CNNOPT: Learning dynamics and CNN chip-specific robustness

Dániel Hillier*,† Samuel Xavier de Souza*, Johan A.K. Suykens*, Joos Vandewalle*
*K.U. Leuven, ESAT-SCD-SISTA, Kasteelpark Arenberg 10, B-3001 Leuven (Heverlee) Belgium
Email: samuel.xavierdesouza, johan.suykens, joos.vandewalle@esat.kuleuven.be
†Jedlik Laboratory, Faculty of Information Technology
Péter Pázmány Catholic University, 1083 Budapest, Práter u. 50/a, Hungary
Email:hillier@digitus.itk.ppke.hu

***Abstract*— A method is presented that unifies previous approaches with the aim of learning new templates with the ability to tune Cellular Nonlinear Network (CNN) templates to individual chip instances in a global optimization framework. The proposed method is built on earlier approaches extending them in three main aspects. First, hardware parameters of the CNN chip are included in the optimization that opens the way to run templates so far believed to be very unstable on chip. Second, a novel global optimization algorithm is used that improves learning speed significantly. Third, the whole method is presented as a new Matlab toolbox so that the only task of the CNN algorithm designer is to formulate the operation to be learned as a training set of the optimization process. Training set design is the most crucial issue of this approach, thus basic rules for the design of training sets are presented. Examples are given in order to illustrate the design issues. We believe that the proposed method can be a valuable tool to find new CNN templates and robustly implement them on chip.**

I. Introduction

Using a Cellular Nonlinear Network (CNN) architecture to process image flows has been proven to outperform conventional digital architectures [1], [2]. However analog Very Large Scale Integration (VLSI) implementations of the CNN architecture still suffer from hardware instabilities – mainly due to manufacturing parameter mismatches. In [3] a recurrent backpropagation based learning method was applied to minimize the effect of parameter deviations across CNN cells.

In [4] a new design method based on a global optimization method was introduced with the aim to design more robust templates targeted to an individual chip instance. In order to find the template that makes the actual chip react as an ideal CNN structure, the output of template executions on the actual CNN chip was used in the cost function. The result of the optimization process is the least sensitive template to the imperfections of the actual chip instance.

This approach can also be considered as a learning of steady-state behavior when only the desired input-output relation is known but not the template values. Moreover, when the trajectory of each cell at specific time intervals is specified in a sequence of images, the same approach is extended to learning of spatiotemporal dynamics (see [5] for details). The additional complexity of this task is that the cost function must be derived so that it assimilates the time instants of the evolution of the output into the set of parameters to be optimized. This allows for learning the desired behaviour with different speeds rather than restricting it to the original speed of the dynamics. This also means that the speed of existing CNN templates might be increased. Furthermore, it also reduces the necessity of generating a perfect training set, which is the most important issue for learning 2-D spatiotemporal behaviour on CNNs.

This paper reports a unified framework addressing the problems mentioned above. An additional novelty of this study compared to [4], [5] is that a more efficient optimization algorithm was applied in order to improve speed of convergence. Moreover, hardware reference values were also considered as optimization parameters. The latter enables chip implementation of templates that were so far known to be quite unstable on the most recent CNN chip implementation (ACE16k version 2 [6]). A new Matlab based toolbox that implements learning and chip specific tuning in a unified global optimization framework giving is also discussed in this paper.

This paper is organized as follows. First the problem of CNN template design and its solution in a global optimization framework is discussed. Second, a brief description of the methods used to solve the template learning/tuning problem follows. Third, specific use-cases are presented with emphasis on design issues of training set design. Finally, a brief overview of the Matlab toolbox solving the template learning/tuning task is presented.

II. Problem statement

We consider a square array of first order CNN cells with space-invariant, nearest neighbor weights defined as:

$$\begin{array}{rcl} \frac{dx_{i,j}}{dt} & = & -x_{i,j}(t) + \sum_{(k,l)\in N_r(i,j)} A_{i+k,j+l} y_{i,j} + \\ & & \sum_{(k,l)\in N_r(i,j)} B_{i+k,j+l} u_{i,j} + z_{i,j} \\ y_{i,j}(t) & = & \frac{1}{2}(|x_{i,j}(t)+1| - |x_{i,j}(t)-1|) \end{array} \quad (1)$$

where i, j denote the position of a cell in the array with M the number of cells in a row/column, N_r defines the neighborhood of a cell, A denotes the feed-back and B the feed-forward cloning templates and z is the current of the cell.

