Super Neurons

Serkan Kiranyaz [ID], *Senior Member, IEEE*, Junaid Malik [ID], Mehmet Yamac [ID], Mert Duman [ID], Ilke Adalioglu [ID], Esin Guldogan, Turker Ince [ID], and Moncef Gabbouj [ID], *Fellow, IEEE*

Abstract—Self-Organized Operational Neural Networks (Self-ONNs) have recently been proposed as new-generation neural network models with nonlinear learning units, i.e., the generative neurons that yield an elegant level of diversity; however, like its predecessor, conventional Convolutional Neural Networks (CNNs), they still have a common drawback: *localized* (fixed) kernel operations. This severely limits the receptive field and information flow between layers and thus brings the necessity for deep and complex models. It is highly desired to improve the receptive field size without increasing the kernel dimensions. This requires a significant upgrade over the generative neurons to achieve the "non-localized kernel operations" for each connection between consecutive layers. In this article, we present superior (generative) neuron models (or super neurons in short) that allow random or learnable kernel shifts and thus can increase the receptive field size of each connection. The kernel localization process varies among the two super-neuron models. The first model assumes *randomly localized* kernels within a range and the second one learns (optimizes) the kernel locations during training. An extensive set of comparative evaluations against conventional and *deformable* convolutional, along with the generative neurons demonstrates that super neurons can empower Self-ONNs to achieve a superior learning and generalization capability with a minimal computational complexity burden. PyTorch implementation of Self-ONNs with super-neurons is now publically shared.

Index Terms—Convolutional neural networks, generative neurons, non-localized kernels, operational neural networks, receptive field.

I. INTRODUCTION

GENERALIZED Operational Perceptrons (GOPs) [1], [2], [3], [4], [5] have been proposed as an advanced model of biological neurons with varying nonlinear synaptic connections. Thanks to such a diverse neuron model, GOPs have achieved a superior learning capability on many challenging problems surpassing conventional Multi-Layer Perceptrons (MLPs) and

even Extreme Learning Machines (ELMs) with a significant performance gap [1], [2], [3], [4], [5]. Following the GOPs' main philosophy, Operational Neural Networks (ONNs) [6], [7], [8], [9] have outperformed CNNs significantly, and achieved a notable learning performance, even on those problems where CNNs entirely fail. Yet, ONNs have the following limitations: 1) strict dependability to the operators in the operator set library, 2) requiring a prior search for the best operator set for each layer/neuron which can be highly time-consuming. Self-organized ONNs (Self-ONNs) [10], [11], [12], [13], [14], [15], [16], [17], [18], [19], [20], [21] have recently been proposed to address these drawbacks with the generative neuron model, which can optimize the nodal operators of each kernel element. Such a capability indeed yields an ultimate neuron heterogeneity that is far superior to what conventional ONNs can offer. Generative neurons can, therefore, replace the traditional "weight optimization" of convolutional neurons with the "nodal function optimization" process. However, their kernels are still "localized" or static, and hence each neuron's receptive field size is determined by its kernel size, and this severely limits the amount of information acquired from the previous layer with such limited and localized kernels. Obviously, using a larger kernel size may be a solution for this; however, it will not only create an increasing complexity issue, but it is also not feasible to determine the optimal kernel size for each connection of the neuron. The aim, therefore, should be to improve the receptive field of each kernel connection by allowing each kernel location to vary while keeping the kernel size the same. Moreover, it would be more beneficial "to learn" or to optimize each kernel location for each connection to the feature maps in the previous layer.

The most prominent approach ever proposed to improve the receptive field size was deformable CNNs [22], [23]. However, the improvements over the regular convolutions were limited or simply none because the kernels of each layer have to be deformed in the same way. Therefore, it is rather a "relocation" operation over each kernel element rather than improving the receptive field size. Furthermore, deformable convolutions further increase the network complexity (number of parameters) and especially the memory overhead significantly. This is why deformable neurons are usually used in only one or a few layers of a (deep) CNN.

To address the aforementioned limitations and drawbacks, the novel and significant contributions of this study can be summarized as follows:

- To accomplish the aim of improving the receptive field size with varying kernel locations, and even optimizing

each kernel location, this study proposes a superior generative neuron model (i.e., super neurons in short) with non-localized kernel operations for Self-ONNs.

- In this article, we propose two super neuron models, each of which has a different kernel localization process: i) random localization within a bias range set for each layer, ii) BP-optimized locations of each kernel.
- While the first model randomly distributes the kernels in a certain "receptive field, the second model simultaneously optimizes kernel location and parameters. This study will reveal the pros and cons of both super neuron models when compared against the generative and convolutional neurons over several challenging problems.
- Finally, an extensive set of experiments reveals that Self-ONNs with super neurons can outperform equivalent or significantly deeper CNNs in many challenging problems.

The rest of the article is organized as follows: Section II will briefly present Self-ONNs with generative neurons while the details of the BP training are presented in Appendix A. Section III presents the two super neuron models with non-localized kernel operations in detail and formulates the forward-propagation (FP) and back-propagation (BP). Comparative evaluations among Self-ONNs with generative and super neurons and CNNs over challenging problems are presented in both Section IV and Appendix C. The computational complexity analysis of these networks for both FP and BP is also presented in Section IV. Finally, Section V concludes the article and suggests topics for future research.

II. SELF-ORGANIZED OPERATIONAL NEURAL NETWORKS

In biological neurons, during the learning process, the neuro-chemical characteristics and connection strengths of the synaptic connections are altered, giving rise to new connections and modifying the existing ones. Inspired by this a generative-neuron model for Self-ONNs is formed where each kernel can have a distinct *nonlinear* nodal-operator generated (optimized) during training without any restrictions. As a result, each kernel element of each generative neuron can "customize" its nodal operator to maximize the learning performance. Both convolutional and operational neurons have *static* (fixed) nodal operators (linear and harmonic, respectively) while the generative neuron has *any* arbitrary nodal function, Ψ, (including possibly standard functions such as linear and harmonic functions) for each kernel element of each connection.

The input map of the ith neuron at the layer $l+1$, x_i^{l+1} for conventional ONNs is composed as follows:

$$x_i^{l+1} = b_i^{l+1} + \sum_{k=1}^{N_l} \boldsymbol{oper2D}\left(y_k^l, w_{ik}^{l+1}, {'NoZeroPad'}\right)$$

$$x_i^{l+1}(m,n)\big|_{(0,0)}^{(M-1,N-1)} = b_i^{l+1}$$

$$+ \sum_{i=1}^{N_{l-1}} \left(P_i^{l+1} \begin{bmatrix} \Psi_i^{l+1}\left(y_k^l(m,n), w_{ik}^{l+1}(0,0)\right), \dots, \\ \Psi_i^{l+1}\left(y_k^l(m+r,n+t), w_{ik}^{l+1}(r,t)\right), \dots \end{bmatrix} \right) \tag{1}$$

where y_k^l are the final output maps of the previous layer neurons *operated* with the corresponding kernels, w_{ik}^{l+1}, with a particular nodal function, Ψ_i^{l+1} such as linear (multiplication), sinusoid, exponential, Gaussian, chirp, Hermitian, etc. A close look at (1) reveals the fact that when the pool operator is a summation, i.e., $P_i^{l+1} = \Sigma$, and the nodal operator is a linear function, $\Psi_i^{l+1}(y_k^l(m+r,n+t), w_{ik}^{l+1}(r,t)) = y_k^l(m+r,n+t) \times w_{ik}^{l+1}(r,t)$, for *all* neurons, then the resulting homogenous ONN will be identical to a CNN. Hence, ONNs are a superset of CNNs as the GOPs are a superset of Multi-layer Perceptrons (MLPs). Self-ONNs differ from ONNs by the following two points:

1) each "fixed-in-advance" nodal operator function with a scalar kernel element, $\Psi_i^{l+1}(y_k^l(m+r,n+t), w_{ik}^{l+1}(r,t))$, is *approximated* by the composite nodal operator, $\boldsymbol{\Psi}(y_k^l(m+r,n+t), \boldsymbol{w_{ik}^{l+1}}(r,t))$, as expressed by the Maclaurin series,

2) the *scalar* kernel parameter, $w_{ik}^{l+1}(r,t)$, of the kernel of an ONN neuron, is replaced by a Q-dimensional array, $\boldsymbol{w_{ik}^l}(r,t)$.

In this way, any nodal operator function can be approximated by the Maclaurin series near the origin as follows:

$$\boldsymbol{\Psi}\left(y, \boldsymbol{w_{ik}^{l+1}}(r,t)\right) = w_{ik}^{l+1}(r,t,0)$$
$$+ w_{ik}^{l+1}(r,t,1)\, y + w_{ik}^{l+1}(r,t,2)\, y^2$$
$$+ \dots + w_{ik}^{l+1}(r,t,Q)\, y^Q \tag{2}$$

where $w_{ik}^{l+1}(r,t,q) = \frac{f^{(q)}(0)}{q!}$ is the qth coefficient of the Q-order polynomial. During BP training, each $\boldsymbol{w_{ik}^{l+1}}(r,t,q)$ is optimized for the learning problem at hand. Thanks to this ability, there is no need for any operator search for Self-ONNs and arbitrary nodal operators can be customized by the training process This results in enhanced flexibility and diversity over an ONN neuron where only a standard nodal operator function has to be used for all kernels, each connected to an output map of a neuron in the previous layer.

Table I presents the formula abbreviations and mathematical symbols used in this article. Back-Propagation (BP) training for Self-ONNs[1] is briefly formulated in Appendix A and further details can be obtained from [10].

III. SUPER NEURONS WITH NON-LOCALIZED KERNEL OPERATIONS

The starting point of this study is the generative neuron model of Self-ONNs [10]. Like its predecessors, ONNs, and CNNs, each kernel connection of a generative neuron to the previous layer output maps is localized, i.e., for a pixel located at (m,n) in a neuron at the current layer, all kernels are located (centered) at the same location over the previous layer output maps. Fig. 1 (bottom-left) illustrates this with 3×3 kernels where a pixel of the ith neuron in layer $l+1$, $x_i^{l+1}(m,n)$, is computed using the 9 pixels of the previous layer output maps,

---

[1]The optimized PyTorch implementation of Self-ONNs with generative and super neurons is publicly shared in http://selfonn.net/.
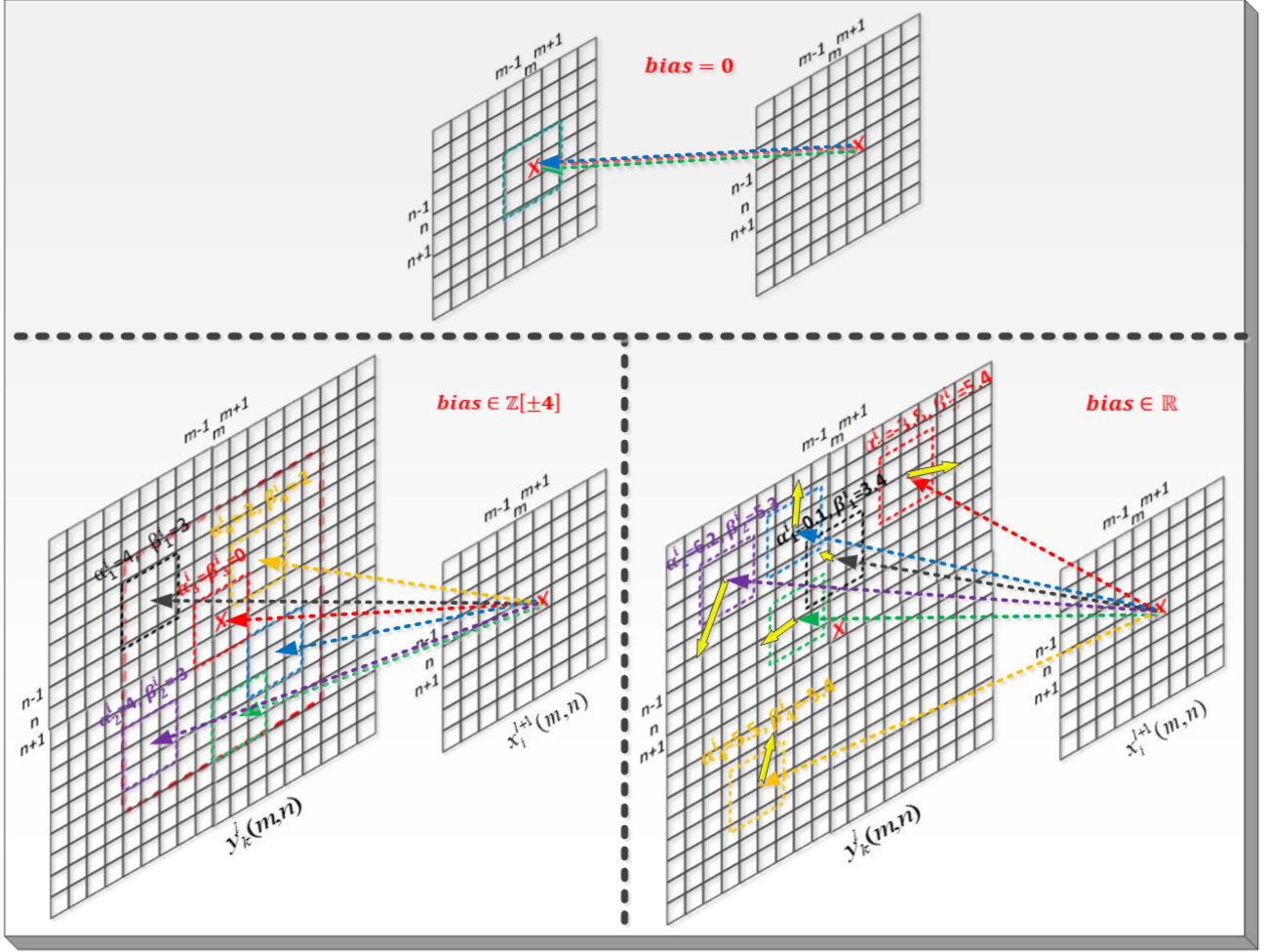
Fig. 1. Localized (top) *vs.* non-localized kernel operations (bottom) to create the pixel, $x_i^{l+1}(m, n)$, from the output maps of the previous layer neurons. At the (bottom) right, *randomly localized* (uniformly distributed) kernels within a spatial *bias* range of $\Gamma = 4$ are shown. At the (bottom) left, the *BP-optimized* locations of each kernel during a BP epoch with bias gradients, $(\Delta\alpha_k^i, \Delta\beta_k^i)$ (yellow vectors) are illustrated.

$y_k^l(m + r, n + t) \; \forall r, t \in [-1, 1]$, for $\forall k \in [1, N_l]$ operated with the kernels centered at the same location, (m, n), given that N_l is the number of neurons in the previous layer, *l*. This gives rise to an obvious limitation since such a static kernel is blinded to the neighboring pixels outside of the kernel boundaries, $\forall r, t \in [-1, 1]$ which may have the potential to contribute to the input pixel, and hence should not be excluded. This study provides two feasible solutions by proposing two super neuron models with *non-localized* kernel operations as illustrated at the bottom of the figure. We define two additional parameters, (α_k^i, β_k^i) as the spatial bias, that is the shift of the kernel from the pixel location, (m, n), towards x- and y-direction. The spatial bias is, therefore, defined for each kernel for both super-neuron models, i.e., in Fig. 1, for the first model, the bottom-left illustration shows the shifted kernel locations for the ith neuron input map at layer $l+1$ connected to the kth output neuron at layer *l*, and the bias values are $[\alpha_k^i, \beta_k^i] \in \mathbb{Z}[\pm\Gamma]$ where the maximum range is determined by the hyperparameter, $\Gamma = 4$ pixels. Therefore, all 3×3 kernels are *randomly* located within a bias range of $[-4, 4]$, and thus, all

pixels within the region of 11×11 pixels can contribute. In the illustration, different colored kernels are for different connections and their corresponding bias values within the 11×11 region (the outer, red-dashed square) are *randomly* set in advance. For instance, the bias for the 1st connection (black) is, $\alpha_1^i = 4, \beta_1^i = 3$ pixels whereas for the 3rd connection (red), it is $\alpha_3^i = 0, \beta_3^i = 0$, respectively.

Finally, for the second model, the illustration at the bottom-right shows the shifted kernel locations by the real-valued bias, $(\alpha_k^i, \beta_k^i) \in \mathbb{R}[\pm\Gamma]$. For this super-neuron model, the bias is iteratively optimized during BP training along with other network parameters. At the end of the training, the bias will converge to a (local) optimum point. So, the bottom-right illustration only shows instantaneous localizations of the kernels at a particular BP iteration.

To formulate a non-localized kernel for the ith neuron in layer $l+1$, connected to the kth neuron in layer *l* with a bias in x- and y-directions, α_k^i and β_k^i, respectively, we will instead shift the output feature map in the previous layer in the opposite direction.

