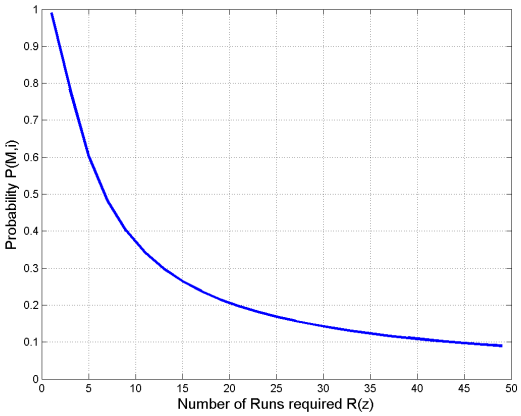


Fig. 1. Relation between the average number of runs $R(z)$, and the probability to obtain a solution with a probability z=99%, given the population size M and the the number of generations i.

problem a solution occurs in every run, it follows from the Eq. 1 that $P(M,i)$ is 99%; if it happens on average every two runs, $P(M,i)$ is 90% and so on.

IV. A GP SYSTEM TO EVOLVE CNN-UM PROGRAMS

A. Main characteristics

The population of our GP system consists of CNN-UM programs, which evolve to find the best solution for the given problem. The templates that can belong to such programs are selected within a set chosen by the designer. Thanks to this feature, any *a priori* knowledge about the problem is used profitably and the search space can be significantly reduced.

The mutation operator is applied to a single randomly selected individual, and it can act according to three different mechanisms: choosing randomly a cloning template of the algorithm and substituting it for another one; changing the value of the variable of a parametric template; or it can selecting randomly a point of the tree representing the CNN-UM program, and then replace the whole branch from this point upwards - that is, to the input level - with a new subtree not related with the previous one.

B. Minor features

During its execution, the GP algorithm may generate subtrees containing redundant information, called introns (see Fig. 2). Whether introns are useful or not is a long-standing

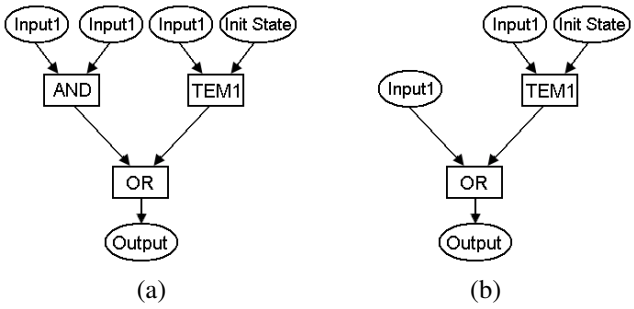


Fig. 2. The tree in (a) contains an intron that can be removed (b).

issue: some authors claim that introns protect useful subtrees by the destructive effects of the operators; others state that introns introduce noise into the system causing a slower convergence. We verified experimentally that in our case introns deteriorate the result, and for this reason we provide a procedure to remove them.

The initial population of CNN-UM programs is set randomly using a Ramped Half-and-Half method [8], which assures a very diverse population composed by trees of several different depths. The designer has the possibility to fix the minimum and maximum size - i.e. maximum and minimum number of levels and nodes - of the individuals, which constraint is enforced also during the evolution. Although other methods can be employed, we found that this one gives the best performances in terms of average number of generations to converge to the solution.

C. Closure and sufficiency of the GP system

In general, in GP the operation set should satisfy the requirements of closure and sufficiency [9]. In this section we show how both of them are met in our system.

Closure means that any operation is well-defined for the arguments it may encounter, and that all the trees created during the evolution process are valid individuals for the GP system. The first point is trivially verified in our system, since all the operations are CNN templates whose input and output are images; the difference between templates operating on binary images and those handling greyscale images can be easily managed successfully. The second point is an indirect consequence of the results about the form of CNN-UM programs presented in Sec. II-B. This property would not be verified if the GP had to deal with trees containing iterations or recursions, as illustrated by the following example.

