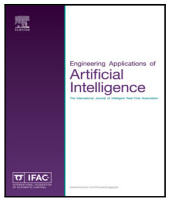
Engineering Applications of Artificial Intelligence 120 (2023) 105863

Contents lists available at ScienceDirect

Engineering Applications of Artificial Intelligence

journal homepage: www.elsevier.com/locate/engappai



Empirical study of the modulus as activation function in computer vision applications

Iván Vallés-Pérez *, Emilio Soria-Olivas, Marcelino Martínez-Sober, Antonio J. Serrano-López, Joan Vila-Francés, Juan Gómez-Sanchís

IDAL, Intelligent Data Analysis Laboratory, University of Valencia, Avenida de la Universitat s/n 46100 Burjassot, Valencia, Spain

ARTICLE INFO

Keywords:
Deep learning
Activation functions
Optimization

ABSTRACT

In this work we propose a new non-monotonic activation function: the *modulus*. The majority of the reported research on nonlinearities is focused on monotonic functions. We empirically demonstrate how by using the *modulus* activation function on computer vision tasks the models generalize better than with other nonlinearities — up to a 15% accuracy increase in *CIFAR100* and 4% in *CIFAR10*, relative to the best of the benchmark activations tested. With the proposed activation function the vanishing gradient and dying neurons problems disappear, because the derivative of the activation function is always 1 or −1. The simplicity of the proposed function and its derivative make this solution specially suitable for *TinyML* and hardware applications.

1. Introduction

The core piece of all deep learning models is the activation function. They enable the models to produce non-linear abstract representations when applying linear transformations in cascade (Goodfellow et al., 2016). The most common choice in modern deep learning models is the Rectified Linear Unit (*ReLU*) (Nair and Hinton, 2010). Among other advantages, *ReLU* nonlinearities allowed us to train deeper neural networks (Xu et al., 2015).

Apart from the *ReLU* in the last years there have appeared many alternative activation functions (Dubey et al., 2022). The following studies represent some of the most popular examples: *Leaky-ReLU* and *PR-ReLU* (Xu et al., 2015), *ELU* (Clevert et al., 2016), *Swish* (Ramachandran et al., 2018), *SELU* (Klambauer et al., 2017), *F/PFLU* (Zhu et al., 2021), *RSigElu* (Kiliçarslan and Celik, 2021) etc. The authors of Agostinelli et al. (2014) studied how to learn adaptive piecewise linear activation functions. All the mentioned studies propose functions that have the following properties in common: nonlinearity, continuity, differentiability and low computational cost (being *Swish* the most computationally expensive option). The majority of the activation functions are monotonic, *Swish*, *GELU* (Hendrycks and Gimpel, 2016), *S-ReLU* (Jin et al., 2016) and *Mish* (Misra, 2019) are some of the most popular exceptions. Despite the large pool of alternatives, *ReLU* still appears as the default choice in many applications due to its simplicity and low computational cost (Nair and Hinton, 2010).

In this work, we investigate the use of the *modulus* function, also known as the absolute value function, as an activation function for deep learning models. Previous research (Karnewar, 2018) has hinted at the potential of the modulus nonlinearity in the design of a specific neural network architecture, but its performance has not been systematically compared to other nonlinearities. Through empirical evaluation, we provide empirical evidence proving that the modulus activation function allows deep learning models to converge to better solutions in *CIFAR10*, *CIFAR100* and *MNIST* datasets. Compared with other modern non-monotonic nonlinearities such as *Mish*, *PFLU* and *FPFLU* or *Swish*, the *modulus* has a very low computational cost (equivalent to *ReLU*). This property makes the *modulus* specially useful for *TinyML* and hardware applications (Sanchez-Iborra and Skarmeta, 2020) and hardware neural networks (Misra and Saha, 2010).

The contributions of this paper are: (1) we propose the modulus activation function, (2) the proposed function empirically shows significantly better results in 75% of our experiments, (3) we contribute to the new trend of non-monotonic activation functions with another example showing good results in the same direction, (4) the simplicity of the modulus activation function makes it very suitable for hardware applications and *TinyML*, and (5) we propose two smooth approximation to the *modulus* function, one of them achieving even superior results than the original *modulus*.

This paper is organized as follows. Section 2 describes the proposed activation function and the ones used for benchmarking. Section 3 describes the experiments conducted and the results achieved. Finally, we summarize our contribution in Section 5, enumerating the main conclusions.

* Correspondence to: Escola Tècnica Superior d'Enginyeria, University of Valencia, Avenida de la Universitat s/n 46100 Burjassot, Valencia, Spain.
E-mail addresses: ivan.valles@uv.es (I. Vallés-Pérez), emilio.soria@uv.es (E. Soria-Olivas), marcelino.martinez@uv.es (M. Martínez-Sober), antonio.j.serrano@uv.es (A.J. Serrano-López), joan.vila@uv.es (J. Vila-Francés), juan.gomez-sanchis@uv.es (J. Gómez-Sanchís).

https://doi.org/10.1016/j.engappai.2023.105863
Received 7 March 2022; Received in revised form 22 December 2022; Accepted 13 January 2023
Available online 18 January 2023

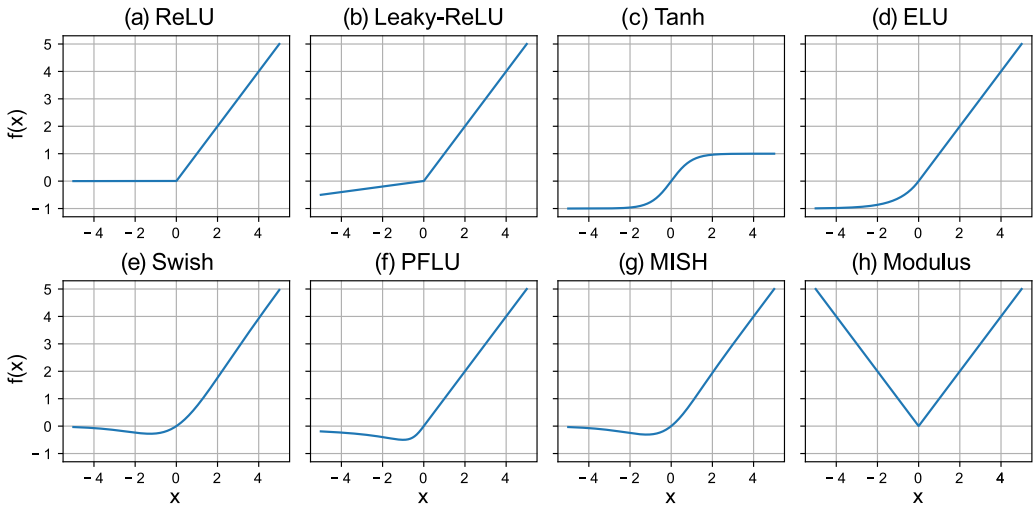


Fig. 1. Nonlinearities used along this study. (a–g) are benchmark activation functions that showed good performance in previous studies. (h) is the *modulus* activation function proposed in this study. For this example, $\beta = 10.0$, $\beta = 1.0$ and $\beta = 1.0$ have been used for the *Leaky-ReLU*, *ELU* and *Swish* activation functions, respectively.