TABLE I
FORMULA ABBREVIATIONS AND DESCRIPTIONS

Symbol	Description
Q	The order of the Maclaurin polynomial
$Kx \times Ky$	The size of a kernel, i.e., $\boldsymbol{w_{ik}^{l+1}}$
$\boldsymbol{w_{ik}^{l+1}}(r,t)$	Q-dimensional array of the kernel element (r,t) from the ith neuron in layer $l+1$ to k^{th} neuron in layer l.
$w_{ik}^{l+1}(r,t,q)$	The qth element of $\boldsymbol{w_{ik}^{l}}(r,t)$.
$[\alpha_k^i, \beta_k^i]$ $\in \mathbb{Z}[\pm\boldsymbol{\Gamma}]$	The integer bias pair in x- and y-directions within the range limit, $\boldsymbol{\Gamma}$ for the ith neuron in the current layer connected to the kth neuron in the previous layer.
(α_k^i, β_k^i) $\in \mathbb{R}[\pm\boldsymbol{\Gamma}]$	The real bias pair in x- and y-directions within the range limit, $\boldsymbol{\Gamma}$ for the ith neuron in
$\vec{y}_k^l$ $= \mathbf{T}^{(\boldsymbol{\alpha_k^i, \beta_k^i})}(y_k^l)$	The shifted version of the output map y_k^l by the bias, $[\alpha_k^i, \beta_k^i]$.
Δ_i^{l+1}	Given the cost function, E, the 2D delta error map of the input feature map, $x_i^{l+1}(m,n)$ at layer $l+1$. Specifically, $$\Delta_i^{l+1}(m,n) = \frac{\partial E}{\partial x_i^{l+1}(m,n)}$$
Δy_k^l	Given the cost function, E, the 2D sensitivity of the output map, $y_k^l(m,n)$. Specifically, $$\Delta y_k^l(m,n) = \frac{\partial E}{\partial y_k^l(m,n)}$$
$\Delta\alpha_k^i, \Delta\beta_k^i$	Given the cost function, E, the individual sensitivities of the bias pair. Specifically, $\Delta\alpha_k^i = \frac{\partial E}{\partial \alpha_k^i}, \Delta\beta_k^i = \frac{\partial E}{\partial \beta_k^i}$
$\nabla_{\boldsymbol{\Psi}}P_i^{l+1}$	The derivative of the pooling function, P_i^{l+1} w.r.t a nodal operator, $\boldsymbol{\Psi}$. Since $P_i^{l+1} = \Sigma$, $\nabla_{\boldsymbol{\Psi}}P_i^{l+1} = 1$
$\nabla_y\boldsymbol{\Psi}$	The derivative of a nodal operator, $\boldsymbol{\Psi}$, w.r.t the output feature map, y.
$\nabla_\alpha\vec{y}$ and $\nabla_\beta\vec{y}$	The derivatives of the shifted output map, $\vec{y}_k^l$, w.r.t the individual bias elements $(\alpha_k^i, \beta_k^i) \in \mathbb{R}[\pm\boldsymbol{\Gamma}]$
$\Delta\alpha_k^i, \Delta\beta_k^i$	Given the cost function, E, the individual sensitivities (derivatives) of the bias elements, $(\alpha_k^i, \beta_k^i) \in \mathbb{R}[\pm\boldsymbol{\Gamma}]$.

For this purpose, let $\mathbf{T}^{(\boldsymbol{\alpha_k^i, \beta_k^i})}$ be the shift operator over y_k^l by the bias, $[\alpha_k^i, \beta_k^i]$. First, we can perform the (opposite direction) shift to obtain, $y_k^l(m + \alpha_k^i, n + \beta_k^i)$ and then operate with the original $Kx \times Ky$ kernel, w_{ik}^{l+1} in a localized manner as expressed in (3). For generative neurons of Self-ONNs recall that $\boldsymbol{\Psi}$ is the composite nodal function which is the Qth order Mac-Laurin series. Over the 1D array of kernel elements, $\boldsymbol{w_{ik}^{l+1}}(r,t)$, $\boldsymbol{\Psi}$ is expressed in (4) where the DC bias term, $w_{ik}^{l+1}(r,t,0)$, is omitted. Therefore, each generative neuron has a 3D kernel matrix where the qth coefficient of the kernel element (r,t) is represented by $w_{ik}^{l+1}(r,t,q)$.

In the next sub-sections, we formulate the forward-propagation (FP) for the two super-neuron models each of which performs non-localized kernel operations, the former with *random* bias and the latter with learnable (real-valued) bias through BP. The details of BP training are covered in Appendices B and C.

$$x_i^{l+1} = b_i^{l+1} + \sum_{k=1}^{N_l} \boldsymbol{oper2D}$$

$$\left(\mathbf{T}^{(\boldsymbol{\alpha_k^i, \beta_k^i})}(y_k^l), \boldsymbol{w_{ik}^{l+1}}, ' NoZeroPad'\right)$$

$$x_i^{l+1}(m,n)\big|_{(0,0)}^{(M-1,N-1)} = b_i^{l+1}$$

$$+ \sum_{k=1}^{N_l} \left(P_i^{l+1}\left[\boldsymbol{\Psi}\left(y_k^l\left(m + \alpha_k^i, n + \beta_k^i\right), \boldsymbol{w_{ik}^{l+1}}(0,0)\right)\right.\right.$$

$$\left.\left. , .., \boldsymbol{\Psi}\left(y_k^l\left(m + \alpha_k^i + r, n + \beta_k^i + t\right), \boldsymbol{w_{ik}^{l+1}}(r,t)\right), \ldots\right]\right)$$

$$\forall r,t \in [0, Kx-1], [0, Ky-1] \qquad (3)$$

$$\boldsymbol{\Psi}\left(y_k^l\left(m + \alpha_k^i + r, n + \beta_k^i + t\right), \boldsymbol{w_{ik}^{l+1}}(r,t)\right)$$

$$= w_{ik}^{l+1}(r,t,1)\, y_k^l\left(m + \alpha_k^i + r, n + \beta_k^i + t\right)$$

$$+ w_{ik}^{l+1}(r,t,2)\, y_k^l\left(m + \alpha_k^i + r, n + \beta_k^i + t\right)^2$$

$$+ \ldots + w_{ik}^{l+1}(r,t,Q)\, y_k^l\left(m + \alpha_k^i + r, n + \beta_k^i + t\right)^Q \quad (4)$$

A. FP for Non-Localized Kernel Operations by the Random Bias

In this super neuron model, the spatial bias consists of an integer pair of shifts in x- and y-directions, $(\alpha_k^i, \beta_k^i) \in \mathbb{Z}[\pm\boldsymbol{\Gamma}]$, each of which is randomly created within the range limit, $\boldsymbol{\Gamma}$, for each kernel connection (to each output map in the previous layer) of each super neuron in the network. Therefore, when a Self-ONN is composed of super neurons that are configured with the non-localized kernel operations by *random* spatial bias, each super neuron will have an array of spatial bias pairs that are randomly assigned in advance and used thereafter as the network parameters. During the FP, the native output map(s) of the input layer are acquired from the training data and the 2D shifted output map(s) by the spatial bias, (α_k^i, β_k^i) are computed as follows:

$$\vec{y}_k^l = \mathbf{T}^{(\boldsymbol{\alpha_k^i, \beta_k^i})}(y_k^l) \qquad (5)$$

where $l = 0$ for the input layer. Then using (3) each input map in the next hidden layer, x_i^{l+1}, $\forall i \in [1, N_{l+1}]$, can be computed. Passing the input map through the activation operator, $f(x)$, first and then the pooling (if up- or down-sampling is performed in that layer), the output map, y_i^{l+1}, is created. Once again, using (5), the shifted output map is created, and the FP proceeds to the next layers. To accommodate such shifts, the boundaries of each output map are zero-padded by $\boldsymbol{\Gamma}$-zeros. In order to speed-up both FP and BP, the qth power of the shifted outputs, $(\vec{y}_k^l)^q$, can also be computed only once (during FP) and stored to be used repeatedly during BP. On the other hand, except for the output maps of the super neurons in the output layer, there is no need to
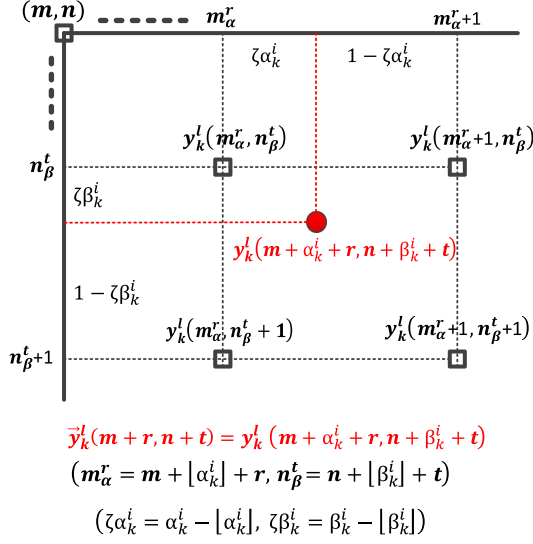
Fig. 2. Composition of the shifted map, $\vec{y}_k^l(m+r, n+t) = y_k^l(m + \alpha_k^i + r, n + \beta_k^i + t)$, by the bias, $(\alpha_k^i, \beta_k^i) \in \mathbb{R}$.

store the original outputs, y_k^l, along with their powers since they are only temporarily needed for non-localized kernel operations.

B. FP for Non-Localized Kernel Operations by the BP-Optimized Bias

The BP optimization of each of the pair of bias shifts in x- and y-directions requires that $(\alpha_k^i, \beta_k^i) \in \mathbb{R}$ and thus, the individual gradients (sensitivities), $\Delta \alpha_k^i = \frac{\partial E}{\partial \alpha_k^i}, \Delta \beta_k^i = \frac{\partial E}{\partial \beta_k^i}$, can be computed. In this case, the bias range, $\mathbf{\Gamma}$, can still be defined for practical reasons (e.g., $(\alpha_k^i, \beta_k^i) \in \mathbb{R}[\pm\mathbf{\Gamma}]$) so that the boundaries of each output map can be zero-padded by Γ-zeros in advance to accommodate the shifts within the allocated memory of the map. As in the random bias approach, during the FP, when an output map, y_k^l, is generated by activating the input map, to create each input map in the next layer, $\boldsymbol{x}_i^{l+1}, \forall i \in [1, N_{l+1}]$, using (3), it will first be shifted by the bias (set earlier by BP) before the 2D (nodal) operation, i.e., $y_k^l \rightarrow \mathbf{T}^{(\alpha_k^i, \beta_k^i)}(y_k^l)$. However, since $(\alpha_k^i, \beta_k^i) \in \mathbb{R}$, the shifted map, $\vec{y}_k^l = \mathbf{T}^{(\alpha_k^i, \beta_k^i)}(y_k^l)$, will exist in the fractional grid as illustrated in Fig. 2. Since the shifted map location, $\vec{y}_k^l(m+r, n+t)$,

is not on the integer grid, but somewhere within the surrounding four grid locations, $(m_\alpha^r, n_\beta^t), \ldots, (m_\alpha^r + 1, n_\beta^t + 1)$ where $(m_\alpha^r = m + \lfloor \alpha_k^i \rfloor + r, n_\beta^t = n + \lfloor \beta_k^i \rfloor + t)$ and $\forall r, t \in [0, Kx - 1], [0, Ky - 1]$ are the kernel indices. One can use bilinear (over 4 pixels) or bicubic (over 16 pixels) interpolation to express the shifted map location, $\vec{y}_k^l(m+r, n+t)$. For simplicity and speed, bilinear interpolation is used as expressed in (6) shown at the bottom of this page. Hence (3) can now be modified for FP with the shifted map location, $\vec{y}_k^l(m+r, n+t)$ by the fractional bias as in (7) shown at the bottom of this page, where $\zeta\alpha_k^i = \alpha_k^i - \lfloor \alpha_k^i \rfloor$ and $\zeta\beta_k^i = \beta_k^i - \lfloor \beta_k^i \rfloor$.

Note that (7) is identical to (1), the one for generative neurons except that the shifted (interpolated) map, $\tilde{y}_k^l$, is now used, which is different from the original output map over which a bilinear interpolation is performed. This is in fact equivalent to a low-pass filtering operation over the actual output maps.

For brevity, BP formulations of the two super neuron models are covered in Appendices B and C, respectively.

IV. RESULTS

In this section, we have tested Self-ONNs with super neurons against both deep and shallow CNN models over three challenging applications. In the next subsection, we performed real-world image denoising experiments over the SIDD Medium benchmark dataset [63] to perform comparative evaluations against the deep (17-layer) Residual CNN (DnCNN [2]) and (DnONN [1]). For the comparisons using the shallow models, the two super neuron models proposed in this study will then be evaluated against the generative neurons, conventional and deformable [22], [23] convolutional neurons over the following challenging problems: 1) Motion and Spatial Deblurring, and 2) Face Segmentation. Finally, to validate the super neurons' ability to learn the true shift, a "Proof-of-Concept" experimentation using a Self-ONN with only one hidden super-neuron will be presented in Appendix D.

A. Real-World Denoising

In real-world denoising experiments, we utilize the SIDD Medium training dataset [63] which consists of 320 high-resolution images. We use the same cropping strategy as adopted in [9] to extract 160 k training patches. For testing, the SIDD validation dataset is used which consists of 1280 noisy clean

$$\vec{y}_k^l(m+r, n+t) = y_k^l\left(m_\alpha^r,\ n_\beta^t\right)\left(1 - \zeta\alpha_k^i\right)\left(1 - \zeta\beta_k^i\right) + y_k^l\left(m_\alpha^r + 1,\ n_\beta^t + 1\right)\zeta\alpha_k^i\zeta\beta_k^i$$
$$+ y_k^l\left(m_\alpha^r + 1,\ n_\beta^t\right)\zeta\alpha_k^i\left(1 - \zeta\beta_k^i\right) + y_k^l\left(m_\alpha^r,\ n_\beta^t + 1\right)\left(1 - \zeta\alpha_k^i\right)\zeta\beta_k^i \tag{6}$$

$$\boldsymbol{x}_i^{l+1} = b_k^l + \sum_{k=1}^{N_l} \boldsymbol{oper2D}(w_{ki}^l, \vec{y}_k^l, 'NoZeroPad')$$

$$\boldsymbol{x}_i^{l+1}(m,n)\Big|_{(0,0)}^{(M-1,N-1)} = b_k^l + \sum_{k=1}^{N_l}\left(P_i^{l+1}\left[\boldsymbol{\Psi}\left(\vec{y}_k^l(m,n), \boldsymbol{w}_{ik}^{l+1}(\boldsymbol{0},\boldsymbol{0})\right), .., \boldsymbol{\Psi}\left(\vec{y}_k^l(m+r, n+t), \boldsymbol{w}_{ik}^{l+1}(\mathbf{r},\mathbf{t})\right), \ldots\right]\right) \tag{7}$$

Model	Param. No.	PSNR (dB)
Super-ONN (Q=3 with tanh)	231K	37.28
Super-ONN (Q=2 with ReLU)	**154K**	37.32
Super-ONN (Q=3 LR-IN with tanh)	231K	37.16
Super-ONN (Q=2 Residual LR-IN)	**154K**	37.58
Super-ONN (Q=2 Reflection ReLU)	177K	**37.81**
DnCNN (ReLU)	556K	36.05
DnONN (Q=3 with tanh)	784K	36.95

TABLE III
COMPARISON OF THE TOTAL NUMBER OF MULTIPLY-ACCUMULATE
OPERATIONS FOR THE NETWORKS USED IN THIS STUDY

Network	Layer 1 Size	Layer 2 Size	*No. of PARs* (K)	Memory Overhead (Kb)
SelfONN (no *bias*)	12	12	7.585	0
SelfONN (*rand. bias*)	12	12	9.265	0
SelfONN (*BP-opt. bias*)	12	12	9.265	0
CNN×4	48	48	21.697	0
DefCNN×4 v1	48	48	37.368	583.2
Def CNN×4 v2	48	48	45.252	874.9

image pairs. In all the experiments, the training-to-validation ratio is set to 9:1. For BP training, we use the ADAM optimizer with the maximum learning rate set to 10^{-3}. All the networks were trained for 100 epochs and the model state which maximized the validation set performance was chosen for evaluation. Model architectures were defined using FastONN [8] library and Pytorch library [64]. All experiments were performed either on an NVIDIA Tesla V100 or an NVIDIA TITAN RTX GPU.

The quantitative results for the real-world denoising problem in terms of average PSNR levels are presented in Table II and the visual results on the SIDD Validation dataset are shown in Fig. 3. In order to test the hyper-parameter variations in Super-ONN models over the performance, we have trained 5 Super-ONN models:

1) *Super-ONN (Q = 3)*: 3-layer Self-ONN with super neurons and tanh activation functions.
2) *Super-ONN (Q = 2)* with ReLU: 3-layer Self-ONN with super neurons and ReLU activation functions.

3) *Super-ONN (Q = 3)* LR-IN: 3-layer Self-ONN with super neurons and ReLU activation functions followed by instance normalization.
4) *Super-ONN (Q = 2)* Residual LR-IN: 3-layer Self-ONN with super neurons, ReLU activation functions followed by instance normalization and a residual input-output connection.
5) *Super-ONN (Q = 2)* Reflection ReLU: 3-layer Self-ONN with super neurons, ReLU activation functions, and reflection padding in the borders of the images.

The results in Table II clearly show that all Super-ONN models significantly outperform both DnCNN and Dn-SelfONN models regardless of their model variations. Especially the performance gap over DnCNN exceeds 1.7 dB in PSNR despite the fact that it has more than 5 times more layers and neurons. This demonstrates the superior learning capability of the super neurons over both generative and convolutional neurons.

Qualitatively speaking, the superior denoising performance of Self-ONNs with super-neurons (Super-ONNs) is once again visible in most output images shown in Figs. 3 and 4. Super-ONNs not only achieve a sharper edge and texture restoration but also recover the smooth regions better than the other networks. In some results the results are indeed similar; however, this overall shows that Super-ONNs can achieve a similar or better restoration performance with less 3 to 4 times less parameters and depth.