In Fig. 3 there are two CNN-UM programs: the one in (a) contains only two templates and a backward loop (iteration), whereas that in (b) is a sequence of three templates. When crossed using the suggested selection points, they produce the offspring of Figs. 3(c) and (d). Due to the fact that in (a) the selection point is inside the loop, the structure for the

individual (d) is not univocal. Should its loop go two templates backward, or rather to the first template after the input? Both cases are compatible with what happens in (a), and there is no possibility to solve this ambiguity.

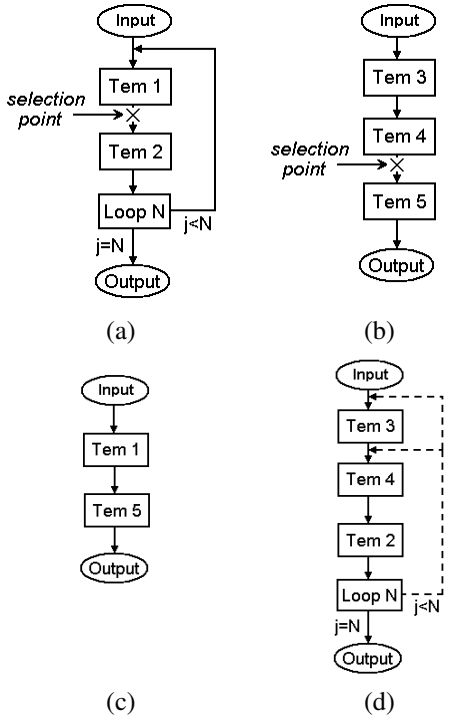


Fig. 3. Two parents (a) and (b) are crossed to obtain an offspring: due to the loop in (a), the tree (d) is not well-defined.

In presence of loops and recursions such situations are common, and they may lead to non-sense trees unless a complex mechanism to detect these anomalies is provided. For these reasons, dealing with functions containing only CNN templates and IF-THEN-ELSE statements assures that each offspring obtained using the GP operators will still be a valid individual, as required by the closure property.

The sufficiency property requires that the operation set be capable of expressing a solution to the problem. In theory we cannot assure that, since new templates can always be included into the operation set, but in practice the designer selects a number of templates that seem to be promising to solve a problem. If all the runs are unsuccessful, the experiments will be repeated adding more CNN templates to the operation set. An application of this approach will be seen in Sec. V-B.

V. Experiments

A. Template decomposition with restricted weights

A field in which our system finds application is the CNN template decomposition with restricted weights [16]. This has both a theoretical and a practical importance, since CNN implementations may accept only a discrete set of possible values. In this paper, we consider the *Horizontal Connected Component Detector (HCCD)* template, whose function is detecting the number of horizontal holes from each row of the image. Due to the nature of the problem, hereafter we reduce the analysis to one dimension, being images and templates horizontal arrays. According to the assumptions made in [17], the elements of the matrices A and B can be $\{0,1\}$, whereas z can assume the values $\{\pm0.5,\pm1.5,\pm2.5,\pm3.5\}$.

Here we consider two different training sets, which in principle have different decompositions. The first is (000,001,010,011,100,101,110,111)→(0,0,1,0,1,0,1,0), it was presented in [17] as well as its decomposition into 14 templates. The second training set is (000,001,010,011,100,101,110,111)→(0,0,1,0,1,0,1,1), proposed in [18] and never decomposed with the previous restrictions. It can be easily proved that both mappings lead to the same final result, but in general the first one has a faster convergence (see the example in Fig. 4).

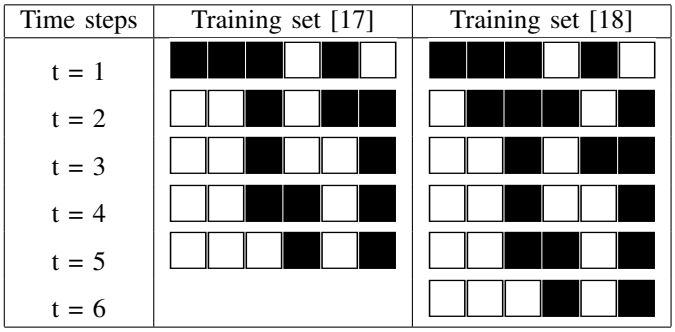


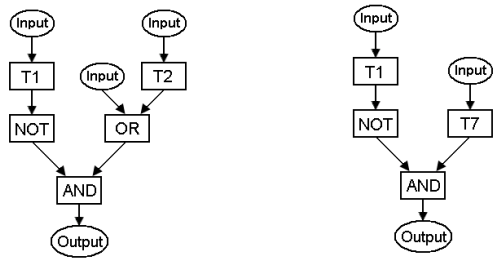
Fig. 4. Example of how the two training set for the HCCD template converge to the same result but with different speed.

