# On Joint Parameterizations of Linear and Nonlinear Functionals in Neural Networks

Abdourrahmane Mahamane Atto, Sylvie Galichet, Dominique Pastor, Nicolas Méger

▶ **To cite this version:**

Abdourrahmane Mahamane Atto, Sylvie Galichet, Dominique Pastor, Nicolas Méger. On Joint Parameterizations of Linear and Nonlinear Functionals in Neural Networks. Neural Networks, 2023, 160, pp.12-21. 10.1016/j.neunet.2022.12.019 . hal-03115681v3

**HAL Id: hal-03115681**
**https://hal.science/hal-03115681v3**

Submitted on 12 Jan 2023

**HAL** is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

# On Joint Parameterizations of Linear and Nonlinear Functionals in Neural Networks

Abdourrahmane Mahamane ATTO[a,*], Sylvie GALICHET[a], Dominique PASTOR[b], Nicolas MÉGER[a]

[a] *Université Savoie Mont Blanc - LISTIC*
*BP 80439 - F-74944 Annecy-le-Vieux Cedex* - FRANCE
[b] *IMT Atlantique, Lab-STICC, Université de Bretagne Loire*
*29238 Brest* - FRANCE

**Abstract**

The paper proposes a new class of nonlinear operators and a dual learning paradigm where optimization jointly concerns both linear convolutional weights and the parameters of these nonlinear operators. The nonlinear class proposed to perform a rich functional representation is composed by functions called rectified parametric sigmoid units. This class is constructed to benefit from the advantages of both sigmoid and rectified linear unit functions, while rejecting their respective drawbacks. Moreover, the analytic form of this new neural class involves scale, shift and shape parameters to obtain a wide range of activation shapes, including the standard rectified linear unit as a limit case. Parameters of this neural transfer class are considered as learnable for the sake of discovering the complex shapes that can contribute to solving machine learning issues. Performance achieved by the joint learning of convolutional and rectified parametric sigmoid learnable parameters are shown to be outstanding in both shallow and deep learning frameworks. This class opens new prospects with respect to machine learning in the sense that main learnable parameters are attached not only to linear transformations, but also to a wide range of nonlinear operators.

**Keywords** – Deep learning ; Ensemble learning ; Sigmoid shrinkage ; Rectified linear unit ; Rectified sigmoid ; Parametric activation ; Convolutional neural network.

## 1. Introduction

Standard neural transfer functions such as Rectified Linear Unit (ReLU) [1] and sigmoid hereafter, denoted respectively $U$ and $S$ with

$$U(x) = x\mathbb{1}_{x>0} = \max(0, x) \tag{1}$$

*Corresponding author
*Email address:* Abdourrahmane.Atto@univ-smb.fr (Abdourrahmane Mahamane ATTO )

$$S(x) = \frac{1}{1+e^{-x}} \tag{2}$$

are non-parametric functions in the sense that their analytic expressions do not depend on unknown parameters or weights. While the sigmoid function has been the leader of the early-stage neural transfer functions, it has been outclassed by the ReLU in most recent deep Convolutional Neural Networks (CNN), see [2, 3, 4, 5, 6, 7, 8] among others.

In terms of machine learning, the first major difference between ReLU and sigmoid is the fact that ReLU output is expected to be a sparse sequence in general, while sigmoid function simply penalizes its entries without forcing non-zero values to zero. Thus, in terms of the compromise between computational complexity and available working memory, ReLU is naturally favored when very deep networks are under consideration.

The second major difference between ReLU and sigmoid concerns their derivatives. The derivative $U'$ of $U$ is the Heaviside unit step function: such a function is stable by composition. However, it admits a singularity at 0 and has the same constant output for both small and large positive values, which may be counterintuitive since if we consider for instance sparse transforms, small and large positives do not carry the same level of information. In addition, because of the zero-forcing operated by ReLU derivative, then learning can be inhibited[1] in a ReLU CNN when the processing implies a large number of negatives.

In contrast with ReLU, the derivative of a standard sigmoid is smooth everywhere. However, it is always strictly less than 1 and this can also lead to a fast decrease to 0 of the sigmoid increments by composition and this, both for positive and negative entries.

One can note that both ReLU and sigmoid admit parametric forms $x \mapsto x\mathbb{1}_{x>0} + \alpha x\mathbb{1}_{x\leqslant 0}$ for parametric ReLU [9] and $x \mapsto S(\alpha x)$ for parametric sigmoid [10], [11]. These parametric forms can solve the limitations highlighted above for specific applications and when $\alpha$ is chosen carefully. It is worth noticing that the use of these parametric forms is limited to specific datasets or specialized networks and their generalization capabilities need to be proven.

In terms of image processing, important properties are invariances by rotation, translation and scaling. It is well known that rotation invariance can be handled by a suitable sequence of convolution filters. For the two remaining invariance properties: on the one hand, both ReLU and sigmoid are translation-variant. On the other hand, only ReLU is scale-invariant, but from a general perspective, translation and scaling invariances can also be obtained from other components of the network such as pooling and convolution layers respectively for the translation and scaling invariances.

This paper provides in the Section 2, new neural transfer functions that possess most of the desirable properties highlighted above, while limiting the undesirable ones. Because biological neurons have non-uniform[2] activation functions, we will propose

[1]Leaky ReLU: $x \mapsto x\mathbb{1}_{x>0} + 0.01x\mathbb{1}_{x\leqslant 0}$ can avoid such issues, however, it is less used in deep neural networks because it raises other issues (such as the arbitrary penalization of negative values, the latter being far from bio-inspired behaviors).

[2]The activation functions depend on the specialization and the depth of the neurons in the brain as diverse inhibition mechanisms in the brain can influence information transfer.

convolutional neural learning frameworks where learning includes the determination of suitable activation functions. Even though this framework leads to a higher computational complexity than using a non-parametric ReLU transfer functions, we will show in Section 3 and 4 that it is highly relevant for shallow and deep learning by providing comparisons with respect to frameworks based on both parametric and non-parametric variants of MISH [12] or SWISH [13] activation functions. Section 5 provides outlooks raised by the joint linear-and-nonlinear parametric learning framework and Section 6 concludes the work.

## 2. Rectified parametric sigmoid shrinking and stretching units

### *2.1. Definitions*

Let $\mathrm{sgn}$ denotes the sign function given by:

$$\mathrm{sgn}(x) = \begin{cases} -1 & \text{for} \quad x < 0 \\ 0 & \text{for} \quad x = 0 \\ 1 & \text{for} \quad x > 0 \end{cases} \tag{3}$$

and $\mathbb{1}_{\mathcal{E}}$ be the indicator of set $\mathcal{E}$ associated with the notation:

$$\mathbb{1}_{\lambda}(x) = \mathbb{1}_{\{x \geqslant \lambda\}} = \begin{cases} 1 & \text{if} \quad x \geqslant \lambda \\ 0 & \text{if} \quad x < \lambda \end{cases} \tag{4}$$

The contributions provided by the paper start from the definitions of 3 transfer functions given below. We first define the Rectified Parametric Sigmoid shrinKage Units (RePSKU or RePSU$^{\triangledown}$) by the form:

$$f_{\lambda,\mu,\sigma,\beta}(x) = \frac{(x-\lambda)\mathbb{1}_{\lambda}(x)}{1 + e^{-\mathrm{sgn}(x-\mu)\left(\frac{|x-\mu|}{\sigma}\right)^{\beta}}} \tag{5}$$

RePSU$^{\triangledown}$ *threshold* $\lambda$ is inspired from the behavior of ReLU functions (see by Eq. (1) in particular for $\lambda = 0$ which implies forcing negative inputs to 0). RePSU$^{\triangledown}$ involves in its exponential term, a *shift* parameter $\mu$, a *scale* parameter $\sigma$ and a *shape* parameter $\beta$: these parameters are inspired from the generalized Gaussian distribution, but integrated as in [14] (Smooth Sigmoid Based Shrinkage functions, SSBS) by using a sigmoid form.