B. Results With Shallow Models

To test the true learning capability of the super neurons, especially against generative neurons, we further apply the following severe restrictions and harsh conditions:

i) Very Low Resolution: 60×60 pixels,
ii) Compact/Shallow Models with only 2 hidden layers and less than 25 neurons: *Inx12x12xOut*,
iii) Scarce Train Data: only 10% of the dataset is for training and the rest is for testing (10xfold cross-validation)
iv) Limited kernel size (3×3 kernels except for the 2nd problem)

For all problems, restriction (ii) is relaxed for the conventional and deformable CNNs that have a configuration $In \times 48 \times 48xOut$, hence labeled as "CNN $\times$ 4" and "DefCNN $\times$ 4". With 4 times more neurons than Self-ONNs (*Inx12x12xOut*), such an unfair comparative evaluation is intended to show the true learning capability of the super neurons. For Self-ONNs, $Q = 3, 5, \text{and } 7$ at the 1st, 2nd hidden, and output layers, respectively. Moreover, the first hidden layer applies sub-sampling by $ssx = ssy = 2$, and the second one applies up-sampling by $usx = usy = 2$. For each regression problem, we used the Signal-to-Noise Ratio (SNR) evaluation metric, which is defined as the ratio of the signal power to the noise power, i.e., $SNR = 10\log(\sigma^2_{signal}/\sigma^2_{noise})$. For the *Image Transformation* problem, we performed 10 experiments each with 4 images transformed to another 4. For *Deblurring* and *Denoising*, the benchmark datasets are partitioned into the train (10%) and test (90%) for 10-fold cross-validation. For each fold, all networks are trained using Stochastic Gradient Descent (SGD) with a
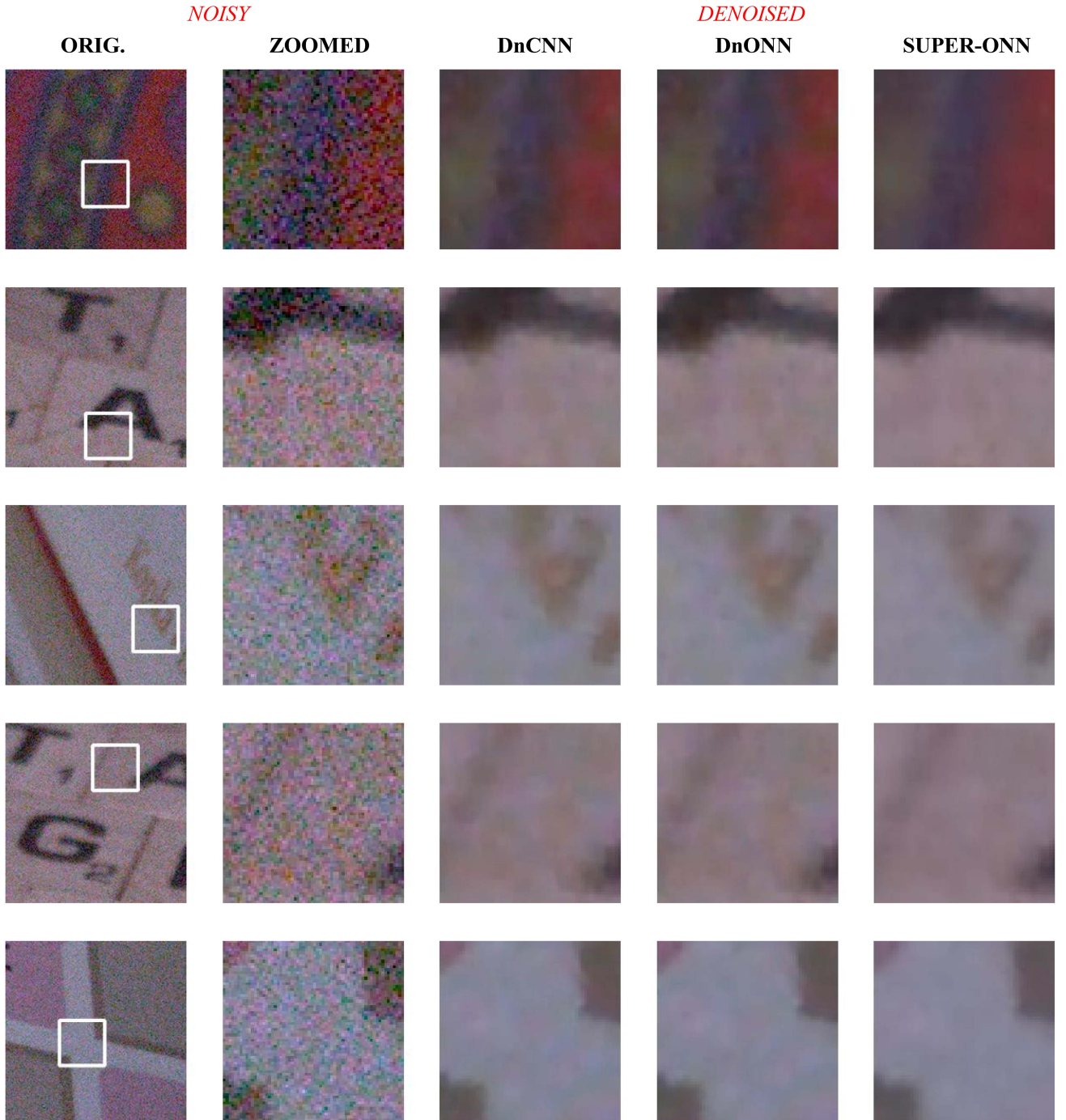
Fig. 3. Some sample images with real-world noise (left), their zoomed sections (2nd column), and the corresponding outputs of the DnCNN (3rd column), DnONN (4th column), and Super-ONN (right) from the validation set of SIDD dataset.

fixed learning parameter as presented in Table IV. Finally, 5 BP runs are performed and the network model that achieved the minimum loss (MSE) during these runs is used for evaluation (tested over the rest of the dataset).

1) Deblurring: Image deblurring [24], [25], [26], [27], [28], [29], [30], [31], [32], [33], [34], [35], [36] can broadly be categorized as kernel-based estimation [24], [25], [26], [27], [28], [29], [30], or an end-to-end system [31], [32], [33], [34], [35], [36]. Deep CNNs have been used for each category but

in this study, we shall evaluate the networks in an "end-to-end" configuration, that does not have to estimate the blurring kernel, rather the blurred image is directly transformed into the restored (deblurred) image.

We expect that super neurons with non-localized kernel operations to achieve a superior performance because image deblurring usually requires a large receptive field for enhanced global knowledge [37] while conventional CNNs (and Self-ONNs) can provide local knowledge limited
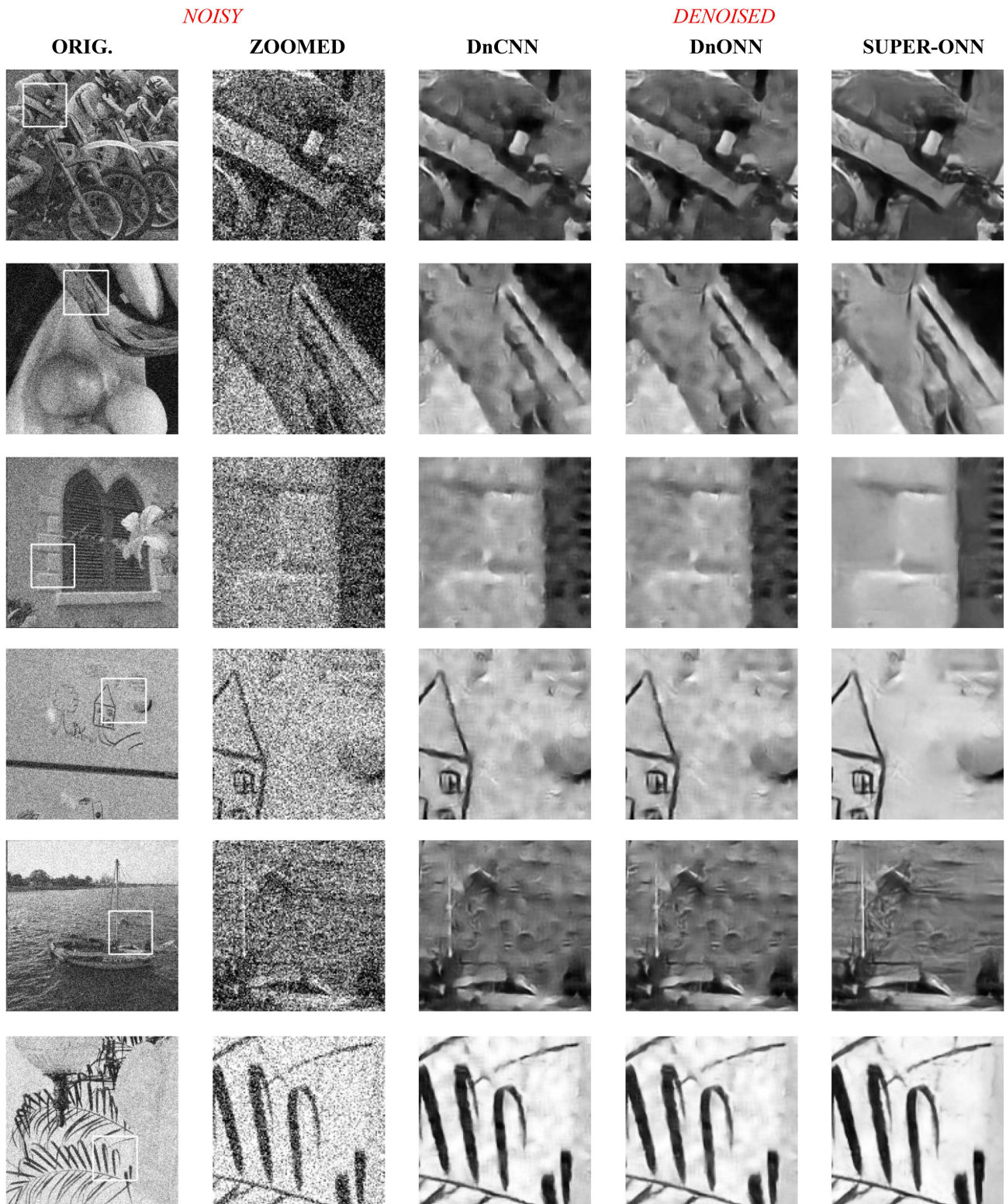
Fig. 4. Some sample images with AWGN corrupted images (left), their zoomed sections (2nd column), and the corresponding outputs of the DnCNN (3rd column), DnONN (4th column), and Super-ONN (right) from the test partition of the KODAK dataset.

with the size of their filters. We consider two blurring problems:

- *Disc(ρ) blurring:* A circular averaging filter (pillbox) within the square matrix of size, $2 \times \rho + 1$.
- *Motion* (Λ, Θ) *blurring:* The linear motion of a camera where Λ specifies the length of the motion and Θ specifies the angle of motion in degrees in a counter-clockwise direction.

In both problems, we aim to evaluate the learning capability of the super neurons in Self-ONNs against the generative, conventional, and especially deformable convolutional neuron models under harsh conditions and for this reason, along with the aforementioned restrictions, we apply a severe blurring with the following parameter settings: $\rho = 5, \Lambda = 11$ and $\Theta = \pi/4$. *Disc(ρ)* basically applies an averaging of 11×11 pixels and *Motion*(Λ, Θ) approximates a linear motion of 11 pixels diagonally. Both blurring artifacts can sometimes cause such severe image degradation that makes it difficult or even infeasible to comprehend the content of the image (e.g., see Fig. 6). Finally, for the random bias ranges for super neurons where $(\alpha_k^i, \beta_k^i) \in \mathbb{Z}[\pm\mathbf{\Gamma}]$, are set as $\mathbf{\Gamma} = \{4, 4, 2\}$ for the 1st, 2nd, and output layers, respectively. Thus, with this setting, for instance, the 1st layer super neurons will have the improved size of the receptive fields as 11×11 pixels, which is significantly larger than the original kernel size of 3x3 pixels.

Fig. 5 shows PSNR and SSIM plots of the best *Disc-5* (top) and motion (bottom) deblurring results per fold over the test partitions. The Self-ONNs with generative neurons having no bias are with the label '0-0-0'), and with super neurons having random bias within $\mathbf{\Gamma} = \{4, 4, 2\}$ are with the label '4-4-2', and with BP-optimized bias are with the label 'Opt', respectively. The conventional and the two versions of the deformable CNNs have the labels 'CNN $\times$ 4', 'DefCNN $\times$ 4 v1', and 'DefCNN $\times$ 4 v2', respectively. The average PSNR and SSIM scores are presented at the end of each corresponding plot.

In both problems, Self-ONNs achieve significantly higher PSNR (around 1 dB) and SSIM ($> 4\%$) levels as compared to the three CNN models with four times more neurons. Though both deformable CNNs achieve slightly higher PSNR and SSIM scores than the conventional CNNs in the majority of the folds, they fail to achieve a higher average performance due to the lowest scores obtained in two folds, indicating a robustness issue. Finally, in both problems, the Self-ONNs with super neurons achieve more than 0.6 dB higher PSNR score on average compared to Self-ONNs with generative neurons.

For a visual evaluation, Figs. 6 and 7 show a set of *Disc-5* and *motion*-blurred (input) images, the target image, and the corresponding outputs of CNN×4, Self-ONNs (with no, random, and BP-optimized spatial biases) from the test partition. We skipped the outputs from both deformable CNNs since they have a very similar or occasionally worse visual quality than the conventional CNN×4 model. The superior deblurring performance of Self-ONNs with super neurons is visible in all outputs.

2) Face Segmentation: Deep CNNs have often been used in face and object segmentation tasks [52], [53], [54], [55], [56], [57], [58], [59], [60], [61]. In this study, we use the benchmark
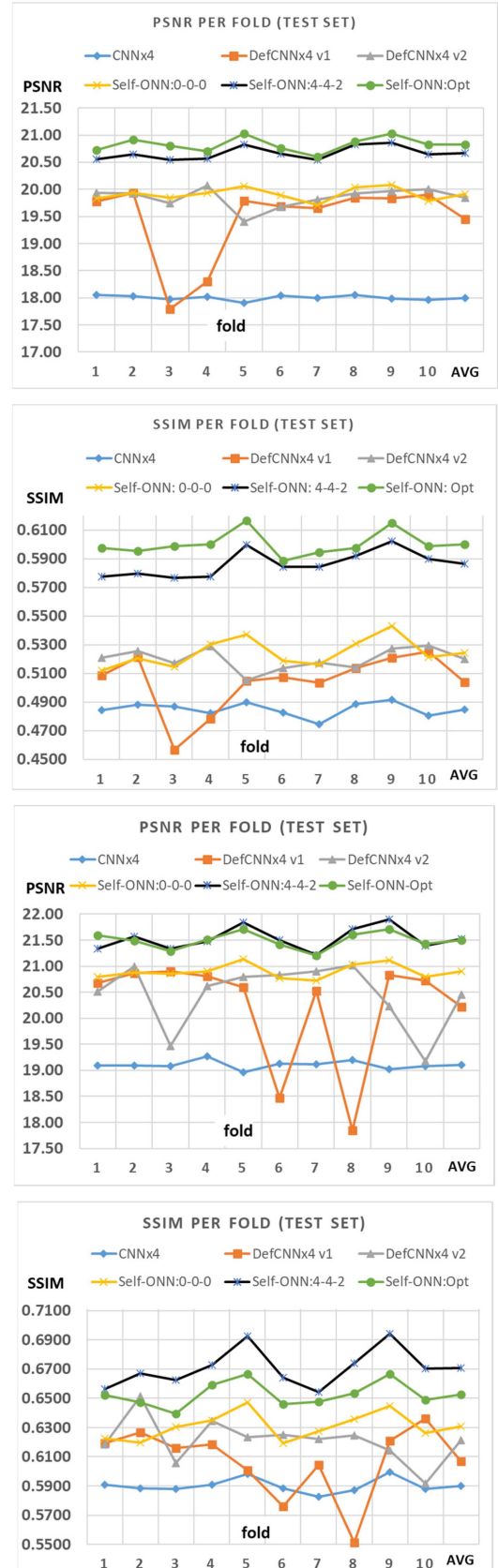

Fig. 5. Best PSNR and SSIM scores for each *Disc-5* (top) and *Motion* (bottom) deblurring fold achieved by the corresponding Self-ONNs (with no, random and BP-optimized spatial biases) and the three CNN×4 configurations over the test set.

Fig. 6. Some typical original (target) and *Disc-5* blurred (input) images and the corresponding outputs of the CNN×4 and the three Self-ONNs (with no, random and BP-optimized spatial bias) from the test partition.

Fig. 7. Some typical original (target) and *Motion* blurred (input) images and the corresponding outputs of the CNN×4 and the three Self-ONNs (with no, random and BP-optimized spatial bias) from the test partition.

FDDB face detection dataset [44], which contains 2000 images with one or many human faces in each image. As per the aforementioned restriction, all images are down-sampled to 60×60 pixels and in this very low resolution, pixel-accurate face segmentation becomes an even more challenging task.

Finally, for the random bias ranges for super neurons where $(\alpha_k^i, \beta_k^i) \in \mathbb{Z}[\pm\mathbf{\Gamma}]$, are set as $\Gamma = \{4, 4, 2\}$ for the 1st, 2nd and output layers, respectively. Thus, with this setting, for instance, the 1st layer super neurons will have the improved size of the receptive fields as 11×11 pixels, which is significantly larger than the original kernel size of 3x3 pixels.

Fig. 8 shows F1 plots of the best (in training) *Face Segmentation* results per fold over the test set. The average test F1 scores achieved by the three Self-ONNs are 80.95% (no

bias), 83.83% (random bias), 84.22% (BP-optimized bias), respectively whilst the CNN × 4 has the F1-score of 75.94%. In both train and test partitions and all folds Self-ONNs achieve significantly higher performance as compared to CNNs. This is despite the fact that it has four times less neurons. In particular, the average performance gap between CNNs and Self-ONNs with super neurons is widened around 8% and 5.6% in train and test partitions, respectively. Finally, the Self-ONNs with super neurons can achieve higher than 3% (train) and 4% (test) on the average than the corresponding performance of the Self-ONNs with generative neurons.

For a visual evaluation, Fig. 9 shows some typical original input images (first column), their (target) ground-truth face maps (last column), and the corresponding outputs of the CNN × 4 and the three Self-ONNs (with no, random and BP-optimized
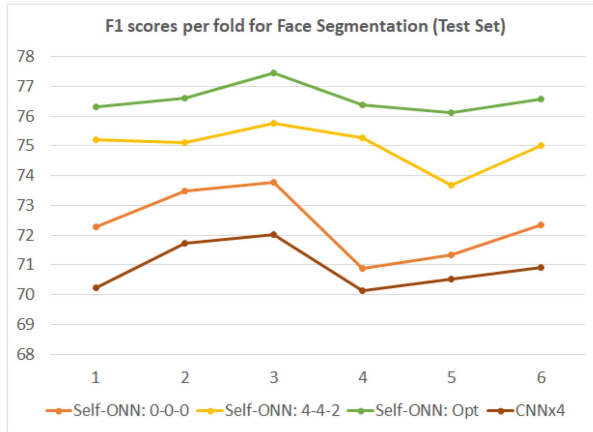
Fig. 8. Best F1 scores for each *face segmentation* fold achieved by the corresponding Self-ONN (with no, random and BP-optimized spatial biases) and CNNx4.

