Trainable Highly-expressive Activation Functions

Irit Chelly*, Shahaf E. Finder*, Shira Ifergane, and Oren Freifeld

The Department of Computer Science, Ben-Gurion University of the Negev, Israel
{tohamy,finders,shiraif}@post.bgu.ac.il, orenfr@cs.bgu.ac.il

Abstract. Nonlinear activation functions are pivotal to the success of deep neural nets, and choosing the appropriate activation function can significantly affect their performance. Most networks use fixed activation functions (*e.g.*, ReLU, GELU, *etc.*), and this choice might limit their expressiveness. Furthermore, different layers may benefit from diverse activation functions. Consequently, there has been a growing interest in trainable activation functions. In this paper, we introduce DiTAC, a trainable highly-expressive activation function based on an efficient diffeomorphic transformation (called CPAB). Despite introducing only a negligible number of trainable parameters, DiTAC enhances model expressiveness and performance, often yielding substantial improvements. It also outperforms existing activation functions (regardless whether the latter are fixed or trainable) in tasks such as semantic segmentation, image generation, regression problems, and image classification. Our code is available at https://github.com/BGU-CS-VIL/DiTAC.

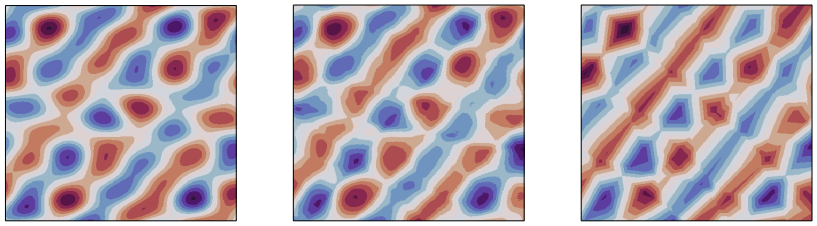
Keywords: Trainable Activation functions · Diffeomorphisms · Deep Learning

1 Introduction

Activation functions (AFs) play a major role in the success of deep neural nets, as they endow the latter with nonlinearity [15,49]. In fact, AFs are crucial to the ability of the networks to approximate almost arbitrarily-complex functions, to learn meaningful feature representations, and to achieve high predictive performance. In addition to their nonlinearity, AFs have various characteristics which directly influence the performance of the network. Traditional AFs, such as the Logistic Sigmoid and Tanh Unit, map their input values into a small range, possibly causing the network gradients to become close to zero [4] and thus, impair training performance [5]. The Rectified Linear Unit (ReLU) [47] and its variants (*e.g.*, LReLU [43] and PReLU [24]) partially solve this issue by mapping the input into an unbounded range in one or two directions. Exponential AFs such as ELU [9] inherit ReLU's benefits but also push the AF responses to zero-mean in order to improve performance [9].

Generally, fixed AFs have limited nonlinearity (hence limited expressiveness) and impose a learning bias on the network. Consequently, adjusting them to

*Equal contribution.



(a) Target function **(b)** MLP+DiTAC **(c)** MLP+PReLU

Fig. 1: Regression-task results of reconstructing a two-dimensional function via a simple MLP, using either DiTAC or (the runner-up) PReLU. Due to its expressiveness, DiTAC manages to fit a smooth function, yielding an evidently-better reconstruction.

varying problem types and data complexity is challenging. As a result, investigating AF design which enhances expressiveness and alleviates such a bias is an open field of research. Trainable activation functions (TAFs) such as PReLU [24], Swish [52] and PELU [62] adjust the shape of a standard fixed AF by adding several learnable parameters. According to [1], this type of functions achieves a minor gain in expressiveness, as those TAFs tend to perform similarly to their base untrainable AFs. A different AF approach is presented by the Maxout Unit [22]. Despite its remarkable improvement over ReLU in solving classification tasks [22], the number of parameters in the Maxout layer increases with the number of the neurons in the network. The aforementioned limitations of existing AFs are part of the motivation behind our paper.

A *diffeomorphism* is a differentiable invertible function with a differentiable inverse. In this paper we propose a **Di**ffeomorphism-based **T**rainable **Ac**tivation function (DiTAC), a differentiable parametric TAF based on highly-expressive and efficient diffeomorphisms (called CPAB [19,20]). DiTAC is highly expressive even though it adds only a negligible amount of trainable parameters. Comparing to existing TAFs which are restricted to learn either only a certain shape [24, 52, 62] or only convex functions [22], DiTAC is capable of learning a variety of shapes (see Fig. 1 and Fig. 3). In [14] it is shown that different AFs are suitable to different types of data and tasks, a fact motivating flexible TAF approaches like ours. In particular, we show that DiTAC achieves significant improvements on various datasets, and various tasks such as semantic segmentation, image generation, image classification, and regression problems.

To summarize, **our contributions are as follows**: (1) To our knowledge, we are the first to propose using flexible diffeomorphisms within TAFs. (2) We present DiTAC, a novel highly-expressive AF that addresses issues of existing TAFs, and can be easily used in any model architecture. (3) We show that DiTAC outperforms existing AFs and TAFs on various tasks and datasets.

2 Related Work

2.1 Activation Functions

AFs are a key component of artificial neural networks. They introduce nonlinearity between the linear operations of the network; *i.e.*, without them the network is simply a linear function. In the early days of neural networks, the most common AFs were the Logistic Sigmoid and the Tanh Unit [37], both are smooth and non-decreasing functions. However, due to their bounded response, the network gradients become generally small and quickly approach zero [4], causing the notorious vanishing-gradient problem. The Rectified Linear Unit (ReLU) [47] function attempts to solve this issue by mapping its positive-values input to an unbounded range. ReLU is widely used due to its computational efficiency and improved performance [21, 43, 47]. To better solve the vanishing-gradient issue, other Rectified AFs such as LReLU [43] and PReLU [24] utilize the negative input values by returning a linear function with a fixed/trainable slope, and hence become unbounded in both directions. Nevertheless, adjusting/learning the right slope may be more suitable for certain tasks than others, or lead to overfitting [14]. Differentiable alternatives to ReLU have also been developed, such as ELU [9], defined as $\mathrm{ELU}(x) = \alpha(e^x - 1)$ for $x \leq 0$, and as the identity function otherwise. Extensions to ELU were proposed [2, 8, 34, 62], where in [8], PDELU learns a zero-mean AF response, allowing steepest descent of gradients and improving training speed and performance [9]. Other differentiable functions are Softplus [16], a smooth approximation of ReLU [1], and ErfReLU [51], a combination of ReLU and the error function (*i.e.*, the cumulative distribution function of a standard normal distribution). In [30–32], the authors propose TAFs for physics-oriented networks. A recent popular AF called GELU [26] utilizes the error function to weight its input. In [52], the authors conducted an automated reinforcement learning-based search to find new AFs, and proposed Swish, defined as $\mathrm{Swish}(x) = x \cdot \mathrm{Sigmoid}(\beta x)$ where β is a trainable parameter. Motivated by those search techniques, [45] introduced Mish, a combination of the tanh and softplus functions, and [38] introduced ASH, a generalized form of Swish.

