



SPLASH: Learnable activation functions for improving accuracy and adversarial robustness

Mohammadamin Tavakoli^{a,*}, Forest Agostinelli^b, Pierre Baldi^a

^a Department of Computer Science, University of California, Irvine, United States of America

^b Department of Computer Science and Engineering, University of South Carolina, United States of America

ARTICLE INFO

Article history:

Received 15 May 2020

Received in revised form 1 February 2021

Accepted 18 February 2021

Available online 4 March 2021

Keywords:

Activation

Neural networks

Accuracy

Robustness

Adversarial

ABSTRACT

We introduce SPLASH units, a class of learnable activation functions shown to simultaneously improve the accuracy of deep neural networks while also improving their robustness to adversarial attacks. SPLASH units have both a simple parameterization and maintain the ability to approximate a wide range of non-linear functions. SPLASH units are: (1) continuous; (2) grounded ($f(0) = 0$); (3) use symmetric hinges; and (4) their hinges are placed at fixed locations which are derived from the data (i.e. no learning required). Compared to nine other learned and fixed activation functions, including ReLU and its variants, SPLASH units show superior performance across three datasets (MNIST, CIFAR-10, and CIFAR-100) and four architectures (LeNet5, All-CNN, ResNet-20, and Network-in-Network). Furthermore, we show that SPLASH units significantly increase the robustness of deep neural networks to adversarial attacks. Our experiments on both black-box and white-box adversarial attacks show that commonly-used architectures, namely LeNet5, All-CNN, Network-in-Network, and ResNet-20, can be up to 31% more robust to adversarial attacks by simply using SPLASH units instead of ReLUs. Finally, we show the benefits of using SPLASH activation functions in bigger architectures designed for non-trivial datasets such as ImageNet.

© 2021 Elsevier Ltd. All rights reserved.

1. Introduction

Nonlinear activation functions are fundamental for deep neural networks (DNNs). They determine the class of functions that DNNs can implement and influence their training dynamics, thereby affecting their final performance. For example, DNNs with rectified linear units (ReLUs) (Nair & Hinton, 2010) have been shown to perform better than logistic and tanh units in several scenarios (e.g. Nair & Hinton, 2010; Nwankpa, Ijomah, Gachagan, & Marshall, 2018; Pedamonti, 2018). Instead of using a fixed activation function, one can use a parameterized activation function and learn its parameters to add flexibility to the model. Piecewise linear functions are a reasonable choice for the parameterization of activation functions (Agostinelli, Hoffman, Sadowski, & Baldi, 2015; Baldi, 2021; He, Zhang, Ren, & Sun, 2015; Jin et al., 2016; Li, Ouyang, & Wang, 2016; Ramachandran, Zoph, & Le, 2017) due to their straightforward parameterization and their ability to approximate non-linear functions (Garvin, Crandall, John, & Spellman, 1957; Stone, 1961). However, in the context

of deep neural networks, the best way to parameterize these piecewise linear activation functions is still an open question. Previous piecewise linear activation functions either sacrifice expressive power for simplicity (i.e. having few parameters) or sacrifice simplicity for expressive power. While expressive power allows deep neural networks to approximate complicated functions, simplicity can make optimization easier by adding useful inductive biases and reducing the size of the hypothesis space. Therefore, we set out to find a parameterized piecewise linear activation function that is as simple as possible while maintaining the ability to approximate a wide range of functions.

Piecewise linear functions, in the most general form, are real-valued functions defined as $S + 1$ line segments with S hinges that denote where one segment ends and the next segment begins. As detailed in Section 3, a function of this most general form requires $3S + 2$ parameters. Many functions in this hypothesis space, such as discontinuous functions, are unlikely to be useful activation functions. Therefore, we significantly reduce the size of the hypothesis space while maintaining the ability to approximate a wide range of useful activation functions. We restrict the form of the piecewise linear function to be continuous and grounded (having an output of zero for an input of zero) with symmetric and fixed hinges. By doing so, we reduce the number of parameters to $S + 1$. Furthermore, we still maintain the ability to approximate almost every successful deep neural network activation function. We call this parameterized piecewise

* Correspondence to: School of Information and Computer Sciences, Office: 248, ICS 2, University of California, Irvine (UCI), Irvine, CA 92697, United States of America.