It is possible to notice that the operation set for the GP can be limited to 10 templates, including only the B matrix and the z. Namely, they are T1=((0,0,1),-0.5); T2=((1,0,0),-0.5); T3=((0,1,1),-0.5); T4=((0,1,1),-1.5); T5=((1,0,1),-0.5); T6=((1,0,1),-1.5); T7=((1,1,0),-0.5); T8=((1,1,0),-1.5); T9=((1,1,1),-0.5); T10=((1,1,1),-1.5). All the other possible templates are trivial. Actually, the whole set can be reduced to only T1 and T2, since all the others can be obtained as a Boolean combination of these two. The fitness function chosen for this experiment is the well-known Hamming distance.

Evolving 100 individuals during 30 generations and repeating the experiment over 200 runs, we converged to a solution 60 times using the first training set, and 97 times using the second one. According to the Eq. 1, the cumulative probability of finding a solution with M=100 and i=30 is P(100,30)=75% for the first example, and P(100,30)=90% for the second example; therefore, the fitness space of the last case is the easiest to explore. It is possible to have a further confirmation of this fact decreasing the number of individuals and generations, and observing that the solution is still found with an high probability. For example, with 50 individuals and 10 generations, the solution was found 45 times out of 200 runs, resulting in P(50,10)=60%.

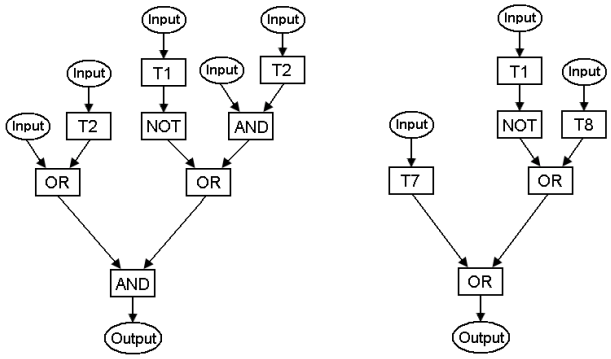
Our system was capable of discovering several different solutions for both cases, and the bests - in terms of

number of templates - are represented in Figs. 5 and 6, in which the tree (a) was found using the operation set {T1,T2,AND,OR,NOT}, and the tree (b) extending the set to {T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,AND,OR,NOT}. Noteworthy, the solution proposed for the training set in [17] is simpler than the one known proposed in the original paper, containing only 3 templates instead of 14.



(a) (b)

Fig. 5. Best solutions - with two different operation sets - for the training set in [17].



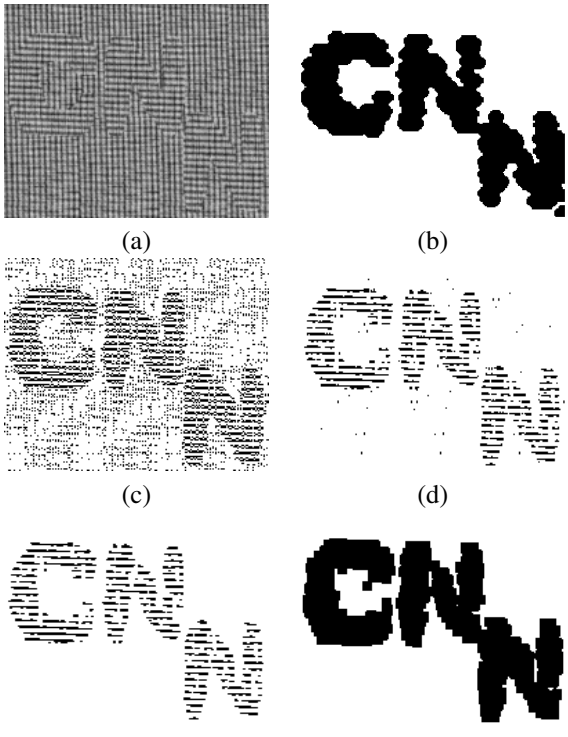
(a) (b)

Fig. 6. Best solutions - with two different operation sets - for the training set in [18].