The aforementioned AFs are either fixed AFs or based on a fixed-shape function, allowing its shape to be adapted by learning several parameters. While adaptable, their expressiveness is still limited as they implicitly inherit the fixed structural form of the AF they were adapted from. Some TAFs avoid this at the cost of more trainable parameters. Unfortunately, however, the number of their added trainable parameters grows with the number of neurons in the network. For instance, the Maxout Unit [22], followed by its stochastic version Probabilistic Maxout [61], computes the maximal value of a set of k trainable linear functions of the same input, leading to a significant increase in the number of total parameters in the network. The ACON family [42] partially mitigates this issue by approximating the general Maxout family. However, in its major proposed variants, the number of added parameters still grows with the number of channels in the network. Another recent TAF that increases the number of

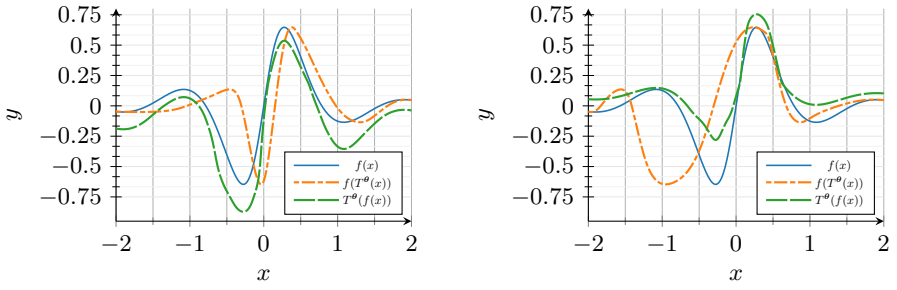


Fig. 2: The CPAB transformation effect when it is applied on each axis. In each panel we use a different $T^{\boldsymbol{\theta}}$. When the x axis is transformed (*i.e.*, $f(T^{\boldsymbol{\theta}}(x))$), the intensity values are unchanged and only the x values are shifted. When the y axis is transformed ($T^{\boldsymbol{\theta}}(f(x))$), the intensity is changed while peaks and valleys' locations are kept.

parameters is the DY-ReLU [7], which forms a piecewise function whose parameters are generated by a hyper function over all input elements. Our work, in contrast to all those works, offers a more expressive nonlinear differentiable TAF using only a negligible amount of added trainable parameters.

2.2 CPAB transformations in Deep Learning

CPAB transformations, proposed by Freifeld *et al.* [19, 20], are efficient and highly-expressive parametric diffeomorphisms. They are called CPAB, short for CPA-Based, as they are based on Continuous Piecewise-Affine (CPA) velocity fields (as we will explain later). Since their inception, those transformations found many applications in DL (*e.g.*, [13,23,33,36,44,48,55–58,60,63]) . A main difference between how all these works used CPAB transformations and our usage of them, is that in those works the CPAB transformations were always applied to the domain of the signals of interest (either the spatial domain in 2D images or the time domain in time series), typically by incorporating them in a Spatial Transformer Net (STN) [29] or a Temporal Transformer Net (TTN) [57], while we apply them (elementwise) to the *range* of feature maps (as an aside, among other things it also means that we do not need to preform grid resampling, a mandatory step in STNs/TTNs). Figure 2 illustrates this difference. In particular, we are the first to use CPAB transformations (or any other highly-expressive family of diffeomorphisms for that matters) for building TAFs.

3 Method

3.1 Preliminaries: 1D CPAB Transformations

Let $T^{\boldsymbol{\theta}}$ be a diffeomorphism parameterized by $\boldsymbol{\theta}$. Working with diffeomorphisms usually involves expensive computations. In our case, since we use diffeomorphisms directly within DL architectures, it is even more important (than in

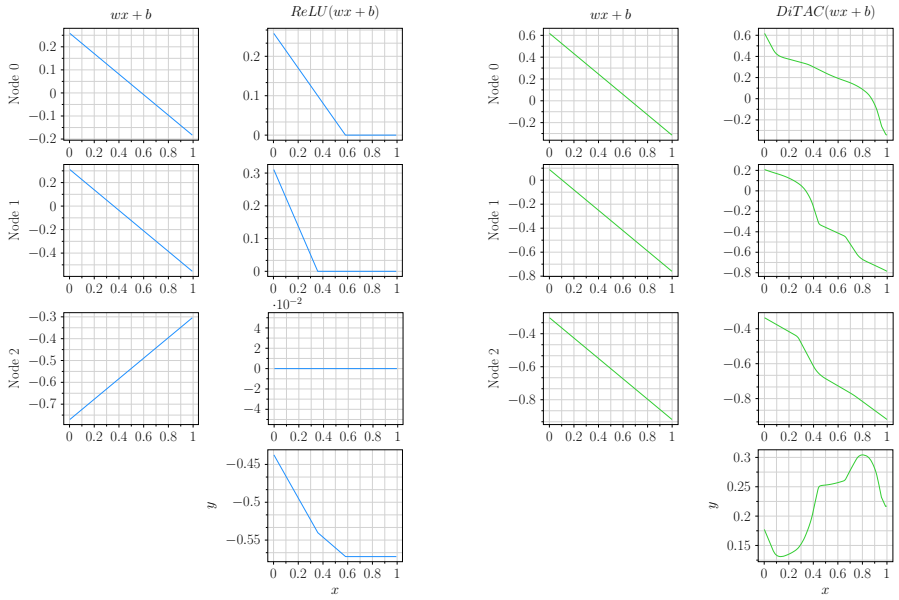


Fig. 3: DiTAC's expressiveness reflected in a 3-node hidden-layer regression network. The first three rows match the three hidden nodes. Two left columns (blue): each node's value before and after *the* ReLU function. Two right columns (green): each node's value before and after *a* DiTAC function. Bottom row: learned 1D regression using ReLU (blue) versus using DiTAC (green).

non-DL applications) to reduce the associated computational burden because during training, the quantities $x \mapsto T^{\boldsymbol{\theta}}(x)$ and $x \mapsto \nabla_{\boldsymbol{\theta}} T^{\boldsymbol{\theta}}(x)$ are computed at multiple values of x and for multiple values of $\boldsymbol{\theta}$.