2. Methods

2.1. Modulus activation function

In this section we introduce our proposed activation function, the *modulus*: $f(x) = |x|$. This nonlinearity is a continuous, piecewise-linear function consisting of an identity mapping for positive values of x, and a negative identity mapping for negative values of x. Its derivative (defined below), as well as the modulus function itself, are hardware-level bit-size operations. This is extremely useful for hardware implementations. Besides, the modulus function can be expressed as $f(x) = \text{sgn}(x) \cdot x$, where $f'(x) = \text{sgn}(x)$, and hence $f(x) = f'(x) \cdot x$. In practice, this means that the derivative can be calculated as part of the forward pass and cached for the *backpropagation*, reducing the total computation. This form is also specially useful in hardware implementations, given that the whole activation function and its derivative is fully represented as a bit-size operation (the sign function). In hardware, the memory and computing requirements are limited, hence having mechanisms that allow for cheap computations (1-bit in the modulus activation) and reusability/caching (part of the gradients computed during the forward pass, in our case) is crucial.

$$f'(x) = \text{sgn}(x) \begin{cases} 1 & \text{if} \quad x > 0 \\ -1 & \text{if} \quad x < 0 \end{cases} \tag{1}$$

Notice that the *modulus* function is differentiable everywhere except in $x = 0$. For practical purposes, we define $f'(0) = 1$ so that the derivative is defined for all the range of x values, similar to the case of *ReLU* (Goodfellow et al., 2016), and to ensure that the norm of the gradient is constant for all values of x. See Fig. 1h for a graphical representation.

The *modulus* activation function belongs, in essence, to the family of rectifier functions (Glorot et al., 2011) . In fact, it is equivalent to a *Leaky-ReLU* with $\beta = -1$ (see Eq. (4)). However *Leaky-ReLU* was originally defined to take values of beta strictly higher than 1 (Xu et al., 2015). Furthermore, we empirically show that the *modulus* achieves significantly superior results than the *Leaky-ReLU*.

The benefit of this activation function with respect to the other rectifiers is that the norm of its gradient is constant ($\|\nabla_x f\| = 1 \quad \forall x$) and hence, interestingly, it does not depend on the value of x. This property is desirable when optimizing the parameters of a neural network with gradient descent algorithms, as there are no input values for which the neuron saturates or explodes (Glorot and Bengio, 2010) (i.e. values of x for which the gradient of the activation function is close to zero, or extremely large). This naturally removes the dying neurons (Lu, 2020) and vanishing gradient problems (Pascanu et al., 2013; Hochreiter, 1998; Hochreiter et al., 2001), which usually appears in activations with zero regions (such as *ReLU*) or with asymptotically saturating regions (such as *tanh*), respectively.

2.2. Smooth approximations of the modulus function

The *modulus* function is not differentiable when $x = 0$. To study if this property harms the performance of the models in any way, two alternative smooth approximations of the *modulus* function have been tested as an additional experiment. See Fig. 2 for a visual representation. The two approximations are defined below.

- Quadratic approximation: referred subsequently as *SoftModulusQ* and defined in Eq. (2) (full derivation in the appendix).

$$f(x) = \begin{cases} x^2 \cdot (2 - |x|) & \text{if} \quad |x| \leq 1 \\ |x| & \text{if} \quad |x| > 1 \end{cases} \tag{2}$$

- Hyperbolic tangent approximation: referred subsequently as *SoftModulusT* and defined in Eq. (3). Starting from the modulus function $f(x) = sgn(x) \cdot x$, we approximate the sgn function as follows $sgn(x) \approx \tanh(x/\beta)$, where $\beta \in [0, 1]$ is a tunable hyperparameter that controls how acute the "V" transition is. The smaller the β, the closest the approximation is to the *modulus*. We used a value of $\beta = 0.01$ in all the models we trained.

$$f(x) = x \cdot \tanh(x/\beta) \tag{3}$$

2.3. Benchmark activation functions

The following activation functions have been used as a benchmark to compare the performance of the proposed one.

- *Tanh*: the hyperbolic tangent has been one of the most popular choices (together with the *sigmoid*), before *ReLU* was proposed (LeCun et al., 2012). Among many other desirable properties, its derivative is very simple: $(\tanh x)' = 1 - \tanh^2 x$. This property was very beneficial specially before automatic differentiation tools appeared. Besides, similar to our proposed activation function, the derivative can be easily calculated during forward pass and cached for the backward pass.
- *ReLU*: this activation function was published in 2010 as an alternative to train *Restricted Boltzmann Machines*. It is defined as $f(x) = \max(0, x)$. It allowed training deeper neural networks by solving the vanishing gradient problems typically happening with

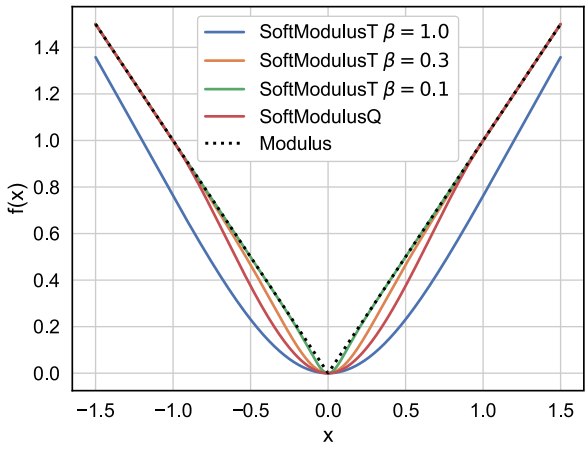


Fig. 2. Representation of the two *SoftModulus* activation functions compared with the original *modulus*: *SoftModulusQ* and *SoftModulusT*.

saturating activation functions. As it can be seen in the formula, ReLU outputs zero if the input is negative. This brings sparsity to the non-linear representation, at the cost of potentially finding optimization problems due to the fact that the derivative when $x < 0$ is zero (dying neurons, for instance).