Fig. 9. Some typical original input images (first column) their (target) ground-truth face maps (last column) and the corresponding outputs of the CNN × 4 and the three Self-ONNs (with no, random and BP-optimized spatial bias) from the test partition.
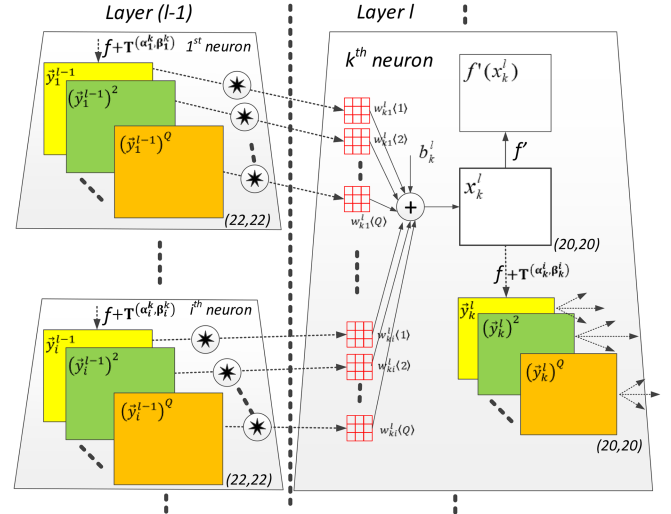

Fig. 10. Illustration of a super-ONN where non-linear operation is accomplished with a set of individual convolutions over the shifted output maps and their powers.

spatial bias) from the test partition. Obviously, the best face segmentation results belong to the Self-ONNs with super neurons while CNNs suffer from severe false-positive regions. The super neurons with BP-optimized spatial bias yield the overall best results with minimal false positives.

C. Computational Complexity Analysis

In this section, the computational complexity of the proposed Self-ONNs with super neurons is analyzed with respect to the parameter-equivalent Self-ONNs with generative neurons and the three CNN models with 4 times more neurons. As assumed in this study, when the pool operator is the "summation", $P_i^l = \Sigma$, in an FP of a Self-ONN with super neurons, (3) can be expressed as follows:

$$x_k^l = b_k^l + \sum_{i=1}^{N_{l-1}} \boldsymbol{oper2D}(\vec{y}_i^{l-1}, w_{ki}^l, {}'NoZeroPad')$$

$$x_k^l(m,n)\Big|_{(0,0)}^{(M-1,N-1)} = b_k^l$$

$$+ \sum_{i=1}^{N_{l-1}} \left(\sum_{r=0}^{K_x-1} \sum_{t=0}^{K_y-1} \boldsymbol{\Psi}\left(\vec{y}_i^{l-1}(m+r,n+t), \boldsymbol{w_{ki}^l}(\mathbf{r},\mathbf{t})\right) \right) \quad (8)$$

where $\boldsymbol{\Psi}$ is the (Taylor series) nodal operator function and $\boldsymbol{w_{ki}^l}(\mathbf{r},\mathbf{t})$ is a Q-dimensional array for the kernel element $(\mathbf{r},\mathbf{t})$. Putting the qth order 2D kernel, $w_{ki}^l\langle q \rangle$ ($q = 1..Q$), which is composed of the *kernel elements*, $w_{ik}^{l+1}(\mathbf{r},\mathbf{t},q)$, then (8) can be simplified as,

$$x_k^l = b_k^l$$

$$+ \sum_{q=1}^{Q} \left\{ \sum_{i=1}^{N_{l-1}} conv2D\left(\left(\vec{y}_i^{l-1}\right)^q, w_{ki}^l\langle q \rangle, {}'NoZeroPad' \right) \right\} \quad (9)$$

Such a 2D convolutional representation of a generative neuron's input map formation is illustrated in Fig. 10. It is straightforward to see that this indeed resembles a multi-output and multi-kernel convolutional neuron. When the shifted powers of the output maps, $\left(\bar{y}_1^{l-1}\right)^q$, for $q = 1, \ldots, Q$, are computed for all hidden neurons in the network, (9) simply turns out to be $(Q \times N_{l-1})$ independent 2D convolutions. Like in conventional CNNs, this can be implemented in a parallel manner, and hence, it will roughly take a similar inference time. We can thus conclude that, in a parallelized implementation, a Self-ONN and a CNN with the same configuration have similar computational complexity. For both super neuron models, the number of parameters, $PARs$, in the Self-ONNs can be expressed as,

$$PARs = \sum_{1}^{L} PARs\,(l)$$
$$= \sum_{1}^{L} \left(\left(\left(N_{l-1} \times (K_x^l \times K_y^l \times Q_l + 2)\right) + 1\right) * N_l\right)$$
$$(10)$$

For each network model, Table III presents the number of network parameters, $PARs$ and the memory overhead, which is the additional memory needed during the FP besides the network parameters and I/O buffers for feature maps. Besides having 4 times more neurons, it is apparent from the table that all CNN $\times$ 4 models have around 2.3 to 6 times more parameters and 2.2 to 5 times higher computational complexity than the Self-ONNs with super neurons that are configured with the non-localized kernel operations by random spatial bias. Whilst having a similar computational complexity, the only overhead cost for super neurons over the generative neurons is about 1.22 times more parameters due to the spatial bias elements. This is true for both models, randomized and BP-optimized bias; however, super neurons with BP-optimized bias have around 1.1 times higher computational complexity than the models with no bias (generative neurons) and random bias. This is due to the bilinear interpolation performed to compute the shifted output maps.

Particularly, for deformable CNN $\times$ 4 models, v1 and v2, the memory overhead, MEM^+, can be expressed as follows:

$$MEM^+ = \sum_{1}^{L} MEMs\,(l)$$
$$= \sum_{1}^{L} \left(K \left(Bs \times Gs \times K_x^l \times K_y^l \times W_x^l \times W_y^l\right)\right)$$
$$(11)$$

where K is constant ($K = 2$ for v1 and $K = 3$ for v2), Bs is the batch size, Gs is the group size, W_x^l and W_y^l are the width and height of the input feature map of the layer, l. The memory overhead can, therefore, be infeasibly large, especially for deep networks with practical settings. As an example, for a single layer with 256 $\times$ 256 pixel feature maps, 3 $\times$ 3 kernels, and $Gs = Bs = 8$, v1 and v2 versions of deformable CNNs will require around 302 Mb and 452 Mb extra memory, respectively only for a single layer.

V. CONCLUSION

The ancient neuron model from the 1950s [38] has been used by the MLPs ever since, and later on shared by its popular derivative, the conventional CNNs. As a linear model, it can only perform linear transformations with "localized" kernels making CNNs entirely homogenous with a static neuron model in terms of transformation and localization. This study is inspired by the well-known proverb, "doing the *right* thing at the *right* place and the *right* time". The Self-ONNs with the generative neuron model can do the "right thing" by customizing each nodal operator on the fly. So, the generative neurons can create the best possible operator for the kernel of each connection during BP training. However, generative neurons can neither locate the "*right* place" for their kernels nor enhance their "receptive field" bounded by the kernel size. To overcome this, the proposed super neurons can be jointly optimized to do the *right* transformation at the *right* (kernel) location of the *right* connection to maximize the learning performance. This study has proposed two models for super neurons: *randomized* and *BP-optimized* kernel localizations of each connection. Both models improve the size of the receptive fields but only the latter one can seek the *right* (kernel) location of each connection. However, we observe that the underlying problem may not require the "right" location and in this case, both approaches are expected to perform either equally well or the former approach can work even slightly better than the latter because it can optimize each nodal operator of each kernel during the entire BP run without altering the location. Whereas the latter approach jointly optimizes both the nodal operator and the location (spatial bias) during the BP run, this is a significantly harder task because the optimization of the nodal operator cannot be finalized while the kernel keeps moving in each BP iteration. In other words, the optimal nodal operator will obviously be different for different kernel locations, and until the location (the spatial bias) is converged the nodal operator optimization cannot be finalized.

Both models of super neurons are evaluated against the conventional and deformable convolutional neurons of CNNs and generative neurons of Self-ONNs. First, shallow Self-ONNs with super-neurons (Super-ONNs in short) have been tested against deep models: DnCNN and DnONN in a Real-World denoising problem. Despite being a significantly shallow model with few neurons, Super-ONNs outperformed both deep models. Then, to reveal the true learning capabilities of super neurons, we purposefully selected challenging learning tasks and applied harsh learning conditions and restrictions such as scarce train data, shallow configurations with few neurons, and minimal kernel size. Despite 4 times more learning units (neurons) being used for all CNN models for comparative evaluations, the results clearly show that Self-ONNs with super neurons can achieve a superior learning and generalization capability thanks to the improved receptive field size they can provide. The computational complexity analysis reveals that an elegant computational efficiency is also achieved in terms of network parameters and memory overhead. In most problems, a notable performance gap is observed over the conventional Self-ONNs with generative neurons without any significant computational burden.

We can foresee that further performance boost can be expected for the Self-ONNs with super-neurons with the following improvements:

- instead of fixing to some naïve values for the two hyper-parameters, Q and Γ, we are aiming to optimize each parameter per layer,
- adapting a better optimization scheme for training, e.g., SGD with momentum [39], AdaGrad [40], RMSProp [41], Adam [42] and its variants [43], all of which should be adapted for Super-ONNs for proper functioning,
- and implementing other kernel operations such as scaling and rotation.

These will be the topics for our future research. The optimized PyTorch implementations of Self-ONNs and Super-ONNs are publicly shared in [62].

APPENDIX

A. Training by Back Propagation for Self-ONNs With Generative Neurons

For Self-ONNs, the contributions of each pixel in the $M \times N$ output map, $y_k^l(m, n)$ on the next layer input map, $x_i^{l+1}(m, n)$, can now be expressed as in (12) shown at the bottom of this page. Using the chain rule, the delta error of the output pixel, $y_k^l(m, n)$, can therefore, be expressed as in (13) shown at the bottom of this page, in the generic form of pool, P_i^{l+1}, and composite nodal operator function, Ψ, of each operational neuron

$i \in [1, .., N_{l+1}]$ in the next layer. In (13), note that the first term,
$$\frac{\partial x_1^{l+1}(m-r,n-t)}{\partial P_i^{l+1}[..,\Psi(y_k^l(m,n),\boldsymbol{w_{ik}^{l+1}}(\boldsymbol{r,t})),..]} = 1.$$

Let $\quad \nabla_{\boldsymbol{\Psi}} P_i^{l+1}(m,n,r,t) = \frac{\partial P_i^{l+1}[..,\Psi(y_k^l(m,n),\boldsymbol{w_{ik}^{l+1}}(\boldsymbol{r,t})),..]}{\partial \Psi(y_k^l(m,n),\boldsymbol{w_{ik}^{l+1}}(\boldsymbol{r,t}))}$

and $\quad \nabla_y \Psi(m,n,r,t) = \frac{\partial \Psi(y_k^l(m,n),\boldsymbol{w_{ik}^{l+1}}(\boldsymbol{r,t}))}{\partial y_k^l(m,n)}.$ Then, (13) simplifies to (14) shown at the bottom of this page. Note further that Δy_k^l, $\nabla_{\Psi_{ki}} P_i^{l+1}$ and $\nabla_y \Psi$ have the same size, $M \times N$ while the next layer delta error, Δ_i^{l+1}, has the size, $(M - K_x + 1) \times (N - K_y + 1)$, respectively. Therefore, to enable this variable 2D convolution in this equation, the delta error, Δ_i^{l+1}, is padded by zeros at all four boundaries ($K_x - 1$ zeros on left and right, $K_y - 1$ zeros on the bottom and top). Thus, $\nabla_y \Psi(m,n,r,t)$ can simply be expressed as in (15) shown at the bottom of this page.

Once the Δy_k^l is computed, using the chain-rule, one can express,

$$\Delta_k^l = \frac{\partial E}{\partial x_k^l} = \frac{\partial E}{\partial y_k^l} \frac{\partial y_k^l}{\partial x_k^l} = \frac{\partial E}{\partial y_k^l} f'(x_k^l) = \Delta y_k^l \, f'(x_k^l) \quad (16)$$

When there is a down-sampling by factors, *ssx* and *ssy*, then the back-propagated delta-error should be first up-sampled to compute the delta-error of the neuron. Let zero order up-sampled map be: $uy_k^l = \underset{ssx,ssy}{\text{up}}(y_k^l)$. Then (16) can be modified, as follows:

$$\Delta_k^l = \frac{\partial E}{\partial x_k^l} = \frac{\partial E}{\partial y_k^l} \frac{\partial y_k^l}{\partial x_k^l} = \frac{\partial E}{\partial y_k^l} \frac{\partial y_k^l}{\partial uy_k^l} \frac{\partial uy_k^l}{\partial x_k^l}$$

$$x_i^{l+1}(m-1,n-1) = \ldots + P_i^{l+1}\left[\boldsymbol{\Psi}\left(y_k^l(m-1,n-1),\boldsymbol{w_{ik}^{l+1}}(\boldsymbol{0,0})\right),\ldots,\boldsymbol{\Psi}\left(y_k^l(m,n),\boldsymbol{w_{ik}^{l+1}}(\boldsymbol{1,1})\right)\right]$$

$$+\ldots x_i^{l+1}(m-1,n) = \ldots + P_i^{l+1}\left[\boldsymbol{\Psi}\left(y_k^l(m-1,n),\boldsymbol{w_{ik}^{l+1}}(\boldsymbol{0,0})\right),\ldots,\boldsymbol{\Psi}\left(y_k^l(m,n),\boldsymbol{w_{ik}^{l+1}}(\boldsymbol{1,0})\right),\ldots\right]$$

$$+\ldots \boldsymbol{x_i^{l+1}}(\boldsymbol{m,n}) = \ldots + P_i^{l+1}\left[\boldsymbol{\Psi}\left(y_k^l(m,n),\boldsymbol{w_{ik}^{l+1}}(\boldsymbol{0,0})\right),..,\boldsymbol{\Psi}\left(y_k^l(m+r,n+t),\boldsymbol{w_{ik}^{l+1}}(\boldsymbol{r,t}),)\ldots\right)\right] + \ldots$$

$$\ldots \ldots$$

$$\therefore \; x_i^{l+1}(m-r,n-t)\Big|_{(1,1)}^{(M-1,N-1)} = b_i^{l+1} + \sum_{k=1}^{N_1} P_i^{l+1}\left[\ldots,\boldsymbol{\Psi}\left(y_k^l(m,n),\boldsymbol{w_{ik}^{l+1}}(\boldsymbol{r,t})\right),\ldots\right] \quad (12)$$

$$\therefore \; \frac{\partial E}{\partial y_k^l}(m,n)\Big|_{(0,0)}^{(M-1,N-1)} = \Delta y_k^l(m,n) = \sum_{i=1}^{N_{l+1}}\left(\sum_{r=0}^{K_x-1}\sum_{t=0}^{K_y-1} \begin{array}{c} \frac{\partial E}{\partial x_i^{l+1}(m-r,n-t)} \times \frac{\partial x_i^{l+1}(m-r,n-t)}{\partial P_i^{l+1}[..,\boldsymbol{\Psi}(y_k^l(m,n),\boldsymbol{w_{ik}^{l+1}}(\boldsymbol{r,t})),..]} \\ \times \frac{\partial P_i^{l+1}[..,\boldsymbol{\Psi}(y_k^l(m,n),\boldsymbol{w_{ik}^{l+1}}(\boldsymbol{r,t})),..]}{\partial \boldsymbol{\Psi}(y_k^l(m,n),\boldsymbol{w_{ik}^{l+1}}(\boldsymbol{r,t}))} \times \frac{\partial \boldsymbol{\Psi}(y_k^l(m,n),\boldsymbol{w_{ik}^{l+1}}(\boldsymbol{r,t}))}{\partial y_k^l(m,n)} \end{array}\right)$$
$$(13)$$

$$\Delta y_k^l(m,n)\Big|_{(0,0)}^{(M-1,N-1)} = \sum_{i=1}^{N_{l+1}}\left(\sum_{r=0}^{K_x-1}\sum_{t=0}^{K_y-1} \Delta_i^{l+1}(m-r,n-t) \times \nabla_{\boldsymbol{\Psi}} P_i^{l+1}(m,n,r,t) \times \nabla_y \boldsymbol{\Psi}(m,n,r,t)\right)$$

$$\text{Let } \nabla_y P_i^{l+1}(m,n,r,t) = \nabla_{\boldsymbol{\Psi}} P_i^{l+1}(m,n,r,t) \times \nabla_y \boldsymbol{\Psi}(m,n,r,t), then$$

$$\Delta y_k^l = \sum_{i=1}^{N_{l+1}} Conv2Dvar\left\{\Delta_i^{l+1}, \nabla_y P_i^{l+1}(m,n,r,t)\right\} \quad (14)$$

$$\nabla_y \boldsymbol{\Psi}(m,n,r,t) = w_{ik}^{l+1}(\text{r,t},1) + 2w_{ik}^{l+1}(\text{r,t},2) y_k^l(m,n) + \ldots + Q w_{ik}^{l+1}(\text{r,t},Q) y_k^l(m,n)^{Q-1} \quad (15)$$

$$= \underset{ssx,ssy}{\text{up}} \left(\Delta y_k^l\right) \beta f'\left(x_k^l\right) \tag{17}$$

where $\beta = \frac{1}{ssx.ssy}$ since each pixel of y_k^l is now obtained by averaging ($ssx.ssy$) number of pixels of the intermediate output, uy_k^l. Finally, when there is a up-sampling by factors, *usx* and *usy*, then let the average-pooled map be: $dy_k^l = \underset{usx,usy}{\text{down}}\left(y_k^l\right)$. Then (17) can be updated as follows:

$$\Delta_k^l = \frac{\partial E}{\partial x_k^l} = \frac{\partial E}{\partial y_k^l}\frac{\partial y_k^l}{\partial x_k^l} = \frac{\partial E}{\partial y_k^l}\frac{\partial y_k^l}{\partial dy_k^l}\frac{\partial dy_k^l}{\partial x_k^l}$$
$$= \underset{usx,usy}{\text{down}}\left(\Delta y_k^l\right)\beta^{-1}f'\left(x_k^l\right) \tag{18}$$

As for the computation of the kernel and bias sensitivities, recall the expression between an individual kernel weight array, $\boldsymbol{w}_{ik}^{l+1}(\mathbf{r},\mathbf{t})$, and the input map of the next layer, $x_i^{l+1}(m,n)$:

$$x_i^{l+1}(m,n)\Big|_{(1,1)}^{(M-1,N-1)} = b_i^{l+1}$$
$$+ \sum_{i=1}^{N_{l-1}} P_i^{l+1}\left[\begin{array}{c}\Psi\left(y_k^l(m,n),\boldsymbol{w}_{ik}^{l+1}(\mathbf{0},\mathbf{0})\right),..,\\ \Psi\left(y_k^l(m+r,n+t),\boldsymbol{w}_{ik}^{l+1}(\mathbf{r},\mathbf{t})\right)...\end{array}\right] \tag{19}$$

where the qth element of the array, $\boldsymbol{w}_{ik}^{l+1}(\mathbf{r},\mathbf{t})$, contributes to all the pixels of $x_i^{l+1}(m,n)$. By using the chain rule of partial derivatives, one can express the weight sensitivities, $\frac{\partial E}{\partial w_{ik}^{l+1}}$, in (20) shown at the bottom of this page. A close look to (20) reveals that, $\frac{\partial \Psi(y_k^l(m+r,n+t),\boldsymbol{w}_{ik}^{l+1}(\mathbf{r},\mathbf{t}))}{\partial w_{ik}^{l+1}(r,t,q)} = y_k^l(m+r,n+t)^q$, which then simplifies to (21) shown at the bottom of this page, Note that in this equation, the first term, $\Delta_1^{l+1}(m,n)$, is independent from the kernel indices, r and t. It will be element-wise multiplied by the other two latter terms, each with the same dimension $(M-Kx+1)x(N-Ky+1)$, and created by derivative functions of nodal and pool operators applied over the shifted pixels of $y_k^l(m+r,n+t)$ and the corresponding weight value, $\boldsymbol{w}_{ik}^{l+1}(\mathbf{r},\mathbf{t})$.

