APPENDICES

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 Eq. (12). Using the chain rule, the delta error of the output pixel, $y_k^l(m,n)$, can therefore, be expressed as in Eq. (13) in the generic form of pool, P_i^{l+1}, and composite nodal operator function, $\boldsymbol{\Psi}$, of each operational neuron $i \in [1,..,N_{l+1}]$ in the next layer. In Eq. (13), note that the first term, $\frac{\partial x_i^{l+1}(m-r,n-t)}{\partial P_i^{l+1}[..,\boldsymbol{\Psi}(y_k^l(m,n),\boldsymbol{w_{ik}^{l+1}}(r,t)),..]} = 1$.

$$
\begin{aligned}
x_i^{l+1}(m-1,n-1) =& \ldots + P_i^{l+1}\left[\boldsymbol{\Psi}\left(y_k^l(m-1,n-1),\boldsymbol{w_{ik}^{l+1}}(\mathbf{0},\mathbf{0})\right),\ldots,\boldsymbol{\Psi}\left(y_k^l(m,n),\boldsymbol{w_{ik}^{l+1}}(\mathbf{1},\mathbf{1})\right)\right]+\ldots \\
x_i^{l+1}(m-1,n) =& \ldots + P_i^{l+1}\left[\boldsymbol{\Psi}\left(y_k^l(m-1,n),\boldsymbol{w_{ik}^{l+1}}(\mathbf{0},\mathbf{0})\right),\ldots,\boldsymbol{\Psi}\left(y_k^l(m,n),\boldsymbol{w_{ik}^{l+1}}(\mathbf{1},\mathbf{0})\right),\ldots\right]+\ldots \\
\boldsymbol{x_i^{l+1}(m,n)} =& \ldots + P_i^{l+1}\left[\boldsymbol{\Psi}\left(y_k^l(m,n),\boldsymbol{w_{ik}^{l+1}}(\mathbf{0},\mathbf{0})\right),..,\boldsymbol{\Psi}\left(y_k^l(m+r,n+t),\boldsymbol{w_{ik}^{l+1}}(\mathbf{r},\mathbf{t}),\right)\ldots\right]+\ldots \\
&\ldots\ldots \\
\therefore\ & x_i^{l+1}(m-r,n-t)\Big|_{(1,1)}^{(M-1,N-1)} = b_i^{l+1} + \sum_{k=1}^{N_1} P_i^{l+1}\left[\ldots,\boldsymbol{\Psi}\left(y_k^l(m,n),\boldsymbol{w_{ik}^{l+1}}(\boldsymbol{r},\boldsymbol{t})\right),\ldots\right]
\end{aligned}
\tag{12}
$$

$$
\therefore\ \frac{\partial E}{\partial y_k^l}(m,n)\Big|_{(0,0)}^{(M-1,N-1)} = \Delta y_k^l(m,n) =
$$

$$
\sum_{i=1}^{N_{l+1}}\left(\sum_{r=0}^{\mathrm{Kx}-1}\sum_{t=0}^{\mathrm{Ky}-1}\frac{\partial E}{\partial x_i^{l+1}(m-r,n-t)}\times\frac{\partial x_i^{l+1}(m-r,n-t)}{\partial P_i^{l+1}\left[..,\boldsymbol{\Psi}\left(y_k^l(m,n),\boldsymbol{w_{ik}^{l+1}}(\boldsymbol{r},\boldsymbol{t})\right),..\right]}\times\frac{\partial P_i^{l+1}\left[..,\boldsymbol{\Psi}\left(y_k^l(m,n),\boldsymbol{w_{ik}^{l+1}}(\boldsymbol{r},\boldsymbol{t})\right),..\right]}{\partial\boldsymbol{\Psi}\left(y_k^l(m,n),\boldsymbol{w_{ik}^{l+1}}(\boldsymbol{r},\boldsymbol{t})\right)}\times\frac{\partial\boldsymbol{\Psi}\left(y_k^l(m,n),\boldsymbol{w_{ik}^{l+1}}(\mathbf{r},\mathbf{t})\right)}{\partial y_k^l(m,n)}\right)
\tag{13}
$$

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

$$
\sum_{i=1}^{N_{l+1}}\left(\sum_{r=0}^{\mathrm{Kx}-1}\sum_{t=0}^{\mathrm{Ky}-1}\Delta_{\mathrm{i}}^{l+1}(m-r,n-t)\times\nabla_{\boldsymbol{\Psi}}\mathrm{P}_i^{l+1}(m,n,r,t)\times\nabla_y\boldsymbol{\Psi}(m,n,r,t)\right)
\tag{14}
$$

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

$$
\Delta y_k^l = \sum_{i=1}^{N_{l+1}} Conv2Dvar\left\{\Delta_{\mathrm{i}}^{l+1},\nabla_y\mathrm{P}_i^{l+1}(m,n,r,t)\right\}
$$