- *Leaky-ReLU*: the *Leaky-ReLU* attempted to solve the problem known as *dying neurons* in the *ReLUs* by adding a small linear term in the negative side of x (see Eq. (4), where $\beta > 1$ is a hyperparameter to be tuned).

$$f(x) = \max(x/\beta, x) \tag{4}$$

- *ELU*: this is a smooth version of *ReLU* that approaches asymptotically to -1 as $x \to -\infty$. Unlike ReLUs, it is differentiable everywhere and can produce negative values. This activation function is formally defined in Eq. (5), where β is a tunable hyperparameter.

$$f(x) = \begin{cases} x & \text{if} \quad x > 0 \\ \beta(e^x - 1) & \text{if} \quad x \leq 0 \end{cases} \tag{5}$$

- *Swish*: this is one of the few non-monotonic activation functions formally published. It gained its popularity because it showed promising results in multiple applications. It is defined as $f(x) = x \cdot \text{sigmoid}(\beta x)$, where β is a hyperparameter to be tuned.
- *Mish*: this is a smooth, non-monotonic and self-regularized activation function, similar to swish, showing superior performance in different benchmarks. It is defined as $f(x) = x \cdot \tanh(\text{softplus}(x))$, where $\text{softplus}(x) = \log(1 + e^x)$ (Dugas et al., 2001).
- *PFLU*: the Power Function Linear Unit (PFLU) is a non-monotonic activation function published on 2020 that showed good performance in convolutional architectures. It is defined as follows: $f(x) = x \cdot \frac{1}{2}\left(1 + \frac{x}{\sqrt{1+x^2}}\right)$.

3. Experiments and results

3.1. Setup

For the following experiments we have used *CIFAR10*, *CIFAR100* (Krizhevsky, 2009) and *MNIST* (LeCun and Cortes, 2010) datasets. *CIFAR10* and *CIFAR100* images are labeled into 10 and 100 classes, respectively. They are of size 32×32 and full color. In both cases, the data sets contain 50,000 images for training and 10,000 images for testing purposes. *MNIST* images belong to one of 10 classes and are in grey scale and 28×28 size. *MNIST* comes with 50,000 images for training and 10,000 images for testing purposes.

The images of the datasets have been normalized so that the minimum and the maximum values are -1 and 1. We have kept untouched the default train/test split provided by *Pytorch* (Paszke et al., 2019), in order to facilitate future potential reproducibility and benchmarking efforts.

Four different architectures have been used to test the performance of the *modulus* activation function against the other nonlinearities. These architectures consist of a multilayer perceptron (named *fully connected*), two convolutional architectures with 2 and 6 convolutional layers (named *conv2* and *conv6* respectively) and the *VGG-16* network (Simonyan and Zisserman, 2015). They are described in more detail in Table 1. The architectures choice has been inspired on the *Lottery Ticket Hypothesis* paper (Frankle and Carbin, 2019), however no pruning methods have been applied in this study.

We have followed the same experimental setup across models, activations and datasets. All the networks have been trained for 100 epochs, and the best accuracy of each run has been reported. Each run has been repeated 30 times with different random weight initializations. No *dropout* (Srivastava et al., 2014) nor *batch normalization* (Ioffe and Szegedy, 2015) have been used. We used *Adam* (Kingma and Ba, 2014) as optimizer, with a learning rate of 10^{-4}, a *gradual warmup* (Gotmare et al., 2019) during the first 5 epochs starting from 10^{-5}, and a *cosine annealing* (Loshchilov and Hutter, 2017) with a target learning rate of 10^{-6} in the last epoch. We used *Python* 3.7.3, *Pytorch* 1.7.1 and *TorchVision* 0.8.2, and all the models have been trained in a single *Nvidia 2080ti* graphics card. The code used can be found in the url of the footnote.[1]

3.2. Classification performance

Table 2 summarizes the results of the classifiers for *CIFAR10*, *CIFAR100* and *MNIST* datasets. As we can see from the tables, the *modulus* activation function outperforms significantly the benchmark activations in 9 out of 12 experiments (4 architectures x 3 datasets). In 4 of these cases, the accuracy improvement is 3% or higher, in relative terms (CIFAR10-Conv2, CIFAR10-Conv6, CIFAR100-Conv2 and CIFAR100-Conv6 with p-value $< 10^{-6}$ in all the cases). If we consider the soft approximations of the modulus activation function, we see that the *SoftModulusT* significantly outperforms the benchmark in 10 out of 12 experiments (with p-value of $6.55 \cdot 10^{-6}$ for MNIST-Conv6, $1.29 \cdot 10^{-2}$ for MNIST-VGG16, $3.99 \cdot 10^{-4}$ for CIFAR100-VGG16 and $< 10^{-6}$ for the rest). All the p-values reported in this study have been obtained using a *Wilcoxon one-sided Rank Sum* test comparing the modulus results against the best of the benchmark activation functions in each case.

Fig. 3 shows the test accuracy at every epoch for the 12 experiments (the curves of the smooth approximations have been included in Fig. 4 to compare them with the *modulus*). As it can be noticed in several cases (e.g. *CIFAR10-Conv2*, *CIFAR10-Conv6*, *CIFAR100-Conv6*), the accuracy of the networks with *modulus* activation function keeps increasing at the 100th epoch while the benchmarks have already stabilized. That observation suggests that, if trained for longer, probably the accuracy would increase even more.

The results of the *SoftModulus* approximations compared with the *modulus* activation function are summarized in Table 3. Additionally, Fig. 4 shows the test accuracy at every epoch for the same activations. From these results, we see that although the *SoftModulusQ* approximation achieves significantly better results than the original modulus in several cases (*FC* model over *CIFAR10* and *MNIST* dataset, and *Conv6* over *MNIST* dataset), it seems to be much more unstable: on VGG16, the gradients of the models with *SoftModulusQ* activations vanish at the beginning of the training process due to the fact that the weights are initialized very close to zero and the gradients are too small, causing numerical problems. This problem may be solved tweaking the initialization. However, the *SoftModulusT* approximation seems to be on-par with the original *modulus* in the majority of cases, while it perform significantly better for deep architectures like *VGG16*.