Eq. (1) can be used to solve many problems in image processing by choosing appropriate values for A, B and z (referred to as template). A nice collection of templates can be found in [7]). However, finding template values for a desired

operation can be a very difficult task. Template design methods for binary input - binary output case have been proposed in [8] [9]. In a more generic approach, template design can be also solved via learning in a global optimization process. In such a case, the problem of finding template values for a desired operation is cast into the problem of minimizing the cost function:

$$\min_{\substack{A,B,z,\Theta,\Delta t_k,\\ \Delta t_{k+1},\cdots,\Delta t_T}} E = \frac{1}{N_S N_T N_R} \sum_{s=1}^{N_S}\sum_{k=1}^{N_T}\sum_{n=1}^{N_R}\sum_{i,j}(y^d_{i,j;s,k,n} - y_{i,j}(A,B,z,\Theta,t_k))^2 \quad (2)$$

where $\Delta t_k = t_k - t_{k-1}\ \forall\ k = 1,\cdots,N_T$ represent the time intervals between two output samples with $t_0 = 0$ and N_T being the number of time instances where the desired output $y^d_{i,j;k}$ is specified. $y^d_{i,j;s,k,n}$ denotes the desired output value of a pixel in the kth image of a given sequence of N_T images. The value $y_{i,j}(A,B,z,\Theta,t_k)$ denotes the output value of a pixel as the system has evolved to the time instant t_k with weight matrices A and B, current z and hardware reference parameters Θ. N_S denotes the number of initial state, input, desired state(s) instances in the training set. N_R denotes the number of repetitions a template is executed on chip. Across repetitions, noise can be added to template and/or initial state and input values.

For the optimization process, initial conditions, i.e. initial states $x_{i,j,s,0,n}$ and inputs $u_{i,j}$ as well as desired outputs $y^d_{i,j;s,k,n}$ are assumed to be given. In order to avoid misinterpretations the following terms will be used as defined below.

- **Training set (TS)**: a collection of input $u_{i,j;s} \in U$, initial state $y_{i,j;s,0,n} \in Y^0$, desired output $y_{i,j;s,k,n} \in Y^d_k$, bias map $z_{i,j;s} \in Z$ and fixed state map images.
- **Training set instance**: a set of images needed to train the template in order to perform the desired functionality. A training set instance contains only one input image U, but may contain several desired outputs $Y^d(k)$ depending on the number of time intervals N_T we specify. A template is said to be learned when the optimization process finishes below the user-specified target threshold of the cost function for all instances in a training set. Optimization can also lead to a result when the target threshold is not reached but a given number of cost function evaluations are exceeded.
- **Template**: in the CNN community the term template is used to denote the variable set A, B, z (linear, space-invariant CNN) and the boundary condition Ω. However these variables do not specify uniquely the function the array has to perform. For the ACE16kv2, additional variables like bias map weight β, hardware references for optical input Θ_{opt}, for template values Θ_{tem} and for image values Θ_{sig} are also to be specified. In the case of dynamic tasks, time Δt_k is also a parameter. In summary, in our framework an extended template composed of the following set of variables must be specified in order to uniquely define a CNN operator: $A, B, z, \beta, \Delta t_k, \Omega, \Theta_{opt}, \Theta_{tem}, \Theta_{sig}$.

The optimization problem formulated in (2) is the fusion of the cost function found in [4], [5]. An important novelty here is that the inclusion of hardware parameters Θ in the optimization process enables correct functioning of templates so far believed to be very unstable when running on VLSI CNN implementations. [9] points out the possibility that the same template with the same input image can converge to different output images depending on the actual voltage values of black pixels and white pixels (saturation region). Hardware parameters Θ_{sig} directly affect these voltage values thus their inclusion in the optimization can promote the correct functioning of the template.