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

$$
\nabla_y\boldsymbol{\Psi}(m,n,r,t) = w_{ik}^{l+1}(\mathrm{r},\mathrm{t},1) + 2w_{ik}^{l+1}(\mathrm{r},\mathrm{t},2)y_k^l(m,n) + \cdots + Qw_{ik}^{l+1}(\mathrm{r},\mathrm{t},\mathrm{Q})y_k^l(m,n)^{Q-1}
\tag{15}
$$

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

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

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: $uy_k^l = \underset{ssx,ssy}{\mathrm{up}}(y_k^l)$. Then Eq. (16) can be modified, as follows:

$$
\begin{aligned}
\Delta_k^l = \frac{\partial E}{\partial x_k^l} = \frac{\partial E}{\partial y_k^l}\frac{\partial y_k^l}{\partial x_k^l} = \frac{\partial E}{\partial y_k^l}\frac{\partial y_k^l}{\partial uy_k^l}\frac{\partial uy_k^l}{\partial x_k^l} \\
= \underset{ssx,ssy}{\mathrm{up}}(\Delta y_k^l)\beta\, f'(x_k^l)
\end{aligned}
\tag{17}
$$

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: $dy_k^l = \underset{usx,usy}{\mathrm{down}}(y_k^l)$. Then Eq. (17) can be updated as follows:

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

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, $x_i^{l+1}(m,n)$:

$$x_i^{l+1}(m,n)\Big|_{(1,1)}^{(M-1,N-1)} = b_i^{l+1} + \sum_{i=1}^{N_{l-1}} P_i^{l+1}\begin{bmatrix}\boldsymbol{\Psi}\left(y_k^l(m,n),\boldsymbol{w_{ik}^{l+1}}(\mathbf{0},\mathbf{0})\right),\ldots,\\ \boldsymbol{\Psi}\left(y_k^l(m+r,n+t),\boldsymbol{w_{ik}^{l+1}}(\mathbf{r},\mathbf{t})\right)\ldots)\end{bmatrix} \quad (19)$$

where the q^{th} 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 Eq. (20). A close look to Eq. (20) reveals that, $\frac{\partial \boldsymbol{\Psi}\left(y_k^l(m+r,n+t),\boldsymbol{w_{ik}^{l+1}}(\boldsymbol{r},\boldsymbol{t})\right)}{\partial w_{ik}^{l+1}(r,t,q)} = y_k^l(m+r,n+t)^q$, which then simplifies to Eq. (21) Note that in this equation, the first term, $\Delta_i^{l+1}(m,n)$, 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 $(\mathrm{M}-\mathrm{Kx}+1)x(N-\mathrm{Ky}+1)$, 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, $\boldsymbol{w_{ik}^{l+1}}(\boldsymbol{r},\boldsymbol{t})$.

$$\frac{\partial E}{\partial w_{ik}^{l+1}}(r,t,q)\Big|_{(0,0,1)}^{(\mathrm{Kx}-1,\mathrm{Ky}-1,\mathrm{Q})} = \frac{\partial E}{\partial x_i^{l+1}(m,n)} \times \frac{\partial x_i^{l+1}(m,n)}{\partial P_i^{l+1}\left[\boldsymbol{\Psi}\left(y_k^l(m,n),\boldsymbol{w_{ik}^{l+1}}(\mathbf{0},\mathbf{0})\right),..,\boldsymbol{\Psi}\left(y_k^l(m+r,n+t),\boldsymbol{w_{ik}^{l+1}}(\mathbf{r},\mathbf{t})\right)\ldots\right]} \times \sum_{m=0}^{M-\mathrm{Kx}+1}\sum_{n=0}^{N-\mathrm{Ky}+1} \frac{\partial P_i^{l+1}\left[\boldsymbol{\Psi}\left(y_k^l(m,n),\boldsymbol{w_{ik}^{l+1}}(\mathbf{0},\mathbf{0})\right),\ldots,\boldsymbol{\Psi}\left(y_k^l(m+r,n+t),\boldsymbol{w_{ik}^{l+1}}(\mathbf{r},\mathbf{t})\right)\ldots\right]}{\partial \boldsymbol{\Psi}\left(y_k^l(m+r,n+t),\boldsymbol{w_{ik}^{l+1}}(\boldsymbol{r},\boldsymbol{t})\right)} \times \frac{\partial \boldsymbol{\Psi}\left(y_k^l(m+r,n+t),\boldsymbol{w_{ik}^{l+1}}(\boldsymbol{r},\boldsymbol{t})\right)}{\partial w_{ik}^{l+1}(r,t,q)} \quad (20)$$

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

$$\therefore \frac{\partial E}{\partial w_{ik}^{l+1}}(r,t,q)\Big|_{(0,0,1)}^{(\mathrm{Kx}-1,\mathrm{Ky}-1,\mathrm{Q})} = \sum_{m=0}^{M-\mathrm{Kx}}\sum_{n=0}^{N-\mathrm{Ky}} \Delta_i^{l+1}(m,n) \times \nabla_{\boldsymbol{\Psi}} \mathrm{P}_i^{l+1}(m+r,n+t,r,t) \times y_k^l(m+r,n+t)^q \quad (21)$$