[1] https://github.com/ivallesp/abs

Table 1
Deep learning architectures used to experiment with different activations. In the dense layers row, the output size has been represented as *C*. The Fully connected architecture (*FC*) is a multilayer perceptron with 2 hidden layers + the output layer. The *Conv2* and *Conv6* architectures are shallow variants of *VGG* described here: (Simonyan and Zisserman, 2015). The last pooling layer of the *VGG-16* original architecture has been trimmed in order to allow this model to work with smaller image sizes.

Network	Fully connected	Conv2	Conv6	VGG-16
Conv layers		64, 64, pool	64, 64, pool 128, 128, pool 256, 256, pool	64, 64, pool 128, 128, pool 256, 256, 256,pool 512, 512, 512, pool 512, 512, 512
Dense layers	256, 256, *C*	256, 256, *C*	256, 256, *C*	4096, 4096, *C*
Filter sizes		3 × 3	3 × 3	3 × 3
Pooling type		max	max	max
# Parameters	269 k–878 k	3.3 M–4.3 M	1.8 M–2.3 M	33.6 M–40.3 M
Size	3.1–11 MB	38–50 MB	21–27 MB	385–462 MB

Table 2
Test accuracy for all the datasets and activation functions tested (rows), and for all the models (columns). The results are expressed as mean ± standard deviation across the 30 random initializations. To facilitate the reading of the table, for each dataset-model combination, we have colored the results as follows: the best result (highest accuracy) has been colored in blue, the second in green, the third in yellow and the fourth in red (first > second > third > fourth). Additionally, we marked in bold those cases where any of the proposed activation functions achieved significantly higher accuracy than the benchmarks, with a significance level of $\alpha = 0.05$, for which we used a *Wilcoxon one-sided Rank Sum* test.

Dataset	Activation	FC	Conv2	Conv6	VGG16
CIFAR10	ReLU	54.65 ± 0.22	71.40 ± 0.26	77.09 ± 1.21	83.66 ± 0.41
	LeakyReLU	54.71 ± 0.24	71.65 ± 0.32	77.38 ± 1.13	83.98 ± 0.34
	Tanh	49.95 ± 0.23	67.46 ± 0.34	77.67 ± 0.24	79.69 ± 0.26
	Swish	55.39 ± 0.29	69.09 ± 0.27	71.66 ± 0.62	80.77 ± 0.50
	ELU	55.16 ± 0.21	69.34 ± 0.34	78.98 ± 0.33	81.21 ± 0.37
	PFLU	55.43 ± 0.30	70.34 ± 0.33	80.77 ± 0.40	81.58 ± 0.38
	Mish	55.44 ± 0.22	69.66 ± 0.35	75.66 ± 0.53	80.98 ± 0.64
	Modulus	53.97 ± 0.24	**73.93 ± 0.42**	**84.22 ± 0.29**	**84.86 ± 0.32**
	SoftModulusQ	54.07 ± 0.29	71.49 ± 0.37	**81.01 ± 1.27**	10.00 ± 0.00
	SoftModulusT	54.04 ± 0.24	**73.95 ± 0.40**	**84.36 ± 0.28**	**85.34 ± 0.36**
CIFAR100	ReLU	27.33 ± 0.25	36.72 ± 0.31	36.35 ± 0.89	44.61 ± 1.11
	LeakyReLU	27.24 ± 0.22	37.03 ± 0.40	37.15 ± 0.77	45.19 ± 1.47
	Tanh	23.63 ± 0.20	35.29 ± 0.47	42.15 ± 0.49	44.14 ± 0.37
	Swish	27.59 ± 0.25	35.20 ± 0.34	35.75 ± 0.41	46.02 ± 1.10
	ELU	27.92 ± 0.26	35.68 ± 0.31	40.74 ± 0.48	47.63 ± 0.71
	PFLU	27.73 ± 0.21	37.51 ± 0.42	42.25 ± 0.45	48.22 ± 0.63
	Mish	27.68 ± 0.25	36.04 ± 0.41	37.63 ± 0.75	48.69 ± 0.69
	Modulus	26.29 ± 0.26	**38.66 ± 0.56**	**48.73 ± 0.62**	45.83 ± 0.80
	SoftModulusQ	26.23 ± 0.25	37.48 ± 0.44	**48.16 ± 1.97**	1.00 ± 0.00
	SoftModulusT	26.32 ± 0.24	**38.69 ± 0.56**	**48.63 ± 0.83**	48.47 ± 0.68
MNIST	ReLU	98.35 ± 0.07	99.27 ± 0.04	99.53 ± 0.03	99.58 ± 0.04
	LeakyReLU	98.37 ± 0.06	99.27 ± 0.04	99.53 ± 0.03	99.58 ± 0.03
	Tanh	98.34 ± 0.07	99.06 ± 0.05	99.48 ± 0.04	99.48 ± 0.04
	Swish	98.36 ± 0.05	99.24 ± 0.04	99.52 ± 0.03	99.53 ± 0.03
	ELU	98.31 ± 0.04	99.16 ± 0.04	99.54 ± 0.03	99.54 ± 0.03
	PFLU	98.42 ± 0.05	99.21 ± 0.04	99.56 ± 0.03	99.57 ± 0.03
	Mish	98.41 ± 0.05	99.23 ± 0.04	99.56 ± 0.03	99.57 ± 0.04
	Modulus	**98.47 ± 0.07**	**99.38 ± 0.04**	**99.60 ± 0.03**	**99.63 ± 0.04**
	SoftModulusQ	**98.51 ± 0.06**	**99.37 ± 0.03**	**99.62 ± 0.03**	11.35 ± 0.00
	SoftModulusT	**98.47 ± 0.06**	**99.39 ± 0.04**	**99.61 ± 0.03**	**99.62 ± 0.03**

We hypothesize that the difference between the two approximations is due to the width of the zero gradient region around $x = 0$: wider regions lead to more training problems (see Fig. 2). The β parameter in the hyperbolic tangent approximation allows for easily adjust this zero-gradient region. We informally tested different values of beta concluding that for high values of β (e.g. $\beta = 1.0$) the model struggles to train due to numerical precision problems (small values of weights lead to tiny gradients). A value of $\beta = 0.01$ seems to work well for all the tested experiments. Finally, we see that the *SoftModulusT* achieves superior results than the original *modulus* when used in the *VGG16* architecture.

4. Discussion

The *modulus* activation function has shown promising results in certain contexts, but it is worth discussing potential criticisms of this approach.

One concern may be that the *modulus* activation does not seem to be biologically inspired by the way in which axon hillocks fire in biological neurons. Despite this, the primary goal of an activation function is to introduce non-linearity into the neural networks, and the *modulus* activation is effective in achieving this. Additionally, the *modulus* produces non-zero gradients on the negative side, which can be beneficial to avoid vanishing gradients and dying neurons.