If $P_i^{l+1} = \Sigma$, then

$$\frac{\partial E}{\partial w_{ik}^{l+1}}(r,t,q)\Bigg|_{(0,0,1)}^{(Kx-1,Ky-1,Q)}$$
$$= \sum_{m=0}^{M-Kx}\sum_{n=0}^{N-Ky}\Delta_1^{l+1}(m,n)\times y_k^l(m+r,n+t)^q$$
$$\therefore \frac{\partial E}{\partial w_{ik}^{l+1}}\langle q\rangle = conv2D\left(\Delta_i^{l+1},\left(y_k^l\right)^q,'NoZeroPad'\right) \tag{22}$$

$$\frac{\partial E}{\partial b_k^l} = \sum_m\sum_n\frac{\partial E}{\partial x_k^l(m,n)}\frac{\partial x_k^l(m,n)}{\partial b_k^l} = \sum_m\sum_n\Delta_k^l(m,n) \tag{23}$$

In (21) there is no need to register a 4D matrix for $\nabla_w\Psi = y_k^l(m+r,n+t)^q$ since it can directly be computed from the outputs of the neurons. Moreover, when the pool operator is the *sum*, then $\nabla_\Psi P_i^{l+1}(m,n,r,t) = 1$ and (21) will simplify to (22) where $\frac{\partial E}{\partial w_{ik}^{l+1}}\langle q\rangle$ is the qth 2D sensitivity kernel, which contains the updates (SGD sensitivities) for the weights of the qth order outputs in Maclaurin polynomial. Finally, the bias sensitivity expressed in (23) is the same for ONNs and CNNs since the bias is the common additive term for all.

Let $w_{ik}^{l+1}\langle q\rangle$ be the qth 2D sub-kernel where $q = 1..Q$ and composed of *kernel elements*, $w_{ik}^{l+1}(r,t,q)$. During each BP iteration, t, the kernel parameters (weights), $w_{ik}^{l+1}\langle q\rangle(t)$, and biases, $b_i^l(t)$, of each neuron in the Self-ONN are updated until a stopping criterion is met. Let, $\varepsilon(t)$, be the learning factor at iteration, t. One can express the update for the weight kernel and bias at each neuron, i, at layer, l as follows:

$$w_{ik}^{l+1}\langle q\rangle(t+1) = w_{ik}^{l+1}\langle q\rangle(t) - \varepsilon(t)\frac{\partial E}{\partial w_{ik}^{l+1}}\langle q\rangle$$
$$b_i^l(t+1) = b_i^l(t) - \varepsilon(t)\frac{\partial E}{\partial b_i^l} \tag{24}$$

As a result, the pseudo-code for BP is presented in Algorithm 1.

$$\frac{\partial E}{\partial w_{ik}^{l+1}}(r,t,q)\Bigg|_{(0,0,1)}^{(Kx-1,Ky-1,Q)} = \sum_{m=0}^{M-Kx+1}\sum_{n=0}^{N-Ky+1}\begin{array}{c}\frac{\partial E}{\partial x_1^{l+1}(m,n)}\times\frac{\partial x_1^{l+1}(m,n)}{\partial P_i^{l+1}\left[\Psi\left(y_k^l(m,n),\boldsymbol{w}_{ik}^{l+1}(\mathbf{0},\mathbf{0})\right),..,\Psi\left(y_k^l(m+r,n+t),\boldsymbol{w}_{ik}^{l+1}(\mathbf{r},\mathbf{t})\right)...\right]}\\ \times\frac{\partial P_i^{l+1}\left[\Psi\left(y_k^l(m,n),\boldsymbol{w}_{ik}^{l+1}(\mathbf{0},\mathbf{0})\right),..,\Psi\left(y_k^l(m+r,n+t),\boldsymbol{w}_{ik}^{l+1}(\mathbf{r},\mathbf{t})\right)...\right]}{\partial\Psi\left(y_k^l(m+r,n+t),\boldsymbol{w}_{ik}^{l+1}(\mathbf{r},\mathbf{t})\right)}\\ \times\frac{\partial\Psi\left(y_k^l(m+r,n+t),\boldsymbol{w}_{ik}^{l+1}(\mathbf{r},\mathbf{t})\right)}{\partial w_{ik}^{l+1}(r,t,q)}\end{array} \tag{20}$$

where $\dfrac{\partial x_1^{l+1}(m,n)}{\partial P_i^{l+1}\left[\Psi\left(y_k^l(m,n),\boldsymbol{w}_{ik}^{l+1}(\mathbf{0},\mathbf{0})\right),..,\Psi\left(y_k^l(m+r,n+t),\boldsymbol{w}_{ik}^{l+1}(\mathbf{r},\mathbf{t})\right)...\right]}$

$= 1$ and $\dfrac{\partial\Psi\left(y_k^l(m+r,n+t),\boldsymbol{w}_{ik}^{l+1}(\mathbf{r},\mathbf{t})\right)}{\partial w_{ik}^{l+1}(r,t,q)} = y_k^l(m+r,n+t)^q$

$$\therefore \frac{\partial E}{\partial w_{ik}^{l+1}}(r,t,q)\Bigg|_{(0,0,1)}^{(Kx-1,Ky-1,Q)} = \sum_{m=0}^{M-Kx}\sum_{n=0}^{N-Ky}\Delta_1^{l+1}(m,n)\times\nabla_\Psi P_i^{l+1}(m+r,n+t,r,t)\times y_k^l(m+r,n+t)^q \tag{21}$$

Algorithm 1: BP Training for Self-ONNs With Generative Neurons.

Input: Self-*ONN*,
    $Stopping\ Criteria\ (maxIter, minMSE)$
Output: Self-*ONN∗* = *BP(Self-ONN*
    $, maxIter, minMSE\ )$

1) **Initialize** network parameters randomly (i.e., $\sim U$(-a, a))
2) **UNTIL** a stopping criterion is reached, **ITERATE**:
  a. **For** each mini-batch in the train dataset, **DO**:
    i. **FP**: Forward propagate from the input layer to the output layer to find q^{th} order outputs, $(y_k^l)^q$ and the required derivatives and sensitivities for BP such as $f'(x_k^l)$, $\nabla_y \Psi_{ki}^{l+1}$, $\nabla_{\Psi_{ki}} P_i^{l+1}$ and $\nabla_w \Psi_{ki}^{l+1}$ of each neuron, k, at each layer, l.
    ii. **BP**: Compute delta error at the output layer and then using (14) and (16) back-propagate the error back to the first hidden layer to compute delta errors of each neuron, k, Δ_k^l at each layer, l.
    iii. **PP**: Find the bias and weight sensitivities using (22) and (23), respectively.
    iv. **Update**: Update the weights and biases with the (cumulation of) sensitivities found in previous step scaled with the learning factor, ε, as in (49):
3) **Return** Self-*ONN∗*

B. BP for Non-Localized Kernel Operations by Random Bias

In a conventional BP, starting from the output (operational) layer, the error is back-propagated to the 1$^{\text{st}}$ hidden layer. For the sake of simplicity, for an image **I** in the training dataset suppose that the error (loss) function is L$_2$-loss or the Mean-Square-Error (MSE) error function, $E(I)$, is used can be expressed as,

$$E(I) = \frac{1}{|I|} \sum_p \left(y_1^L(I_p) - T(I_p) \right)^2 \quad (25)$$

where I_p is the pixel p of the image I, T is the target output and y_1^L is the predicted output. The delta error in the output layer of the input map can then be expressed in (26).

$$\Delta_1^L = \frac{\partial E}{\partial x_1^L} = \frac{\partial E}{\partial y_1^L} \frac{\partial y_1^L}{\partial x_1^L} = \frac{2}{|I|} \left(y_1^L(I) - T(I) \right) f'\left(x_1^L(I) \right) \quad (26)$$

For Self-ONNs with super neurons, the contributions of each shifted pixel in the output map, $y_k^l(m + \alpha_k^i, n + \beta_k^i)$, on the next layer input map, $x_i^{l+1}(m, n)$, can now be expressed as in (27) shown at the bottom of the next page, (highlighted in red for clarity). So for the hidden operational layers, a close look at (27) will reveal the fact that the contributions of each pixel in the $(M + 2\Gamma) \times (N + 2\Gamma)$ shifted output map, $y_k^l(m + \alpha_k^i, n + \beta_k^i)$ on the next layer input maps, $x_i^{l+1}(m, n), i \in [1, N_{l+1}]$, depend solely on the bias of each connection, (α_k^i, β_k^i). Therefore, the delta error of the output pixel, $y_k^l(m, n)$, should be computed for each connection and then cumulated. Using the chain rule, the delta error of the output pixel, $y_k^l(m, n)$, can therefore, be expressed as in (28) shown at the bottom of the next page, in the generic form

of pool, P_i^{l+1}, and composite nodal operator function, Ψ, of the ith super neuron, $i \in [1, .., N_{l+1}]$. In (28), note that the first term, $\frac{\partial x_i^{l+1}(m-r, n-t)}{\partial P_i^{l+1}[.., \Psi(y_k^l(m+\alpha_k^i, n+\beta_k^i), w_{ik}^{l+1}(r,t)),..]} = 1$. Let the (shifted) 4D matrices $\nabla_{\Psi} P_i^{l+1}(m + \alpha_k^i, n + \beta_k^i, r, t) = \frac{\partial P_i^{l+1}[.., \Psi(y_k^l(m+\alpha_k^i, n+\beta_k^i), w_{ik}^{l+1}(r,t)),..]}{\partial \Psi(y_k^l(m+\alpha_k^i, n+\beta_k^i), w_{ik}^{l+1}(r,t))}$ and $\nabla_y \Psi(m + \alpha_k^i, n + \beta_k^i, r, t) = \frac{\partial \Psi(y_k^l(m+\alpha_k^i, n+\beta_k^i), w_{ik}^{l+1}(r,t))}{\partial y_k^l(m+\alpha_k^i, n+\beta_k^i)}$. Then, (28) simplifies to (29) shown at the bottom of the next page, where $\nabla_y \Psi(m + \alpha_k^i, n + \beta_k^i, r, t)$ can be directly computed as,

$$\begin{aligned} \nabla_y \Psi \left(m + \alpha_k^i, n + \beta_k^i, r, t \right) &= w_{ik}^{l+1}(\text{r}, \text{t}, 1) \\ &+ 2w_{ik}^{l+1}(\text{r}, \text{t}, 2) \, y_k^l \left(m + \alpha_k^i, n + \beta_k^i \right) + \ldots \\ &+ Q w_{ik}^{l+1}(\text{r}, \text{t}, Q) \, y_k^l \left(m + \alpha_k^i, n + \beta_k^i \right)^{Q-1} \end{aligned} \quad (30)$$

Now let $\nabla_y P_i^{l+1}(m + \alpha_k^i, n + \beta_k^i, r, t) = \nabla_{\Psi} P_i^{l+1}(m + \alpha_k^i, n + \beta_k^i, r, t) \times \nabla_y \Psi(m + \alpha_k^i, n + \beta_k^i, r, t)$. In this study, the *summation* is used as the pool operator for the sake of simplicity, i.e., $P_i^{l+1} = \Sigma$, so, $\nabla_{\Psi} P_i^{l+1}(m, n, r, t) = 1$ and thus, $\nabla_y P_i^{l+1}(m + \alpha_k^i, n + \beta_k^i, r, t) = \nabla_y \Psi(m + \alpha_k^i, n + \beta_k^i, r, t)$ then the delta error computed for the connection to the ith neuron at layer $l+1$ can be expressed as,

$$\begin{aligned} \mathbf{T}^{(\boldsymbol{\alpha_k^i}, \boldsymbol{\beta_k^i})} \left(\Delta y_k^l \right) &= \Delta y_k^l \left(m + \alpha_k^i, n + \beta_k^i \right) \\ &= Conv2Dvar \left\{ \Delta_i^{l+1}, \mathbf{T}^{(\boldsymbol{\alpha_k^i}, \boldsymbol{\beta_k^i})} \left(\nabla_y \Psi \right) \right\} \end{aligned} \quad (31)$$

Finally, the overall delta error for the output map, Δy_k^l, is computed as the cumulation of the back-shifted individual delta-errors, i.e.,

$$\begin{aligned} \Delta y_k^l(m, n) \big|_{(0,0)}^{(M-1, N-1)} \\ = \sum_{i=1}^{N_{l+1}} \mathbf{T}^{(-\boldsymbol{\alpha_k^i}, -\boldsymbol{\beta_k^i})} \left(\Delta y_k^l \left(m + \alpha_k^i, n + \beta_k^i \right) \right) \end{aligned} \quad (32)$$

Once the Δy_k^l is computed, using the chain-rule, one can finalize the back-propagation of the delta error from layer $l+1$ to layer l, as follows:

$$\Delta_k^l = \frac{\partial E}{\partial x_k^l} = \frac{\partial E}{\partial y_k^l} \frac{\partial y_k^l}{\partial x_k^l} = \frac{\partial E}{\partial y_k^l} f'\left(x_k^l \right) = \Delta y_k^l \, f'\left(x_k^l \right) \quad (33)$$

When there is a pooling (down-sampling) by factors, *ssx,* and *ssy,* then the back-propagated delta-error by (33) should be first up-sampled to compute the delta-error of the neuron. Let zero-order up-sampled map be: $uy_k^l = \underset{ssx, ssy}{\text{up}} (y_k^l)$. Then (33) can be modified, as follows:

$$\begin{aligned} \Delta_k^l = \frac{\partial E}{\partial x_k^l} &= \frac{\partial E}{\partial y_k^l} \frac{\partial y_k^l}{\partial x_k^l} = \frac{\partial E}{\partial y_k^l} \frac{\partial y_k^l}{\partial uy_k^l} \frac{\partial uy_k^l}{\partial x_k^l} \\ &= \underset{ssx, ssy}{\text{up}} \left(\Delta y_k^l \right) \beta f'\left(x_k^l \right) \end{aligned} \quad (34)$$

where $\beta = \frac{1}{ssx.ssy}$ since each pixel of y_k^l is now obtained by averaging $(ssx.ssy)$ number of pixels of the intermediate

output, uy_k^l. Finally, when there is an up-sampling by factors, *usx,* and *usy,* then let the average-pooled map be: $dy_k^l =$ down (y_k^l). Then (33) can be updated as follows:
$$\Delta_k^l = \frac{\partial E}{\partial x_k^l} = \frac{\partial E}{\partial y_k^l}\frac{\partial y_k^l}{\partial x_k^l} = \frac{\partial E}{\partial y_k^l}\frac{\partial y_k^l}{\partial dy_k^l}\frac{\partial dy_k^l}{\partial x_k^l}$$
$$= \underset{usx,usy}{\text{down}} \left(\Delta y_k^l\right)\beta^{-1}f'\left(x_k^l\right) \quad (35)$$

As for the computation of the sensitivities for kernel parameters, $\frac{\partial E}{\partial w_{ik}^{l+1}}(r,t,q)\Big|_{(0,0,1)}^{(Kx-1,Ky-1,Q)}$, and bias, $\frac{\partial E}{\partial b_k^l}$, (3) indicates that the qth element of the array, $\boldsymbol{w_{ik}^{l+1}}(\mathbf{r,t})$, contributes to all the pixels of $x_i^{l+1}(m,n)$. Once again by using

the chain rule of partial derivatives, the sensitivities for kernel parameters can be expressed in (36) shown at the bottom of this page. Since $\frac{\partial\Psi(y_k^l(m+\alpha_k^i+r,n+\beta_k^i+t),\boldsymbol{w_{ik}^{l+1}}(\boldsymbol{r,t}))}{\partial w_{ik}^{l+1}(r,t,q)} = y_k^l(m+\alpha_k^i+r,n+\beta_k^i+t)^q$, $P_i^{l+1} = \Sigma$, and $\nabla_\Psi P_i^{l+1}(m+\alpha_k^i+r,n+\beta_k^i+t) = 1$. then (36) simplifies to (37) shown at the bottom of this page.