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

$$\frac{\partial E}{\partial w_{ik}^{l+1}}(r,t,q)\Big|_{(0,0,1)}^{(\mathrm{Kx}-1,\mathrm{Ky}-1,\mathrm{Q})} = \sum_{m=0}^{M-\mathrm{Kx}}\sum_{n=0}^{N-\mathrm{Ky}} \Delta_i^{l+1}(m,n) \times y_k^l(m+r,n+t)^q \quad (22)$$

$$\therefore \frac{\partial E}{\partial w_{ik}^{l+1}}\langle q\rangle = conv2D\left(\Delta_i^{l+1},\left(\mathrm{y_k^l}\right)^{\mathrm{q}},'NoZeroPad'\right)$$

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

In Eq. (21) there is no need to register a 4D matrix for $\nabla_w \boldsymbol{\Psi} = y_k^l(m+r,n+t)^q$ since it can directly be computed from the outputs of the neurons. Moreover, when the pool operator is the *sum*, then $\nabla_{\boldsymbol{\Psi}} \mathrm{P}_i^{l+1}(m,n,r,t) = 1$ and Eq. (21) will simplify to Eq. (22) where $\frac{\partial E}{\partial w_{ik}^{l+1}}\langle q\rangle$ is the q^{th} 2D sensitivity kernel, which contains the updates (SGD sensitivities) for the weights of the q^{th} order outputs in Maclaurin polynomial. Finally, the bias sensitivity expressed in Eq. (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 q^{th} 2D sub-kernel where q=1..Q and composed of *kernel elements,* $w_{ik}^{l+1}(\mathrm{r,t,q})$. 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, ε(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:

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

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

Algorithm 1: BP training for Self-ONNs with generative neurons

Input: Self-*ONN*, $Stopping\ Criteria\ (maxIter, minMSE)$

Output: Self-*ONN** = $BP(Self\text{-}ONN, maxIter, minMSE)$

1) **Initialize** network parameters randomly (i.e., ~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 $f'(x_k^l)$, $\nabla_y \Psi_{ki}^{l+1}$, $\nabla_{\Psi_{ki}} \mathrm{P}_i^{l+1}$ 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 Eqs. (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 Eqs. (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 Eq. (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 1st hidden layer. For the sake of simplicity, for an image $\mathbf{I}$ in the training dataset suppose that the error (loss) function is L2-loss or the Mean-Square-Error (MSE) error function, $E(I)$, is used can be expressed as,

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

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 Eq. (26).

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

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 Eq. (27)

$$\begin{gathered}
x_i^{l+1}(m-1,n-1) = \ldots + \\
P_i^{l+1}\left[\boldsymbol{\Psi}\left(y_k^l(m+\alpha_k^i-1, n+\alpha_k^i-1), \boldsymbol{w_{ik}^{l+1}}(\mathbf{0},\mathbf{0})\right), \ldots, \boldsymbol{\Psi}\left(y_k^l(m+\alpha_k^i, n+\beta_k^i), \boldsymbol{w_{ik}^{l+1}}(\mathbf{1},\mathbf{1})\right)\right] + \ldots \\
x_i^{l+1}(m-1,n) = \ldots + \\
P_i^{l+1}\left[\boldsymbol{\Psi}\left(y_k^l(m+\alpha_k^i-1, n+\alpha_k^i), \boldsymbol{w_{ik}^{l+1}}(\mathbf{0},\mathbf{0})\right), \ldots, \boldsymbol{\Psi}\left(y_k^l(m+\alpha_k^i, n+\beta_k^i), \boldsymbol{w_{ik}^{l+1}}(\mathbf{1},\mathbf{0})\right), \ldots\right] + \ldots \\
\boldsymbol{x_i^{l+1}(m,n)} = \ldots + \\
P_i^{l+1}\left[\boldsymbol{\Psi}\left(y_k^l(m+\alpha_k^i, n+\beta_k^i), \boldsymbol{w_{ik}^{l+1}}(\mathbf{0},\mathbf{0})\right), .., \boldsymbol{\Psi}\left(y_k^l(m+\alpha_k^i+r, n+\alpha_k^i+t), \boldsymbol{w_{ik}^{l+1}}(\mathbf{r},\mathbf{t})\right)\ldots\right] + \ldots \\
\ldots \\
\therefore\ x_i^{l+1}(m-r,n-t)\Big|_{(1,1)}^{(M-1,N-1)} = b_i^{l+1} + \sum_{k=1}^{N_1} P_i^{l+1}\left[\ldots, \boldsymbol{\Psi}\left(y_k^l(m+\alpha_k^i, n+\beta_k^i), \boldsymbol{w_{ik}^{l+1}}(\boldsymbol{r},\boldsymbol{t})\right), \ldots\right]
\end{gathered} \tag{27}$$