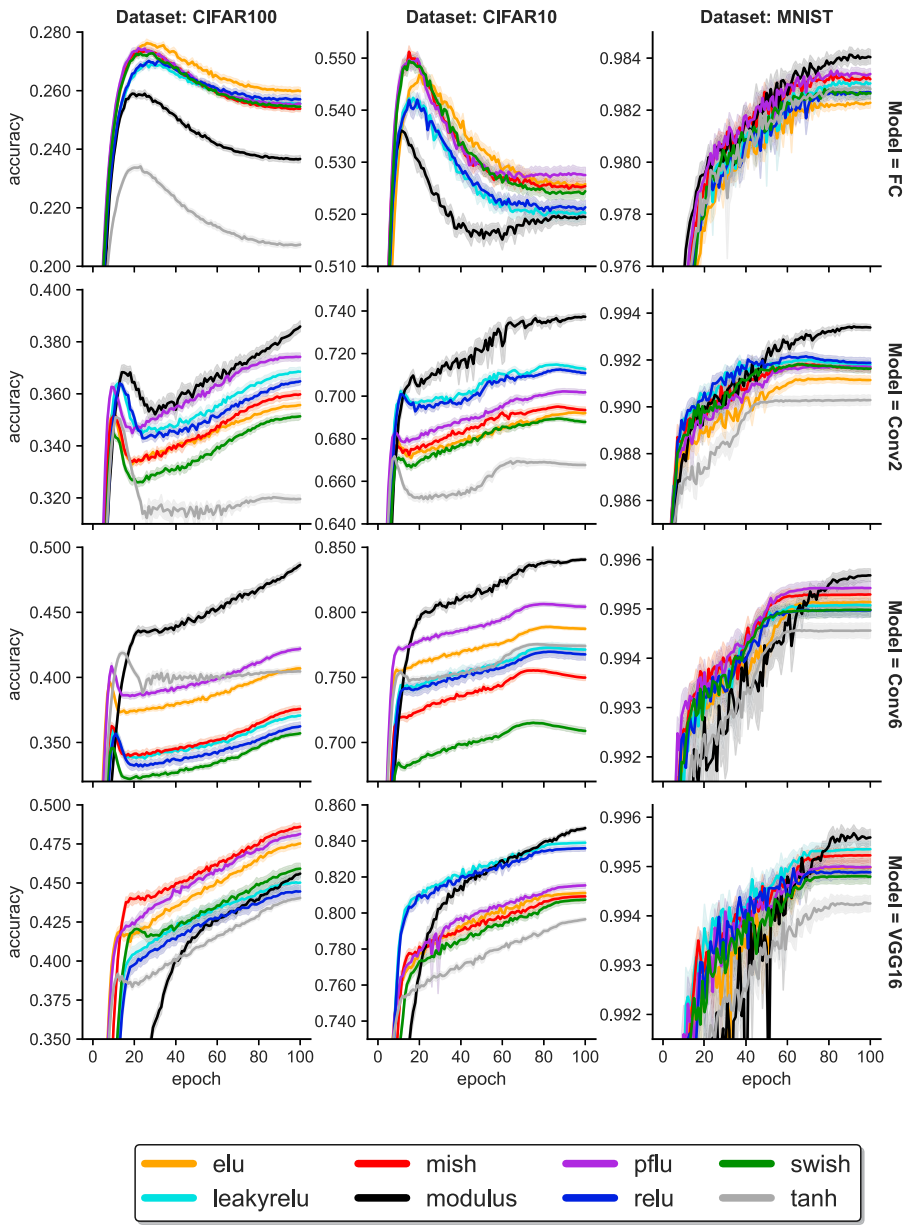


Fig. 3. Learning curves of all the trained models. Each line represents the average test accuracy of 30 runs of a model. The shade behind to each line shows the 95% confidence interval around the mean. Notice that the scales in the y-axis are not shared to allow for better zoom in each figure, comparing the performance across models is not the objective of this study. Details like the shades are better viewed in a screen.

Table 3
Accuracy comparison for the soft approximations of the *modulus* function, compared with the original *modulus* definition. The results are expressed as mean ± standard deviation across the 30 random initializations. In each column we highlighted the model with higher accuracy in bold. We added a star to those cases where a *SoftModulus* activation function achieved significantly higher results than the *modulus*, with a significance level of $\alpha = 0.05$.

Dataset	Activation	FC	Conv2	Conv6	VGG16
CIFAR10	Modulus	53.97 ± 0.24	73.93 ± 0.42	84.22 ± 0.29	84.86 ± 0.32
	SoftModulusQ	**54.07 ± 0.29** *	71.49 ± 0.37	81.01 ± 1.27	10.00 ± 0.00
	SoftModulusT	54.04 ± 0.24	**73.95 ± 0.40**	**84.36 ± 0.28**	**85.34 ± 0.36** *
CIFAR100	Modulus	26.29 ± 0.26	38.66 ± 0.56	**48.73 ± 0.62**	45.83 ± 0.80
	SoftModulusQ	26.23 ± 0.25	37.48 ± 0.44	48.16 ± 1.97	1.00 ± 0.00
	SoftModulusT	**26.32 ± 0.24**	**38.69 ± 0.56**	48.63 ± 0.83	**48.47 ± 0.68** *
MNIST	Modulus	98.47 ± 0.07	99.38 ± 0.04	99.60 ± 0.03	**99.63 ± 0.04**
	SoftModulusQ	**98.51 ± 0.06** *	99.37 ± 0.03	**99.62 ± 0.03** *	11.35 ± 0.00
	SoftModulusT	98.47 ± 0.06	**99.39 ± 0.04**	99.61 ± 0.03	99.62 ± 0.03