Secondly, the Rectified Parametric Sigmoid stretcHage Units (RePSHU or RePSU$^{\triangle}$) are defined as the RePSU$^{\triangledown}$ dependent functions given by:

$$g_{\lambda,\mu,\sigma,\beta}(x) = 2x\mathbb{1}_{\lambda}(x) - f_{\lambda,\mu,\sigma,\beta}(x) \tag{6}$$

RePSU$^{\triangle}$ can be considered as a *complement* to RePSU$^{\triangledown}$ in the following specific sense: the average of RePSU$^{\triangledown}$ and RePSU$^{\triangle}$ activations is $[f_{\lambda,\mu,\sigma,\beta}(x) + g_{\lambda,\mu,\sigma,\beta}(x)]/2 = x$ for all $x \geqslant \lambda$.

Finally, both RePSU$^{\triangledown}$ and RePSU$^{\triangle}$ are encapsulated to form a larger parametric transfer class called Rectified Parametric Sigmoid Unit (RePSU) and given by:

$$A_{\lambda,\mu,\sigma,\beta,\alpha}(x) = \alpha g_{\lambda,\mu,\sigma,\beta}(x) + (1-\alpha) f_{\lambda,\mu,\sigma,\beta}(x) \tag{7}$$

Examples of RePSU graphs are given in Figure 1 for a given input shape and different parameters $(\sigma, \beta, \alpha)$, the latter being considered as positive real values hereafter. The RePSU outputs displayed in this figure correspond to RePSU$^{\triangledown}$ (respectively RePSU$^{\triangle}$) when the graphs are located under (respectively over) the input data transformed (yellow curve represents the identity).

**Remark**

If $\alpha = 0$ and $\beta = 1$, then function $u_{\lambda,\mu,\xi} = A_{\lambda,\mu,1/\xi,1,0}$ has the following form:

$$u_{\lambda,\mu,\xi}(x) = \begin{cases} \frac{x-\lambda}{1+e^{-\xi(x-\mu)}} & \text{if} \quad x \geqslant \lambda \\ 0 & \text{if} \quad x < \lambda \end{cases} \tag{8}$$

where we have assumed $\xi = 1/\sigma$. Thus, RePSU$^{\triangledown}_{\lambda,\mu,\sigma,1,0}$ corresponds to the restriction on $\mathbb{R}^+$ of the SSBS activation functions [14]. Furthermore, the restriction on $\mathbb{R}^+$ of the SSBS class includes the SWISH [13] (when $\mu = 0$) and SiLU [15] (for $\mu = 0$ and $\xi = 1$) activations[3]. Moreover, RePSU class includes standard ReLU since $u_{\lambda\to 0,\mu,\xi\to+\infty} = U$ where $U$ is the ReLU function defined by Eq. (1). Thus $A_{\lambda,\mu,\sigma,\beta,\alpha}$ can be seen as a generalization of several standard transfer functions.

### *2.2. RePSU: penalized attention mechanisms*

Attention mechanisms consist in constraining an optimization problem by specifying some relevant property to reach desirable solutions (that are generally different from those of the unconstrained problem). For instance, if we consider the following unconstrained least squares minimization problem:

$$\min_{\boldsymbol{y}} ||\boldsymbol{y} - \boldsymbol{d}||^2_{\ell_2}$$

where $\boldsymbol{d}$ is a vector corresponding to the data observed, then its standard attention based variant is the so-called *penalized least squares* problem given by [16]:

$$\min_{\boldsymbol{y}} ||\boldsymbol{y} - \boldsymbol{d}||^2_{\ell_2} + P(\boldsymbol{y})$$

In this problem, the choice $P(\boldsymbol{y}) = \lambda||\boldsymbol{y}||_{\ell_1}$ is known as a regularized least squares penalization problem (sparsity driven attention).

Consider now the transformation induced by RePSU$^{\triangledown}$ in Eq. (5) and let:

$$P_{\lambda,\mu,\sigma,\beta}(x) = 2\int_0^x \left(f^{-1}_{\lambda,\mu,\sigma,\beta}(z) - z\right) \mathrm{d}z \tag{9}$$

for $x \geqslant 0$, where $f^{-1}$ is the inverse[4] of $f$. Then, for $d \geqslant 0$, $f_{\lambda,\mu,\sigma,\beta}(d)$ is the solution of the minimization problem

$$\min_{y\geqslant 0}(y-d)^2 + P_{\lambda,\mu,\sigma,\beta}(y)$$

[3]The restriction of SWISH and SiLU on $\mathbb{R}^-$ is composed by negligible values that are not forced to zero: a limitation in terms of sparsity that is avoided by the ReLU-like behavior of $u_{0,\mu,\xi}(x)$ on $\mathbb{R}^-$.

[4]Either the natural inverse or the generalized (principal sub-solution) respectively if $\lambda = 0$ or if $\lambda > 0$.

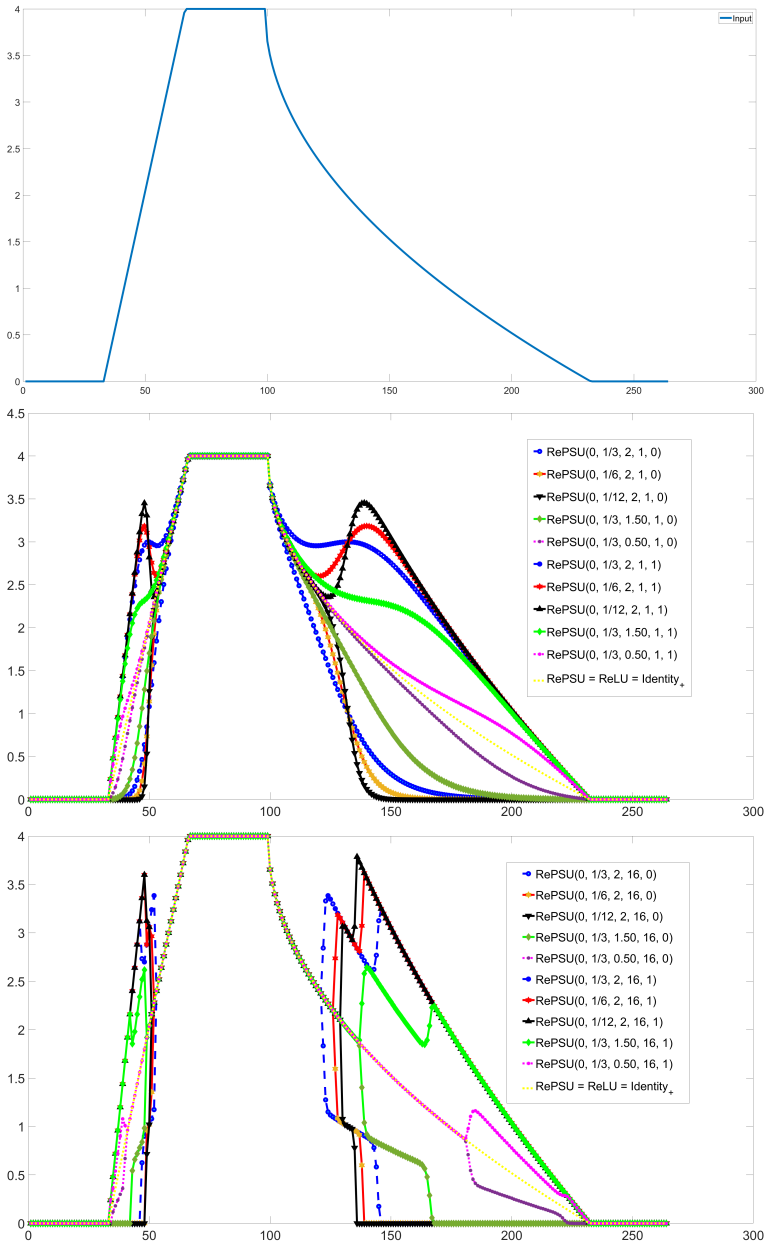


Figure 1: Transform $y = A_{\lambda=0,\mu,\sigma,\beta,\alpha}(x)$ operated by RePSU when $x$ =Input is given at the top.