E-mail addresses: mohamadt@uci.edu (M. Tavakoli), foresta@cse.sc.edu (F. Agostinelli), pfbaldi@uci.edu (P. Baldi).

linear activation function SPLASH (Simple Piecewise Linear and Adaptive with Symmetric Hinges).

Typically, learned activation functions are evaluated in terms of accuracy on a test set. We compare the classification accuracy of SPLASH units to nine other learned and fixed activation functions and show that SPLASH units consistently give superior performance. We also perform ablation studies to gain insight into why SPLASH units improve performance and show that the flexibility of the SPLASH units during training significantly affects the final performance. In addition, we also evaluate the robustness of SPLASH units to adversarial attacks (Goodfellow, Shlens, & Szegedy, 2014; Nguyen, Yosinski, & Clune, 2015; Szegedy et al., 2013). When compared to ReLUs, SPLASH units reduce the success of adversarial attacks by up to 31%, without any modifications to how they are parameterized or learned.

2. Related work

Variants of ReLUs, such as leaky-ReLUs (Maas, Hannun, & Ng, 2013), exponential linear units (ELUs) (Clevert, Unterthiner, & Hochreiter, 2015), and scaled exponential linear units (SELUs) (Klambauer, Unterthiner, Mayr, & Hochreiter, 2017) have been shown to improve upon ReLUs. ELUs and SELUs encourage the outputs of the activation functions to have zero mean while SELUs also encourage the outputs of the activation functions to have unit variance. Neural architecture search (Ramachandran et al., 2017) has also discovered novel activation functions, in particular, the Swish activation function. The Swish activation function is defined as $f(x) = x * (1 + e^{-\beta x})^{-1}$ and performs slightly better than ReLUs. It is worth mentioning that, in Lin, Chen, and Yan (2013), the authors proposed the network-in-network approach where they replace activation functions in convolutional layers with small multi-layer perceptrons. Theoretically, due to universal approximation theorem (Csáji et al., 2001), this is the most expressive activation function; however, it requires many more parameters.

Some of the early attempts to learn activation functions in neural networks can be found in Khan, Ahmad, Khan, and Miller (2013), Poli (1996), and Weingaertner, Tatai, Gudwin, and Von Zuben (2002), where the authors proposed learning the best activation function per neuron among a pool of candidate activation functions using genetic and evolutionary algorithms. Maxout (Goodfellow, Warde-Farley, Mirza, Courville, & Bengio, 2013) has been introduced as an activation function aimed at enhancing the model averaging properties of dropout (Srivastava, Hinton, Krizhevsky, Sutskever, & Salakhutdinov, 2014). However, not only is it limited to approximating convex functions, but it also requires a significant increase in the number of parameters.

APL units (Agostinelli et al., 2015), P-ReLUs (He et al., 2015) and S-ReLUs (Jin et al., 2016) are adaptive activation functions from the piecewise linear family that can mimic both convex and non-convex functions. Of these activation functions, APL units are the most general. However, they require a parameter for the slope of each line segment as well as for the location of each hinge. Additionally, APL units give more expressive power to the left half of the input space than to the right half. Furthermore, the locations of the hinges are randomly assigned, therefore, it is possible that some line segments may go unused. S-ReLUs also learn the slopes of the line segments and the locations of the hinges, however, the initial locations of the hinges are determined by the data. S-ReLUs have less expressive power than APL units as the form of the function is restricted to only have two hinges. P-ReLUs are the simplest of these activation functions with one fixed hinge where only the slope of one of the line segments is learned. On the other hand, SPLASH units can have few or many hinges and the locations of the hinges are fixed and derived from the data. Therefore, only the slopes of the line segments have to be learned. Furthermore, SPLASH units give equal expressive power to the left and the right half of the input space.

3. From piecewise linear functions to SPLASH units

3.1. Family of piecewise linear functions

Given $S + 1$ line segments and S hinges, piecewise linear functions can be parameterized with $2(S + 1) + S = 3S + 2$ parameters: one parameter for the slope and one parameter for the y-intercept of each segment, plus S parameters for the locations of the hinges. We go from the most general case to SPLASH units by reducing the number of parameters to $S + 1$ while still being able to approximate a wide range of functions by restricting the activation function to be **continuous** and **grounded** with **symmetric** and **fixed** hinges.