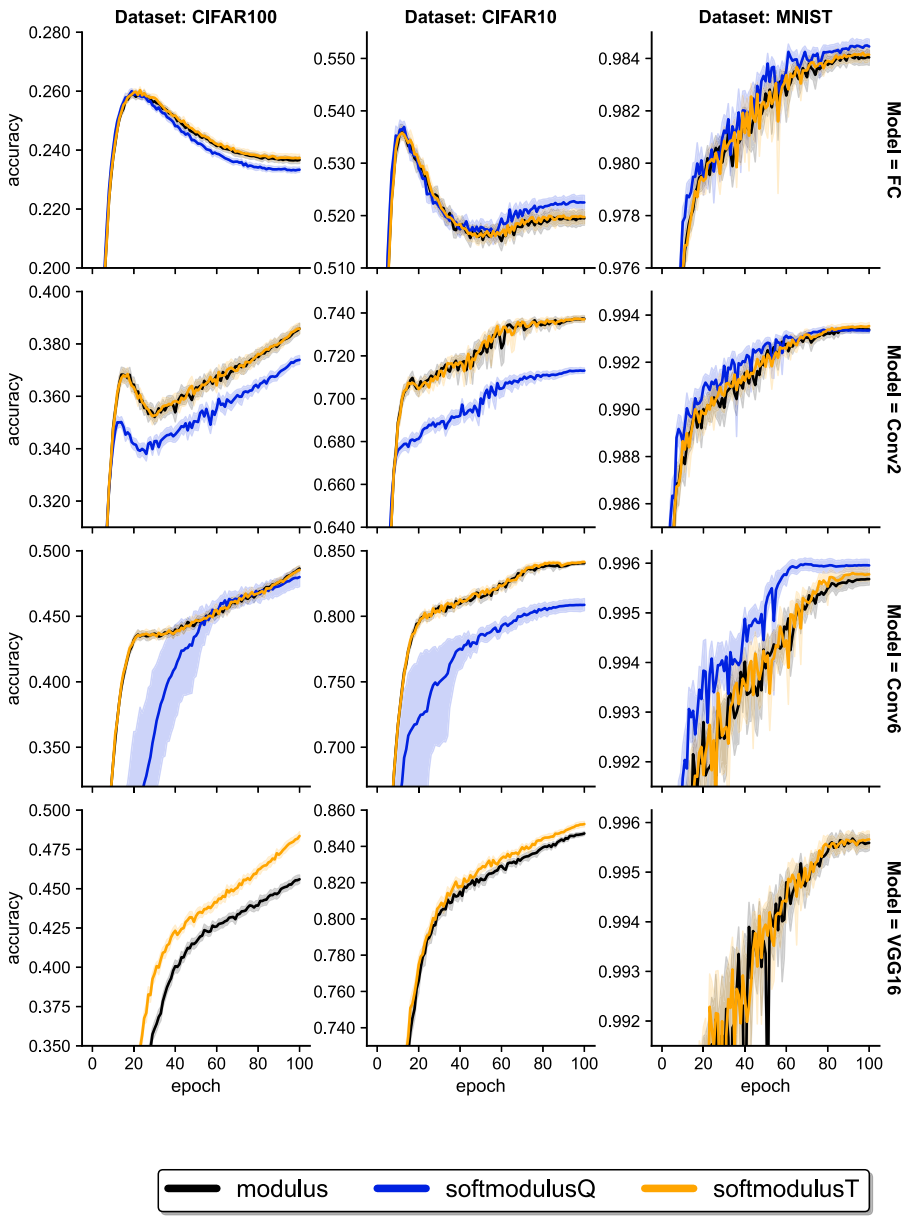


Fig. 4. Learning curves for smooth approximations compared with the *modulus*. Each line represents the average test accuracy of 30 runs of a model. The shade behind to each line shows the 95% confidence interval around the mean. Notice that the scales in the y-scales are not shared to allow for better zoom in each figure, comparing the performance across models is not the objective of this study. Details like the shades are better viewed in a screen.

Despite the intuition that non-monotonic activations can lead to neurons which parameters are more difficult to optimize, there are several examples in the literature (as mentioned in the introduction) that show that when using those nonlinearities the models tend to achieve better results. One potential explanation for this is that non-monotonic activations can introduce more complex nonlinearities into the network, allowing it to model complex relationships in the data more easily, potentially improving the performance of the resulting model. Another possible reason is that non-monotonic activation functions may have improved gradient properties, making optimization easier. For instance, certain non (strictly) monotonic activation functions, such as *ReLU* and the *modulus*, have very simple gradients.

Finally, another criticism of the *modulus* function may be its potential to introduce symmetry in neural networks, which may result in oscillation or instability. While in our experiments we did not observe evidences of this behavior, it may be a contributing factor to the slower convergence of networks with the *modulus* function (see Fig. 3). Another potential explanation for this observation is the lack of initialization methods specially designed for the *modulus* activation. It is worth noting that symmetry in neural networks can lead to duplication of ways to approximate the desired function, which translates to multiple equally valid solutions in the loss landscape. This may be advantageous, as it results in more regions of the loss landscape containing feasible solutions. Other potential advantage is that symmetry may improve the generalization ability of the network, as it allows to learn patterns that are invariant under certain transformations (e.g. input sign flip in dense layers without bias terms $f_\theta(X) = f_\theta(-X)$). The symmetry property introduced by the *modulus* can have more advantages, disadvantages or applications. Further research could explore the potential side effects of symmetry in more depth, as well as its impact on the performance and stability of the network in other tasks and applications.

5. Conclusions

We have shown how the new *modulus* activation function outperforms the benchmark activation functions in 75% of our experiments.

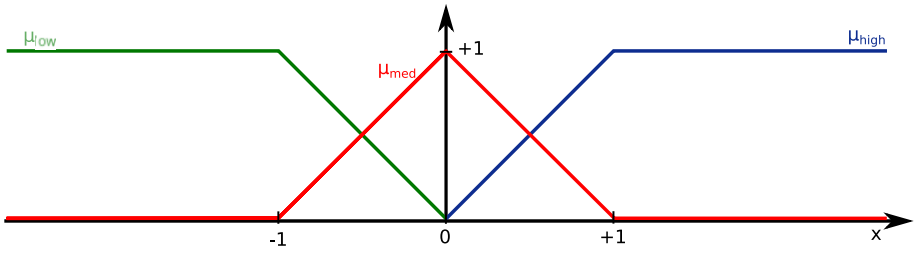


Fig. 5. Membership functions used to combine the quadratic and modulus functions.

The improvement achieved by the networks with this activation function is often significantly higher (3% or higher in one third of the experiments we conducted). Additionally, we have proposed a smooth version of the *modulus* activation function which performs better than the first one, at a slightly higher computational cost.

As a next step, it would be beneficial to evaluate the proposed *modulus* activation function on a wider range of problems beyond image classification. This would provide insight into the generalizability and potential utility of the *modulus* function in a variety of tasks. Further investigation in this direction could help to identify the specific areas in which the *modulus* function is especially effective, and highlight new applications for this non-monotonic activation function.