III. Methods

Fig. 1 visualizes the main modules of the proposed optimization framework. There exists a rich family of methods for solving optimization problems. Methods relying on gradient information converge quickly but can be stuck in local optima. In our case, due to the piece-wise linear nature of the non-linear output function of the CNN cells, gradient information of the network can only be approximated. In addition, there is no evidence available that would support that the parameter space is convex in order to justify the use of gradient based methods. Due to these issues, gradient based methods are not suitable to solve this problem.

Stochastic methods do not rely on gradient information. Instead, the parameter space is probed according to some strategy and the resulting cost values are used to direct the random search. Fig. 1 depicts the main building blocks of the template learning/tuning problem in a stochastic optimization framework. Eq. (2) is implemented by modules in the shaded area, (1) is implemented in the "Run template on chip" module.

A. Optimization algorithm

In [4] it was suggested that stochastic optimization methods are more suitable for tuning CNN templates than gradient based optimization techniques. In [10] a new global optimization algorithm called Coupled Simulated Annealing (CSA) is introduced. In CSA the annealing temperatures of several Simulated Annealing processes are interconnected in order to improve performance in convergence speed and to increase the probability of exploring all basins of attraction in a given number of cost function evaluations. In addition, the coupling also allows the control of the variance of the acceptance probabilities by the acceptance temperature. The sensitivity of the acceptance temperature on its initial value is therefore reduced. In addition, variance control guides optimization to quasi-optimal runs, i.e. it balances global search done at high temperatures with gradually switching to local search with the temperature cooling. The number of cost function evaluations per individual processes decreases exponentially when the number of optimizers is increased linearly. This is another advantage of CSA since interactions between solutions

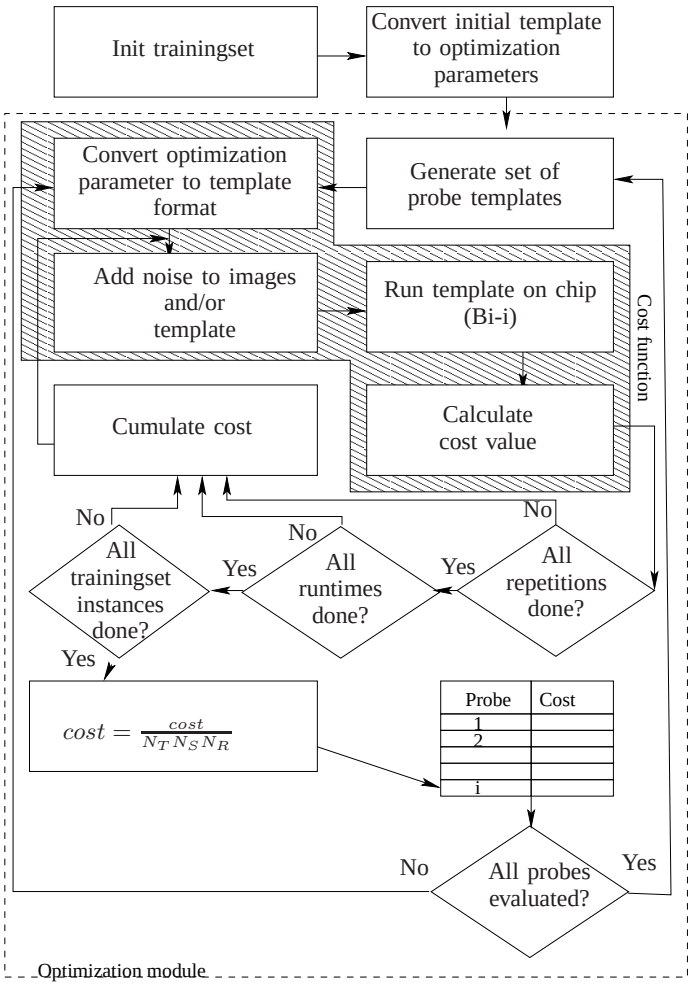


Fig. 1. Block diagram of the optimization framework. Eq. (2) is implemented by modules in the shaded area, (1) is implemented in the "Run template on chip" module.

decrease the number of cost function evaluations to reach a given energy threshold.