Continuous

The general form of piecewise linear functions allows for discontinuous functions. Because virtually all successful activation functions are continuous, we argue that continuous learnable activation functions will still provide sufficient flexibility for DNNs. For a continuous piecewise linear function, we need to specify the y-intercept of one segment, the slopes of the $S + 1$ segments, as well as the locations of the S hinges, reducing the number of parameters to $2S + 2$.

Grounded

Furthermore, we restrict the function to be grounded, that is, having an output of zero for an input of zero. Since the y-intercept is fixed at zero, we no longer have to specify the y-intercept for any of the segments, reducing the number of parameters to $2S + 1$.

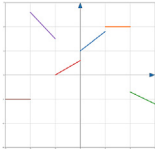
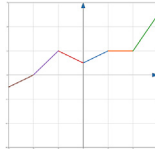
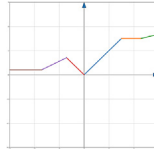
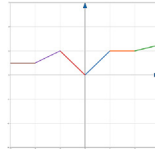
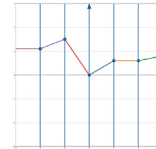
Symmetric hinges

In our design, we place the hinges in symmetric locations on the positive and negative halves on the x -axis, giving equal expressive power to each half. This allows, if need be, the activation function to approximate both even and odd functions. Because the location of one hinge determines the location of another, we can reduce the number of parameters for the hinges to $\lfloor \frac{S}{2} \rfloor$. In the case of an odd number of hinges, one hinge will be fixed at zero to maintain symmetry. This reduces the number of parameters to $S + 1 + \lfloor \frac{S}{2} \rfloor$.

Fixed hinges

Finally, we address the issue of where to set the exact location of each segment. It is important that each segment has the potential to influence the output of the function. The distribution of the input could be such that only some of the segments influence the output while others remain unused. In the worst case, the input could be concentrated on a single segment, reducing the activation function to just a linear function. To ensure that each segment is able to play a role in the output of the function, we train our DNNs using batch normalization (Ioffe & Szegedy, 2015). At the beginning of training, batch normalization ensures that, for each batch, the input to the activation function has a mean of zero and a standard deviation of one. In addition, this facilitates the uniform calibration of the location of the hinges in units of standard deviations. For instance, placing the hinges at $0, \pm 1$, corresponds to placing them at the origin and at one standard deviation from the origin. With the location of the hinges fixed, the number of parameters is reduced to $S + 1$. This activation function can approximate the vast majority of existing activation functions, such as tanh units, ReLUs, leaky ReLUs, ELUs, and, with the use of a bias, logistic units. We show the different types of piecewise linear functions that we have described in Table 1.

Table 1
Different types of piecewise linear functions defined on N intervals. The rightmost function is what we use to parameterize our SPLASH activation functions.

Type	General	Continuous	Continuous grounded	Continuous grounded symmetric hinges	Continuous grounded symmetric hinges fixed hinges
# Params	$3S + 2$	$2S + 2$	$2S + 1$	$S + 1 + \lfloor \frac{S}{2} \rfloor$	$S + 1$
Viz					

3.2. SPLASH units

We formulate the activation of a hidden unit h as the summation of $S + 1$ max functions with S symmetric offsets, where S is an odd number and one of the offsets is zero:

$$h(x) = \sum_{s=1}^{(S+1)/2} a_+^s \max(0, x - b^s) + \sum_{s=1}^{(S+1)/2} a_-^s \max(0, -x - b^s) \quad (1)$$

The first summation contains max functions with a non-zero output starting at $x \geq 0$ and continuing to infinity. The second summation contains max functions with a non-zero output starting at $x \leq 0$ and continuing to negative infinity. When summed together, these max functions form $S + 1$ continuous and grounded line segments with hinges located at b^s and $-b^s$. To ensure the function has symmetric and fixed hinges, we use the same b^s in both summations, where $b^s \geq 0$ for all s ; furthermore, we have the values of b^s remain fixed during training. Since we are using batch normalization, we fix the positions of the hinges b^s for each s to be a predetermined number of standard deviations away from the mean. We ensure there is always one hinge at zero by setting b^1 to be zero. The summation of the learned parameters a_+^s and a_-^s , $\sum_1^i a_+^i$ or $\sum_1^i a_-^i$ determines the slope of i th line segment. These parameters are shared across all units in a layer. Therefore, SPLASH units add $S + 1$ parameters per layer. We study the effect of different initializations as well as the effect of the number of hinges, S , on training accuracy. From our experiments, we found that initializing SPLASH units to have the shape of a ReLU and setting S to 7 gave the best results (see Appendix for additional details).