$$\begin{gathered}
\therefore \frac{\partial E}{\partial y_k^l}(m+\alpha_k^i, n+\beta_k^i)\Big|_{(0,0)}^{(M-1,N-1)} = \Delta y_k^l(m+\alpha_k^i, n+\beta_k^i) = \\
\sum_{r=0}^{Kx-1}\sum_{t=0}^{Ky-1}\frac{\partial E}{\partial x_i^{l+1}(m-r,n-t)} \times \frac{\partial x_i^{l+1}(m-r,n-t)}{\partial P_i^{l+1}\left[..,\boldsymbol{\Psi}\left(y_k^l(m+\alpha_k^i, n+\beta_k^i), \boldsymbol{w_{ik}^{l+1}}(\boldsymbol{r},\boldsymbol{t})\right),..\right]} \times \\
\frac{\partial P_i^{l+1}\left[..,\boldsymbol{\Psi}\left(y_k^l(m+\alpha_k^i, n+\beta_k^i), \boldsymbol{w_{ik}^{l+1}}(\boldsymbol{r},\boldsymbol{t})\right),..\right]}{\partial \boldsymbol{\Psi}\left(y_k^l(m+\alpha_k^i, n+\beta_k^i), \boldsymbol{w_{ik}^{l+1}}(\boldsymbol{r},\boldsymbol{t})\right)} \times \frac{\partial \boldsymbol{\Psi}\left(y_k^l(m+\alpha_k^i, n+\beta_k^i), \boldsymbol{w_{ik}^{l+1}}(\mathbf{r},\mathbf{t})\right)}{\partial y_k^l(m+\alpha_k^i, n+\beta_k^i)}
\end{gathered} \tag{28}$$

$$\begin{gathered}
\Delta y_k^l(m+\alpha_k^i, n+\beta_k^i) = \\
\sum_{r=0}^{K_x-1}\sum_{t=0}^{K_y-1}\Delta_i^{l+1}(m-r,n-t) \times \nabla_{\boldsymbol{\Psi}} P_i^{l+1}(m+\alpha_k^i, n+\beta_k^i, r, t) \times \nabla_y \boldsymbol{\Psi}(m+\alpha_k^i, n+\beta_k^i, r, t)
\end{gathered} \tag{29}$$

where $\nabla_y \boldsymbol{\Psi}(m+\alpha_k^i, n+\beta_k^i, r, t)$ can be directly computed as,

$$\begin{gathered}
\nabla_y \boldsymbol{\Psi}(m+\alpha_k^i, n+\beta_k^i, r, t) = w_{ik}^{l+1}(\mathrm{r},\mathrm{t},1) + 2w_{ik}^{l+1}(\mathrm{r},\mathrm{t},2)y_k^l(m+\alpha_k^i, n+\beta_k^i) + \cdots \\
+Qw_{ik}^{l+1}(\mathrm{r},\mathrm{t},Q)y_k^l(m+\alpha_k^i, n+\beta_k^i)^{Q-1}
\end{gathered} \tag{30}$$

Now let $\nabla_y P_i^{l+1}(m+\alpha_k^i, n+\beta_k^i, r, t) = \nabla_{\boldsymbol{\Psi}} P_i^{l+1}(m+\alpha_k^i, n+\beta_k^i, r, t) \times \nabla_y \boldsymbol{\Psi}(m+\alpha_k^i, n+\beta_k^i, r, t)$. In this study, the *summation* is used as the pool operator for the sake of simplicity, i.e., $P_i^{l+1} = \Sigma$, so, $\nabla_{\boldsymbol{\Psi}} P_i^{l+1}(m,n,r,t) = 1$ and thus, $\nabla_y P_i^{l+1}(m+\alpha_k^i, n+\beta_k^i, r, t) = \nabla_y \boldsymbol{\Psi}(m+\alpha_k^i, n+\beta_k^i, r, t)$ then the delta (highlighted in red for clarity). So for the hidden operational layers, a close look at Eq. (27) will reveal the fact that the contributions of each pixel in the $(M+2\boldsymbol{\Gamma}) \times (N+2\boldsymbol{\Gamma})$ 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 Eq. (28) in the generic form of pool, P_i^{l+1}, and composite nodal operator function, $\boldsymbol{\Psi}$, of the i^{th} super neuron, $i \in [1, .., N_{l+1}]$. In Eq. (28), note that the first term, $\frac{\partial x_1^{l+1}(m-r,n-t)}{\partial P_i^{l+1}\left[..,\boldsymbol{\Psi}\left(y_k^l(m+\alpha_k^i, n+\beta_k^i), \boldsymbol{w_{ik}^{l+1}}(\boldsymbol{r},\boldsymbol{t})\right),..\right]} = 1$. Let the (shifted) 4D matrices $\nabla_{\boldsymbol{\Psi}} P_i^{l+1}(m+\alpha_k^i, n+\beta_k^i, r, t) = \frac{\partial P_i^{l+1}\left[..,\boldsymbol{\Psi}\left(y_k^l(m+\alpha_k^i, n+\beta_k^i), \boldsymbol{w_{ik}^{l+1}}(\boldsymbol{r},\boldsymbol{t})\right),..\right]}{\partial \boldsymbol{\Psi}\left(y_k^l(m+\alpha_k^i, n+\beta_k^i), \boldsymbol{w_{ik}^{l+1}}(\boldsymbol{r},\boldsymbol{t})\right)}$ and $\nabla_y \boldsymbol{\Psi}(m+\alpha_k^i, n+\beta_k^i, r, t) = \frac{\partial \boldsymbol{\Psi}\left(y_k^l(m+\alpha_k^i, n+\beta_k^i), \boldsymbol{w_{ik}^{l+1}}(\boldsymbol{r},\boldsymbol{t})\right)}{\partial y_k^l(m+\alpha_k^i, n+\beta_k^i)}$. Then, Eq. (28) simplifies to Eq. (29).