The choice of a particular optimization algorithm - i.e. strategy used to generate the probing templates - does influence how fast the global optimum will be reached. Since convergence to the global optimum cannot be ensured in a finite number of iterations, the choice of stochastic optimization method is a very important issue. Optimization methods are more and more relying on some kind of parallelism in order to perform more efficiently. The Probe–Cost table in Fig. 1 refers to the use of multi-probe based algorithms (CSA or various kinds of evolutionary methods), although any kind of optimization algorithm can be embedded into the proposed framework.

B. Reducing the search space

The main disadvantage of global optimization is the need for a high number of function evaluations. When the number of allowed cost function evaluations is limited, the size of the search space influences drastically the chance of finding the global optimum. Modules in Fig. 1 doing conversion between the actual extended CNN template and the vector of optimization parameters must therefore ensure that the number and range of parameters needed to learn/tune the template is not larger than the essential.

The basis to reduce the number of parameters to be optimized is the user's initial knowledge about the spatial structure of the template/function to be tuned/learned. A priori knowledge such as template stability constraints [11], fixed template values, or dependence between template positions have to be taken into account. For example the simulator version of the trigger-wave template [12] consists of an A template where the surrounding elements are the same. In the B template only the central element is different from zero. In this example, exploiting these two a priori known constraints eliminates 15 unnecessary parameters from the optimization.

Chip implementations may involve hard-wired limitations of the CNN model. For instance, on the ACE16kv2 either the surrounding template elements of the A template or those of the B template cannot be used. This is a hardware related constraint that limits the range of template types and must be automatically imposed on template learning/tuning processes.

The search space can be cut down further by setting bounds on the range of each optimized variable. Firstly, in accordance with the actual VLSI implementation, hardware bounds apply to all template values. Further bounds can be derived mainly in the case of chip-specific tuning, i.e. either when the simulator version of a template is known or when a first optimization run had already found a working template and another optimization is to be launched in order to improve the robustness of the solution - preferably with additional noise added. In order to avoid unnecessary manual work, limiting the search range of each individual parameter is not needed. Instead, parameter bounds can be applied to specific subsets of template and hardware parameters denoted as $\nu = [a_n, a_5, b_n, b_5, z, b, \Theta_{tem}, \Theta_{sig}]^T$, where a_n and b_n represent the off-center elements of the A and B template respectively, and a_5 and b_5 represent the central elements.

C. Defining a Training Set

Either for learning, or chip-specific tuning, constructing a good training set (TS) is a key issue in order to obtain a well performing template. The topology of the cost surface is affected by the actual TS used in the optimization. In addition, the metric used in (2) to calculate the error between the desired output and the output of the probed template also alters the topology of the cost surface. A well defined TS must ensure that only the desired operation can correspond to the global optimum. In other words, a good understanding of the operation to be solved by the template is needed to design the TS properly.

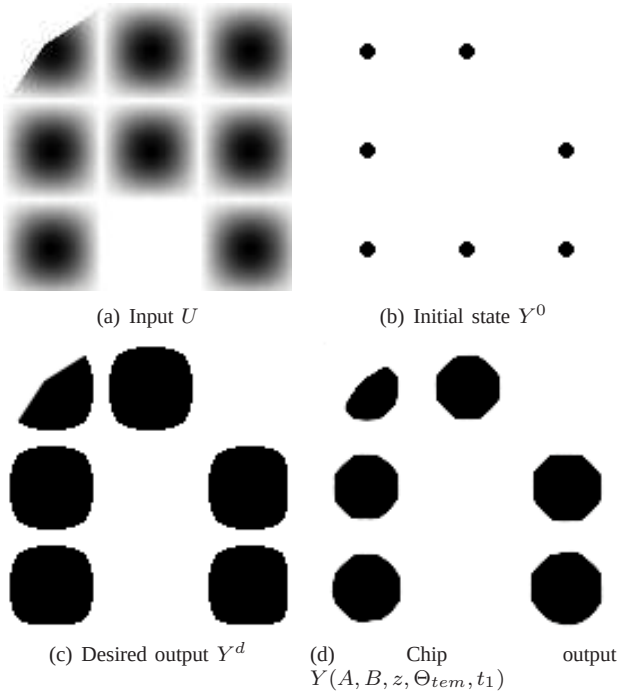
A wrongly designed TS is reflected in a cost surface where the desired operation is assigned to a local optimum but the global optimum corresponds to an operation that is specific to the actual TS. In such a case, the learned template will not work with other TS instances, i.e. the result template does not