The main reason why we chose CPAB transformations [19, 20] as the diffeomorphism family to be used within our method is that they are both expressive and efficient. Throughout the remainder of this paper all the CPAB transformations are assumed to be in 1D (for the more general case, see [20]). In a presentation based on [20], we now briefly explain what CPAB transformations are, as well as their name.

Let $\Omega = [a, b] \subset \mathbb{R}$ be a finite interval and let $\mathcal{V}$ be a space of continuous functions, from Ω to $\mathbb{R}$, that are also piecewise-affine w.r.t. some fixed partition of Ω into sub-intervals. Note that $\mathcal{V}$ is a finite-dimensional linear space. Let $d = \dim(\mathcal{V})$, let $\boldsymbol{\theta} \in \mathbb{R}^d$, and let $v^{\boldsymbol{\theta}} \in \mathcal{V}$ denote the generic element of $\mathcal{V}$, parameterized by $\boldsymbol{\theta}$. The space of CPAB transformations obtained via the integration of elements of $\mathcal{V}$, is defined as

$$\mathcal{T} \triangleq \Big\{ T^{\boldsymbol{\theta}} : x \mapsto \phi^{\boldsymbol{\theta}}(x; 1) \text{ s.t. } \phi^{\boldsymbol{\theta}}(x; t) \text{ solves the integral equation}$$
$$\phi^{\boldsymbol{\theta}}(x; t) = x + \int_0^t v^{\boldsymbol{\theta}}(\phi^{\boldsymbol{\theta}}(x; \tau))\, \mathrm{d}\tau \text{ where } v^{\boldsymbol{\theta}} \in \mathcal{V} \Big\}. \tag{1}$$

It can be shown that every $T^{\boldsymbol{\theta}} \in \mathcal{T}$ is an order-preserving transformation (*i.e.*, it is monotonically increasing) and a diffeomorphism [20]. Note that while $v^{\boldsymbol{\theta}} \in \mathcal{V}$ is CPA, the CPAB $T^{\boldsymbol{\theta}} \in \mathcal{T}$ is not (*e.g.*, $T^{\boldsymbol{\theta}}$ is differentiable, unlike any non-trivial CPA function). Equation 1 also implies that the elements of $\mathcal{V}$ are viewed as velocity fields.

Particularly useful for us are the following facts: 1) The finer the partition of Ω is, the more expressive the CPAB family becomes (which also means that d increases). 2) CPAB transformations lend themselves to fast and accurate computations in closed form of both $x \mapsto T^{\boldsymbol{\theta}}(x)$ [20] and the gradient, $x \mapsto \nabla_{\boldsymbol{\theta}} T^{\boldsymbol{\theta}}(x)$ [18,44]. Together, these facts mean that *CPAB transformations provide us with a convenient and an efficient way to parameterize, and optimize over, nonlinear monotonically-increasing functions.*

3.2 The DiTAC Activation Function

Our proposed TAF, called DiTAC, is a TAF derived from a CPAB transformation. DiTAC includes a negligible amount of trainable parameters, and yet it is highly expressive. Unlike existing TAFs which dedicate a parameter for each input's channel, DiTAC's expressiveness stems from the expressiveness of CPAB transformations. For illustration, in Fig. 3 we show how nonlinearity evolves step-by-step in a regression MLP with a 3-node hidden layer, when using either ReLU or DiTAC. While in the ReLU case, the expressiveness is mostly reflected only after summing all of the activation responses (and the resulting function is non-differentiable in several locations), DiTAC's expressiveness (and differentiability) is distinctly shown already in the very first data transformation each neuron goes through. Importantly, the availability of closed-form expressions for the CPAB transformation and its gradient makes it easy to use DiTAC as a drop-in replacement instead of any AF in any DL architecture.

We now explain how the DiTAC is built. Recall that a CPAB transformation, $T^{\boldsymbol{\theta}}$, is defined on a finite interval, $[a, b]$. Its co-domain is also a finite interval, which may or may not coincide with Ω (depending whether one imposes zero-boundary conditions on $\boldsymbol{v}^{\boldsymbol{\theta}}$ or not; see [20]). As some of the AF's input can fall outside of $[a, b]$, our main version of DiTAC combines $T^{\boldsymbol{\theta}}$ with GELU, a recent widely-used AF in state-of-the-art models. Recall that GELU [26] is given by $\mathrm{GELU}(x) = x \cdot \Phi(x)$ where Φ is the cumulative distribution function of a standard normal distribution.

The GELU-like DiTAC function is

$$\mathrm{DiTAC}(x) = \tilde{x} \cdot \Phi(x)\,, \qquad \tilde{x} = \begin{cases} T^{\boldsymbol{\theta}}(x) & \text{If } a \leq x \leq b \\ x & \text{Otherwise} \end{cases} \tag{2}$$

where $T^{\boldsymbol{\theta}}$ is a (learnable) CPAB transformation while $\Omega = [a, b]$, the domain of $T^{\boldsymbol{\theta}}$, is user-defined. This main DiTAC version is the one used in the experiments in the paper (while the appendix also contains other versions; see below).

Other versions of DiTAC can also be built by combining $T^{\boldsymbol{\theta}}$ with a variety of other AFs, not just with GELU. For instance, we present Leaky-DiTAC, where

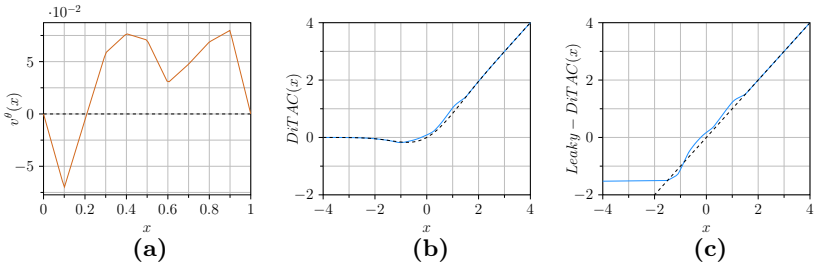


Fig. 4: Here we display (a) $\boldsymbol{v}^{\boldsymbol{\theta}}$, a CPA velocity field, (b) DiTAC (solid line) versus GELU (dashed line), and (c) a Leaky DiTAC (solid line) versus the identity function (dashed line). The two DiTAC functions were derived from the CPAB transformation $T^{\boldsymbol{\theta}}$ that corresponds to $\boldsymbol{v}^{\boldsymbol{\theta}}$ in (a).