For the bias sensitivity, the chain rule yields:

$$\frac{\partial E}{\partial b_k^l} = \sum_{m=0}^{M-1}\sum_{n=0}^{N-1}\frac{\partial E}{\partial x_k^l(m,n)}\frac{\partial x_k^l(m,n)}{\partial b_k^l} = \sum_{m=0}^{M-1}\sum_{n=0}^{N-1}\Delta_k^l(m,n) \quad (38)$$

$$x_i^{l+1}(m-1,n-1) = \ldots +$$
$$P_i^{l+1}\left[\Psi\left(y_k^l\left(m+\alpha_k^i-1,n+\alpha_k^i-1\right),\boldsymbol{w_{ik}^{l+1}}(\boldsymbol{0,0})\right),\ldots,\Psi\left(y_k^l\left(m+\alpha_k^i,n+\beta_k^i\right),\boldsymbol{w_{ik}^{l+1}}(\boldsymbol{1,1})\right)\right]+\ldots$$
$$x_i^{l+1}(m-1,n) = \ldots +$$
$$P_i^{l+1}\left[\Psi\left(y_k^l\left(m+\alpha_k^i-1,n+\alpha_k^i\right),\boldsymbol{w_{ik}^{l+1}}(\boldsymbol{0,0})\right),\ldots,\Psi\left(y_k^l\left(m+\alpha_k^i,n+\beta_k^i\right),\boldsymbol{w_{ik}^{l+1}}(\boldsymbol{1,0})\right),\ldots\right]+\ldots$$
$$\boldsymbol{x_i^{l+1}(m,n)} = \ldots +$$
$$P_i^{l+1}\left[\Psi\left(y_k^l\left(m+\alpha_k^i,n+\beta_k^i\right),\boldsymbol{w_{ik}^{l+1}}(\boldsymbol{0,0})\right),..,\Psi(y_k^l\left(m+\alpha_k^i+r,n+\alpha_k^i+t\right),\boldsymbol{w_{ik}^{l+1}}(\boldsymbol{r,t})\ldots)\right]+\ldots$$
$$\ldots$$
$$\therefore x_i^{l+1}(m-r,n-t)\Big|_{(1,1)}^{(M-1,N-1)} = b_i^{l+1} + \sum_{k=1}^{N_1}P_i^{l+1}\left[\ldots,\Psi\left(y_k^l\left(m+\alpha_k^i,n+\beta_k^i\right),\boldsymbol{w_{ik}^{l+1}}(\boldsymbol{r,t})\right),\ldots\right] \quad (27)$$
$$\therefore \frac{\partial E}{\partial y_k^l}\left(m+\alpha_k^i,n+\beta_k^i\right)\Big|_{(0,0)}^{(M-1,N-1)} = \Delta y_k^l\left(m+\alpha_k^i,n+\beta_k^i\right)$$
$$= \sum_{r=0}^{Kx-1}\sum_{t=0}^{Ky-1}\frac{\frac{\partial E}{\partial x_i^{l+1}(m-r,n-t)}\times\frac{\partial x_i^{l+1}(m-r,n-t)}{\partial P_i^{l+1}[..,\Psi(y_k^l(m+\alpha_k^i,n+\beta_k^i),\boldsymbol{w_{ik}^{l+1}}(\boldsymbol{r,t})),..]}}{\times\frac{\partial P_i^{l+1}[..,\Psi(y_k^l(m+\alpha_k^i,n+\beta_k^i),\boldsymbol{w_{ik}^{l+1}}(\boldsymbol{r,t})),..]}{\partial\Psi(y_k^l(m+\alpha_k^i,n+\beta_k^i),\boldsymbol{w_{ik}^{l+1}}(\boldsymbol{r,t}))}\times\frac{\partial\Psi(y_k^l(m+\alpha_k^i,n+\beta_k^i),\boldsymbol{w_{ik}^{l+1}}(\boldsymbol{r,t}))}{\partial y_k^l(m+\alpha_k^i,n+\beta_k^i)}} \quad (28)$$
$$\Delta y_k^l\left(m+\alpha_k^i,n+\beta_k^i\right) = \sum_{r=0}^{K_x-1}\sum_{t=0}^{K_y-1}\Delta_i^{l+1}(m-r,n-t)\times\nabla_\Psi P_i^{l+1}\left(m+\alpha_k^i,n+\beta_k^i,r,t\right)\times\nabla_y\Psi\left(m+\alpha_k^i,n+\beta_k^i,r,t\right) \quad (29)$$

$$\frac{\partial E}{\partial w_{ik}^{l+1}}(r,t,q)\Big|_{(0,0,1)}^{(Kx-1,Ky-1,Q)} = \sum_{m=0}^{M-Kx+1}\sum_{n=0}^{N-Ky+1}\left(\begin{array}{c}\frac{\partial E}{\partial x_1^{l+1}(m,n)}\times\frac{\partial x_1^{l+1}(m,n)}{\partial P_i^{l+1}[..,\Psi(y_k^l(m+\alpha_k^i+r,n+\beta_k^i+t),\boldsymbol{w_{ik}^{l+1}}(\boldsymbol{r,t}))...)]}\\\times\frac{\partial P_i^{l+1}[..,\Psi(y_k^l(m+\alpha_k^i+r,n+\beta_k^i+t),\boldsymbol{w_{ik}^{l+1}}(\boldsymbol{r,t})),...)]}{\partial\Psi(y_k^l(m+\alpha_k^i+r,n+\beta_k^i+t),\boldsymbol{w_{ik}^{l+1}}(\boldsymbol{r,t}))}\\\times\frac{\partial\Psi(y_k^l(m+\alpha_k^i+r,n+\beta_k^i+t),\boldsymbol{w_{ik}^{l+1}}(\boldsymbol{r,t}))}{\partial w_{ik}^{l+1}(r,t,q)}\end{array}\right) \quad (36)$$

$$\frac{\partial E}{\partial w_{ik}^{l+1}}(r,t,q)\Big|_{(0,0,1)}^{(Kx-1,Ky-1,Q)} = \sum_{m=0}^{M-Kx}\sum_{n=0}^{N-Ky}\Delta_1^{l+1}(m,n)\times\nabla_\Psi P_i^{l+1}\left(m+\alpha_k^i+r,n+\beta_k^i+t\right)\times y_k^l\left(m+\alpha_k^i+r,n+\beta_k^i+t\right)^q$$

$$= \sum_{m=0}^{M-Kx}\sum_{n=0}^{N-Ky}\Delta_1^{l+1}(m,n)\times y_k^l\left(m+\alpha_k^i+r,n+\beta_k^i+t\right)^q$$

$$\therefore \frac{\partial E}{\partial w_{ik}^{l+1}}\langle q\rangle = conv2D\left(\Delta_i^{l+1},\left(T^{(\alpha_k^i,\beta_k^i)}\left(y_k^l\right)\right)^q,'NoZeroPad'\right) \quad (37)$$

C. BP for Non-Localized Kernel Operations by the BP-Optimized Bias

Recall that (6) allows us to compute the derivatives of the output map w.r.t the individual bias elements, as expressed in (41). These derivatives will be needed in the BP formulation that will be covered in this section.

The delta error in the output layer of the input map is the same as in (25). With $\vec{y}_k^l(m,n) = y_k^l(m + \alpha_k^i, n + \beta_k^i)$ (28) can be simplified as in (42) shown at the bottom of the next page, and with $P_i^{l+1} = \Sigma$, it yields (43) shown at the bottom of the page, where $\nabla_{\vec{y}} P_i^{l+1}(m,n,r,t) = \nabla_{\Psi} P_i^{l+1}(m,n,r,t) \times \nabla_{\vec{y}} \Psi(m,n,r,t) = \nabla_{\vec{y}} \Psi(m,n,r,t)$ and $\nabla_{\vec{y}} \Psi(m,n,r,t)$ can be directly computed as in (44) shown at the bottom of the next page. Finally, the delta error of $\vec{y}_k^l$ (from its contribution to x_i^{l+1} alone) can be computed as,

$$\mathbf{T}^{(\alpha_k^i, \beta_k^i)}\left(\Delta y_k^l\right) = \Delta \vec{y}_k^l = Conv2Dvar\left\{\Delta_i^{l+1}, (\nabla_{\vec{y}}\Psi)\right\} \tag{39}$$

Basically, in these equations, we are using the grid of $\vec{y}_k^l(m,n)$ - not the original grid of y_k^l. However, we need to compute individual Δy_k^l from the $\Delta \vec{y}_k^l$ for each connection in the next layer so that we can cumulate them to compute the overall delta error for y_k^l. To accomplish this, as in the earlier approach with random (integer) bias, the overall delta error for the output map, Δy_k^l, will be computed as the cumulation of the back-shifted individual delta-errors, $\Delta \vec{y}_k^l$ computed for each connection, i.e.,

$$\Delta y_k^l(m,n)\big|_{(0,0)}^{(M-1,N-1)}$$
$$= \sum_{i=1}^{N_{l+1}} \mathbf{T}^{(-\alpha_k^i, -\beta_k^i)}\left(\Delta y_k^l\left(m + \alpha_k^i, n + \beta_k^i\right)\right)$$
$$= \sum_{i=1}^{N_{l+1}} \mathbf{T}^{(-\lfloor\alpha_k^i\rfloor, -\lfloor\beta_k^i\rfloor)}\left(\Delta y_k^l\left(m_\alpha, n_\beta\right)\right) \tag{40}$$

where $m_\alpha = m + \lfloor\alpha_k^i\rfloor$ and $n_\beta = n + \lfloor\beta_k^i\rfloor$. Since the bias elements are not an integer, we should now use the reverse-interpolation to compute first, $\Delta y_k^l(m + \lfloor\alpha_k^i\rfloor, n + \lfloor\beta_k^i\rfloor)$ as illustrated in Fig. 11. Once again using bilinear interpolation, $\Delta y_k^l(m_\alpha, n_\beta)$ can be computed as expressed in (45) shown at the bottom of the next page. As in the random bias approach, the overall delta error for the output map, Δy_k^l, is computed as the cumulation of the back-shifted individual delta-errors using (40). Once on the integer grid, it is straightforward to compute Δy_k^l using (32).

After the (overall) Δy_k^l is computed, using (33) (or (34) or (35) in case down- or up-sampling is performed), the delta error, Δ_k^l, can be computed and hence, the back-propagation of the (delta) error from layer $l+1$ to the kth neuron at layer l is completed.

Once the back-propagation of delta errors is completed, then weight and bias sensitivities can be computed using (37) and (38) with the same simplifications. Note that $\vec{y}_k^l = \mathbf{T}^{(\alpha_k^i, \beta_k^i)}(y_k^l)$ is the shifted (interpolated) output map as before with the only difference that $(\alpha_k^i, \beta_k^i) \in \mathbb{R}$.

Finally, for the spatial bias sensitivities, $\Delta \alpha_k^i = \frac{\partial E}{\partial \alpha_k^i}$, $\Delta \beta_k^i = \frac{\partial E}{\partial \beta_k^i}$, the spatial bias pair, (α_k^i, β_k^i), shifts only the pixels of the
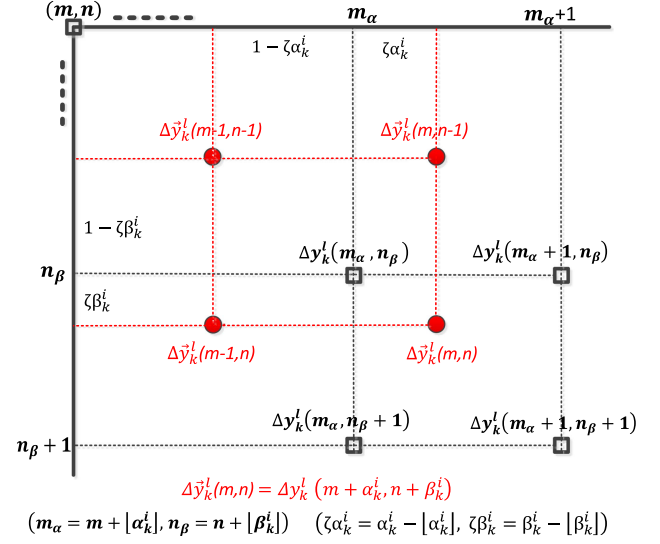

Fig. 11. Reverse interpolation from the shifted delta error, $\Delta \vec{y}_k^l(m,n) = \Delta y_k^l(m + \alpha_k^i, n + \beta_k^i)$ by the bias, $(\alpha_k^i, \beta_k^i) \in \mathbb{R}$, to the original delta error with integer shifts, $\Delta y_k^l(m + \lfloor\alpha_k^i\rfloor, n + \lfloor\beta_k^i\rfloor)$ where $(\lfloor\alpha_k^i\rfloor, \lfloor\beta_k^i\rfloor) \in \mathbb{Z}$.

output map, y_k^l, to contribute to all pixels of x_i^{l+1}. By using the chain rule of partial derivatives, the sensitivities of the spatial bias pair can be expressed in (46) shown at the bottom of the next page. Let $\nabla_\alpha \vec{y}(m,n) = \frac{\partial \vec{y}_k^l(m,n)}{\partial \alpha_k^i}$, which was expressed in (41), (46) finally simplifies to (47) shown at the bottom of the next page, where $\Delta \alpha_k^i$ is a scalar and $\nabla_\alpha \vec{y} \otimes \Delta \vec{y}_k^l$ is the 2D cross-correlation between $\nabla_\alpha \vec{y}$ and $\Delta \vec{y}_k^l$.

$$\frac{\partial \vec{y}_k^l(m+r, n+t)}{\partial \alpha_k^i} = \left(1 - \zeta\beta_k^i\right)\left(y_k^l\left(m_\alpha^r + 1,\ n_\beta^t\right)\right.$$
$$\left. - y_k^l\left(m_\alpha^r,\ n_\beta^t\right)\right)$$
$$+ \zeta\beta_k^i\left(y_k^l\left(m_\alpha^r + 1,\ n_\beta^t + 1\right) - y_k^l\left(m_\alpha^r,\ n_\beta^t + 1\right)\right)$$

$$\frac{\partial \vec{y}_k^l(m+r, n+t)}{\partial \beta_k^i} = \left(1 - \zeta\alpha_k^i\right)\left(y_k^l\left(m_\alpha^r,\ n_\beta^t + 1\right)\right.$$
$$\left. - y_k^l\left(m_\alpha^r,\ n_\beta^t\right)\right)$$
$$+ \zeta\alpha_k^i\left(y_k^l\left(m_\alpha^r + 1,\ n_\beta^t + 1\right) - y_k^l\left(m_\alpha^r + 1,\ n_\beta^t\right)\right) \tag{41}$$

Similarly, it is straightforward to show that the sensitivity, $\Delta \beta_k^i = \frac{\partial E}{\partial \beta_k^i}$, can be expressed as,

$$\Delta \beta_k^i = \frac{\partial E}{\partial \beta_k^i} = \sum_{m=0}^{M-1}\sum_{n=0}^{N-1} \nabla_\beta \vec{y}(m,n) \times \Delta \vec{y}_k^l(m,n)$$
$$= \nabla_\beta \vec{y} \otimes \Delta \vec{y}_k^l \tag{48}$$

where $\nabla_\beta \vec{y}(m,n) = \frac{\partial \vec{y}_k^l(m,n)}{\partial \beta_k^i}$ as expressed in (41). It is interesting to see that both spatial bias sensitivities depend on the cross-correlation of two distinct gradients, the shifted (interpolated) output map delta error and its direct derivative w.r.t the

Parameter	Symbol	Default	Description
Max. no. of iterations	$maxIter$	200	BP stopping criterion
Target min. MSE	$minMSE$	10^{-3}	BP stopping criterion
Range for rand. initialization	$U(-\varpi, \varpi)$	$\varpi = 0.1$	for kernel parameters
	$U(-\Gamma, \Gamma)$	$\Gamma = 8$	for spatial bias
Learning factors	ε	$\varepsilon(t) = 0.1$	for kernel parameters
	γ	$\gamma(t) = 10$	for spatial bias pairs

corresponding bias element. This means that during BP iterations, the ongoing gradient descent operation, e.g., Stochastic Gradient Descent (SGD), will keep updating the kernel location until either correlation between these two gradients vanishes (e.g., they become uncorrelated) or when the (magnitude of the) delta errors diminishes eventually at the final stages of the BP (e.g., convergence of the gradient descent). In other words, the local optimal location of a particular kernel of a particular connection -if exists for the particular problem at hand- will be converged when either of the conditions is satisfied (i.e., when $\Delta\alpha_k^i, \Delta\beta_k^i \approx 0$).

During each BP iteration, t, the kernel parameters, $w_{ik}^{l+1}\langle q\rangle(t)$, and biases, $b_i^l(t)$, (spatial) bias pairs, $\alpha_k^i(t), \beta_k^i(t)$, of each super neuron in the Self-ONN are updated until a stopping criterion is met. Let, $\varepsilon(t)$ and $\gamma(t)$ be the learning factors at

iteration, t, of weights and spatial bias pairs, respectively. One can express the SGD update for the kernel parameters, bias, and the kernel location of each super neuron, i, at the layer, l, in (49) shown at the bottom of the next page. The parameters of a Self-ONN for BP training via SGD are presented in Table IV.