B. Improving a CNN-UM algorithm for texture segmentation

The second experiment we considered regards the texture segmentation by means of CNNs, solved in [19] through an algorithm including a CNN template expressly designed for the problem. Unfortunately, such template is sensitive to the noise, and strongly connected to the example used to design it. Nevertheless, here we show how the method illustrated in this paper finds automatically a solution composed exclusively by standard robust templates.

The input image and the result proposed in [19] are in Figs. 7(a) and (b), respectively. As for our system, we evolved 150 individual during 30 generations, using the following template set: {*AND, OR, NOT, VerticalLineRemover, HorizontalLineRemover, Erosion, Dilation, PointRemoval, HorizontalErosion, VerticalErosion*}. The reason for such a choice is that in this case we clearly have to discern horizontal and vertical lines, and remove as much noise as possible; however, other choices are also possible. All the templates are included in the standard library [20], except for the *HorizontalErosion* whose parameters are A=(0 0 0; 0 2 0; 0 0 0), B=(0 0 0; 1 1 1, 0 0 0); z=-4. Moreover, the *Dilation* and the *Erosion* templates were embedded in a subroutine, usually known with the name of *Closing operation*. We propose a fitness function based on the Hamming distance, but slightly modified to make a distinction between false negatives and false positives: A greater weight was given to the firsts in order to avoid holes in the image. The intermediate and final results are in Fig. 7, and the algorithm found is depicted in Fig. 8. Comparing Fig. 7(b) to Fig. 7(f), it is evident that the two solutions are very similar, but ours has the advantage of using only standard templates, and then being easily implementable on a real device.



(a) (b) (c) (d) (e) (f)

Fig. 7. Texture segmentation: (a) input image, (b) result found in [19], (c) *VerticalLineRemover*, (d) Erosion, (e) *HorizontalErosion*, (f) output of our algorithm.

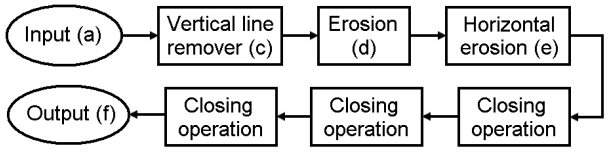


Fig. 8. Algorithm for the texture segmentation. The letters within brackets refer to Fig. 7.

VI. Conclusion

In this paper we drew an analogy between the CNN-UM and a paradigm called GM+IM machine, which is capable of duplicating the functionality of an arbitrary Turing machine. Such analysis yielded three different results.

First, we presented a proof of the universality of the CNN-UM without making use of the *Game of Life*.

Second, we proved that all the CNN-UM programs must fit into a fixed structure, composed by a CNN-UM algorithm with no iterations nor recursions contained within a repeat-until statement. Obviously, the fact that all the CNN-UM programs *can* be written in such a way does not exclude that they *may* be represented using alternative expressions too.

Third, we showed that Genetic Programming is a natural candidate for the automatic evolution of CNN-UM programs, since it handles structures with a one-to-one correspondence with UMF diagrams. Furthermore, the property of closure and sufficiency for GP when applied to analogic algorithms is for the first time proved formally.

In the two examples examined - CNN template decomposition and texture segmentation - the GP system was capable of finding new efficient solutions. Moreover, through the analysis of the runs, it also gave additional information about the difficulty of performing automatically a certain task. Such examples, though simple and not requiring if-then-else structures, help to foresee the potentialities of this approach, which will be applied in the near future to new and more complex cases.

Acknowledgment

G. E. Pazienza thanks DURSI and the 'Generalitat de Catalunya' for the FI grant.

References

[1] P. Venetianer, P. Szolgay, K. Crounse, T. Roska, and L. Chua, "Analog combinatorics and cellular automata - Key algoritms and layout designs," *International Journal of Circuit Theory and Applications*, vol. 24, pp. 145–164, 1996.