Indeed, the derivative of $(y-d)^2+P_{\lambda,\mu,\sigma,\beta}(y)$ with respect to $y$ is $2(y-d)+P'_{\lambda,\mu,\sigma,\beta}(y)$ and this derivative is reduced to $2f^{-1}_{\lambda,\mu,\sigma,\beta}(y) - 2d$ since from Eq. (9), we have: $P'_{\lambda,\mu,\sigma,\beta}(y) = 2f^{-1}_{\lambda,\mu,\sigma,\beta}(y) - 2y$. Thus, the above derivative is 0 (necessary condi-

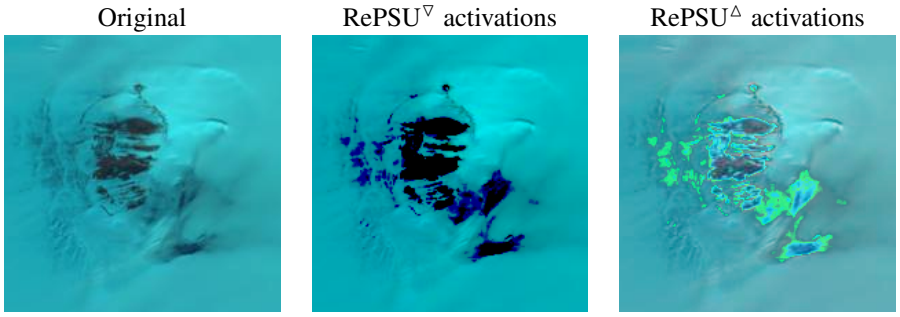


Figure 2: Original: Landsat 8 OLI image at Antarctica's Marie Bird Land. RePSU$^{\nabla}$: shrink small signals and highlight medial variations. RePSU$^{\Delta}$: amplify small signals and highlight medial variations. These behaviors can be seen as attention mechanisms. Note that for both RePSU$^{\nabla}$ and RePSU$^{\Delta}$, very small and very large values are left almost unchanged.

tion for optimality) iff $f^{-1}_{\lambda,\mu,\sigma,\beta}(y) = d$, that is if $y = f_{\lambda,\mu,\sigma,\beta}(d)$. Note that we have considered non-negative values in the elements of attention evidence given above. This is because neural activations are considered to be positive in this paper. For a generalization to negative outputs, antisymmetric extension of RePSU can be considered.

Concerning the practical interpretation of the RePSU attention focus, note that from Eq. (5) (see also Figure 1), it follows that parameter $\mu$ controls the inflection location from which differentiated penalties are applied to the input. The penalty intuition is the following: activations larger than $\mu$ remain quasi-unchanged whereas those smaller than $\mu$ can be either attenuated (case of RePSU$^{\nabla}$, because they do not carry significant information) or amplified (case corresponding to RePSU$^{\Delta}$, because they are associated with weak signals carrying some information of interest) before transfer to the upstream part of the neural network. Figure 2 highlights this behavior when RePSU$^{\nabla}$ and RePSU$^{\Delta}$ are applied to transform a satellite image.

### *2.3. Derivatives of RePSU*

When assuming learnable parameters for RePSU in a gradient descent based minimization problem, we need to compute RePSU partial derivatives with respect to variables $\lambda, \mu, \sigma, \beta, \alpha$. Note that from (7), we have:

$$\frac{\partial A_{\lambda,\mu,\sigma,\beta,\alpha}(x)}{\partial \alpha} = g_{\lambda,\mu,\sigma,\beta}(x) - f_{\lambda,\mu,\sigma,\beta}(x) \tag{10}$$

Figure 3 shows the behaviors of the above partial derivatives when the input data are those given in Figure 1: the derivatives are almost zeros for large values of $x$, except in the neighborhood of the penalization regions associated with the value $\mu$, where the differentiate penalization shapes involved are governed by $\sigma$ and $\beta$.

Regarding the partial derivatives with respect to $\lambda, \mu, \sigma, \beta$ and due to the simple dependencies of $A$ and $g$ with respect to $f$ (see Equations (7) and (6) respectively), we will focus on derivatives of $f$ hereafter. From Eq. (5) and after some steps of calculus, we have:

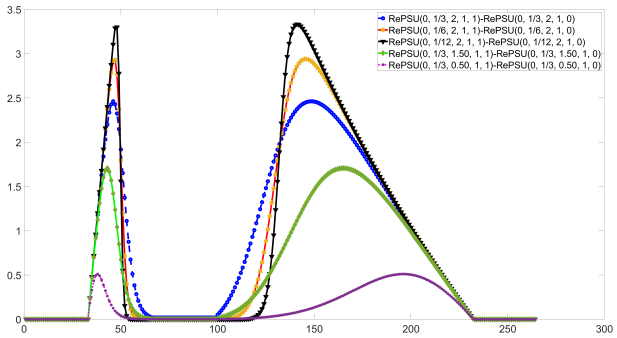


Figure 3: Graphs $\partial/\partial\alpha\left(A_{\lambda,\mu,\sigma,\beta,\alpha}(x)\right) = A_{\lambda,\mu,\sigma,\beta,1}(x) - A_{\lambda,\mu,\sigma,\beta,0}(x)$ when $x$ is given at the top of Figure 1.

$(i)$ Derivative with respect to the activation threshold $\lambda$:

$$\frac{\partial f_{\lambda,\mu,\sigma,\beta}(x)}{\partial\lambda} = \frac{-\mathbb{1}_{\lambda}(x)}{1+e^{-\operatorname{sgn}(x-\mu)\left(\frac{|x-\mu|}{\sigma}\right)^{\beta}}} \tag{11}$$

$(ii)$ Derivative with respect to the location parameter $\mu$ (inflection point for penalization):

$$\begin{aligned}\frac{\partial f_{\lambda,\mu,\sigma,\beta}(x)}{\partial\mu} = &-\frac{1}{4\sigma^{\beta}}(x-\lambda)\mathbb{1}_{\lambda}(x)\times \\ &\left(2\delta(x-\mu)|x-\mu|^{\beta}+\beta(x-\mu)\operatorname{sgn}(x-\mu)|x-\mu|^{\beta-2}\right) \\ &\times\operatorname{sech}^2\left(\frac{1}{2}\operatorname{sgn}(x-\mu)\left(\frac{|x-\mu|}{\sigma}\right)^{\beta}\right)\end{aligned} \tag{12}$$

where *sech* is the hyperbolic secant function $\operatorname{sech}(x) = 2\Big/\left(e^{x}+e^{-x}\right)$,
$(iii)$ Derivative with respect to the scale parameter $\sigma$ (penalization weight):

$$\frac{\partial f_{\lambda,\mu,\sigma,\beta}(x)}{\partial\sigma} = \frac{-\beta}{2\sigma}\frac{(x-\lambda)\mathbb{1}_{\lambda}(x)\operatorname{sgn}(x-\mu)\left(\frac{|x-\mu|}{\sigma}\right)^{\beta}}{1+\cosh\left(\operatorname{sgn}(x-\mu)\left(\frac{|x-\mu|}{\sigma}\right)^{\beta}\right)} \tag{13}$$

where *cosh* is the hyperbolic cosine function $\cosh(x) = \left(e^{x}+e^{-x}\right)\Big/2$ and
$(iv)$ Derivative with respect to the shape parameter $\sigma$ (penalization form):

$$\begin{aligned}\frac{\partial f_{\lambda,\mu,\sigma,\beta}(x)}{\partial\beta} = &\frac{1}{4\sigma^{\beta}}(x-\lambda)\mathbb{1}_{\lambda}(x) \\ &\left(\operatorname{sgn}(x-\mu)|x-\mu|^{\beta}\log\left(\frac{|x-\mu|}{\sigma}\right)\right) \\ &\operatorname{sech}^2\left(\frac{1}{2}\operatorname{sgn}(x-\mu)\left(\frac{|x-\mu|}{\sigma}\right)^{\beta}\right)\end{aligned} \tag{14}$$

In some situations, one may want to fix parameters $\lambda, \mu, \sigma, \beta, \alpha$ (set them as non-learnable for the sake of transfer learning for example) and in this case, the requirement for the global gradient descent updating is the derivative:

$$f'_{\lambda,\mu,\sigma,\beta}(x) = \frac{\partial f_{\lambda,\mu,\sigma,\beta}(x)}{\partial x} = \frac{\beta}{4\sigma}(x-\lambda)\mathbb{1}_{\lambda}(x) \times \left(\operatorname{sgn}(x-\mu)\left(\frac{|x-\mu|}{\sigma}\right)^{\beta-1}\right) \times \operatorname{sech}^2\left(\frac{1}{2}\operatorname{sgn}(x-\mu)\left(\frac{|x-\mu|}{\sigma}\right)^{\beta}\right) \tag{15}$$

$$+\frac{\mathbb{1}_{\lambda}(x)}{1+e^{-\operatorname{sgn}(x-\mu)\left(\frac{|x-\mu|}{\sigma}\right)^{\beta}}} \tag{16}$$

RePSU derivatives are smooth: the general behavior is a "no-jump" property[5] which implies introducing only small variabilities[6] between close objective values. In contrast, at the RePSU limits corresponding to translated versions of ReLU (when $\xi \to +\infty$ and for fixed $\mu$), the derivatives shift rapidly from 0 to 1 similarly to ReLU. Thus to summarize, the outstanding RePSU derivative property is that the graph of the derivative is flat for large activations (the derivatives tend to 1 at infinity): this yields a stable composition for the few very large activations received by the activation function.