The following theorem shows that SPLASH units can approximate any non-linear and uniformly continuous function that has an output of zero for an input of zero in a closed interval of real numbers.

Theorem 3.1. For any function $f : [A, B] \rightarrow \mathbb{R}$ and $\epsilon \in \mathbb{R}^+$, $\exists S \in \mathbb{N}$, where $|f(x) - \text{SPLASH}(x)| \leq \epsilon$, assuming:

- A and B are finite real numbers.
- f is uniformly continuous.

Proof. Uniform continuity of f implies that for every $\epsilon \in \mathbb{R}^+$, $\exists \delta > 0$ such that for every x and $y \in [A, B]$ where $|x - y| \leq \delta$, then we have $|f(x) - f(y)| \leq \epsilon$. Placing S equally distanced hinges $\{H_1, \dots, H_S\}$ on the interval $[A, B]$, divides this into $S + 1$ equal sub-intervals $[H_i, H_{i+1}]$. We choose S to be greater than $\frac{B-A}{\delta} - 1$, so the length of each sub-interval would be smaller than δ . For any of the sub-intervals starting at $H_i \in \{H_1, \dots, H_S\}$, we approximate f by a line segment which connects $f(H_i)$ to $f(H_i + \frac{B-A}{S+1})$. Due to the linear form of SPLASH(x) for $x \in [H_i, H_i + \frac{B-A}{S+1}]$:

$$|f(x) - \text{SPLASH}(x)| \leq \max(\max_x |f(x) - f(H_i)|, \max_x |f(x) - f(H_{i+1})|) \quad (2)$$

f is uniformly continuous, so:

$$|f(x) - \text{SPLASH}(x)| \leq \epsilon \quad (3)$$

Now we need to show that SPLASH function (i.e., Eq. (1)) is able to connect $f(H_i)$ to $f(H_i + \frac{B-A}{S+1})$ for $H_i \in \{H_1, \dots, H_S\}$. We do so by a simple induction as follows: Suppose that $f(H_i)$ connected to $f(H_{i+1})$ for $i \in \{1, \dots, i - 1\}$. The slope of SPLASH in the sub-interval $[H_{i-1}, H_i]$ are set to be $\sum_1^i a_+^i$ or $\sum_1^i a_-^i$ (depending on the sign of the sub-interval). However, the slope of SPLASH in the sub-interval $[H_i, H_{i+1}]$ is either $\sum_1^{i+1} a_+^i$ or $\sum_1^{i+1} a_-^i$. In both cases, the extra term a_+^{i+1} or a_-^{i+1} can change the slope to any arbitrary value. This fact plus the assumption of continuity of SPLASH guarantees that $f(H_i)$ can be connected to $f(H_{i+1})$ which was our proposed approximation. The last thing to mention is that since SPLASH is grounded ($\text{SPLASH}(0) = 0$), this approximation by line segments can only approximate functions f where $f(0) = 0$. \square

4. Accuracy

4.1. Comparison to other activation functions

In order to demonstrate the benefits provided by SPLASH units in deep learning, we compare them to other well-known activation functions across several different deep learning architectures and data sets. We train LeNet5 (LeCun, Bottou, Bengio, & Haffner, 1998), Network-in-Network (Lin et al., 2013), All-CNN (Springenberg, Dosovitskiy, Brox, & Riedmiller, 2014), and ResNet-20 (He, Zhang, Ren, & Sun, 2016), on three different datasets: MNIST (LeCun et al., 1998), CIFAR-10, and CIFAR-100 (Krizhevsky, Hinton, et al., 2009). We use batch normalization before each SPLASH layer so the input to the SPLASH layer has a mean of zero ($\mu = 0.0$) and a standard deviation of one ($\sigma = 1.0$). We empirically searched the space of hinge locations and found that setting $S = 7$ with hinges at $x = 0.0, \pm 1.0\sigma, \pm 2.0\sigma, \pm 2.5\sigma$, works well in practice where $\sigma = 1.0$ is the standard deviation of the input distribution. a_+^1 is initialized to 1 and the remaining slopes are initialized to 0. With this initialization, the starting shape of a SPLASH unit mimics the shape of a ReLU.