$T^{\boldsymbol{\theta}}$ is applied on $[a, b]$ while the rest of the data goes through a Leaky-ReLU (LReLU) function. That is,

$$\text{Leaky DiTAC}(x) = \begin{cases} T^{\boldsymbol{\theta}}(x) & \text{If } a \leq x \leq b \\ \text{LReLU}(x) & \text{Otherwise} \end{cases} \tag{3}$$

For an illustration of both these DiTAC types, see Fig. 4. Additional DiTAC's versions and their illustrations can be found in the Appendix.

To stabilize the training and to prevent learning too-extreme transformations, we also use a regularization over the velocity fields. Concretely, our regularization term (which was also used in, *e.g.*, [44, 57, 59]) is

$$\mathcal{L}_{\text{reg}} = \sum\nolimits_{l=1}^{L} \boldsymbol{\theta}_l{}^T \boldsymbol{\Sigma}_{\text{CPA}}^{-1} \boldsymbol{\theta}_l \tag{4}$$

where L is the number of activation layers in the network, $\boldsymbol{\theta} \in \mathbb{R}^d$ is the DiTAC parameters, and $\boldsymbol{\Sigma}_{\text{CPA}}^{-1}$ is a $d \times d$ covariance matrix associated with a Gaussian smoothness prior (proposed in [19]) over CPA velocity fields. That matrix has two hyperparameters: λ_{var}, which controls the variance of the velocity fields, and λ_{smooth}, which controls the similarity of the velocities in different subintervals and hence the smoothness (in the machine-learning sense) of the field.

3.3 How to Drastically Reduce the Computational Cost

As is usual in DL, training involves a very high number of calls to the AFs. Consider a tensor of size (b, c, h, w), where b is the batch size, c is the number of channels, and (h, w) is the height and width. Applying a CPAB transformation to each entry in the tensor will naturally require evaluating it $b{\cdot}c{\cdot}h{\cdot}w$ times. *E.g.*, the AF of the last bottleneck block of ResNet-50 [25], with a batch size of 32, operates over ∼800K entries. Thus, and although CPAB transformations provide an efficient solution for representing diffeomorphisms, using such transformations

naively here could still incur a significant computational cost during training and be too slow. Luckily, there is a better way. Below we provide a solution that, during the learning, dramatically alleviates this cost. Moreover, during inference that solution renders DiTAC just as efficient as other AFs.

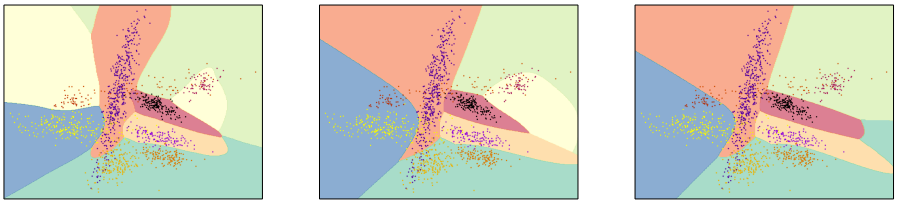
To drastically reduce the cost during learning, we quantize the interval $[a, b]$ (on which the CPAB transformation is applied) to n discrete values uniformly. Although losing some information, quantizing activations in a neural network can show little to no impact on accuracy when using a large enough number of elements (in our case, usually 2^8 suffice) [17, 46]. In this approach, we use the CPAB transformation on the set of quantized elements and create a lookup table, which we can then use to transform the values of all the entries in the input tensor. That is, we output $y_i = T^{\boldsymbol{\theta}}(Q(x_i))$ where $Q(\cdot)$ is the quantization function and $Q(x_i) \in \{a + k\Delta\}_{k=0}^{n}$ for $\Delta = \frac{b-a}{n}$. For backpropagation, we utilize a variation of the Straight Through Estimator [3]. We calculate the CPAB derivatives only for the outputs of the quantized values, which we then broadcast as derivative estimations for the x_i's,

$$\frac{\partial T^{\boldsymbol{\theta}}(x_i)}{\partial x_i} \approx \frac{\partial T^{\boldsymbol{\theta}}(q_j)}{\partial q_j}, \quad \frac{\partial T^{\boldsymbol{\theta}}(x_i)}{\partial \theta_\ell} \approx \frac{\partial T^{\boldsymbol{\theta}}(q_j)}{\partial \theta_\ell}, \quad \text{where } q_j = Q(x_i). \tag{5}$$

Revisiting that ResNet-50 example, we need to transform only a much smaller set of entries (*e.g.*, $2^{10} = 1024 \ll 800\text{K}$) for the same input while achieving, empirically, nearly identical results. During the learning, we (quickly) build such a lookup table every time $\boldsymbol{\theta}$ changes. Once the learning is done and before the inference, a single lookup table (per each DiTAC function) is calculated and then reused during inference as many times as needed. An experiment measuring the computational costs during inference is provided in the Appendix.

4 Results

We present an extensive evaluation of DiTAC on several challenging tasks using state-of-the-art architectures. In § 4.1 we analyze DiTAC's performance in classification and regression tasks on synthetic and toy data. In § 4.2 we continue evaluating DiTAC on real-data (*e.g.*: ImageNet-1K [53]; ADE20K [68]), on image classification, semantic segmentation, and image generation tasks. In addition we show comprehensive comparisons between DiTAC and the following fixed AFs: ReLU, LReLU, GELU, ELU, Softplus, Mish, and the following TAFs: Swish, PReLU, PDELU and Meta-ACON, on various tasks. In all of our experiments, the trainable parameters of DiTAC (and of other TAF competitors) are trained without weight decay to avoid pushing the learned values to zero during training. We found that a 10-cell partition of Ω (the domain of the CPAB function) suffices for good results. This translates to an addition of only 9 trainable parameters in each DiTAC instance in the recipient network.



(a) DiTAC **(b)** Swish **(c)** GELU

Fig. 5: Decision boundaries learned by DiTAC, Swish, and GELU in a 2D-GMM classification, along with test-batch data points colored by their ground-truth classes. Evidently, DiTAC learns more accurate boundaries and identifies more classes.