Some useful additional invariance properties inherited from RePSU$^{\triangledown}$ are given below.

### *2.4. Intra-class translation invariance for RePSU$^{\triangledown}$*

The invariance highlighted below applies at RePSU$^{\triangledown}$ subclass level. Assumes $x-\tau$ is the input of the RePSU$^{\triangledown}$ class. Then one can note that for RePSU$^{\triangledown}$ functions defined by Eq. (8), we have:

$$u_{\lambda,\mu,\xi}(x-\tau) = u_{\tau+\lambda,\tau+\mu,\xi}(x)$$

This implies that the output of a shifted input can be deducted directly from parameter shifts of the RePSU$^{\triangledown}$. Thus, translation invariance can be achieved by a series of RePSU$^{\triangledown}$ functions associated with different parameters. In comparison with ReLU for which translation can induce forcing to zero, RePSU$^{\triangledown}$ has the capability to either keep invariant or force to zero a given value depending on the training objective.

### *2.5. Intra-class scaling conservatives for RePSU$^{\triangledown}$*

Scaling is present at different stages in image processing. For instance, dividing an 8-bit coded image by a positive constant changes the scaling, but does not affect pixel distribution shapes. For RePSU$^{\triangledown}$ functions given by Eq. (8), we have the following property:

$$u_{\lambda,\mu,\xi}(\alpha x) = \alpha u_{\lambda/\alpha,\mu/\alpha,\alpha\xi}(x)$$

[5]The derivative has no-jump, expected for parameter limits.

[6]Discontinuities are known to generate a high variance in iterative processing.

Thus, re-scaling a value can be inferred by re-scaling RePSU$^{\triangledown}$ outputs thanks to scaled parameters set as $(\lambda', \mu', \xi') = (\lambda/\alpha, \mu/\alpha, \alpha\xi)$.

The following addresses performance of CNN involving RePSU nonlinearities. We will use $\beta = 1$ for all experimental results so that learning concerns:

- parameters $(\lambda, \mu, \sigma)$ for the shrinkage activation RePSU$^{\triangledown}$ (given by Eq. (5)) and the stretchage activation RePSU$^{\triangle}$ (given by Eq. (6));
- parameters $(\lambda, \mu, \sigma, \alpha)$ for the more general RePSU form defined by Eq. (7).

## 3. Learning activation function parameters: shallow networks

The second main contribution provided by the paper is the joint learning of standard convolutional linearities and parameters of the nonlinear ReSPU class: ReSPU parameters $\lambda, \mu, \sigma, \beta, \alpha$ (see Eq. (7)) are assumed learnable hereafter. The issue addressed in this section is then measuring the performance provided by the learning of ReSPU nonlinearities, when the latter are considered to activate convolution outputs of a shallow CNN.

We recall that the main goal of the paper is to prove the interest in learning optimal nonlinear activations from a parametric family generalizing/including the ReLU, in contrast with using only (non-learnable) ReLU activations. Thus, a good comparison could be limited to an opposition between a "learnable RePSU" and a "purely ReLU" CNNs. However, we extend the comparison by testing both fixed and parameterized forms of some recent RELU's alternatives: the **Parametric MISH (PMISH)** by [12]:

$$\begin{aligned} M(x) &= x \tanh(\text{softplus}_{\xi}(x)) \\ &= x \tanh\left(\frac{1}{\xi}\log(1 + e^{\xi x})\right) \end{aligned} \tag{17}$$

and for the **Parametric SWISH (PSWISH)** used in [13] and also called SiLU (Sigmoid Linear Unit) in [15]:

$$S(x) = x\sigma(\xi x) = \frac{x}{1 + e^{-\xi x}} \tag{18}$$

In this respect, we will also consider learning the activation parameter $\xi$ involved in PMISH and PSWISH. We will keep the terminologies of **MISH** and **SWISH** in the standard (non-learnable) cases corresponding to $\xi = 1$ in Eqs. (17) and (18).

### *3.1. Fast and efficient learning from scratch with respect to shallow CNNs*

We consider a standard, so-called MNIST, handwritten digit recognition problem described in [17]. CNN architectures are known to be efficient for solving such a problem and therefore, what we propose here is to test the learning speed on different CNN structures and several hyperparameter settings. Thereby, the number of training epochs is fixed respectively to either 1 or 2 and a Monte Carlo simulation framework is applied to avoid a biased comparison that can be due to parameter initialization.

Table 1: Shallow CNN-1-$X$ frameworks where $X \in \{\text{ReLU, RePSU}^{\triangledown}\text{, MISH, PMISH, SWISH, PSWISH}\}$ corresponds to the specific activation function used at layer 4. FC denotes a *Fully Connected* layer. We have considered $N_2 = 32$ for CNN-1. The number of parametric activation functions used is $N_2$ (number of convolution filters used downstream), that is one function per convolution filter.

| Layer | Content | $\#N$ of Elements | Element size | Learnable |
|---|---|---|---|---|
| 1 | Inputs<br>Images | $N_1$ | $M_1^x \times M_1^y \times M_1^c$ | No |
| 2 | 'Convolve' | $N_2$ | $M_2^x \times M_2^y$ | Yes |
| 3 | 'Normalize' | Standard / Mini-batch | | |
| 4 | ReLU | $N_2$ | - | No |
| | RePSU$^{\triangledown}$ | | 3 | Yes |
| | MISH | | - | No |
| | PMISH | | 1 | Yes |
| | SWISH | | - | No |
| | PSWISH | | 1 | Yes |
| 5 | 'FC' | [Output size $L$ ] | | |
| 6 | 'Softmax' | Probabilities with respect to $L$ outputs | | |
| 7 | 'Classify' | Cross-entropy (Output: category) | | |

More precisely, training and testing concern the shallow CNN described by Table 1 and the Monte Carlo trials apply on the number of training epochs and the sizing[7] of layer 2 in by Table 1. The recognition tasks are associated with the following experimental setup:

- Split the handwritten digit database in training and testing sets;
- Specify a number of epochs and perform iteratively, the following:
  - initialize convolution parameters from real random numbers,
  - initialize RePSU$^{\triangledown}$, PMISH, PSWISH parameters from positive real random numbers,
  - perform training[8] with respect to the number of epochs, then testing
  - save testing score and reiterate;
- Compute average performance over the correspond 100 Monte Carlo trials.

Experimental results are given in Table 2. It appears that RePSU$^{\triangledown}$ based CNNs deliver high performance faster than the ReLU, MISH, PMISH, SWISH and PSWISH based CNNs. We recall that finding a favorable initialization to any of the given CNNs is often just a matter of trials in intensive simulation environments: it is not the problem addressed in this section. The motivation is testing the behavior of any CNN in generic

[7]Numbers and sizes of convolution filters, which determine the number of additional parameters used.

[8]In RePSU case, the corresponding cross-entropy depends on RePSU parameters: the latter (except $\beta$, fixed to $\beta = 1$ because $\beta$ is at the power of an exponential term), are updated thanks to the behavior of the cross-entropy and the RePSU derivatives by using the back-propagation algorithm with respect to a gradient descent method, similarly as when updating convolution weights.