To initiate the BP training by SGD over a dataset, a Self-ONN is first configured according to the network parameters, i.e., number of layers (L) and hidden neurons (N_l), the kernel-size (Kx, Ky), the pooling type and the (polynomial) order for each layer/neuron are set in advance. Let *Self-ONN(0)* be the initially configured network ready for BP training. In the pseudo-code for BP training presented in Algorithm 1, five consecutive stages in an iterative loop are visible: 1) BP initialization (Step 1), 2) Forward-Propagation (FP) of each image in the batch where native and shifted (interpolated) output maps, derivatives and

$$\frac{\partial E}{\partial y_k^l}\left(m + \alpha_k^i, n + \beta_k^i\right)\Bigg|_{(0,0)}^{(M-1,N-1)} = \Delta y_k^l\left(m + \alpha_k^i, n + \beta_k^i\right) = \Delta \vec{y}_k^l\left(m, n\right)$$

$$= \sum_{r=0}^{Kx-1}\sum_{t=0}^{Ky-1} \begin{array}{c} \frac{\partial E}{\partial x_i^{l+1}(m-r,n-t)} \times \frac{\partial x_i^{l+1}(m-r,n-t)}{\partial P_i^{l+1}\left[..,\Psi\left(\vec{y}_k^l(m,n),\boldsymbol{w}_{ik}^{l+1}(\boldsymbol{r},\boldsymbol{t})\right),..\right]} \\ \times \frac{\partial P_i^{l+1}\left[..,\Psi\left(\vec{y}_k^l(m,n),\boldsymbol{w}_{ik}^{l+1}(\boldsymbol{r},\boldsymbol{t})\right),..\right]}{\partial\Psi\left(\vec{y}_k^l(m,n),\boldsymbol{w}_{ik}^{l+1}(\boldsymbol{r},\boldsymbol{t})\right)} \times \frac{\partial\Psi\left(\vec{y}_k^l(m,n),\boldsymbol{w}_{ik}^{l+1}(\boldsymbol{r},\boldsymbol{t})\right)}{\partial\vec{y}_k^l(m,n)} \end{array} \tag{42}$$

$$\Delta y_k^l\left(m + \alpha_k^i, n + \beta_k^i\right) = \Delta\vec{y}_k^l\left(m, n\right) = \sum_{r=0}^{K_x-1}\sum_{t=0}^{K_y-1} \Delta_i^{l+1}\left(m-r, n-t\right) \times \nabla_{\vec{y}}\Psi\left(m, n, r, t\right) \tag{43}$$

$$\nabla_{\vec{y}}\Psi\left(m, n, r, t\right) = w_{ik}^{l+1}\left(\mathrm{r}, \mathrm{t}, 1\right) + 2w_{ik}^{l+1}\left(\mathrm{r}, \mathrm{t}, 2\right)\vec{y}_k^l\left(m, n\right) + \ldots + Qw_{ik}^{l+1}\left(\mathrm{r}, \mathrm{t}, Q\right)\vec{y}_k^l\left(m, n\right)^{Q-1} \tag{44}$$

$$\Delta y_k^l\left(m_\alpha^r, n_\beta^t\right) = \Delta\vec{y}_k^l\left(m, n\right)\left(1 - \zeta\alpha_k^i\right)\left(1 - \zeta\beta_k^i\right) + \Delta\vec{y}_k^l\left(m-1, n-1\right)\zeta\alpha_k^i\zeta\beta_k^i +$$

$$\Delta\vec{y}_k^l\left(m-1, n\right)\zeta\alpha_k^i\left(1 - \zeta\beta_k^i\right) + \Delta\vec{y}_k^l\left(m, n-1\right)\left(1 - \zeta\alpha_k^i\right)\zeta\beta_k^i \tag{45}$$

$$\frac{\partial E}{\partial\alpha_k^i} = \Delta\alpha_k^i = \sum_{m=0}^{M-1}\sum_{n=0}^{N-1}\left(\sum_{r=0}^{Kx-1}\sum_{t=0}^{Ky-1} \begin{array}{c} \frac{\partial E}{\partial x_i^{l+1}(m-r,n-t)} \times \frac{\partial x_i^{l+1}(m-r,n-t)}{\partial P_i^{l+1}\left[..,\Psi\left(\vec{y}_k^l(m,n),\boldsymbol{w}_{ik}^{l+1}(\boldsymbol{r},\boldsymbol{t})\right),..\right]} \\ \times \frac{\partial P_i^{l+1}\left[..,\Psi\left(\vec{y}_k^l(m,n),\boldsymbol{w}_{ik}^{l+1}(\boldsymbol{r},\boldsymbol{t})\right),..\right]}{\partial\Psi\left(\vec{y}_k^l(m,n),\boldsymbol{w}_{ik}^{l+1}(\boldsymbol{r},\boldsymbol{t})\right)} \times \frac{\partial\Psi\left(\vec{y}_k^l(m,n),\boldsymbol{w}_{ik}^{l+1}(\boldsymbol{r},\boldsymbol{t})\right)}{\partial\vec{y}_k^l(m,n)} \times \frac{\partial\vec{y}_k^l(m,n)}{\partial\alpha_k^i} \end{array}\right) \tag{46}$$

$$\Delta\alpha_k^i = \frac{\partial E}{\partial\alpha_k^i} = \sum_{m=0}^{M-1}\sum_{n=0}^{N-1} \nabla_\alpha\mathbf{y}\left(m, n\right)$$

$$\times \left(\sum_{r=0}^{K_x-1}\sum_{t=0}^{K_y-1} \Delta_i^{l+1}\left(m-r, n-t\right) \times \nabla_\Psi P_i^{l+1}\left(m + \alpha_k^i, n + \beta_k^i, r, t\right) \times \nabla_{\vec{y}}\Psi\left(m + \alpha_k^i, n + \beta_k^i, r, t\right)\right)$$

$$= \sum_{m=0}^{M-1}\sum_{n=0}^{N-1} \nabla_\alpha\vec{y}\left(m, n\right) \times \Delta\vec{y}_k^l\left(m, n\right) = \nabla_\alpha\vec{y} \otimes \Delta\vec{y}_k^l \tag{47}$$

Algorithm 2: Back-Propagation by SGD for Self-ONNs With Super Neurons.

Input: **Self-ONN(0)**, $Hyper\ Parameters, Train\ Parameters : Stopping\ Criteria\ (maxIter, minMSE), \varpi, \mathbf{\Gamma}, \varepsilon, \gamma$
Output: **Self-ONN∗** = **BP** (**Self-ONN(0)**, SGD, $Hyper\ Parameters, Train\ Parameters$)

1) **Initialize** network parameters of each super neuron:
   a. $w_{ik}^{l+1}\langle q\rangle(0) = U(-\varpi, \varpi), b_i^l(t+1) = U(-\varpi, \varpi)$ for $\forall i \in [1, N_{l+1}], \forall k \in [1, N_l], \forall q \in [1, Q]$
   b. $\alpha_k^i(0) = U(-\mathbf{\Gamma}, \mathbf{\Gamma}), \beta_k^i(0) = U(-\mathbf{\Gamma}, \mathbf{\Gamma})$ for $\forall i \in [1, N_{l+1}], \forall k \in [1, N_l]$
2) **UNTIL** either stopping criterion is reached, **ITERATE** $(t = 1 : maxIter)$:
   a. **For** each batch in the train dataset, **DO**:
      i. **Init**: Assign next item, I_p, directly as the output map(s) in the input layer neurons and using (6) create the shifted output map(s) along with their powers, $(\bar{y}_k^0)^q$, $\forall q \in [1, Q_1]$ where Q_1 is the polynomial order of the super neurons in the 1st hidden layer.
      ii. **FP**: From the previous layer (shifted) output maps, compute each input map in the 1st hidden layer, x_i^1, $\forall i \in [1, N_1]$ using (7), then the native output maps, y_i^1 and finally, the shifted output maps along with their powers, $(\bar{y}_i^1)^q$ $\forall q \in [1, Q_1]$.
      iii. **FP**: Then compute the required derivatives and sensitivities for each hidden layer, such as $f'(x_k^l)$, $\nabla_y \Psi_{ik}^l$, and $\nabla_w \Psi_{ik}^l$ of each neuron, i, at each layer, l. $(\nabla_{\Psi_{ik}} \mathrm{P}_i^l = 1)$
      iv. **FP**: Repeat (ii) until the output layer is reached. Compute the output map(s), $y_1^L(I_p)$, of the neurons in the output layer and then, compute the MSE and delta error, Δ_1^L, using (25) and (26), respectively.
      v. **BP**: For each hidden neuron at the last hidden layer, using (39) compute delta error for the shifted output map and then using (45), perform reverse-interpolation (and shift) to compute the delta error of the actual output map for each connection to the next layer.
      vi. **BP**: Using (32) compute the overall delta error for the output map, Δy_k^l, as the cumulation of the back-shifted individual delta errors.
      vii. **BP**: Finally, using (33) (or (34) or (35) in case down- or up-sampling is performed), compute the delta error at this level, Δ_k^l.
      viii. **PP**: Compute sensitivities for the kernel parameters, bias, and spatial bias pair using (37), (38), (47), and (48) respectively.
      ix. **Update**: Update for the kernel parameters, bias, and the kernel location of each super neuron in the network with the (cumulation of) sensitivities found in step (viii) scaled with the current learning factors, $\varepsilon(t)$ and $\gamma(t)$, using (49).
3) **Return Self-ONN∗**

---

output MSE and delta error are computed (in Step 2.a, i–iii), 3) Back-Propagation (BP) of the delta error from the output layer to the first hidden layer (in Step 2.a, v–vii), 4) post-processing (PP) where the kernel parameter and bias sensitivities, the sensitivities of the spatial bias pair are computed for each image in the batch and cumulated, and 5) Update: when all images in the batch are processed, then the kernel, bias and the kernel location of each super neuron in the network are updated and this is repeated for the other batches and iterations. The pseudo-code In Algorithm 1 can be used for a Self-ONN with super neurons that are configured with the non-localized kernel operations by random spatial bias, the following steps should be modified accordingly. First, the initialization of bias elements should be an integer in 1.b, i.e., $\alpha_k^i(0) = \lfloor U(-\mathbf{\Gamma}-1, \mathbf{\Gamma}+1)\rfloor$ and $\beta_k^i(0) = \lfloor U(-\mathbf{\Gamma}-1, \mathbf{\Gamma}+1)\rfloor$ for $\forall i \in [1, N_{l+1}], \forall k \in [1, N_l]$. Then since the spatial bias elements are integers now, (3) can be used instead

of (7) for FP. Steps 2.a.iv and 2.a.vii are identical for both approaches. The main difference in BP is step 2.a.v where (31) should be used instead of (39) for the delta error computed for the connection to the ith neuron at layer $l+1$ and there is no need for reverse interpolation, hence (45) is simply omitted. Obviously for post-processing (PP) at step 2.a.vii, and Update at step 2.a.xi, (47), (48), and (49) are, too, omitted since there is no gradient computation for the spatial bias pair, α_k^i, and β_k^i, as they are fixed as integers during step 1. Since the spatial bias elements are integers now, (3) can be used instead of (7) for FP. Steps 2.a.iv and 2.a.vii are identical for both approaches. The main difference in BP is step 2.a.v where (31) should be used instead of (39) for the delta error computed for the connection to the ith neuron at layer $l+1$ and there is no need for reverse interpolation, hence (45) is simply omitted. Obviously for post-processing (PP) at step 2.a.vii, and update at step 2.a.xi, (47), (48), and (49) are,

$$w_{ik}^l\langle q\rangle(t+1) = w_{ik}^l\langle q\rangle(t) - \varepsilon(t)\frac{\partial E}{\partial w_{ik}^l}\langle q\rangle, \qquad q \in [1, Q], i \in [1, N_l], k \in [1, N_{l-1}]$$
$$b_i^l(t+1) = b_i^l(t) - \varepsilon(t)\frac{\partial E}{\partial b_i^l}, \qquad\qquad i \in [1, N_l]$$
$$\alpha_k^i(t+1) = \alpha_k^i(t) - \gamma(t)\Delta\alpha_k^i, \qquad\qquad i \in [1, N_l], k \in [1, N_{l-1}]$$
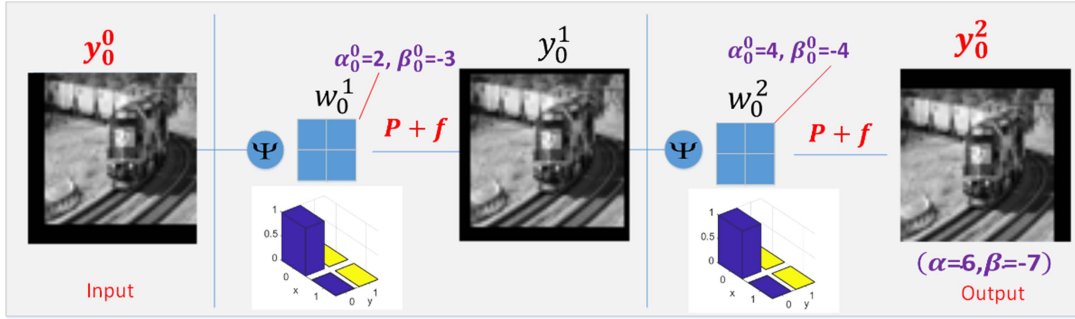$$\beta_k^i(t+1) = \beta_k^i(t) - \gamma(t)\Delta\beta_k^i, \qquad\qquad i \in [1, N_l], k \in [1, N_{l-1}] \qquad (49)$$

Fig. 12. Sample Self-ONN with a single (hidden) super neuron over the toy problem. The perfect regression of the target is illustrated (SNR = ∞) for an ideal case.
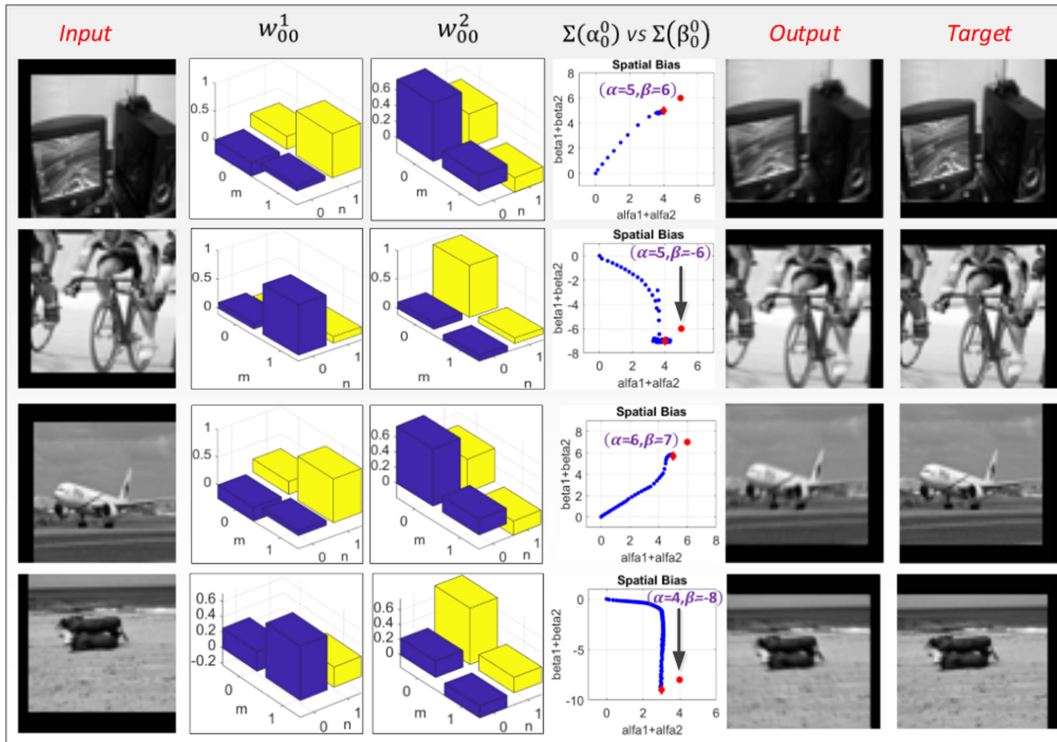

Fig. 13. Four "proof of concept" verification experiments where the target images are created with random shifts are shown at each row. The 2nd and 3rd columns show bar plots of the kernels and the 4th column shows the plots of the cumulative bias elements (hidden and output super neurons) in each BP iteration with a blue point. The cumulative, $\sum(\alpha_0^0, \beta_0^0)$, and target shifts (α, β) are shown with the red circles on the plot. The BP is stopped at the iteration when the SNR is reached to 35 dB.

too, omitted since there is no gradient computation for the spatial bias pair, α_k^i, and β_k^i, as they are fixed as integers.

D. Proof of Concept

In order to validate the super neurons' ability to learn the true shift using BP-optimization of the spatial bias pair, a Self-ONN network with one hidden layer and a single neuron is trained over a toy problem where the network aims to learn to regress (transform) an input image to an output image, which is the shifted version of the input image by $(\alpha, \beta) \in \mathbb{Z}[-\Gamma, \Gamma]$, i.e., $y_0^2(m, n) = y_0^0(m + \alpha, n + \beta)$. Therefore, with this setup, we can now validate whether the super neurons with the non-localized kernels are able to learn the true shift collectively

during the BP training, and if so, whether the Self-ONN is able to generate the target (shifted) image perfectly well. Fig. 12 illustrates this over a sample image where the output image is the shifted version of the input image with $(\alpha = 6, \beta = -7)$ pixels. In this ideal regression case, the cumulative bias shift of the two super neurons in x- and y-directions indeed is equal to the target shift, i.e., $\sum(\alpha_0^0, \beta_0^0) = (6, -7)$ where the 1st order learned kernels are impulses, i.e., $w_{00}^1(r, t) = w_{00}^2(r, t) = \delta(r, t)$. Since this is a validation experiment where the cumulative bias convergence is compared against the actual shift, we keep $Q = 1$ to avoid the higher-order (nonlinear) operations and thus to achieve a perfect reconstruction by linear convolution.

The ideal regression case illustrated in Fig. 12 shows the configuration of *only* one of the possible BP-optimized super neurons in a Self-ONN. Another ideal output can also be achieved,

for instance, when $\sum(\alpha_0^0, \beta_0^0) = (5, -8)$ and the 1st order learned kernels are impulses are $w_{00}^1(r, t) = \delta(r, t)$, $w_{00}^2(r, t) = \delta(r-1, t-1)$ *or* $w_{00}^1(r, t) = \delta(r-1, t-1)$, $w_{00}^2(r, t) = \delta(r, t)$, or even, $w_{00}^1(r, t) = \delta(r-1, t)$, $w_{00}^2(r, t) = \delta(r, t-1)$. In this case, the cumulative bias shifts are converged to the close vicinity of the actual shift (with an offset of (11) pixels) while the kernels of the hidden and output super neurons with the shifted impulses accommodate for the offset left out by the biases.

Over the 40 input images randomly selected in the Pascal dataset, we created the target images with random shifts by $\Gamma = 8$ pixels. Fig. 13 shows four examples of this verification experiment where the input, output, and target images are shown in the first and the last two columns, respectively. The 2nd and 3rd columns show bar plots of the kernels and the 4th column shows the plots of the cumulative bias elements (hidden and output super neurons) in each BP iteration with a blue point. The cumulative, $\sum(\alpha_0^0, \beta_0^0)$, and target shifts, (α, β), are shown with the red circles on the plot. The spatial bias pair is initially set as, $(\alpha_0^0, \beta_0^0) = (0, 0)$. The BP iterations are stopped when the regression SNR reaches 35 dB. In all experiments including the four shown in the figure, the cumulative bias converged to the close vicinity of the actual shift and we observed that offsets such as $(0, 1), (1, 0)$ or $(1, 1)$ pixels are accommodated by the 2x2 kernels with shifted impulses. This is also visible in the figure where the offset is $(1, 1)$ pixels. In the experiments shown in the first and third rows, the kernel functions in the 1st and 2nd (output) layers are: $w_{00}^1(r, t) \cong \delta(r-1, t-1)$ and $w_{00}^2(r, t) \cong \delta(r, t)$ while the one in the fourth row, they are: $w_{00}^1(r, t) \cong \delta(r, t-1)$ and $w_{00}^2(r, t) \cong \delta(r-1, t)$. Since the early stopping criterion is set as SNR = 35 dB, the kernels are only approximating the (shifted) impulses. A common observation in all experiments is that the spatial bias elements usually converged during the early stages of the BP, i.e., within around 20–50 iterations while the optimization of the kernels was initiated afterwards.