With the exception of the All-CNN architecture, moderate data augmentation is performed as it is explained in He et al. (2016). Moderate data augmentation adds horizontally flipped examples of all images to the training set as well as random translations with a maximum translation of 5 pixels in each dimension. For the All-CNN architecture, we use heavy data augmentation (Springenberg et al., 2014). More details about the hyperparameters are given in Appendix.

We compare SPLASH units to ReLUs, leaky-ReLUs, PReLUs, APL units, tanh units, sigmoid units, ELUs, maxout units with nine features, and Swish units. The hyperparameters for each DNN are tuned using ReLUs and use the same hyperparameters for each activation function. The results of the experiments are shown in Table 2. We report the average and the standard deviation of the error rate on the test set across five runs. As reported

Table 2

Results of training four different DNN architectures from the literature. For each architecture, the first row represents the results of our implementation of the version described in the literature. The second row represents the best performance of the same architecture when trained using the following nine activation functions: ReLU, leaky-ReLU, tanh, sigmoid, ELU, maxout with nine features, Swish ($\beta = 0.2$), and APL with $S = 5$. Finally, the last row represents the results of the same architecture when trained using SPLASH activation functions. In all cases, each architecture is trained and tested five times. The mean and standard deviation are reported, and the corresponding significance metrics are given in Table 11 in the Appendix.

Activation	MNIST	CIFAR-10		CIFAR-100	
		–	D-A	–	D-A
LeNet5 + ReLU	1.11 ± 0.09	30.98 ± 1.14	23.41 ± 1.31	–	–
LeNet5 + PReLU*	1.13 ± 0.04	30.71 ± 0.69	23.33 ± 0.88	–	–
LeNet5 + SPLASH	1.03 ± 0.07	30.14 ± 0.99	22.93 ± 1.24	–	–
Net in Net + ReLU	–	9.71 ± 0.69	8.11 ± 0.81	36.06 ± 0.84	32.98 ± 1.10
Net in Net + APL* (Agostinelli et al., 2015)	–	9.59 ± 0.24	7.51 ± 0.14	34.40 ± 0.16	30.83 ± 0.24
Net in Net + SPLASH	–	9.21 ± 0.55	7.29 ± 0.93	33.91 ± 0.97	30.32 ± 0.66
All-CNN + ReLU	–	9.24 ± 0.48	7.42 ± 0.59	34.11 ± 0.79	32.43 ± 0.73
All-CNN + maxout*	–	9.19 ± 0.51	7.45 ± 0.41	34.21 ± 0.88	32.33 ± 0.91
All-CNN + SPLASH	–	9.02 ± 0.33	7.18 ± 0.41	33.14 ± 0.71	32.06 ± 0.66
ResNet-20 + ReLU	–	10.65 ± 0.55	8.71 ± 0.51	34.54 ± 0.88	32.63 ± 0.67
ResNet-20 + APL*	–	10.29 ± 0.71	8.59 ± 0.58	34.62 ± 0.79	32.51 ± 0.81
ResNet-20 + SPLASH	–	9.98 ± 0.42	8.18 ± 1.02	33.97 ± 0.51	32.12 ± 0.77

in Table 2, our implementations of the architectures mentioned above, using SPLASH activation functions, outperform the original results reported in the literature for the same architectures with different activation functions (He et al., 2016; Li, 2017; Lin et al., 2013; Springenberg et al., 2014). Table 2 also shows that SPLASH units have the best performance across all datasets and architectures, reducing relative classification error by up to 10% when compared to ReLUs (see Appendix for additional details on each experiment).