Table 2: Mean accuracies in percentages over 100 Monte Carlo trials of the handwritten digit recognition issue: impact of the number of epochs, the Convolution Filter Size (CFS) and Number of Convolution Filters (NCF) with respect to shallow CNN-1-$X$ frameworks defined in Table 1 where $X \in \{$ ReLU, RePSU, MISH, PMISH, SWISH, PSWISH $\}$.

| Non-learnable activations | | | | | | | | Learnable activations | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| CFS (columns) $\in \{2, 3, 4, 5, 6, 7\}$ | | | | | | | | | | | | | | |
| EPOCH = 1 | | | | | | | | | | | | | | |
| | 2 | 3 | 4 | 5 | 6 | 7 | NCF | 2 | 3 | 4 | 5 | 6 | 7 | |
| CNN-1-ReLU | 72.67 | 72.70 | 72.23 | 72.35 | 72.40 | 72.25 | 10 | 78.07 | 76.92 | 78.07 | 77.48 | 76.99 | 77.03 | CNN-1-RePSU |
| | 77.08 | 77.57 | 77.45 | 77.10 | 77.40 | 77.47 | 20 | 84.99 | 85.84 | 83.92 | 85.48 | 84.91 | 84.21 | |
| | 78.27 | 78.18 | 78.34 | 78.07 | 78.28 | 77.85 | 30 | 86.75 | 87.91 | 88.42 | 88.49 | 88.69 | 85.83 | |
| | 77.96 | 77.97 | 77.79 | 77.93 | 77.72 | 77.55 | 40 | 89.73 | 88.91 | 87.09 | 88.81 | 88.99 | 88.16 | |
| | 77.62 | 77.76 | 77.62 | 77.36 | 77.62 | 77.29 | 50 | 89.74 | 87.55 | 89.71 | 89.74 | 90.62 | **90.55** | |
| | 2 | 3 | 4 | 5 | 6 | 7 | NCF | 2 | 3 | 4 | 5 | 6 | 7 | |
| CNN-1-MISH | 75.14 | 75.13 | 75.64 | 74.98 | 75.03 | 74.83 | 10 | 66.38 | 66.60 | 66.71 | 66.98 | 66.68 | 66.77 | CNN-1-PMISH |
| | 81.69 | 81.83 | 81.85 | 81.51 | 81.80 | 81.44 | 20 | 68.68 | 68.42 | 68.84 | 68.29 | 68.46 | 68.59 | |
| | 84.24 | 84.19 | 84.13 | 84.12 | 83.93 | 84.07 | 30 | 68.55 | 69.00 | 68.72 | 68.47 | 68.46 | 68.52 | |
| | 84.88 | 85.02 | 84.62 | 84.83 | 85.00 | 84.75 | 40 | 68.27 | 68.02 | 67.96 | 68.02 | 68.00 | 68.07 | |
| | 84.89 | 84.96 | 84.80 | 84.69 | 85.01 | 84.93 | 50 | 67.28 | 67.57 | 67.22 | 66.89 | 67.11 | 67.55 | |
| | 2 | 3 | 4 | 5 | 6 | 7 | NCF | 2 | 3 | 4 | 5 | 6 | 7 | |
| CNN-1-SWISH | 75.40 | 75.19 | 75.11 | 74.98 | 75.16 | 75.27 | 10 | 70.23 | 70.37 | 70.14 | 70.16 | 70.49 | 70.79 | CNN-1-PSWISH |
| | 82.17 | 82.03 | 81.82 | 81.89 | 82.23 | 81.78 | 20 | 75.81 | 75.79 | 76.19 | 75.71 | 75.75 | 76.08 | |
| | 84.70 | 84.46 | 84.33 | 84.29 | 84.57 | 84.63 | 30 | 78.19 | 78.06 | 77.80 | 78.20 | 77.86 | 77.98 | |
| | 85.51 | 85.47 | 85.48 | 85.41 | 85.42 | 85.57 | 40 | 78.55 | 78.76 | 78.63 | 78.87 | 78.59 | 78.61 | |
| | 85.82 | 85.76 | 85.81 | 85.46 | 85.65 | 85.89 | 50 | 79.27 | 79.29 | 78.81 | 79.41 | 79.12 | 79.24 | |
| EPOCH = 2 | | | | | | | | | | | | | | |
| | 2 | 3 | 4 | 5 | 6 | 7 | NCF | 2 | 3 | 4 | 5 | 6 | 7 | |
| CNN-1-ReLU | 84.37 | 84.07 | 84.11 | 84.34 | 84.48 | 84.58 | 10 | 88.15 | 88.11 | 88.12 | 88.91 | 88.03 | 88.79 | CNN-1-RePSU |
| | 2089.10 | 89.09 | 89.23 | 88.94 | 89.15 | 89.58 | 20 | 93.35 | 91.42 | 89.45 | 91.29 | 94.11 | 90.40 | |
| | 3090.56 | 90.46 | 90.79 | 90.47 | 90.65 | 90.54 | 30 | 94.73 | 93.82 | 90.96 | 90.94 | 93.84 | 91.95 | |
| | 4090.93 | 90.99 | 90.86 | 91.22 | 90.95 | 91.11 | 40 | 94.49 | 88.68 | 91.55 | 88.59 | 92.46 | 94.54 | |
| | 5091.12 | 91.18 | 91.15 | 91.15 | 91.07 | 91.03 | 50 | 92.87 | 91.81 | 90.84 | 91.83 | 94.49 | **95.64** | |
| | 2 | 3 | 4 | 5 | 6 | 7 | NCF | 2 | 3 | 4 | 5 | 6 | 7 | |
| CNN-1-MISH | 86.74 | 86.54 | 86.76 | 86.28 | 86.51 | 86.43 | 10 | 73.67 | 74.18 | 73.47 | 74.28 | 74.52 | 74.37 | CNN-1-PMISH |
| | 91.96 | 92.11 | 91.99 | 92.20 | 92.00 | 91.95 | 20 | 76.85 | 77.06 | 77.37 | 76.92 | 77.03 | 77.07 | |
| | 93.44 | 93.43 | 93.62 | 93.66 | 93.51 | 93.51 | 30 | 77.90 | 77.28 | 77.61 | 77.40 | 77.53 | 77.64 | |
| | 94.14 | 94.07 | 94.07 | 94.15 | 94.12 | 94.10 | 40 | 77.70 | 77.61 | 77.25 | 77.59 | 77.04 | 77.37 | |
| | 94.09 | 94.30 | 94.44 | 94.34 | 94.40 | 94.29 | 50 | 76.80 | 76.63 | 77.11 | 76.95 | 76.59 | 76.82 | |
| | 2 | 3 | 4 | 5 | 6 | 7 | NCF | 2 | 3 | 4 | 5 | 6 | 7 | |
| CNN-1-SWISH | 86.30 | 86.60 | 86.47 | 86.34 | 86.58 | 86.37 | 10 | 80.30 | 80.39 | 80.78 | 80.54 | 80.24 | 80.69 | CNN-1-PSWISH |
| | 92.10 | 91.99 | 92.09 | 91.87 | 91.68 | 91.95 | 20 | 86.57 | 86.28 | 86.48 | 86.49 | 86.84 | 86.63 | |
| | 93.64 | 93.63 | 93.63 | 93.58 | 93.64 | 93.67 | 30 | 88.86 | 88.87 | 88.67 | 88.72 | 88.73 | 88.67 | |
| | 94.22 | 94.28 | 94.26 | 94.39 | 94.18 | 94.40 | 40 | 89.58 | 89.76 | 89.65 | 89.38 | 89.45 | 89.72 | |
| | 94.54 | 94.58 | 94.54 | 94.52 | 94.68 | 94.61 | 50 | 90.13 | 89.96 | 89.87 | 89.79 | 89.69 | 89.79 | |

Monte Carlo setups with respect to the sizing of a convolutional layer and the adjoined activation functions. Table 2 clearly shows that RePSU$^{\triangledown}$ learns very quickly (it reaches 95% on MNIST in 2 epochs), an advantage that can be exploited to help complex networks in faster discovery of satisfactory solutions.

**Findings:** it is well known that convolution filters can tend to reduce/blur informations and this has implied putting a lot of convolution filters per layer in modern CNNs for creating diversity. We see from Table 2 that unreasonably increasing the number of convolution filters of a layer is not a necessity: an alternative is to design a parametric activation functions that can penalize or amplify relevant informations.

### *3.2. Fast transfer learning as shallow parametric decision heads for deep CNNs*

From Subsection 3.1, we have seen, on a relatively simple classification task, that shallow RePSU$^{\triangledown}$ CNNs learn quickly, without the need of seeing the same example a hundred times. We now consider a more complex classification task to assess the relevance of learning parametric activation forms including the ReLU function as a special case: object recognition setup related to the CIFAR-10 challenge [18].