CRediT authorship contribution statement

Iván Vallés-Pérez: Conceptualization, Methodology, Software, Investigation, Validation, Writing – original draft. **Emilio Soria-Olivas:** Conceptualization, Resources, Writing – review & editing, Supervision, Validation. **Marcelino Martínez-Sober:** Writing – review & editing, Supervision, Validation. **Antonio J. Serrano-López:** Writing – review & editing, Supervision, Validation. **Joan Vila-Francés:** Writing – review & editing, Supervision, Validation. **Juan Gómez-Sanchís:** Writing – review & editing, Supervision, Validation.

Declaration of competing interest

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: Juan Gomez-Sanchis, Antonio J. Serrano-Lopez reports financial support was provided by Spanish ministry of science and innovation MCIN AEI projects PID2019-107347RR-C33 and PID2021-127946OB-00.

Data availability

Data used can be freely downloaded.

Acknowledgments

This work has been partially funded by the Spanish ministry of science and innovation MCIN/AEI projects PID2019-107347RR-C33 and PID2021-127946OB-I00.

Appendix. SoftModulusQ derivation

We chose a fuzzy set approach to combine the modulus function with a quadratic function. For that, three membership functions are defined in Fig. 5 and Eqs. (6), (7), (8).

$$\mu^x_{\text{low}} = \begin{cases} 1 & \text{if} \quad x < -1 \\ -x & \text{if} \quad -1 \le x < 0 \\ 0 & \text{if} \quad x \ge 0 \end{cases} \tag{6}$$

$$\mu^x_{\text{med}} = \begin{cases} 0 & \text{if} \quad x < -1 \\ x+1 & \text{if} \quad -1 \le x < 0 \\ 1-x & \text{if} \quad 0 \le x < 1 \\ 0 & \text{if} \quad x \ge 1 \end{cases} \tag{7}$$

$$\mu^x_{\text{high}} = \begin{cases} 0 & \text{if} \quad x < 0 \\ x & \text{if} \quad 0 \le x < 1 \\ 1 & \text{if} \quad x \ge 1 \end{cases} \tag{8}$$

Then we define $f^x_{\text{low}} = -x$, $f^x_{\text{med}} = x^2$ and $f^x_{\text{high}} = x$. If we combine these functions with the membership functions we get the following.

- For $x < -1$: we get $\hat{f}(x) = \frac{f^x_{\text{low}} \cdot \mu^x_{\text{low}}}{\mu^x_{\text{low}}} = -x$
- For $-1 \le x < 0$: we get $\hat{f}(x) = \frac{f^x_{\text{low}} \cdot \mu^x_{\text{low}} + f^x_{\text{med}} \cdot \mu^x_{\text{med}}}{\mu^x_{\text{low}} + \mu^x_{\text{med}}} = \frac{(x+1)x^2 + (-x)(-x)}{1+x-x} = x^3 + 2x^2$
- For $0 \le x < 1$: we get $\hat{f}(x) = \frac{f^x_{\text{med}} \cdot \mu^x_{\text{med}} + f^x_{\text{high}} \cdot \mu^x_{\text{high}}}{\mu^x_{\text{med}} + \mu^x_{\text{high}}} = \frac{(1-x)x^2 + x \cdot x}{1-x+x} = -x^3 + 2x^2$
- For $x \ge 1$: we get $\hat{f}(x) = \frac{f^x_{\text{high}} \cdot \mu^x_{\text{high}}}{\mu^x_{\text{high}}} = x$

Combining and simplifying all the above expressions we get the final *SoftModulusQ* activation function.

$$f(x) = \begin{cases} x^2 \cdot (2 - |x|) & \text{if} \quad |x| \le 1 \\ |x| & \text{if} \quad |x| > 1 \end{cases} \tag{9}$$

References

Agostinelli, F., Hoffman, M., Sadowski, P., Baldi, P., 2014. Learning activation functions to improve deep neural networks. In: 3rd International Conference on Learning Representations, 2015.

Clevert, D., Unterthiner, T., Hochreiter, S., 2016. Fast and accurate deep network learning by exponential linear units (ELUs). In: Bengio, Y., LeCun, Y. (Eds.), 4th International Conference on Learning Representations. ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings, URL http://arxiv.org/abs/1511.07289.

Dubey, S.R., Singh, S.K., Chaudhuri, B.B., 2022. Activation functions in deep learning: A comprehensive survey and benchmark. Neurocomputing 503, 92–108. http://dx.doi.org/10.1016/j.neucom.2022.06.111, URL https://www.sciencedirect.com/science/article/pii/S0925231222008426.

Dugas, C., Bengio, Y., Bélisle, F., Nadeau, C., Garcia, R., 2001. Incorporating second-order functional knowledge for better option pricing. In: Leen, T., Dietterich, T., Tresp, V. (Eds.), Advances in Neural Information Processing Systems, Vol. 13. MIT Press, URL https://proceedings.neurips.cc/paper/2000/file/44968aece94f667e4095002d140b5896-Paper.pdf.

Frankle, J., Carbin, M., 2019. The lottery ticket hypothesis: Finding sparse, trainable neural networks. In: 7th International Conference on Learning Representations. ICLR, OpenReview.net, URL http://dblp.uni-trier.de/db/conf/iclr/iclr2019.html#FrankleC19.

Glorot, X., Bengio, Y., 2010. Understanding the difficulty of training deep feedforward neural networks. In: Teh, Y.W., Titterington, M. (Eds.), Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics. In: Proceedings of Machine Learning Research, vol.9, PMLR, Chia Laguna Resort, Sardinia, Italy, pp. 249–256, URL http://proceedings.mlr.press/v9/glorot10a.html.

Glorot, X., Bordes, A., Bengio, Y., 2011. Deep sparse rectifier neural networks. In: Gordon, G., Dunson, D., Dudí k, M. (Eds.), Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics. In: Proceedings of Machine Learning Research, vol.15, PMLR, Fort Lauderdale, FL, USA, pp. 315–323, URL http://proceedings.mlr.press/v15/glorot11a.html.

Goodfellow, I., Bengio, Y., Courville, A., 2016. Deep Learning. MIT Press, http://dx.doi.org/10.1007/s10710-017-9314-z.

Gotmare, A., Shirish Keskar, N., Xiong, C., Socher, R., 2019. A closer look at deep learning heuristics: Learning rate restarts, warmup and distillation. In: 7th International Conference on Learning Representations. ICLR, URL https://openreview.net/forum?id=r14EOsCqKX.