4.1 Toy Data

In this section, we report the results of several experiments in classification and regression tasks on simple datasets. We use simple MLPs with at most two dense layers, changing only the AF between configurations. We run each configuration with several learning rates and select the best performance for each competitor. In all toy-data analyses, we omit Meta-ACON (as it works in a channel-wise manner, which is impossible to apply in a fully-connected network) while for PDELU we use its channel-shared version.

Table 1: A comparison of DiTAC with existing AFs and TAFs. We report the top-1 classification accuracy (in %) along with the number of model parameters used by each AF, on the 2D-GMM and MNIST datasets. The network is a simple MLP.

AF/TAF	2D-GMM		MNIST	
	Param.	Top-1	Param.	Top-1
ReLU	11,410	84.0	109,386	97.75
LReLU	11,410	84.0	109,386	97.77
GELU	11,410	84.0	109,386	97.99
ELU	11,410	84.0	109,386	97.82
Softplus	11,410	81.4	109,386	97.8
Mish	11,410	84.0	109,386	97.97
Swish	11,412	86.8	109,388	97.93
PReLU	11,412	83.93	109,388	97.84
PDELU	11,412	83.86	109,388	97.68
DiTAC	11,428	**89.06**	109,404	**98.29**

Classification. For this task, we use two datasets, a two-dimensional Gaussian-Mixture-Model (2D-GMM) and MNIST. We construct a 2D GMM as a 10-component mixture, where the parameters of each Gaussian component (*i.e.*, the

mean vector and covariance matrix) are drawn from a normal-inverse-Wishart distribution. We then draw $5{\cdot}10^3$ points from the GMM, with a split of 70%/30% for train/test data.

For each dataset, we use a simple MLP with two hidden layers. For MNIST we set their sizes to 128 and 64, and for 2D-GMM we set both to 100. Note that the number of trained parameters in DiTAC is negligible w.r.t. the network size. *E.g.*, the 2D-GMM network has $11,410$ trainable parameters, and by using DiTAC, we add only 18 parameters (9 per each of the two AFs in that MLP) to the model. More details about the training procedure and data generation can be found in the Appendix.

The results appear in Table 1 and show a significant advantage to using DiTAC. In this comparison, we report the top-1 accuracy along with network size generated by each AF. In addition, in Fig. 5 we visualize the learned decision boundaries of DiTAC, Swish, and GELU in the 2D-GMM classification task. DiTAC outperforms the two state-of-the-art competitors by learning a better distinction between the classes.

Regression. Continuing the comparison, we now experiment with regression tasks of reconstructing one- and two-dimensional functions using an MLP with one hidden layer containing 30 and 50 neurons respectively. The first function is $\sin(\exp 6x)$, and the second is a sum of sines with various frequencies. We train the model for 40K iterations, ensuring training convergence of all of the competitors. More training details can be found in the Appendix. Here, we had to omit PDELU from the results table due to its unstable training process.

Table 2: Regression-task results of learning the one- and two-dimensional functions mentioned in the text, along with the number of model parameters used by each AF.

AF/TAF	**1D Func.**			**2D Func.**		
	Param.	MSE ↓	R2 ↑	Param.	MSE ↓	R2 ↑
ReLU	91	0.117	76.47	201	0.023	88.86
LReLU	91	0.051	89.78	201	0.011	93.13
GELU	91	0.138	72.73	201	0.054	65.27
ELU	91	0.207	58.30	201	0.118	25.11
Softplus	91	0.206	59.56	201	0.112	24.96
Mish	91	0.175	66.71	201	0.114	27.28
Swish	92	0.061	88.33	202	0.130	25.47
PReLU	92	0.062	88.30	202	0.008	96.09
DiTAC	100	**0.020**	**96.26**	210	**0.004**	**98.37**

Table 2 shows that when using DiTAC, the MLP fits more accurately to the functions, despite using only an insignificant increase in the number of trainable parameters. DiTAC's expressiveness and flexibility become even more evident

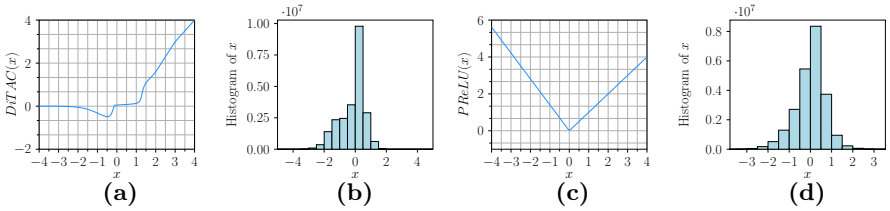


Fig. 6: We demonstrate DiTAC's expressiveness by showing the learned DiTAC (a) and its input-data histogram (b), and the learned PReLU (c) and its input-data histogram (d), in the 2D regression problem with a simple MLP.

when visualizing the learned DiTAC function compared to PReLU (the runner-up) for the two-dimensional regression in Fig. 6: while PReLU is restricted in the function shapes it can learn, DiTAC ended up learning a certain function curvature that fits to the problem. This is also demonstrated qualitatively in Fig. 1, where it is evident that PReLU's elevation lines are more rigid than DiTAC's, which indicates its difficulty in fitting the smooth function. A detailed analysis of the experiments above, together with evaluation on additional regression datasets, is presented in the Appendix.

4.2 Real-World Data

Small-scale Classification Experiment. We first conduct a small-scale classification experiment using ImageNet-50, a subset of 50 classes from ImageNet [53]. We train several MobileNet-V3 [28] and ResNet [25] configurations of different sizes, and compare DiTAC to various competitors by replacing all AF instances in each architecture with each AF/TAF candidate. As for ResNet, the training procedure is inspired by [65]. The models were trained for 300 epochs on 224×224 resolution on a single GPU, using an AdamW optimizer. We use a batch size of 256, initial learning rate of $5\cdot10^{-3}$, weight decay of 0.02 and a 5 epoch linear warmup. With regards to MobileNet-V3, we follow the training setup proposed in [28] with a few adjustments, considering a smaller dataset and a single GPU. We trained the models for 300 epochs, using AdamW optimizer, batch size of 256, initial learning rate of $1\cdot10^{-3}$, weight decay of $4\cdot10^{-5}$ and a 5-epoch linear warmup. We use the implementation of `timm` [64] for training and evaluation.

In Table 3 we report the top-1 accuracy of the various AFs/TAFs. The results show that DiTAC consistently outperforms existing AFs and TAFs in a variety of model configurations and sizes, obtaining an improvement of up to 0.6% in MobileNet-V3 and 0.12% in ResNet.