4.2. Insights into why SPLASH units improve accuracy

Fig. 1 shows how the shape of the SPLASH units change during training for the ResNet-20 architecture. From these figures, we can see that, during the early stages of training, the SPLASH units have a negative output for a negative input and a positive output for a positive input. During the later stages of training, SPLASH units have a positive output for both a negative input and a positive input. SPLASH units look similar to that of a leaky-ReLU during the early stages of training and look similar to a symmetric function during the later stages of training.

To better understand why SPLASH units lead to better performance, we used the final shape of the SPLASH units as a fixed activation function to train another ResNet-20 architecture. In Fig. 2, we can see that the performance is only as good as that of ReLUs. This leads us to believe that the evolution of the shape of the SPLASH units during training is crucial to obtaining improved performance. Since we observed that SPLASH units would first give a negative output for a negative input and then give a positive output for a negative input, we train ResNet-20 with SPLASH units under two different conditions: (1) the first slope on the negative half of the input (a^{\perp}) is forced to be only positive, yielding a negative output for the line segment at zero (SPLASH-negative units) and (2) the first slope on the negative half of the input (a^{\perp}) is forced to be only negative, yielding a positive output for the line segment at 0 (SPLASH-positive units).

The performance of SPLASH-positive and SPLASH-negative units is shown in Fig. 2. The figure shows that, although SPLASH-positive units have the ability to mimic the final learned shape of SPLASH units, it performs worse than SPLASH units and only slightly better than ReLUs. This shows that the ability to give a negative output for a negative input is crucial for SPLASH units. Furthermore, SPLASH-negative units perform better than SPLASH-positive units, but still worse than SPLASH units. In addition, we see that SPLASH-negative units exhibit a relatively quick decrease in the training loss, similar to that of SPLASH units, but do not

reach the final training loss of SPLASH units. These observations suggest that the flexibility of the learnable activation function plays a crucial role in the final performance.

4.3. Tradeoffs

The benefits of SPLASH units come at the cost of longer training time. The average per epoch training time (in seconds) and the final accuracy of a variety of fixed and learned activation functions are reported in Table 3. The table shows that training with SPLASH units can take between 1.2 and 3 times longer, depending on S and the chosen architecture. We observe that the error does not significantly decrease beyond $S = 7$. Therefore, we chose $S = 7$ for our experiments. While, for many deep learning algorithms, obtaining better performance often comes at the cost of longer training times, in Section 5, we show that SPLASH units also improve the robustness of deep neural networks to adversarial attacks.

5. Robustness to adversarial attacks

DNNs have been shown to be vulnerable to many types of adversarial attacks (Goodfellow et al., 2014; Szegedy et al., 2013). Research suggests that activation functions are a major cause of this vulnerability (Brendel, Rauber, & Bethge, 2017; Rozsa & Boul, 2019; Zantedeschi, Nicolae, & Rawat, 2017). For example, Zhang, Weng, Chen, Hsieh, and Daniel (2018) bounded a given activation function using linear and quadratic functions with adaptive parameters and applied a different activation for each neuron to make neural networks robust to adversarial attacks. Wang et al. (2018) proposed a data-dependent activation function and empirically showed its robustness to both black-box and gradient-based adversarial attacks. Other studies such as Dhillon et al. (2018), Rakin, Yi, Gong, and Fan (2018), and Song, Chen, Cheung, and Kuo (2018) focused on other properties of activation functions, such as quantization and pruning, and showed that they can improve the robustness of DNNs to adversarial examples.

Among all these proposed activation functions, most of them are optimized through an adversarial training task which is extremely time-consuming and computationally expensive. There are also several studies focusing on designing activation functions without adversarial training. One of the most successful activation functions designed to improve adversarial robustness is the Tent activation function (Rozsa & Boul, 2019). This activation