generalize to the whole set of possible U, Y^0 images. A good illustration is the grayscale-constrained isotropic trigger wave template. Consider just a single circular grayscale constraint element on Fig. 2(a). If the whole input image would be composed of such elements and the desired output would have a black circle in every corresponding position, a simple thresholding operation would easily match the desired output. In addition, thresholding would create more circular object than the trigger wave operator on the chip output. So in this case, a wrong template design would assign an unwanted operation to the global optimum although the desired operation would still be present on the cost surface as a local optimum.

Another case of bad TS design is when the desired operation corresponds to the global optimum, but the difference between the cost of the global optimum and a number of local optima is not high or the basin of attraction of the global optimum is too narrow. A more careful TS design can alleviate this problem, however a more discriminative metric provides a better solution.

In order to minimize the effects of badly designed training sets and maximize convergence rate and performance of the template optimization, we define here a few rules of thumb:

- Analyze the functionality that the template operation has to learn and ensure all pivotal input-output mappings are encompassed by your training set.
- For many functionalities, besides the input image, other images are also important to achieve what is desired. Analyze how the input, initial state, bias map, fixed state map alter the desired output.
- Make sure your desired output is well balanced. In the case of metric used in (2), the average of pixel values should be about zero in order to avoid that the optimization gets trapped into fully black (all pixels $\rightarrow$ +1) or fully white (all pixels $\rightarrow$ -1) images.
- It is not necessary to create one TS instance for each input-output mapping of the operator to be learned. A single TS instance can carry most of the relevant information. Embed as many input-output mappings as possible to your TS instance. This can be done in a spatially distributed way (see Fig. 2. In addition, encoding various pivotal mappings can improve the balance of black/white pixels of your desired output.
- For learning, when the functionality requires both input and initial state, it is difficult to be sure about the contents of the input and initial state. Equivalent functionalities can be achieved with different templates by exchanging the input and initial state images. E.g. for the shadow template, the object to be shadowed can be put either in the input or in the initial state or in both. In the simulator, it is enough to load the object as initial state and define B with all 0. When implementing templates on chip often the robustness, speed, and/or generalization are stronger in one of the cases. Also, for single input functionalities, input or initial state images can assume -1 or $+1$ globally, or be purely arbitrary. Try different configuration since many times it is not clear at first which configuration is best.
- Finally, for the learning of dynamics or spatiotemporal behavior, apply the same rules above to each time step and make sure that desired dynamics is clearly visible between consecutive images.



(a) Input U (b) Initial state Y^0

(c) Desired output Y^d (d) Chip output $Y(A, B, z, \Theta_{tem}, t_1)$

Fig. 2. A sample training set for learning steady-state behavior. The desired output was obtained using the simulator version of the template in the matCNN toolbox. Initial temperature $= 1$, parameters included in optimization: $a_n, a_5, b_5, z, t_1, \Theta_{tem}$. Initial values were: $a_n = a_5 = 0, b_5 = 3, z = 3, t_1 = 100, \Omega = -1$. Initial values of Θ were those found in the hardware manual of the ACE16k v2.

D. Choice of the metric

Consider the metric proposed in (2). Desired output images where the proportion between the number of black and white pixels is strongly unbalanced - i.e. fully black or fully white output images - represent an inherently false optimum on the cost surface. A viable approach to make the optimization less dependent on TS design - i.e. diminish the number of strong local minima - would be to improve the metric used in (2).