error computed for the connection to the i^{th} neuron at layer l+1 can be expressed as,

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

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.,

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

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:

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

When there is a pooling (down-sampling) by factors, *ssx,* and *ssy*, then the back-propagated delta-error by Eq. (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 Eq. (33) can be modified, as follows:

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

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: $dy_k^l = \underset{usx,usy}{\text{down}}(y_k^l)$. Then Eq. (33) can be updated as follows:

$$\Delta_k^l = \frac{\partial E}{\partial x_k^l} = \frac{\partial E}{\partial y_k^l}\frac{\partial y_k^l}{\partial x_k^l} = \frac{\partial E}{\partial y_k^l}\frac{\partial y_k^l}{\partial dy_k^l}\frac{\partial dy_k^l}{\partial x_k^l} = \underset{usx,usy}{\text{down}}(\Delta y_k^l)\beta^{-1} f'(x_k^l) \quad (35)$$

As for the computation of the sensitivities for kernel parameters, $\frac{\partial E}{\partial w_{ik}^{l+1}}(r,t,q)\Big|_{(0,0,1)}^{(\text{Kx}-1,\text{Ky}-1,\text{Q})}$, and bias, $\frac{\partial E}{\partial b_k^l}$, Eq. (3) indicates that the q^{th} element of the array, $\boldsymbol{w_{ik}^{l+1}}(\mathbf{r},\mathbf{t})$, contributes to all the pixels of $x_i^{l+1}(m,n)$. Once again by using the chain rule of partial derivatives, the sensitivities for kernel parameters can be expressed in Eq. (36). Since $\frac{\partial \boldsymbol{\Psi}\left(y_k^l(m+\alpha_k^i+r,n+\beta_k^i+t),\boldsymbol{w_{ik}^{l+1}}(\boldsymbol{r},\boldsymbol{t})\right)}{\partial w_{ik}^{l+1}(r,t,q)} = y_k^l(m+\alpha_k^i+r,n+\beta_k^i+t)^q$, $\text{P}_\text{i}^{\text{l+1}} = \Sigma$, and $\nabla_{\boldsymbol{\Psi}}\text{P}_i^{l+1}\left(m+\alpha_k^i+r,n+\beta_k^i+t\right) = 1$. then Eq. (36) simplifies to Eq. (37).

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

$$\frac{\partial E}{\partial w_{ik}^{l+1}}(r,t,q)\Big|_{(0,0,1)}^{(\text{Kx}-1,\text{Ky}-1,\text{Q})} = \sum_{m=0}^{M-\text{Kx}}\sum_{n=0}^{N-\text{Ky}}\Delta_\text{i}^{l+1}(m,n)\times\nabla_{\boldsymbol{\Psi}}\text{P}_i^{l+1}\left(m+\alpha_k^i+r,n+\beta_k^i+t\right)\times y_k^l(m+\alpha_k^i+r,n+\beta_k^i+t)^q$$
$$= \sum_{m=0}^{M-\text{Kx}}\sum_{n=0}^{N-\text{Ky}}\Delta_\text{i}^{l+1}(m,n)\times y_k^l\left(m+\alpha_k^i+r,n+\beta_k^i+t\right)^q \quad (37)$$
$$\therefore \frac{\partial E}{\partial w_{ik}^{l+1}}\langle q\rangle = conv2D\left(\Delta_i^{l+1},\left(\text{T}^{\left(\alpha_k^i,\beta_k^i\right)}\left(\text{y}_\text{k}^\text{l}\right)\right)^\text{q},'NoZeroPad'\right)$$

For the bias sensitivity, the chain rule yields:

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

C. BP for Non-localized Kernel Operations by the BP-optimized Bias

Recall that Eq. (6) allows us to compute the derivatives of the output map w.r.t the individual bias elements, as expressed in Eq. (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 Eq. (25). With $\vec{y}_k^l(m,\, n) = y_k^l\left(m + \alpha_k^i,\, n + \beta_k^i\right)$ Eq. (28) can be simplified as in Eq. (42) and with $\mathrm{P}_i^{l+1} = \Sigma$, it yields Eq. (43) where $\nabla_{\bar{y}}\mathrm{P}_i^{l+1}(m,n,r,t) = \nabla_{\boldsymbol{\Psi}}\mathrm{P}_i^{l+1}(m,n,r,t) \times \nabla_{\bar{y}}\boldsymbol{\Psi}(m,n,r,t) = \nabla_{\bar{y}}\boldsymbol{\Psi}(m,n,r,t)$ and $\nabla_{\bar{y}}\boldsymbol{\Psi}(m,n,r,t)$ can be directly computed as in Eq. (44). Finally, the delta error of $\vec{y}_k^l$ (from its contribution to $\boldsymbol{x_i^{l+1}}$ alone) can be computed as,

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

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.,

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

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, $\Delta y_k^l\left(m + \lfloor\alpha_k^i\rfloor, n + \lfloor\beta_k^i\rfloor\right)$ as illustrated in Figure 11. Once again using bilinear interpolation, $\Delta y_k^l\left(m_\alpha,\, n_\beta\right)$ can be computed as expressed in Eq. (45). 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 Eq. (40). Once on the integer grid, it is straightforward to compute Δy_k^l using Eq. (32).

After the (overall) Δy_k^l is computed, using Eq. (33) (or Eq. (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 k^{th} neuron at layer l is completed.

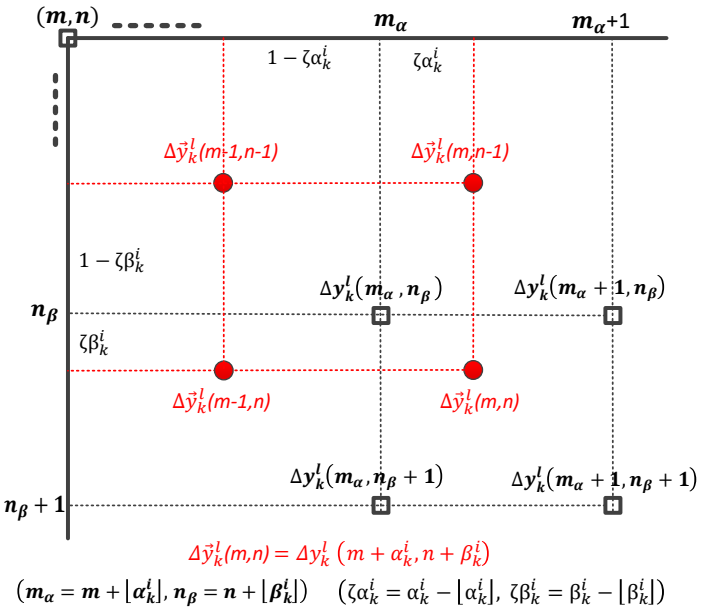

Figure 11: The reverse interpolation from the shifted delta error, $\Delta\vec{y}_k^l(m,\, n) = \Delta y_k^l\left(m + \alpha_k^i,\, n + \beta_k^i\right)$ by the bias, $\left(\alpha_k^i, \beta_k^i\right) \in \mathbb{R}$, to the original delta error with integer shifts, $\Delta y_k^l\left(m + \lfloor\alpha_k^i\rfloor,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

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

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, $\left(\alpha_k^i, \beta_k^i\right)$, shifts only the pixels of the 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 Eq. (46). Let $\nabla_\alpha\vec{y}(m,n) = \frac{\partial\vec{y}_k^l(m,n)}{\partial\alpha_k^i}$, which was expressed in Eq. (41), Eq. (46) finally simplifies to Eq. (47) 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 $\Delta\vec{y}_k^l$.

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

$$\frac{\partial E}{\partial y_k^l}\left(m+\alpha_k^i, n+\beta_k^i\right)\Big|_{(0,0)}^{(M-1,N-1)} = \Delta y_k^l\left(m+\alpha_k^i, n+\beta_k^i\right) = \Delta\vec{y}_k^l(m,n)$$
$$= \sum_{r=0}^{\mathrm{Kx}-1}\sum_{t=0}^{\mathrm{Ky}-1} \frac{\partial E}{\partial x_i^{l+1}(m-r,n-t)} \times \frac{\partial x_i^{l+1}(m-r,n-t)}{\partial P_i^{l+1}\left[..,\boldsymbol{\Psi}\left(\vec{y}_k^l(m,n), \boldsymbol{w_{ik}^{l+1}}(\boldsymbol{r},\boldsymbol{t})\right),..\right]} \times \frac{\partial P_i^{l+1}\left[..,\boldsymbol{\Psi}\left(\vec{y}_k^l(m,n), \boldsymbol{w_{ik}^{l+1}}(\boldsymbol{r},\boldsymbol{t})\right),..\right]}{\partial \boldsymbol{\Psi}\left(\vec{y}_k^l(m,n), \boldsymbol{w_{ik}^{l+1}}(\boldsymbol{r},\boldsymbol{t})\right)} \times \frac{\partial \boldsymbol{\Psi}\left(\vec{y}_k^l(m,n), \boldsymbol{w_{ik}^{l+1}}(\mathbf{r},\boldsymbol{t})\right)}{\partial \vec{y}_k^l(m,n)} \tag{42}$$