Hendrycks, D., Gimpel, K., 2016. Bridging nonlinearities and stochastic regularizers with Gaussian error linear units. In: 4th International Conference on Learning Representations. ICLR, URL https://openreview.net/forum?id=Bk0MRI5lg.

Hochreiter, S., 1998. The vanishing gradient problem during learning recurrent neural nets and problem solutions. Int. J. Uncertain. Fuzziness Knowl.-Based Syst. 6 (2), 107–116. http://dx.doi.org/10.1142/S0218488598000094.

Hochreiter, S., Bengio, Y., Frasconi, P., Schmidhuber, J., 2001. Gradient flow in recurrent nets: the difficulty of learning long-term dependencies. In: A Field Guide to Dynamical Recurrent Neural Networks. IEEE Press, URL https://ml.jku.at/publications/older/ch7.pdf.

Ioffe, S., Szegedy, C., 2015. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In: 32nd International Conference on Machine Learning,Vol. 37. In: International Conference of Machine Learning, JMLR.org, pp. 448–456, URL http://proceedings.mlr.press/v37/ioffe15.html.

Jin, X., Xu, C., Feng, J., Wei, Y., Xiong, J., Yan, S., 2016. Deep learning with S-shaped rectified linear activation units. In: Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence. AAAI '16, AAAI Press, pp. 1737–1743, URL https://arxiv.org/abs/1512.07030.

Karnewar, A., 2018. AANN: Absolute artificial neural network. In: 2018 3rd International Conference for Convergence in Technology. I2CT, pp. 1–6. http://dx.doi.org/10.1109/I2CT.2018.8529552.

Kiliçarslan, S., Celik, M., 2021. RSigELU: A nonlinear activation function for deep neural networks. Expert Syst. Appl. 174, 114805. http://dx.doi.org/10.1016/j.eswa.2021.114805, URL https://www.sciencedirect.com/science/article/pii/S0957417421002463.

Kingma, D.P., Ba, J., 2014. Adam: A method for stochastic optimization. In: 3rd International Conference of Learning Representations. ICLR, URL https://arxiv.org/abs/1412.6980.

Klambauer, G., Unterthiner, T., Mayr, A., Hochreiter, S., 2017. Self-normalizing neural networks. In: Guyon, I., von Luxburg, U., Bengio, S., Wallach, H.M., Fergus, R., Vishwanathan, S.V.N., Garnett, R. (Eds.), NIPS. pp. 971–980, URL http://dblp.uni-trier.de/db/conf/nips/nips2017.html#KlambauerUMH17.

Krizhevsky, A., 2009. Learning multiple layers of features from tiny images. Technical Report, Computer Science, University of Toronto, URL https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf.

LeCun, Y.A., Bottou, L., Orr, G.B., Müller, K.-R., 2012. Efficient BackProp. In: Montavon, G., Orr, G.B., Müller, K.-R. (Eds.), Neural Networks: Tricks of the Trade: Second Edition. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 9–48. http://dx.doi.org/10.1007/978-3-642-35289-8_3.

LeCun, Y., Cortes, C., 2010. MNIST handwritten digit database. URL http://yann.lecun.com/exdb/mnist/.

Loshchilov, I., Hutter, F., 2017. SGDR: Stochastic gradient descent with warm restarts. In: 5th International Conference of Learning Representations. ICLR, URL https://ml.informatik.uni-freiburg.de/~staeglis/deploy/papers/17-ICLR-SGDR.pdf.

Lu, L., 2020. Dying ReLU and initialization: Theory and numerical examples. Commun. Comput. Phys. 28 (5), 1671–1706. http://dx.doi.org/10.4208/cicp.oa-2020-0165.

Misra, D., 2019. Mish: A self regularized non-monotonic neural activation function. arXiv preprint arXiv:1908.08681.

Misra, J., Saha, I., 2010. Artificial neural networks in hardware: A survey of two decades of progress. Neurocomputing 74 (1), 239–255. http://dx.doi.org/10.1016/j.neucom.2010.03.021, Artificial Brains URL https://www.sciencedirect.com/science/article/pii/S092523121000216X.

Nair, V., Hinton, G.E., 2010. Rectified linear units improve restricted Boltzmann machines. In: Fürnkranz, J., Joachims, T. (Eds.), Proceedings of the 27th International Conference on Machine Learning. ICML-10, pp. 807–814, URL https://www.cs.toronto.edu/~fritz/absps/reluICML.pdf.

Pascanu, R., Mikolov, T., Bengio, Y., 2013. On the difficulty of training recurrent neural networks. In: Dasgupta, S., McAllester, D. (Eds.), Proceedings of the 30th International Conference on Machine Learning, no. 3. In: Proceedings of Machine Learning Research, vol.28, PMLR, Atlanta, Georgia, USA, pp. 1310–1318, URL http://proceedings.mlr.press/v28/pascanu13.html.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., Chintala, S., 2019. PyTorch: An imperative style, high-performance deep learning library. In: Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché Buc, F., Fox, E., Garnett, R. (Eds.), Advances in Neural Information Processing Systems,Vol. 32. Curran Associates, Inc., pp. 8024–8035, URL http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

Ramachandran, P., Zoph, B., Le, Q., 2018. Searching for activation functions. In: 6th International Conference on Learning Representations. ICLR, URL https://openreview.net/forum?id=SkBYYyZRZ.

Sanchez-Iborra, R., Skarmeta, A.F., 2020. TinyML-enabled frugal smart objects: Challenges and opportunities. IEEE Circuits Syst. Mag. 20 (3), 4–18. http://dx.doi.org/10.1109/MCAS.2020.3005467.

Simonyan, K., Zisserman, A., 2015. Very deep convolutional networks for large-scale image recognition. In: 3rd International Conference on Learning Representations. ICLR, URL https://arxiv.org/abs/1409.1556.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., Salakhutdinov, R., 2014. Dropout: A simple way to prevent neural networks from overfitting. J. Mach. Learn. Res. 15 (1), 1929–1958, URL https://jmlr.org/papers/v15/srivastava14a.html.

Xu, B., Wang, N., Chen, T., Li, M., 2015. Empirical evaluation of rectified activations in convolutional network. CoRR URL https://arxiv.org/pdf/1505.00853.pdf.

Zhu, M., Min, W., Wang, Q., Zou, S., Chen, X., 2021. PFLU and FPFLU: Two novel non-monotonic activation functions in convolutional neural networks. Neurocomputing 429, 110–117. http://dx.doi.org/10.1016/j.neucom.2020.11.068, URL https://www.sciencedirect.com/science/article/pii/S0925231220318749.