The CIFAR-10 dataset contains 60000 tiny images (size $32 \times 32 \times 3$) associated with 10 classes being: airplane, automobile, bird, cat, deer, dog, frog, horse, ship and truck. The dataset involves 6000 images per class, 50000 training images and 10000 test images. Additional information on the dataset can be found in the aforementioned reference.

It is worth noticing that CIFAR-10 challenge is running since a decade and several networks have been specifically designed to reach the highest reasonable performance on the associated classification task. We have now reached fewer than 6 classification errors over 1000 tested samples and this, from several transformer networks (see for instance [19] for a survey of classification networks and performance results on CIFAR-10).

Our goal is to compare learnable and non-learnable activation functions, without searching for a best-specific architecture. In this respect, we will not care on finding or reusing the best network for CIFAR-10. Instead, we aim to evaluate the CIFAR-10 performance of user-friendly state-of-the-art deep learning networks endowed with either non-parametric or parametric nonlinear activation heads. More precisely, we consider a shallow feature-based ensemble learning framework summarized in Table 3, where the input features are computed by using three base networks being EfficientNetB0 [20], NasNetMobile [21], and Xception [22], all being pre-trained on ImageNet [23]. These networks are very deep and consequently, we omit here their descriptions (the reader is asked to refer to the corresponding literature for more information). We only mention the feature layers feeding into the head of the ensemble model:

- EfficientNet-B0: layer called 'efficientnet-b0|model|head|MulLayer', located at operating position 286 and performing element-wise multiplication;
- NasNetMobile: depth concatenation layer called 'normal_concat_12' and located at operating position 908;
- Xception: layer called 'add_12', located at operating position 158 and performing element-wise addition.

Table 3: Shallow Ensemble Learning SEL-$Z$ frameworks where $Z \in \{\text{ReLU, PMISH, PSWISH, RePSU}^{\nabla}\}$ and the output sizes are given in parentheses. This framework we used implies that any parametric activation learner has $311680 \times N$ parameters, where $N$ is the number of parameters of the corresponding activation function. Specifically, nc is the number of output classes.

| Property | Processing | | |
|---|---|---|---|
| | Image input layer (32, 32, 3) | | |
| | Image resizing layer (299, 299, 3) | | |
| Non-learnable (base models) | EfficientNet-B0 'efficientnet-b0\|model\|head\| ...MulLayer' (9, 9, 1280) | NasNetMobile 'normal_concat_12' (10, 10, 1056) | Xception 'add_12' (10, 10, 1024) |
| Fusion | Concatenation (total: 311680 feature values) | | |
| Nonlinear learnable | ReLU or [RePSU$^{\nabla}$ or PMISH or PSWISH layer (311680)] | | |
| Linear learnable | Fully connected layer (nc) | | |
| | Softmax, Cross entropy | | |

where *operating position* refers to an enumeration of the distinct operators encapsulated per layer in a network. In this enumeration, we proceed from top to bottom and, from left to right when there exist parallel branches. The aim of the model concatenation described in Table 3 is providing a diversity of features (311680 activation values) to the nonlinear transfer forms. This amounts to using approximately a 558x558 image as input of the head models, which is reasonable.

Table 4 provides performance results on CIFAR-10 for a:

- one-pass ensemble learning (only one epoch for training), where the ensemble framework is that of Table 3;
- hyperparameter variation over:
  - minibatch sizes varying from 32 to 256 and
  - Monte Carlo trials with respect to initializations of the nonlinear activation parameters and the fully connected layer neurons.

Table 4: Accuracies in percentages for one CIFAR-10 pass training with respect to shallow SEL-$Z$ frameworks defined in Table 3 where $Z \in \{$ ReLU, PMISH, PSWISH, RePSU$^{\nabla}\}$.

| Network: | SEL-ReLU | SEL-PMISH | SEL-PSWISH | SEL-RePSU$^{\nabla}$ |
|---|---|---|---|---|
| Accuracy (%): | 86.78 | 85.48 | 83.31 | 94.80 |
| Time elapsed (mm:ss): | 06:48 | 06:58 | 06:50 | 07:08 |

The mean time elapsed per epoch (on an NVIDIA RTX A6000 card) is given in this table for information only: to avoid computation bottlenecks that can be due to the massive feature extraction step at any iteration, it has been necessary to write an optimized code by converting the CIFAR-10 dataset into "EfficientNet-NasNetMobile-Xception" feature vectors which are stored in separate files. The consequence is a huge gain in loading time on the GPU and a reduction in communications between GPU and CPU.

As it can be seen in Table 4, the penalized learning of activation parameters performed by RePSU$^{\triangledown}$ guaranty a straightforward learning in the sense of reaching 95% accuracy despite the fact that the examples are only seen once. It is worth noticing that the experimental computation time depends on many external parameters (tiny mini-batch load, server load, ...). Thus, time elapsed is provided for information only, to emphasize that the bottlenecks in computational complexity are not necessarily related to the nonlinearities adjoined.

The relevance of attenuating strong signals or amplifying weak signals (motivation of RePSU parameterization) can also be measured in a context where it is necessary to disregard class heterogeneity. For testing this, we have merged the CIFAR-10 classes to form 2 heterogeneous classes:

- Artificial creatures = {airplane, automobile, ship, truck};
- Natural creatures = {bird, cat, deer, dog, frog, horse}.

This dataset will be called CIFAR-10-BIN and the retrieval of man-made creatures from CIFAR-10-BIN leads to the performance results summarized in Table 5. This

Table 5: Accuracies in percentages for CIFAR-10's two-pass training with respect to shallow SEL-$Z$ frameworks defined in Table 3 where $Z \in \{$ ReLU, PMISH, PSWISH, RePSU$^{\triangledown}\}$.

| Network: | SEL-ReLU | SEL-PMISH | SEL-PSWISH | SEL-RePSU$^{\triangledown}$ |
|---|---|---|---|---|
| Accuracy: | 93.97 | 97.53 | 97.54 | 98.46 |

table confirms that for a given network configuration, inserting learnable activations increases transfer learning performance. Thus, it avoids going deeper in network architectures to achieve a desirable performance, knowing that the deeper the network, the less explicable this network is.

## 4. Learning activation parameters: deep CNNs

The first set of experiments presented in Section 3 has highlighted the contribution of RePSU in boosting performance through shallow learning. The second set of experiments presented in this section concerns the study of a potential RePSU contributions in deep learning from networks admitting millions of learnable parameters. It is obvious that in a very deep network, a rule of simplicity consists in alternating convolutional linearities with simple non-learnable activation functions. We will not question this rule here: the goal is not replacing convolutional parameters by RePSU ones, but presenting parametric forms which, when placed in a surgical way, will create the diversity that can boost learning or even improve the performance of classification. Therefore, only a single RePSU$^{\triangledown}$, RePSU$^{\triangle}$ or RePSU layer will be considered hereafter in CNNs, whatever the deepness of the CNN: additional nonlinearities will be composed by ReLU in order to limit the computational complexity of the framework.

### 4.1. *Efficient parametric decision heads for learning from scratch in deep CNNs*

The first experimental setup concerns a training from scratch to classify CIFAR-10 objects, when using the Deep Ensemble Learning (DEL) model described in Table 6. In contrast with the SEL transfer networks of Table 3, the base models imported in

Table 6: Deep Ensemble Learning DEL-$Z$ frameworks for $Z \in \{$ReLU, PMISH, PSWISH, RePSU$^{\nabla}\}$, any DEL-$Z$ having 1370 layers (operator-specific counting) and approximately 30 million learnable parameters. The abbreviations GAP and *eb0* are used to denote "Global Average Pooling" and "*efficientnet-b0*".

| Property | Processing | | |
|---|---|---|---|
| | Image input layer (32, 32, 3) | | |
| | Image resizing layer (299, 299, 3) | | |
| *Connection* | Split: one input node, 3 output nodes | | |
| **Structure** | EfficientNet-B0 | NasNetMobile | Xception |
| Root node | 'eb0\|model\|stem\|conv2d' | 'stem_conv1' | 'block1_conv1' |
| Terminal node | 'eb0\|model\|head\|MulLayer' | 'normal_concat_12' | 'block14_sepconv2_bn' |
| Output size | (9, 9, 1280) | (10, 10, 1056) | (10, 10, 2048) |
| Layers | 286 | 908 | 165 |
| Learnables | 4.0 million | 4.2 million | 20.8 million |
| **Activations** | $Z$ (1280) | $Z$ (1056) | $Z$ (2048) |
| Output sizes | (9, 9, 1280) | (10, 10, 1056) | $Z$ (10, 10, 2048) |
| **Pooling** | 2D GAP | 2D GAP | 2D GAP |
| Output size | (1, 1, 1280) | (1, 1, 1056) | (1, 1, 2048) |
| **Fusion** | FC (200, 1280) | FC (200, 1056) | FC (200, 2048) |
| | Concatenation, total: (4384, 1) feature vector) | | |
| **Fusion** | FC (10, 4384) | | |
| | Softmax, Cross entropy, 10 output classes | | |

Table 6 from EfficientNetB0, NasNetMobile and Xception are no longer in inference mode: they now involve learnable parameters. Experimental results are presented in Table 7.