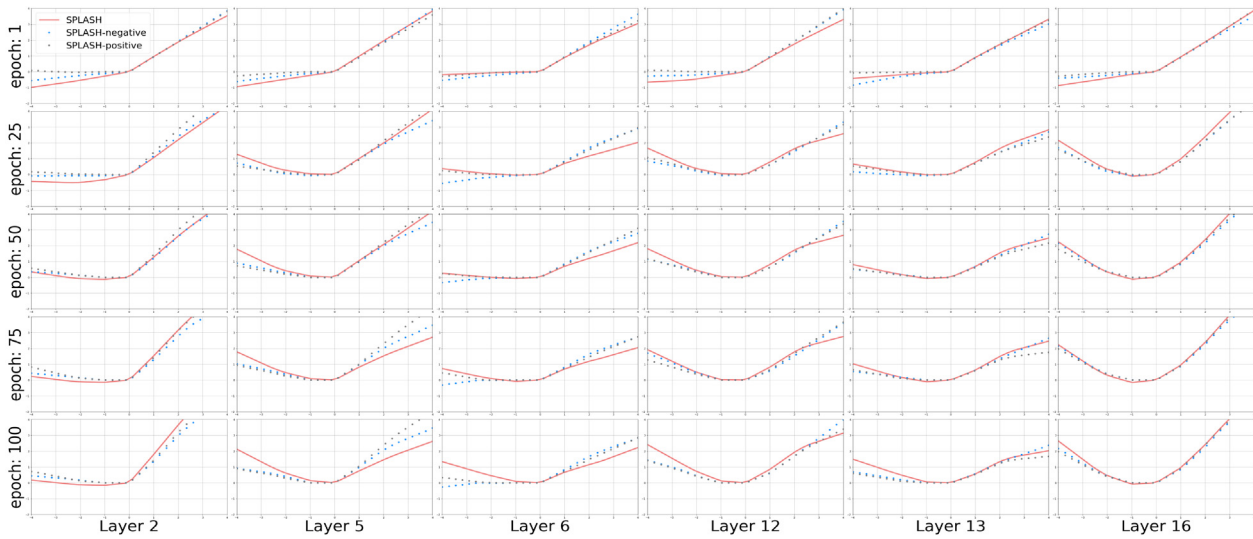


Fig. 1. The shape of the SPLASH units in six different layers of the ResNet-20 architecture during training on the CIFAR-10 dataset. Each SPLASH layer is placed after a 2D convolution layer and batch normalization layer. In the early stages of training, the shape of SPLASH units looks visually similar to that of a leaky-ReLU. However, during the later stages of training, the shape of SPLASH units looks visually similar to that of a symmetric function.