Classification. Next, we train ConvNeXt-T [40] and Swin-T [39] with DiTAC on ImageNet-100 and ImageNet-1K [53], and compare to its baseline version with a GELU function. As for the ImageNet-1K experiment, we follow the regular training setup mentioned in [40] and [39] for ConvNeXt and Swin training

Table 3: A comparison of DiTAC with existing AFs/TAFs on a small-scale classification problem. We show the classification top-1 accuracy (in %) on ImageNet-50 on various MobileNet-V3 (MN) and ResNet (RN) configurations. For PDELU, we were unable to report the accuracy in several cases due to its unstable training process.

AF/TAF	MN 0.5	MN 0.75	MN 1.0	RN-18	RN-34	RN-50
ReLU	77.36	79.76	80.83	89.64	91.96	93.00
LReLU	77.60	79.88	81.99	89.64	91.60	**93.24**
GELU	79.32	82.12	83.51	89.92	92.12	92.84
ELU	76.00	79.00	80.83	88.12	92.20	93.12
Softplus	75.12	79.40	80.43	89.04	91.36	92.40
Mish	78.96	82.00	83.19	88.96	92.56	92.60
Swish	77.20	81.52	82.99	88.48	91.40	92.32
PReLU	77.92	81.16	81.68	88.96	91.60	92.56
PDELU	77.00	80.96	82.28	–	–	–
Meta-ACON	79.20	81.96	84.04	89.64	92.08	92.20
DiTAC	**79.92**	**82.48**	**84.19**	**90.04**	**92.64**	**93.24**

procedures, respectively. Both models were trained using 300-epochs schedule and 4-GPUs setup. We use an AdamW optimizer with a momentum of 0.9, a weight decay of 0.05 and 20 epochs of linear warmup. In ConvNeXT, we set the initial learning rate to $4 \cdot 10^{-3}$, and use an effective batch size of 4096. As for Swin, we use an initial learning rate of $1 \cdot 10^{-3}$ and an effective batch size of 1024. The training details of ImageNet-100 appear in the Appendix, along with additional information about the training process on ImageNet-1K. We use the implementation of `timm` [64] for training and evaluation.

In Table 4 we show the top-1 accuracy of DiTAC compared to the baseline AF. In ImageNet-100 we train the baseline AF ourselves, whereas in ImageNet-1K we compare to the published results in ConvNeXT and Swin papers. DiTAC noticeably surpasses the baseline performance, gaining an improvement of 0.3% for ConvNeXT-T and 0.2% for Swin-T in ImageNet-1K. DiTAC's advantage is consistent also in the ImageNet-100 experiments.

Table 4: Classification top-1 accuracy (in %) on ImageNet-100 (IN-100) and ImageNet-1K (IN-1K) using a 300-epoch training schedule, on ConvNeXt-T and Swin-T. We compare DiTAC with GELU, the baseline AF, in both architectures.

Configuration	IN-100	IN-1K
ConvNeXt-T	92.4	82.1
ConvNeXt-T+DiTAC	**92.5**	**82.4**
Swin-T	91.4	81.3
Swin-T+DiTAC	**91.7**	**81.5**

Semantic Segmentation. We extend our research and evaluate DiTAC on a semantic segmentation task by comparing the performance of the baseline AF (ReLU) with DiTAC's and GELU's on the segmentation-framework. In the two first experiments we train UperNet [66] as a segmentation framework on the ADE20K dataset [68], where in one configuration we use ConvNeXT-T and in the other we use Swin-T as the backbone. In the third experiment we train a PSPNet [67] segmentation framework using ResNet-50 [25] as the backbone, on the Cityscapes dataset [12]. In all experiments in this section we use the implementation of `mmseg` [11] for training and evaluation, and compare DiTAC's and GELU's performance to `mmseg`'s published results. More details about the training procedures can be found in the Appendix.

In Table 5 we report the mean intersection over union (mIoU) of the described experiments. We observe an improved performance by replacing the baseline AF of each segmentation framework with DiTAC. We notice that GELU behaves differently on each configuration, as opposed to its consistent success in classification models such as ConvNeXT and Swin-Transformer.

Table 5: Segmentation results on UperNet with ConvNeXT-T and Swin-T backbones on ADE20K dataset, and PSPNet with ResNet-50 backbone on Cityscapes dataset. We compare DiTAC with the baseline AF and GELU.

Configuration	Dataset	mIoU
ConvNeXT-T + UperNet	ADE20K	46.1
ConvNeXT-T + UperNet+GELU	ADE20K	45.9
ConvNeXT-T + UperNet+DiTAC	ADE20K	**46.2**
Swin-T + UperNet	ADE20K	44.4
Swin-T + UperNet+GELU	ADE20K	**44.7**
Swin-T + UperNet+DiTAC	ADE20K	**44.7**
ResNet-50 + PSPNet	Cityscapes	77.9
ResNet-50 + PSPNet+GELU	Cityscapes	78.0
ResNet-50 + PSPNet+DiTAC	Cityscapes	**78.4**

To reveal the reason for this gap, in Fig. 7 we display GELU and ReLU (baseline) functions vs. the learned DiTAC in the last layer of the ResNet-50+PSPNet configuration, at time of convergence. It can be seen that DiTAC has learned a unique function shape which led to its improved performance.

Generative Networks. We further evaluate DiTAC's performance on the image generation task by training two types of GAN architectures, DCGAN [50] as an unconditional GAN, and BigGAN [6] as a conditional GAN, on the CelebA [41] and CIFAR-10 [35] datasets, respectively. Similar to the semantic-segmentation experiments, we compare the performance of the baseline AFs with DiTAC's and GELU's on both the Discriminator (D) and Generator (G) modules. DC-

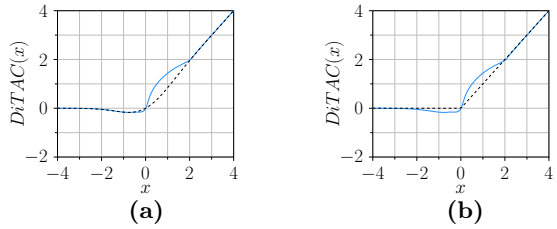


Fig. 7: Here we display (a) the learned DiTAC (solid line) vs. GELU (dashed line), and (b) the learned DiTAC (solid line) vs. ReLU (dashed line), on the semantic segmentation task, trained on ResNet-50+PSPNet. We associate DiTAC's improved performance to its unique function.

GAN uses ReLU in G and LReLU in D as the baseline AFs, whereas BigGAN uses ReLU as the baseline AF in both D and G. We utilize the implementation of `mmgen` [10] for training and evaluation, and follow their suggested training procedure. Additional training details are provided in the Appendix.