Table 7: Accuracies in percentages for CIFAR-10 with respect to DEL-$Z$ frameworks defined in Table 6 where $Z \in \{$ ReLU, PMISH, PSWISH, RePSU$^{\nabla}\}$.

| Max Epochs | DEL-ReLU | DEL-PMISH | DEL-PSWISH | DEL-RePSU$^{\nabla}$ |
|---|---|---|---|---|
| 10 | 62,10 | 62,77 | 55.00 | 71.42 |
| 100 | 92,67 | 93,28 | 90.32 | 94.95 |

As it can be seen from Table 7, learning both linearities and RePSU$^{\nabla}$ nonlinearities in the deep CNN framework is, again, more efficient than using only a ReLU activation. Furthermore, a performance gain similar[9] to that of Table 7 is obtained in a two-pass training, when native parameters of the base DEL used in Table 6 are imported (transfer), instead of using random initializations.

### 4.2. *Efficient parametric base activations in learning from scratch for deep CNNs*

The second experimental setup deployed to analyze the contribution of RePSU for promoting performance in deep CNNs concerns texture analysis. We will use

[9]Performance: test accuracy of 94.59% for DEL-RePSU$^{\nabla}$ when using 2 training epochs.

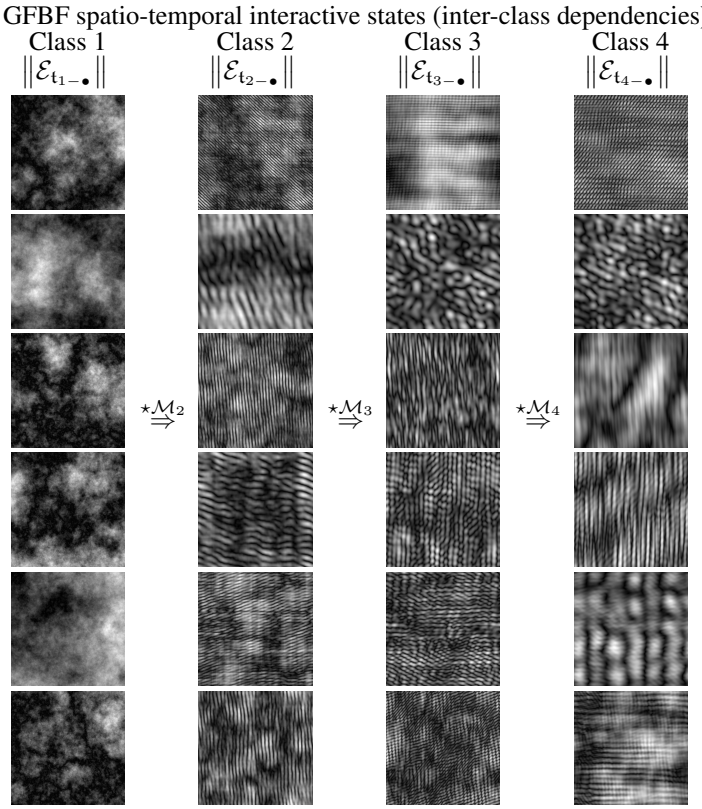


Figure 4: Sample elements of GFBF database $\mathcal{D}$. Fractional field evolution is governed by the temporal interactions which create inter-class statistical similarities: the learning system has to make abstraction with respect to these inter-class similarities and focus on intra-class ones.

a synthetic database [24] available at Mendeley data hyperlink where the concept of true class does not lead to any confusion[10].

This database is composed by Generalized Fractional Brownian Fields (GFBF, [25]). GFBF is a nonstationary random field $\mathcal{E}_{t_Q}$ associated with convolutions of interacting modulated fractional Brownian fields $(\mathcal{M}_q)_{q=1,2,\ldots,Q}$. Any of the modulated fractional Brownian fields is a long spatial memory process characterized by a given Hurst exponent and a singular spectral point. These models make synthesis of evolution fields with rich structural content possible by using a series of spatial convolutions (linearities) and shift/modulation operators (nonlinearities). GFBF are considered hereafter in an interaction framework where $Q$ is associated with the number of different modulated Brownian fields in interaction. Examples of evolution factors and synthesized fields are given in Figure 4 when the GFBF involves respectively $Q = 1, 2, 3$ and $4$ interactions.

[10]Expert based labeling is far from being perfect, except in certain trivial contexts.

The problem addressed is then the design of a system capable of learning the evolution factor $Q$, given an arbitrary GFBF field $\mathcal{E}$. A total of 4800 GFBF images (1200 images per class) is considered hereafter, when the number $Q$ of interacting modulated Brownian fields pertains to the class labels $\{1, 2, 3, 4\}$, this parameter $Q$ defining the class property. For any class, poles and Hurst parameters have been generated randomly. An overview of the intricacy of the concept of class associated with this database is shown in Figure 4, where textures pertaining to the same column pertain to the same class: this figure shows interclass dependency, a scenario that limits learning capabilities as confusion is possible between intra-class similarities (number of interactions) and inter-class similarities (remaining dependencies after field evolution). Such a challenging classification problem justifies the use of a deep CNN framework.

Table 8: Deep CNN-2-$Y$ frameworks where $Y \in \{$ ReLU, RePSU, MISH, PMISH, SWISH, PSWISH $\}$ corresponds to the specific activation function used in layer 4. FC denotes a *Fully Connected* layer. We have considered $N_2 = 96$ for CNN-2.

| Layer | Content | $\#N$ of Elements | Element size | Learnable |
|---|---|---|---|---|
| 1 | 'Inputs' (images) | $N_1$ | $M_1^x \times M_1^y \times M_1^c$ | No |
| 2 | 'Convolve-1' | $N_2$ | $3 \times 3$ | Yes |
| 3 | 'Normalize-1' | Standard / Mini-batch | | |
| 4 | ReLU | $N_2$ | - | No |
| | RePSU$^{\nabla}$/RePSU$^{\Delta}$/RePSU | | 3 / 3 / 4 | Yes |
| | MISH | | - | No |
| | PMISH | | 1 | Yes |
| | SWISH | | - | No |
| | PSWISH | | 1 | Yes |
| 5 | 'Convolve-2' | 128 | $5 \times 5$ | Yes |
| 6 | 'Normalize-2' | Standard / Mini-batch | | |
| 7 | 'ReLU' | | 3 | No |
| 8 | 'Convolve-3' | 384 | $7 \times 7$ | Yes |
| 9 | 'Normalize-3' | Standard / Mini-batch | | |
| 10 | 'ReLU' | | 3 | No |
| 11 | 'Convolve-4' | 192 | $5 \times 5$ | Yes |
| 12 | 'Normalize-4' | Standard / Mini-batch | | |
| 13 | 'ReLU' | | 3 | No |
| 14 | 'Convolve-5' | 128 | $3 \times 3$ | Yes |
| 15 | 'Normalize-5' | Standard / Mini-batch | | |
| 16 | 'ReLU' | | 3 | No |
| 17 | 'FC-1' | [Output size: 4096 ] | | |
| 18 | 'ReLU' | 4096 | 3 | No |
| 19 | 'FC-2' | [Output size: $L$ ] | | |
| 20 | 'Softmax' | Probability with respect to $L$ outputs | | |
| 21 | 'Classify' | Cross-entropy (Output: category) | | |

When using 800 textures for learning and 400 for validation per class and when learning from RePSU and ReLU based networks of Table 8, then the corresponding validation losses and accuracies are given in Table 9. Similarly to the handwritten digit

recognition results of Section 3.1 and in comparison with the standard CNN paradigm associated with non-learnable activations (ReLU, MISH, SWISH), the learnable activation frameworks show higher performance in general and RePSU based CNN outperforms these CNNs in terms of faster convergence to a desirable solution (increase of the validation accuracy) and the decrease validation loss.