Currently the most obvious metric is used in the cost function, that is a measure of symmetrical difference between two images and can be considered as the degree of coincidence of two point sets $\mathcal{P}$ and $\mathcal{Q}$. In the binary case this is the same as counting the number of different points between images. This metric between binary images is often referred to as Hamming distance. Another often-used distance is the Hausdorff distance that measures the distance of a point on $\mathcal{P}$ that is farthest from any point on $\mathcal{Q}$.

Although the Hamming and Hausdorff distances are commonly used in image processing applications for object comparison and classification, they have several disadvantages. Hamming distance measures only the area difference, but does

not reveal anything about shape difference. In addition, it is sensitive to object shift and noise. Hausdorff metric measures the mismatch between two shapes but also cannot tell anything about shape properties. A one pixel sized noisy spot can drastically modify the Hausdorff distance.

In [1] the so-called non-linear wave metric was introduced that inherently measures both area and shape differences between two binary objects. Let a binary wave be started from $\mathcal{P} \bigcap \mathcal{Q}$ and spreading only to the points of $\mathcal{P} \bigcup \mathcal{Q}$. The time required for the wave to occupy $\mathcal{P} \bigcup \mathcal{Q}$ measures the difference between the shapes $\mathcal{P}$ and $\mathcal{Q}$. The result is a grayscale map where values are related to the time required for the wave to reach a given position. If these so-called local Hausdorff distances are summarized then the wave-type metric takes both Hamming and Hausdorff distances into account. In addition to capture both area and shape differences, this technique has parallel implementation with about $10\mu s$ running time on [6].

IV. Examples

Depending on the particular choice of N_T and N_R in (2), learning of dynamical operations or learning of steady-state behavior of an operation or tuning for chip-specific robustness of an existing template is performed. The choice of N_S is important in a different aspect. Setting it to a number greater than 1 - i.e. using more than one training set instances - is beneficial to ensure the final template is not depending on our choice of particular images in the training set.

A. Tuning for chip-specific robustness

Tuning or learning of a steady-state behavior operation is performed when N_T is set to 1. Robustness of a template is ensured if N_R is set higher than 1, and a proper amount of noise is added to images in the TS and to each probed template.

For all use-cases error tolerance tests can be introduced that influence the robustness of the final template. In order to minimize the time needed to find the optimal template, it is better to split optimizations into two epochs. First, a rough optimum is searched ensuring the template performs the desired functionality. Then, a second optimization starting from the result of the first will introduce noise into template and/or image values in order to increase robustness. The definition of the variance of the noise involves some experimentation. Variance set too high can corrupt the correct functioning of the template whereas too low variance does not improve robustness enough. In theory, the higher the number of repetitions the more robust the final template. Note however that optimization time increases linearly with the number of repetitions.

B. Learning of steady-state behavior

In our context, learning means the identification of specific values for all variables in the extended template so that the desired functionality (image processing operation) is performed on the CNN array. This task differs from tuning in that we do not have an initial guess of the values of the extended template. This implies that the whole parameter space must be searched.

C. Spatio-temporal learning

Learning of spatio-temporal dynamics means that there exist more than one trajectories the array can follow in order to transform the initial state to the final state. Specifying intermediate points $Y_k^d, k \leq N_T, N_T > 1$ on this trajectory assigns the lowest cost of the optimization process to one single trajectory.

A related case is when the operation is difficult to be learned, e.g. operations that need an asymmetric A template. An example is the grayscale constrained anisotropic trigger wave operator [2]. This operator is simple to be implemented in simulator but it is unknown whether hardware realization can do the job. So with complex operations, two problems make the learning difficult: first, we are not sure the hardware can perform the job or whether the specific trajectory can be followed. Second, since the A template is not symmetric, the parameter space to be searched is much higher than for the isotropic case.

Learning the trajectory of the operation incrementally, both issues may be addressed. It is important to emphasize that both in the case of tuning, template learning and trajectory learning, the objective is to get one single template.

For incremental learning, the desired outputs Y_k^d are not specified for a particular time instant during the evolution of the CNN array. Only the order of which desired images are learned matters, i.e. the succession of N_T images (Y_k^d).