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

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

$$\Delta y_k^l\left(m_\alpha^r,\ n_\beta^t\right) = \Delta\vec{y}_k^l(m,\ n)\left(1-\zeta\alpha_k^i\right)\left(1-\zeta\beta_k^i\right) + \Delta\vec{y}_k^l(m-1,\ n-1)\zeta\alpha_k^i\zeta\beta_k^i + \Delta\vec{y}_k^l(m-1,\ n)\zeta\alpha_k^i\left(1-\zeta\beta_k^i\right) + \Delta\vec{y}_k^l(m,\ n-1)\left(1-\zeta\alpha_k^i\right)\zeta\beta_k^i \tag{45}$$

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

$$\Delta\alpha_k^i = \frac{\partial E}{\partial \alpha_k^i} = \sum_{m=0}^{M-1}\sum_{n=0}^{N-1} \nabla_\alpha \mathbf{y}(m,n) \times$$
$$\left(\sum_{r=0}^{\mathrm{K_x}-1}\sum_{t=0}^{\mathrm{K_y}-1} \Delta_{\mathrm{i}}^{l+1}(m-r,n-t) \times \nabla_{\boldsymbol{\Psi}}\mathrm{P}_i^{l+1}(m+\alpha_k^i, n+\beta_k^i, r, t) \times \nabla_{\vec{y}}\boldsymbol{\Psi}(m+\alpha_k^i, n+\beta_k^i, r, t)\right) \tag{47}$$
$$= \sum_{m=0}^{M-1}\sum_{n=0}^{N-1} \nabla_\alpha\vec{y}(m,n) \times \Delta\vec{y}_k^l(m,n) = \nabla_\alpha\vec{y} \otimes \Delta\vec{y}_k^l$$

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 Eq. (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 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, $\varepsilon(\mathrm{t})$ and $\gamma(\mathrm{t})$ 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 Eq. (49). The parameters of a Self-ONN for BP training via SGD are presented in Table 4.

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

Table 4: The train parameters of a Self-ONN with super neurons.

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*
	$U(-\mathbf{\Gamma},\mathbf{\Gamma})$	$\mathbf{\Gamma} = \mathbf{8}$	*for spatial bias*
Learning factors	ε	$\varepsilon(t) = 0.1$	*for kernel parameters*
	γ	$\gamma(t) = 10$	*for spatial bias pairs*

Algorithm 1: 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 Eq. (6) create the shifted output map(s) along with their powers, $(\vec{y}_k^0)^q,\ \forall q \in [1,Q_1]$ 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 Eq. (7), then the native output maps, y_i^1 and finally, the shifted output maps along with their powers, $(\vec{y}_i^1)^q\ \forall q \in [1,Q_1]$.
      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 Eqs. (25) and (26), respectively.
      v. **BP**: For each hidden neuron at the last hidden layer, using Eq. (39) compute delta error for the shifted output map and then using Eq. (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 Eq. (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 Eq. (33) (or Eq. (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 Eqs. (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 Eq. (49).
3) **Return Self-*ONN****

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 Alg. 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 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 Alg. 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., $\alpha_k^i(0) = \lfloor U(-\boldsymbol{\Gamma} - \mathbf{1}, \boldsymbol{\Gamma} + \mathbf{1}) \rfloor$ and $\beta_k^i(0) = \lfloor U(-\boldsymbol{\Gamma} - \mathbf{1}, \boldsymbol{\Gamma} + \mathbf{1}) \rfloor$ for $\forall i \in [1, N_{l+1}], \forall k \in [1, N_l]$. Then since the spatial bias elements are integers now, Eq. (3) can be used instead of Eq. (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 Eq. (31) should be used instead of Eq. (39) for the delta error computed for the connection to the i^{th} neuron at layer l+1 and there is no need for reverse interpolation, hence Eq. (45) is simply omitted. Obviously for post-processing (PP) at step 2.a.vii, and Update at step 2.a.xi, Eqs. (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, Eq. (3) can be used instead of Eq. (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 Eq. (31) should be used instead of Eq. (39) for the delta error computed for the connection to the i^{th} neuron at layer l+1 and there is no need for reverse interpolation, hence Eq. (45) is simply omitted. Obviously for post-processing (PP) at step 2.a.vii, and update at step 2.a.xi, Eqs. (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.