[2] R. Dogaru and L. Chua, "Universal CNN cells," *International Journal of Bifurcation and Chaos*, vol. 9, no. 1, pp. 1–48, 1999.

[3] L. O. Chua, T. Roska, and P. Venetianer, "The CNN is universal as the Turing machine," *IEEE Trans. Circuits Syst. I*, vol. 40, no. 4, pp. 289–291, Apr. 1993.

[4] Z. Galias, "Designing cellular neural networks for the evaluation of local boolean functions," *IEEE Trans. Circuits Syst. II*, vol. 40, pp. 219–223, Mar. 1993.

[5] T. Roska and L. O. Chua, "The CNN universal machine: an analogic array computer," *IEEE Trans. Circuits Syst. II*, vol. 40, pp. 163–173, Mar. 1993.

[6] E. Berlekamp, J. H. Conway, and R. K. Guy, *Winning ways for your mathematical plays*. New York: Academic Press, 1982.

[7] K. R. Crounse and L. O. Chua, "The CNN universal machine is as universal as a Turing machine," *IEEE Trans. Circuits Syst. II*, vol. 43, no. 4, pp. 353–355, Apr. 1996.

[8] J. Koza, *Genetic programming - on the programming of computers by means of natural selection*. Cambridge, MA: MIT-Press, 1992.

[9] R.Poli, W. B. Langdon, N. F. McPhee, and J. R. Koza, "Genetic programming: An introductory tutorial and a survey of techniques and applications," University of Essex, UK, Tech. Rep. CES-475, Oct. 2007.

[10] A. Teller, "Turing completeness in the language of genetic programming with indexed memory," in *Proceedings of the 1994 IEEE World Congress on Computational Intelligence*, vol. 1. Orlando, Florida, USA: IEEE Press, 27-29 June 1994, pp. 136–141.

[11] S. Handley, "On the use of a directed acyclic graph to represent a population of computer programs," in *Proceedings of the 1994 IEEE World Congress on Computational Intelligence*, vol. 1, Orlando, Florida, USA, 27-29 June 1994, pp. 154–159.

[12] G. Pazienza, K. Karacs, and X. Vilasis-Cardona, "An automatic tool to design CNN-UM programs," in *Proc. 2007 European Conference on Circuit Theory and Design (ECCTD'07)*, Seville, Spain, 2007.

[13] T. Roska, "Computational and computer complexity of analogic cellular wave computers," *Journal of Circuits, Systems, and Computers*, vol. 12, no. 4, pp. 539–562, 2003.

[14] V. Preciado, M. Preciado, and M. Jaramillo, *Genetic Programming for Automatic Generation of Image Processing Algorithms on the CNN Neuroprocessing Architecture*. Springer, 2004, vol. 3040, pp. 374–383.

[15] A. E. Magurran, *Ecological diversity and its measurement*. Princeton, NJ: Princeton University Press, 1988.

[16] Y. Lin and J. Hsieh, "Robust template decomposition with restricted weigts for cellular neural networks implementing and arbitrary boolean function," *International Journal of Bifurcation and Chaos*, vol. 17, no. 9, pp. 3151–3169, 2007.

[17] M. Laiho, A. Paasio, J. Flak, and K. Halonen, "Template design for cellular nonlinear networks with one-bit weights," *IEEE Trans. Circuits Syst. I*, to be published.

[18] H. Harrer and J. Nossek, "Discrete-time cellular neural networks," *International Journal of Circuit Theory and Applications*, vol. 20, pp. 453–467, 1992.

[19] T. Szirany and M. Csapodi, "Texture classification and segmentation by cellular neural network using genetic learning," *Computer vision and Image Understanding*, vol. 71, no. 3, pp. 255–270, 1998.

[20] L. Kék, K. Karacs, and T. R. (Eds.). (2007) Cellular wave computing library, version 2.1 (templates, algorithms and programs). [Online]. Available: http://cnn-technology.itk.ppke.hu/Library_v2.1b.pdf