In contrast, the actual chip outputs $y_{i,j}(A, B, z, t_k)$ do depend on the particular choice of time instant t_k. However Δt_k are also parameters to be optimized and can therefore allow some flexibility on the dynamics of the CNN array. For example when a trajectory like $\sin(t)$ has to be learned, any approximation of $\sin(\omega t)$ is a solution, i.e. the desired trajectory is not only $\sin(t)$ but all time scaled versions of it.

The right choice for the value of T will depend on how difficult the learning problem is and on how much time and processing resources are available.

Fig. 2 presents a TS instance for tuning/learning the grayscale constrained isotropic trigger wave operator.

Table I summarizes the hardware resources needed for each use-case.

V. Implementation - Matlab Toolbox

The unified framework for learning/tuning CNN templates has been implemented as a Matlab Toolbox. Simple analogic routines can be designed using the Bi-i Vision System [13] via its own low-level programming language called analogic macro code (AMC) [14]. More complex algorithms that also need digital routines can be designed in a relatively easy way via the software development kit (SDK). However, additional software and knowledge is required involving the purchase of Texas Code Composer Studio and DSP programming experience. Matlab is considered as a good trade-off between

TABLE I

ILLUSTRATIVE FIGURES FOR TUNING, LEARNING OF STEADY STATE OPERATION WITH AND WITHOUT A PRIORI KNOWLEDGE OF TEMPLATE STRUCTURE OF THE GRAYSCALE CONSTRAINED ISOTROPIC TRIGGER-WAVE TEMPLATE.

	Tuning	Learning with a priori	"Hard" learning
U	1	1	N_S
Y^d	1	1	1 * N_T
template executions	1*N_R	1*N_R	N_S*N_R*N_T
parameters optimized	b_5, z, Θ_{tem}	a_n, a_5, b_5, z, Θ_{tem}, t_1	A, b_5, z, Θ_{tem}, t_1
parameters	6	9	16
indiv. optimizers in CSA	12	18	48
probe-set evaluations	100	100	1000
approx. optimization time	9.6 min	13 min	390 min

required programming knowledge and flexibility. The benefit of the proposed Matlab Toolbox is that it relieves the user from all the intricate programming issues needed to design algorithms via the Bi-i SDK.

A. Performance considerations

Although chip robustness will most probably improve in future VLSI implementations, this issue will have crucial importance when industrial applications are targeted. For application fields where environmental conditions are stable (like in many surveillance tasks), a robust CNN based algorithm can be used with one-time chip-specific tuning of the templates used. Where conditions are varying in an a priori known range and one single template tuning cannot ensure correct operation in the whole range, a viable solution is re-calibration of the system in constant intervals or when change in environmental conditions is detected. Clearly, during re-calibration, running of the original algorithm is suspended that cannot be tolerated in most cases. However two Bi-i systems performing the same task but get re-calibrated in an alternated way, resolves this issue with introducing an important decrease of possible hazards due to hardware failure.

VI. DISCUSSION AND CONCLUSION

We presented a unified framework for template learning/chip specific tuning based on the coupled simulated annealing global optimization algorithm. The current method shows much better convergence rates than previous work. Using CSA we experienced highly acceptable convergence times during tuning/training. When the process failed to converge, the underlying cause was a not properly designed training set.

We demonstrated that templates believed to be unstable on the ACE16k v2 can be made robust when hardware reference values are included in the optimization process.

Definition of a training set remains the key issue for successful template learning. Designing a good TS is an iterative process during which the user gets more and more insight into how the set of all possible input-output mappings of an operation can be condensed into the TS. Guidelines on training set design were given and dependence of convergence rate on the training set and metric used was discussed. Although good solution is not guaranteed to be found by global optimization in a limited number of cost function evaluations, the proposed method can be a valuable tool to implement new operators on CNN.

A Matlab toolbox is proposed to the community that relieves the user from as much hassle as possible related to code interfacing and hardware hazards so that CNN algorithm design can be put in focus (**http://www.esat.kuleuven.be/sista/chaoslab/cnnopt**).