Table 9: Mean validation loss and mean validation accuracy every ten epochs for the GFBF class identification issue with respect to RePSU and ReLU based deep CNN presented in Table 8.

| Max Epochs | 5 | 10 | 15 | 20 | 25 | 50 | 75 | 100 | |
|---|---|---|---|---|---|---|---|---|---|
| | Validation accuracy | | | | | | | | Time elapsed (hh:mm:ss) |
| CNN-2-ReLU | 53.75 | 59.90 | 60.94 | 64.58 | 63.75 | 58.13 | 58.13 | 58.96 | 06:17:10 |
| CNN-2-MISH | 54.06 | 58.33 | 60.42 | 62.92 | 69.38 | 71.25 | 66.15 | 66.25 | 06:35:30 |
| CNN-2-PMISH | 39.58 | 48.12 | 49.38 | 60.31 | 65.31 | 61.88 | 71.04 | 68.54 | 07:18:37 |
| CNN-2-SWISH | 44.17 | 52.40 | 60.31 | 61.35 | 62.60 | 62.81 | 64.17 | 60.73 | 06:25:57 |
| CNN-2-PSWISH | 46.25 | 56.98 | 62.81 | 63.75 | 67.81 | 60.42 | 69.58 | 69.58 | 06:30:53 |
| RePSU (general), RePSU$^{\nabla}$ ($\alpha = 0$) and RePSU$^{\Delta}$ ($\alpha = 1$) | | | | | | | | | |
| CNN-2-RePSU$^{\nabla}$ | 50.10 | 52.60 | 67.08 | 68.02 | 74.69 | 70.83 | 71.77 | 72.40 | 07:25:43 |
| CNN-2-RePSU$^{\Delta}$ | 45.31 | 60.94 | 59.38 | 62.50 | 71.88 | 72.29 | 70.83 | 71.56 | 07:54:28 |
| CNN-2-RePSU | 48.54 | 62.19 | 58.96 | 69.79 | 71.77 | 73.54 | 73.65 | 74.27 | 08:59:29 |

## 5. Discussion: limitations and open issues

### *5.1. RePSU*

We can reasonably expect to improve RePSU based CNN performance by taking more RePSU layers into account. However, computational complexity then explodes and the best strategy has been a combination of RePSU and ReLU: such a combination creates sufficient activation diversity while limiting both the computational time and feature space exploration. One can finally note that RePSU's parameter $\beta$ is very sensitive and difficult to optimize in practice: only this parameter has been set to 1 during the experiments. A specific updating strategy, requiring very small gradient increments, needs to be developed for learning an optimal estimate of this parameter.

### *5.2. PSWISH and PMISH*

The main issue raised by PSWISH (defined in [13] for the parametric form and in [15] for the non-parametric form) is the fact that PSWISH output is not 0 even for very large negative inputs. This implies well-known limitations associated (similar to those of the sigmoid) in terms of very small but non-null gradients. PMISH [12] suffers from the same default as is it non-zero almost everywhere. Open prospect regarding learnable PSWISH and PMISH imply not only adding a few more parameters, but also rectifying the output so as to focus on significant values. This can solve vanishing gradient issues for PSWISH and PMISH when $\xi$ tends to zero or is initialized close to zero.

## 6. Conclusion

In this work, we have proposed a family of nonlinear transfer functions, the RePSU functions. These functions are constructed to inherit from best qualities of ReLU and SSBS functions. RePSU based CNN involves learning nonlinear activation weights because parametric transfer forms have been considered. The experimental results show that RePSU functions can be used to create a diversity of activations in a CNN or to achieve a higher learning performance for a classification task.


### ACKNOWLEDGEMENT

The authors address special thanks to the anonymous reviewers of the *Elsevier Neural Networks Journal* for their constructive comments and insightful advices. The authors are very grateful to the C.N.E.S. (Centre National d'Études Spatiales, France) for the support provided thanks to SITS-DEEP and SHARE projects. The authors also wish to thank the ANR (Agence Nationale de la Recherche, France) for the support associated with the grant PRCE C2R-IA. Numerical simulations have been performed thanks to the facilities offered by MUST computing center of Université Savoie Mont Blanc.


## References


[1] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," in *Proceedings of the 27th International Conference on International Conference on Machine Learning*, ser. ICML'10. Madison, WI, USA: Omnipress, 2010, pp. 807–814.

[2] Y.-G. Yoon, P. Dai, J. Wohlwend, J.-B. Chang, A. H. Marblestone, and E. S. Boyden, "Bvlc alexnet model," *https://github.com/BVLC/caffe/tree/master/models/bvlc_alexnet*, 2015.

[3] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *CoRR*, vol. abs/1409.1556, 2014.

[4] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[5] C. Szegedy, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015, pp. 1–9.

[6] J. Carreira and A. Zisserman, "Quo vadis, action recognition? a new model and the kinetics dataset," in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, vol. 00, July 2017, pp. 4724–4733.

[7] W. Shi, Y. Gong, X. Tao, and N. Zheng, "Training dcnn by combining max-margin, max-correlation objectives, and correntropy loss for multilabel image classification," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 29, no. 7, pp. 2896–2908, July 2018.

[8] S. Lin, R. Ji, Y. Li, C. Deng, and X. Li, "Toward compact convnets via structure-sparsity regularized filter pruning," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 31, no. 2, pp. 574–588, Feb 2020.

[9] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," in *The IEEE International Conference on Computer Vision (ICCV)*, December 2015.

[10] W. Little and G. L. Shaw, "Analytic study of the memory storage capacity of a neural network," *Mathematical Biosciences*, vol. 39, no. 3, pp. 281 – 290, 1978. [Online]. Available: http://www.sciencedirect.com/science/article/pii/0025556478900585

[11] J. Han and C. Moraga, "The influence of the sigmoid function parameters on the speed of backpropagation learning," in *From Natural to Artificial Neural Computation*, J. Mira and F. Sandoval, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 1995, pp. 195–201.

[12] D. Misra, "Mish: A self regularized non-monotonic activation function," 2020.

[13] P. Ramachandran, B. Zoph, and Q. V. Le, "Searching for activation functions," 2017.

[14] A. M. Atto, D. Pastor, and G. Mercier, "Smooth sigmoid wavelet shrinkage for non-parametric estimation," *IEEE International Conference on Acoustics, Speech, and Signal Processing, ICASSP,* Las Vegas, Nevada, USA, 30 march - 4 april, 2008.

[15] D. Hendrycks and K. Gimpel, "Gaussian error linear units (gelus)," 2020.

[16] A. Antoniadis, "Wavelet methods in statistics: Some recent developments and their applications," *Statistics Surveys*, vol. 1, pp. 16 – 55, 2007.

[17] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. G. Murray, B. Steiner, P. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu, and X. Zheng, "Tensorflow: A system for large-scale machine learning," in *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation*, ser. OSDI'16. Berkeley, CA, USA: USENIX Association, 2016, pp. 265–283. [Online]. Available: http://dl.acm.org/citation.cfm?id=3026877.3026899

[18] A. Krizhevsky, "Learning multiple layers of features from tiny images," Tech. Rep., 2009.

[19] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby, "An image is worth 16x16 words: Transformers for image recognition at scale," in *International Conference on Learning Representations*, 2021. [Online]. Available: https://openreview.net/forum?id=YicbFdNTTy

[20] M. Tan and Q. V. Le, "Efficientnet: Rethinking model scaling for convolutional neural networks," 2020.

[21] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le, "Learning transferable architectures for scalable image recognition," 2018.

[22] F. Chollet, "Xception: Deep learning with depthwise separable convolutions," in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017, pp. 1800–1807.

[23] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "Imagenet large scale visual recognition challenge," *Int. J. Comput. Vision*, vol. 115, no. 3, pp. 211–252, Dec. 2015.

[24] A. M. Atto, "Generalized fractional brownian texture dataset," *Mendeley Data*, vol. V1, 2020.

[25] A. M. Atto, Z. Tan, O. Alata, and M. Moreaud, "Non-stationary texture synthesis from random field modeling," in *IEEE International Conference on Image Processing (ICIP)*, Oct 2014, pp. 4266–4270.