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 $(\alpha, \beta) \in \mathbb{Z}[-\boldsymbol{\Gamma}, \boldsymbol{\Gamma}]$, 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. Figure 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 Figure 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 (1,1) 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 $\boldsymbol{\Gamma} = \mathbf{8}$ pixels**.** Figure 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 35dB. 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 $(1,1)$ pixelsare 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=35dB, 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.

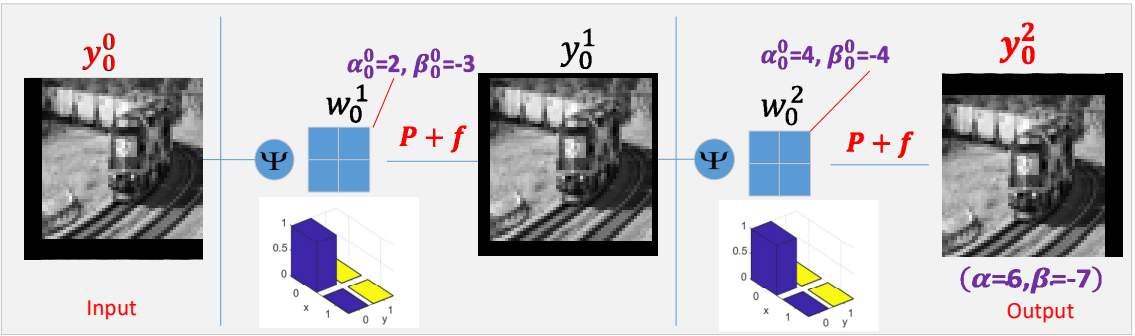

Figure 12: A sample Self-ONN with a single (hidden) super neuron over the toy problem. The perfect regression of the target is illustrated (SNR = ∞) for an ideal case.

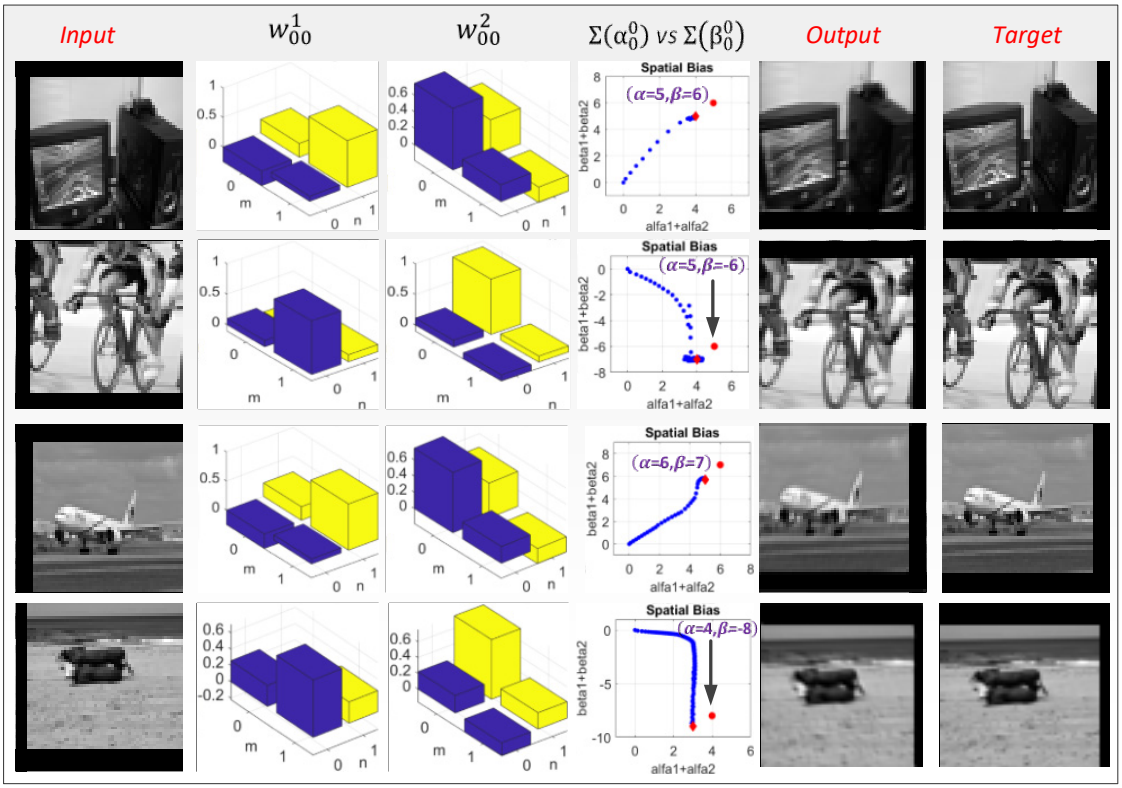

Figure 13: Four "Proof of Concept" verification experiments where the target images are created with random shifts are shown at each row. The 2nd and 3rd columns show bar plots of the kernels and the 4th column shows the plots of the cumulative bias elements (hidden and output super neurons) in each 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 35dB.

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.