ACKNOWLEDGMENT

The authors would like to thank Michiel Van Dyck and the entire team of Analogic Computers Ltd. Research supported by K.U.Leuven: GOA-AMBioRICS, CoE EF/05/006 Optimization in Engineering, Flemish Government: FWO: G.0211.05 (Nonlinear Systems), G.0226.06 (Cooperative Systems) Belgian Federal Science Policy Office IUAP P5/22.

REFERENCES

[1] I. Szatmari, A. Schultz, C. Rekeczky, T. Kozek, T. Roska, and L. O. Chua, "Morphology and autowave metric on CNN applied to bubble-debris classification," *IEEE Trans. Neural. Networks*, vol. 11, no. 6, pp. 1385–1393, 2000.

[2] D. Hillier, V. Binzberger, D. L. Vilarino, and C. Rekeczky, "Topographic cellular active contour techniques: theory, implementations and comparisons," *Int. Journal. of Circ. Theory and Appl.*, vol. 34, no. 2, pp. 183 – 216, 2006.

[3] R. Tetzlaff, R. Kunz, and D. Wolf, "Minimzing the effects of parameter deviations on cellular neural networks," *Int. Journal. of Circ. Theory and Appl.*, vol. 7, no. 1, pp. 77–86, 1999.

[4] S. Xavier-de-Souza, M. E. Yalcin, J. A. K. Suykens, and J. Vandewalle, "Toward CNN Chip-specific Robustness," *IEEE Trans. Circ. and Systems-I:Fundamental Theory and Appl.*, vol. 51, no. 5, pp. 892–902, May 2004.

[5] S. Xavier-de-Souza, J. A. K. Suykens, and J. Vandewalle, "Learning of spatiotemporal behaviour in cellular neural networks," *Int. Journal. of Circ. Theory and Appl.*, vol. 34, no. 1, pp. 127–140, Jan 2006.

[6] A. Rodriguez-Vazquez, G. Linan-Cembrano, L. Carranza, E. Roca-Moreno, R. Carmona-Galan, F. Jimenez-Garrido, R. Dominguez-Castro, and S. Meana, "ACE16k: the third generation of mixed-signal SIMD-CNN ACE chips toward VSoCs," *Trans. Circ. and Systems-I:Fundamental Theory and Appl.*, vol. 51, no. 5, pp. 851– 863, 2004.

[7] T. Roska, L. Kék, L. Nemes, and Á. Zarándy, "CNN software library: templates and algorithms," Comp. and Auto. Ins. of the Hung. Acad. of Sci, Tech. Rep., 2005. [Online]. Available: lab.analogic.sztaki.hu/Candy/csl

[8] A. Zarándy, "The art of CNN template design," *International Journal of Circuit Theory and Applications*, vol. 27, no. 1, pp. 5–23, 1999.

[9] M. Gilli and P. Paolo-Civalleri, "Template design methods for binary stable cellular neural networks," *International Journal of Circuit Theory and Applications*, vol. 30, no. 2-3, pp. 211–230, 2002.

[10] S. Xavier-de-Souza, J. A. K. Suykens, J. Vandewalle, and D. Bollé, "Cooperative behavior in coupled simulated annealing processes with variance control," accepted for publication, NOLTA, 2006.

[11] P. Paolo-Civalleri and M. Gilli, "On stability of cellular neural networks," *The Journal of VLSI Signal Processing*, vol. 23, no. 2-3, pp. 429 – 435, 1999.

[12] C. Rekeczky and L. O. Chua, "Computing with front propagation: Active contour and skeleton models in continuous-time CNN," *The Journal of VLSI Signal Processing*, vol. 23, no. 2-3, pp. 373 – 402, 1999.

[13] A. Zarándy and C. Rekeczky, "Bi-i: a standalone ultra high speed cellular vision system," *IEEE Circ. and Systems Magazine*, vol. 5, no. 2, pp. 36–45, 2005.

[14] T. Kozek, A. Zarándy, S. Zöld, T. Roska, and P. Szolgay, "Analogic macro code (AMC) extended assembly language for CNN computers," MTA SzTAKI, Tech. Rep. DNS-10, 1998.