To approximate measures of sample quality, we employ the popular standard metrics Fréchet Inception Distance (FID) [27] and Inception Score (IS) [54]. In Table 6 we report the FID and IS results for the above experiments. The results demonstrate a clear advantage to DiTAC on the two architecture types.

Table 6: Image generation performance of DiTAC vs. baseline AFs and GELU, on DCGAN and BigGAN.

Model	Dataset	FID ↓	IS ↑
DCGAN	CelebA	133.5	2.4
DCGAN+GELU	CelebA	516.5	1.0
DCGAN+DiTAC	CelebA	**34.6**	**2.6**
BigGAN	CIFAR-10	10.3	9.5
BigGAN+GELU	CIFAR-10	27.6	7.6
BigGAN+DiTAC	CIFAR-10	**10.0**	**9.8**

5 Conclusion

In this paper we presented DiTAC, a novel highly-expressive TAF that is based on efficient diffeomorphisms. We showed that DiTAC outperforms existing AFs and TAFs in various challenging tasks, while adding only a negligible amount of trainable parameters. The main limitation of DiTAC is its hypothetical computational cost during training. However, this limitation is mostly theoretical as we also provided a technical solution that alleviates this cost in training, and eliminates it completely in inference.

Acknowledgements

This work was supported in part by the Lynn and William Frankel Center at BGU CS, by Israel Science Foundation Personal Grant #360/21, and by the Israeli Council for Higher Education (CHE) via the Data Science Research Center at BGU. I.C. and S.E.F. were also funded in part by the Kreitman School of Advanced Graduate Studies and by BGU's Hi-Tech Scholarship. I.C. was also supported by the Israel's Ministry of Technology and Science Aloni Scholarship.

References

1. Apicella, A., Donnarumma, F., Isgrò, F., Prevete, R.: A survey on modern trainable activation functions. Neural Networks (2021)
2. Barron, J.T.: Continuously differentiable exponential linear units. arXiv preprint arXiv:1704.07483 (2017)
3. Bengio, Y., Léonard, N., Courville, A.: Estimating or propagating gradients through stochastic neurons for conditional computation. arXiv preprint arXiv:1308.3432 (2013)
4. Bengio, Y., Simard, P., Frasconi, P.: Learning long-term dependencies with gradient descent is difficult. IEEE Transactions on Neural Networks (1994)
5. Bishop, C.: Pattern recognition and machine learning. Springer google schola (2006)
6. Brock, A., Donahue, J., Simonyan, K.: Large scale gan training for high fidelity natural image synthesis. arXiv preprint arXiv:1809.11096 (2018)
7. Chen, Y., Dai, X., Liu, M., Chen, D., Yuan, L., Liu, Z.: Dynamic relu. In: European Conference on Computer Vision. pp. 351–367. Springer (2020)
8. Cheng, Q., Li, H., Wu, Q., Ma, L., Ngan, K.N.: Parametric deformable exponential linear units for deep neural networks. Neural Networks (2020)
9. Clevert, D.A., Unterthiner, T., Hochreiter, S.: Fast and accurate deep network learning by exponential linear units (elus). arXiv preprint arXiv:1511.07289 (2015)
10. Contributors, M.: MMGeneration: Openmmlab generative model toolbox and benchmark. https://github.com/open-mmlab/mmgeneration (2021)
11. Contributors, M.: MMSegmentation: Openmmlab semantic segmentation toolbox and benchmark. https://github.com/open-mmlab/mmsegmentation (2020)
12. Cordts, M., Omran, M., Ramos, S., Rehfeld, T., Enzweiler, M., Benenson, R., Franke, U., Roth, S., Schiele, B.: The cityscapes dataset for semantic urban scene understanding. In: CVPR (2016)
13. Detlefsen, N.S., Freifeld, O., Hauberg, S.: Deep diffeomorphic transformer networks. In: CVPR (2018)
14. Dubey, S.R., Singh, S.K., Chaudhuri, B.B.: Activation functions in deep learning: A comprehensive survey and benchmark. Neurocomputing (2022)
15. Duch, W., Jankowski, N.: Survey of neural transfer functions. Neural Computing Surveys (1999)
16. Dugas, C., Bengio, Y., Bélisle, F., Nadeau, C., Garcia, R.: Incorporating second-order functional knowledge for better option pricing. NeurIPS (2000)
17. Finder, S.E., Zohav, Y., Ashkenazi, M., Treister, E.: Wavelet feature maps compression for image-to-image cnns. NeurIPS (2022)
18. Freifeld, O.: Deriving the CPAB derivative. Tech. rep., Ben-Gurion University (2018)