In brief, such a "Proof of Concept" demonstration shows a unique capability of the super neurons in a regression problem, i.e., only with a single hidden neuron, from an arbitrary input image, the network can perfectly regress the output image which is the shifted version of the input image. Such an image transformation is not possible for any conventional CNNs, or even Self-ONNs with generative neurons, unless the effective receptive field is expanded by using sufficiently deep and complex networks.

REFERENCES

[1] S. Kiranyaz, T. Ince, A. Iosifidis, and M. Gabbouj, "Generalized model of biological neural networks: Progressive operational perceptrons," in *Proc. Int. Joint Conf. Neural Netw.*, 2017, pp. 2477–2485.

[2] S. Kiranyaz, T. Ince, A. Iosifidis, and M. Gabbouj, "Progressive operational perceptrons," *Neurocomputing*, vol. 224, pp. 142–154, 2017.

[3] D. T. Tran, S. Kiranyaz, M. Gabbouj, and A. Iosifidis, "Progressive operational perceptron with memory," *Neurocomputing*, vol. 379, pp. 172–181, 2019, doi: 10.1016/j.neucom.2019.10.079.

[4] D. T. Tran, S. Kiranyaz, M. Gabbouj, and A. Iosifidis, "Heterogeneous multilayer generalized operational perceptron," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 31, no. 3, pp. 710–724, Mar. 2020.

[5] D. T. Tran, S. Kiranyaz, M. Gabbouj, and A. Iosifidis, "Knowledge transfer for face verification using heterogeneous generalized operational perceptrons," in *Proc. IEEE Int. Conf. Image Process.*, 2019, pp. 1168–1172.

[6] S. Kiranyaz, T. Ince, A. Iosifidis, and M. Gabbouj, "Operational neural networks," *Neural Comput. Appl.*, vol. 32, pp. 6645–6668, Mar. 2020, doi: 10.1007/s00521-020-04780-3.

[7] S. Kiranyaz, J. Malik, H. B. Abdallah, T. Ince, A. Iosifidis, and M. Gabbouj, "Exploiting heterogeneity in operational neural networks by synaptic plasticity," *Neural Comput. Appl.*, vol. 33, pp. 7997–8015, Jan. 2021, doi: 10.1007/s00521-020-05543-w.

[8] J. Malik, S. Kiranyaz, and M. Gabbouj, "FastONN–Python based open-source GPU implementation for operational neural networks," 2020, *arXiv:2006.02267*.

[9] J. Malik, S. Kiranyaz, and M. Gabbouj, "Operational vs convolutional neural networks for image denoising," 2020, *arXiv:2009.00612*.

[10] S. Kiranyaz, J. Malik, H. B. Abdallah, T. Ince, A. Iosifidis, and M. Gabbouj, "Self-organized operational neural networks with generative neurons," *Neural Netw.*, vol. 140, pp. 294–308, Mar. 2021, doi: 10.1016/j.neunet.2021.02.028.

[11] J. Malik, S. Kiranyaz, and M. Gabbouj, "Self-organized operational neural networks for severe image restoration problems," *Neural Netw.*, vol. 135, pp. 201–211, Jan. 2021, doi: 10.1016/j.neunet.2020.12.014.

[12] O. C. Devecioglu, J. Malik, T. Ince, S. Kiranyaz, E. Atalay, and M. Gabbouj, "Real-time glaucoma detection from digital fundus images using Self-ONNs," *IEEE Access*, vol. 9, pp. 140031–140041, 2021, doi: 10.1109/ACCESS.2021.3118102.

[13] T. Ince et al., "Early bearing fault diagnosis of rotating machinery by 1D self-organized operational neural networks," *IEEE Access*, vol. 9, pp. 139260–139270, 2021, doi: 10.1109/ACCESS.2021.3117603.

[14] S. Kiranyaz et al., "Blind ECG restoration by operational cycle-GANs," *IEEE Trans. Biomed. Eng.*, vol. 69, no. 12, pp. 3572–3581, Dec. 2022, doi: 10.1109/TBME.2022.3172125.

[15] M. Uzair, S. Kiranyaz, and M. Gabbouj, "Global ECG classification by self-operational neural networks with feature injection," *IEEE Trans. Biomed. Eng.*, vol. 70, no. 1, pp. 205–215, Jan. 2023, doi: 10.1109/TBME.2022.3187874.

[16] M. Gabbouj et al., "Robust peak detection for holter ECGs by self-organized operational neural networks," *IEEE Trans. Neural Netw. Learn. Syst.*, early access, pp. 1–12, Mar. 28, 2022, doi: 10.1109/TNNLS.2022.3158867.

[17] A. Degerli, S. Kiranyaz, M. E. H. Chowdhury, and M. Gabbouj, "OSeg-Net: Operational segmentation network for COVID-19 detection using chest X-ray images," in *Proc. IEEE Int. Conf. Image Process.*, 2022, pp. 2306–2310.

[18] M. Ahishali, S. Kiranyaz, I. Ahmad, and M. Gabbouj, "SRL-SOA: Self-representation learning with sparse 1D-operational autoencoder for hyperspectral image band selection," in *Proc. IEEE Int. Conf. Image Process.*, 2022, pp. 2296–2300.

[19] A. Rahman et al., "Robust biometric system using session invariant multimodal EEG and keystroke dynamics by the ensemble of Self-ONNs," *Comput. Biol. Med.*, vol. 142, Mar. 2022, Art. no. 105238, doi: 10.1016/j.compbiomed.2022.105238.

[20] J. Malik, O. C. Devecioglu, S. Kiranyaz, T. Ince, and M. Gabbouj, "Real-time patient-specific ECG classification by 1D self-operational neural networks," *IEEE Trans. Biomed. Eng.*, vol. 69, no. 5, pp. 1788–1801, May 2022, doi: 10.1109/TBME.2021.3135622.

[21] J. Malik, S. Kiranyaz, and M. Gabbouj, "BM3D vs 2-layer ONN," in *Proc. IEEE Int. Conf. Image Process.*, 2021, pp. 1994–1998.

[22] J. Dai et al., "Deformable convolutional networks," in *Proc. IEEE Int. Conf. Comput. Vis.*, 2017, pp. 764–773, doi: 10.1109/ICCV.2017.89.[21].

[23] X. Zhu, H. Hu, S. Lin, and J. Dai, "Deformable ConvNets v2: More deformable, better results," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2018, pp. 9300–9308.

[24] A. Chakrabarti, "A neural approach to blind motion deblurring," in *Proc. 14th Eur. Conf. Comput. Vis.*, 2016, pp. 221–235.

[25] D. Gong et al., "From motion blur to motion flow: A deep learning solution for removing heterogeneous motion blur," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2017, pp. 2319–2328.

[26] M. Noroozi, P. Chandramouli, and P. Favaro, "Motion deblurring in the wild," in *Proc. 39th German Conf. Pattern Recognit.*, 2017, pp. 65–77.

[27] C. J. Schuler, M. Hirsch, S. Harmeling, and B. Sch ̈olkopf, "Learning to deblur," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 38, no. 7, pp. 1439–1451, Jul. 2016.

[28] J. Sun, W. Cao, Z. Xu, and J. Ponce, "Learning a convolutional neural network for non-uniform motion blur removal," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2015, pp. 769–777.

[29] U. Schmidt, C. Rother, S. Nowozin, J. Jancsary, and S. Roth, "Discriminative non-blind deblurring," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2013, pp. 604–611.

[30] C. J. Schuler, H. C. Burger, S. Harmeling, and B. Scholkopf, "A machine learning approach for non-blind image deconvolution," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2013, pp. 1067–1074.

[31] O. Kupyn, V. Budzan, M. Mykhailych, D. Mishkin, and J. Matas, "DeblurGAN: Blind motion deblurring using conditional adversarial networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2018, pp. 8183–8192.

[32] S. Nah, T. H. Kim, and K. M. Lee, "Deep multi-scale convolutional neural network for dynamic scene deblurring," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2017, pp. 3883–3891.

[33] S. Ramakrishnan, S. Pachori, A. Gangopadhyay, and S. Raman, "Deep generative filter for motion deblurring," in *Proc. IEEE Int. Conf. Comput. Vis. Workshops*, 2017, pp. 2993–3000.

[34] X. Tao, H. Gao, X. Shen, J. Wang, and J. Jia, "Scale-recurrent network for deep image deblurring," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2018, pp. 8174–8182.

[35] J. Zhang et al., "Dynamic scene deblurring using spatially variant recurrent neural networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2018, pp. 2521–2529.

[36] A. Odena, V. Dumoulin, and C. Olah, "Deconvolution and checkerboard artifacts," *Distill*, vol. 1, 2016, Art. no. e3. [Online]. Available: http://distill.pub/2016/deconv-checkerboard

[37] S. Sahu, M. K. Lenka, and P. K. Sa, "Blind deblurring using deep learning: A survey," 2019, *arXiv:1907.10128*.

[38] W. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *Bull. Math. Biophys.*, vol. 5, pp. 115–133, 1943.

[39] N. Qian, "On the momentum term in gradient descent learning algorithms," *Neural Netw.*, vol. 12, pp. 145–151, 1999, doi: 10.1016/S0893-6080(98)00116-6.

[40] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization," in *Proc. 23rd Conf. Learn. Theory*, 2010, pp. 257–269.

[41] T. Tieleman and G. Hinton, "Lecture 6.5 - RMSProp, neural networks for machine learning | coursera," (n.d.).

[42] K. Diederik and J. L. Ba, "ADAM: A method for stochastic optimization," in *Proc. Amer. Inst. Phys. Conf.*, 2014, pp. 58–62, doi: 10.1063/1.4902458.

[43] S. Ruder, "An overview of gradient descent optimization algorithms," 2016.

[44] V. Jain and E. Learned-Miller, "FDDB: A benchmark for face detection in unconstrained settings," Univ. Massachusetts, Boston, MA, USA, Tech. Rep. UM-CS2010-009, 2010.

[45] P. Isola, J. Y. Zhu, T. Zhou, and A. A. Efros, "Image-to-image translation with conditional adversarial networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2017, pp. 1125–1134.

[46] J. Y. Zhu, T. Park, P. Isola, and A. A. Efros, "Unpaired image-to-image translation using cycle-consistent adversarial networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2017, pp. 2223–2232.

[47] V. Jain and S. Seung, "Natural image denoising with convolutional networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2009, pp. 769–776.

[48] S. Lefkimmiatis, "Non-local color image denoising with convolutional neural networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2017, pp. 3587–3596.

[49] D. Yang and J. Sun, "BM3D-Net: A convolutional neural network for transform-domain collaborative filtering," *IEEE Signal Process. Lett.*, vol. 25, no. 1, pp. 55–59, Jan. 2018, doi: 10.1109/LSP.2017.2768660.

[50] Y. Chen, W. Yu, and T. Pock, "On learning optimized reaction diffusion processes for effective image restoration," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2015, pp. 5261–5269.

[51] P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, and P.-A. Manzagol, "Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion," *J. Mach. Learn. Res.*, vol. 11, pp. 3371–3408, 2010.

[52] S. Kiranyaz, M.-A. Waris, I. Ahmad, R. Hamila, and M. Gabbouj, "Face segmentation in thumbnail images by data-adaptive convolutional segmentation networks," in *Proc. Int. Conf. Image Process.*, 2016, pp. 2306–2310.

[53] Y. Li, H. Qi, J. Dai, X. Ji, and Y. Wei, "Fully convolutional instance aware semantic segmentation," in *Proc. Comput. Vis. Pattern Recognit.*, 2017, pp. 4438–4446.

[54] T. Lin et al., "Microsoft COCO: Common objects in context," in *Proc. Eur. Conf. Comput. Vis.*, 2014, pp. 740–755.

[55] H. Zhu, F. Meng, J. Cai, and S. Lu, "Beyond pixels: A comprehensive survey from bottom-up to semantic image segmentation and cosegmentation," *J. Vis. Commun. Image Representation*, vol. 34, pp. 12–27, 2016.

[56] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," in *Proc. IEEE Conf. Comput. Vis. Pattern Rec.*, 2015, pp. 3431–3440.

[57] O. Ronneberger, P. Fischer, and T. Brox, "UNet: Convolutional networks for biomedical image segmentation," in *Proc. Int. Conf. Med. Image Comput. Comput.-Assist. Interv.*, 2015, pp. 234–241.

[58] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2015, pp. 3431–3440, doi: 10.1109/CVPR.2015.7298965.

[59] E. Learned-Miller, G. B. Huang, A. RoyChowdhury, H. Li, and G. Hua, "Labeled faces in the wild: A survey," in *Advances in Face Detection and Facial Image Analysis*. Berlin, Germany: Springer, 2016, pp. 189–248.

[60] F. Yu and V. Koltun, "Multi-scale context aggregation by dilated convolutions," 2015, *arXiv. 1511.07122*.

[61] P. Luc, C. Couprie, S. Chintala, and J. Verbeek, "Semantic segmentation using adversarial networks," 2016, *arXiv. 1611.08408*.

[62] Self-ONNs, 2021. [Online]. Available: http://selfonn.net/

[63] A. Abdelrahman, S. Lin, and M. S. Brown, "A high-quality denoising dataset for smartphone cameras," in *Proc. IEEE Comput. Vis. Pattern Recognit.*, 2018, pp. 1692–1700.

[64] "Torch.nn.functional.grid_sample PyTorch 1.9.0 documentation," 2018. Accessed: Aug. 24, 2021. [Online]. Available: https://pytorch.org/docs/stable/generated/torch.nn.functional.grid_sample.html#torch.nn.functional.grid_sample

Serkan Kiranyaz (Senior Member, IEEE) is currently a Professor with Qatar University, Doha, Qatar. He authored or coauthored two books, seven book chapters, ten patents/applications, more than 100 journal articles in several IEEE Transactions and other high impact journals, and more than 120 papers in international conferences. His principal research field is machine learning and signal processing. He made significant contributions on bio-signal analysis, classification and segmentation, computer vision with applications to recognition, classification, multimedia retrieval, evolving systems and evolutionary machine learning, swarm intelligence, and evolutionary optimization.

Junaid Malik received the bachelor's degree in electrical engineering from NUST, Islamabad, Pakistan, in 2013, and the M.Sc. degree in information technology from the Tampere University of Technology, Tampere, Finland, in 2017. He is also with SAMI Research Group as a Doctoral Researcher. His research interests include object segmentation, image restoration, and operational neural networks.

Mehmet Yamac received the B.S. degree in electrical and electronics engineering from Anadolu University, Eskisehir, Turkey, in 2009, and the M.S. degree in electrical and electronics engineering from Bogaziçi University, Istanbul, Turkey, in 2014. He is currently working toward the Ph.D. degree with the Department of Computing Sciences, Tampere University, Tampere, Finland. He was a Research and Teaching Assistant with Bogazici University from 2012 to 2017 and a Researcher with Tampere University from 2017 to 2020. He is currently a Senior Researcher with Huawei Technologies Oy, Tampere. He has coauthored more than 35 papers, two of them nominated for the Best (or Student Best) Paper Award in EUVIP 2018 and EUSIPCO 2019. His research interests include computer and machine vision, machine learning, and compressive sensing.

Mert Duman received the B.S. degree from Bilkent University, Ankara, Turkey, in 2020. He is working toward the M.Sc. degree in machine learning with Tampere University, Tampere, Finland. He is currently a Researcher with the Signal Analysis and Machine Intelligence Group, Tampere University. His research interests include generative models, image restoration, artificial neuron design, anomaly detection, zero-shot learning, and data analysis.

Ilke Adalioglu received the B.S. degree in electrical and Electronics engineering from Bilkent University, Ankara, Turkey, in 2021. She is working toward the M.Sc. degree in signal processing and machine learning with Tampere University, Tampere, Finland. She is currently a Researcher with the Signal Analysis and Machine Intelligence Research Group. Her research interests include machine learning, signal processing, image analysis, and processing of biomedical data.

Esin Guldogan received the M.S. and Ph.D. degrees from the Signal Processing Department, Tampere University of Technology, Tampere, Finland, in 2003 and 2009, respectively. Her research interests include machine learning, computer vision, imaging algorithms, and 3D image processing.

Turker Ince received the B.S. degree in electrical engineering from the Bilkent University, Ankara, Turkey, in 1994, the M.S. degree in electrical engineering from Middle East Technical University, Ankara, Turkey, in 1996, and the Ph.D. degree in electrical engineering from the University of Massachusetts, Amherst (UMass- Amherst), in 2001. From 1996 to 2001, he was a Research Assistant with the Microwave Remote Sensing Laboratory, UMass-Amherst. He was a Design Engineer with Aware, Inc., Boston, from 2001 to 2004, and Texas Instruments, Inc., Dallas, from 2004 to 2006. In 2006, he joined the Faculty of Engineering with the Izmir University of Economics, Turkey, where he is currently a Professor with Electrical & Electronics Engineering Department. His research interests include machine learning, signal processing, biomedical engineering, predictive analytics, and remote sensing.

Moncef Gabbouj (Fellow, IEEE) is currently a Professor with the Department of Computing Sciences, Tampere University, Tampere, Finland. He was an Academy of Finland Professor. His research interests include Big Data analytics, multimedia analysis, artificial intelligence, machine learning, pattern recognition, nonlinear signal processing, video processing, and coding. Dr. Gabbouj is a Fellow of the Asia-Pacific Artificial Intelligence Association. He is a Member of the Academia Europaea, Finnish Academy of Science and Letters, and Finnish Academy of Engineering Sciences.