19. Freifeld, O., Hauberg, S., Batmanghelich, K., Fisher III, J.W.: Highly-expressive spaces of well-behaved transformations: Keeping it simple. In: ICCV (2015)
20. Freifeld, O., Hauberg, S., Batmanghelich, K., Fisher III, J.W.: Transformations based on continuous piecewise-affine velocity fields. IEEE TPAMI (2017)
21. Glorot, X., Bordes, A., Bengio, Y.: Deep sparse rectifier neural networks. In: AISTATS (2011)
22. Goodfellow, I., Warde-Farley, D., Mirza, M., Courville, A., Bengio, Y.: Maxout networks. In: ICML. PMLR (2013)
23. Hauberg, S., Freifeld, O., Larsen, A.B.L., III, J.W.F., Hansen, L.K.: Dreaming more data: Class-dependent distributions over diffeomorphisms for learned data augmentation. In: AISTATS (2016)
24. He, K., Zhang, X., Ren, S., Sun, J.: Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In: ICCV (2015)
25. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: CVPR (2016)
26. Hendrycks, D., Gimpel, K.: Gaussian error linear units (gelus). arXiv preprint arXiv:1606.08415 (2016)
27. Heusel, M., Ramsauer, H., Unterthiner, T., Nessler, B., Hochreiter, S.: Gans trained by a two time-scale update rule converge to a local nash equilibrium. Advances in neural information processing systems **30** (2017)
28. Howard, A., Sandler, M., Chu, G., Chen, L.C., Chen, B., Tan, M., Wang, W., Zhu, Y., Pang, R., Vasudevan, V., et al.: Searching for mobilenetv3. In: ICCV (2019)
29. Jaderberg, M., Simonyan, K., Zisserman, A., et al.: Spatial transformer networks. In: NeurIPS (2015)
30. Jagtap, A.D., Kawaguchi, K., Em Karniadakis, G.: Locally adaptive activation functions with slope recovery for deep and physics-informed neural networks. Proceedings of the Royal Society A **476**(2239), 20200334 (2020)
31. Jagtap, A.D., Kawaguchi, K., Karniadakis, G.E.: Adaptive activation functions accelerate convergence in deep and physics-informed neural networks. Journal of Computational Physics **404**, 109136 (2020)
32. Jagtap, A.D., Shin, Y., Kawaguchi, K., Karniadakis, G.E.: Deep kronecker neural networks: A general framework for neural networks with adaptive activation functions. Neurocomputing **468**, 165–180 (2022)
33. Kaufman, I., Weber, R.S., Freifeld, O.: Cyclic diffeomorphic transformer nets for contour alignment. In: ICIP. IEEE (2021)
34. Klambauer, G., Unterthiner, T., Mayr, A., Hochreiter, S.: Self-normalizing neural networks. NeurIPS (2017)
35. Krizhevsky, A., Hinton, G., et al.: Learning multiple layers of features from tiny images. Tech. rep., Toronto, ON, Canada (2009)
36. Kryeem, A., Raz, S., Eluz, D., Itah, D., Hel-Or, H., Shimshoni, I.: Personalized monitoring in home healthcare: An assistive system for post hip replacement rehabilitation. In: ICCV Workshops (2023)
37. LeCun, Y., Bottou, L., Bengio, Y., Haffner, P.: Gradient-based learning applied to document recognition. Proceedings of the IEEE (1998)
38. Lee, K., Yang, J., Lee, H., Hwang, J.Y.: Stochastic adaptive activation function. Advances in Neural Information Processing Systems **35**, 13787–13799 (2022)
39. Liu, Z., Lin, Y., Cao, Y., Hu, H., Wei, Y., Zhang, Z., Lin, S., Guo, B.: Swin transformer: Hierarchical vision transformer using shifted windows. In: ICCV (2021)
40. Liu, Z., Mao, H., Wu, C.Y., Feichtenhofer, C., Darrell, T., Xie, S.: A convnet for the 2020s. In: CVPR (2022)

41. Liu, Z., Luo, P., Wang, X., Tang, X.: Deep learning face attributes in the wild. In: ICCV (2015)
42. Ma, N., Zhang, X., Liu, M., Sun, J.: Activate or not: Learning customized activation. In: CVPR (2021)
43. Maas, A.L., Hannun, A.Y., Ng, A.Y., et al.: Rectifier nonlinearities improve neural network acoustic models. In: ICML (2013)
44. Martinez, I., Viles, E., Olaizola, I.G.: Closed-form diffeomorphic transformations for time series alignment. In: ICML. PMLR (2022)
45. Misra, D.: Mish: A self regularized non-monotonic activation function. arXiv preprint arXiv:1908.08681 (2019)
46. Nagel, M., Fournarakis, M., Amjad, R.A., Bondarenko, Y., Van Baalen, M., Blankevoort, T.: A white paper on neural network quantization. arXiv preprint arXiv:2106.08295 (2021)
47. Nair, V., Hinton, G.E.: Rectified linear units improve restricted boltzmann machines. In: ICML (2010)
48. Neifar, N., Ben-Hamadou, A., Mdhaffar, A., Jmaiel, M., Freisleben, B.: Leveraging statistical shape priors in gan-based ECG synthesis. arXiv preprint arXiv:2211.02626 (2022)
49. Prince, S.J.: Understanding Deep Learning. MIT press (2023)
50. Radford, A., Metz, L., Chintala, S.: Unsupervised representation learning with deep convolutional generative adversarial networks. arXiv preprint arXiv:1511.06434 (2015)
51. Rajanand, A., Singh, P.: Erfrelu: Adaptive activation function for deep neural network. arXiv preprint arXiv:2306.01822 (2023)
52. Ramachandran, P., Zoph, B., Le, Q.V.: Searching for activation functions. arXiv preprint arXiv:1710.05941 (2017)
53. Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., et al.: Imagenet large scale visual recognition challenge. IJCV (2015)
54. Salimans, T., Goodfellow, I., Zaremba, W., Cheung, V., Radford, A., Chen, X.: Improved techniques for training gans. Advances in neural information processing systems **29** (2016)
55. Schwöbel, P., Warburg, F.R., Jørgensen, M., Madsen, K.H., Hauberg, S.: Probabilistic spatial transformer networks. In: UAI (2022)
56. Shacht, G., Danon, D., Fogel, S., Cohen-Or, D.: Single pair cross-modality super resolution. In: CVPR (2021)
57. Shapira Weber, R.A., Eyal, M., Skafte, N., Shriki, O., Freifeld, O.: Diffeomorphic temporal alignment nets. NeurIPS (2019)
58. Shapira Weber, R.A., Freifeld, O.: Regularization-free diffeomorphic temporal alignment nets. In: ICML. PMLR (2023)
59. Skafte Detlefsen, N., Freifeld, O., Hauberg, S.: Deep diffeomorphic transformer networks. In: CVPR (2018)
60. Skafte Detlefsen, N., Hauberg, S.: Explicit disentanglement of appearance and perspective in generative models. NeurIPS (2019)
61. Springenberg, J.T., Riedmiller, M.: Improving deep neural networks with probabilistic maxout units. arXiv preprint arXiv:1312.6116 (2013)
62. Trottier, L., Giguere, P., Chaib-Draa, B.: Parametric exponential linear unit for deep convolutional neural networks. In: IEEE ICMLA (2017)
63. Wang, H., Liu, F., Zhou, Q., Yi, R., Tan, X., Ma, L.: Continuous piecewise-affine based motion model for image animation. In: AAAI (2024)

64. Wightman, R.: Pytorch image models. https://github.com/rwightman/pytorch-image-models (2019). https://doi.org/10.5281/zenodo.4414861
65. Wightman, R., Touvron, H., Jégou, H.: Resnet strikes back: An improved training procedure in timm. arXiv preprint arXiv:2110.00476 (2021)
66. Xiao, T., Liu, Y., Zhou, B., Jiang, Y., Sun, J.: Unified perceptual parsing for scene understanding. In: ECCV (2018)
67. Zhao, H., Shi, J., Qi, X., Wang, X., Jia, J.: Pyramid scene parsing network. In: CVPR (2017)
68. Zhou, B., Zhao, H., Puig, X., Xiao, T., Fidler, S., Barriuso, A., Torralba, A.: Semantic understanding of scenes through the ade20k dataset. IJCV (2019)