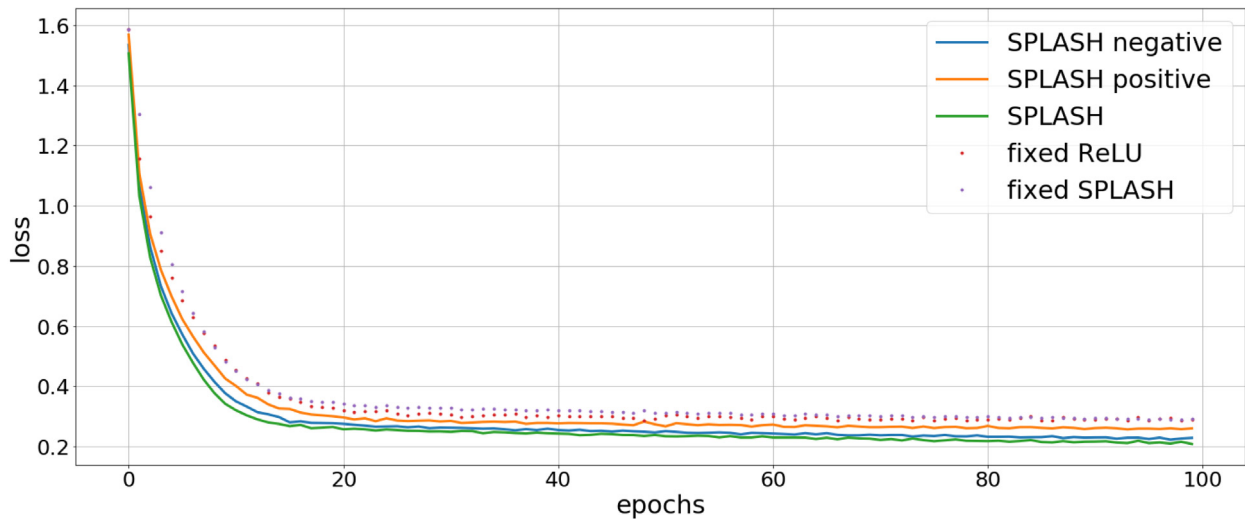


Fig. 2. Training loss for ReLUs and different types of SPLASH units for the ResNet-20 architecture on CIFAR-10. SPLASH units converge faster and also have the lowest final loss. Fixed SPLASH is a fixed activation function that mimics the final shape of the SPLASH units trained on the ResNet-20 architecture. Fixed SPLASH performs only about as well as ReLUs. SPLASH-negative units perform better than SPLASH-positive units, however, they perform worse than SPLASH units. Furthermore, although SPLASH-positive units have the ability to mimic the final shape of SPLASH units, they perform worse.

Table 3

Per-epoch training time is reported in seconds. The benefits of SPLASH come at the cost of slower training time. All models are trained using NVIDIA TITAN V GPU with 12036MiB memory and 850MHz. Maxout is trained with six features and APL is set to have five hinges. For the sake of brevity, T and E are corresponding to per-epoch training time and error rate respectively.

Activation		SPLASH					Tanh	maxout	ReLU	Swish	APL
		S = 3	S = 5	S = 7	S = 9	S = 11					
MNIST (LeNet5)	T	18.1	20.2	24.0	28.5	31.8	15.4	19.4	12.1	12.5	19.3
	E	1.14	1.10	1.03	1.01	1.05	1.25	1.31	1.11	1.18	1.14
CIFAR-10 (LeNet5)	T	21.3	24.7	29.7	33.1	35.4	19.6	22.1	17.2	17.6	24.0
	E	30.79	30.57	30.20	30.14	30.11	31.14	31.01	30.88	30.69	30.66

function is designed based on the hypothesis that adversarial attacks exploit the open space risk of classic monotonic activation functions such as ReLU. The Tent activation function bounds the open space risk and improves the adversarial robustness of DNNs without the need for adversarial training.

Also, recently, authors in [Zhao and Griffin \(2016\)](#) theoretically showed that DNNs with symmetric activations are less likely to

get fooled. The authors proved that “symmetric units suppress unusual signals of exceptional magnitude which result in robustness to adversarial fooling and higher expressibility”. This fact can also be seen in the design of the Tent activation function which is inherently symmetric.

Because SPLASH units are capable of approximating a symmetric function and bounding the open space risk, they may also

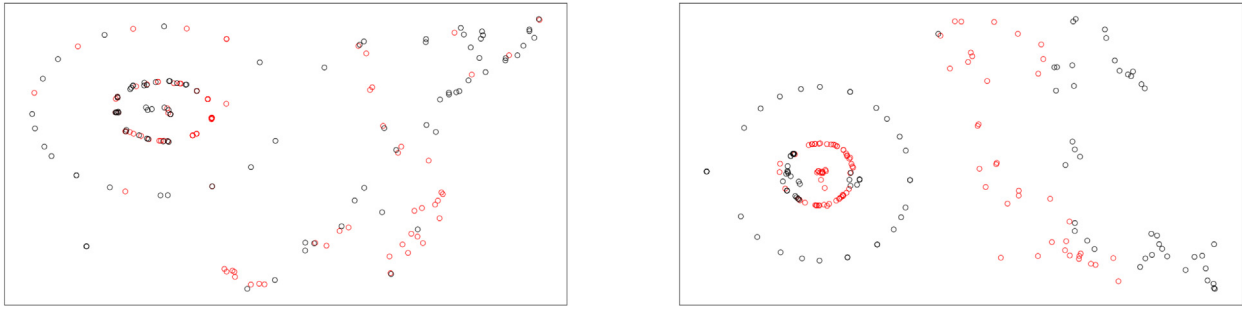


Fig. 3. tSNE visualization of the pre-softmax layer's outputs for the LeNet5 architecture trained on CIFAR-10. Left: Trained with ReLUs. Right: Trained with SPLASH units. Red and black points are 100 random samples from the frog and ship images of the CIFAR-10 dataset. The figures show that the samples from these two classes are better separated using the DNN trained with SPLASH units.

be capable of increasing the robustness of DNNs to adversarial attacks. In this section, we show that SPLASH units greatly improve the robustness of DNNs to adversarial attacks with no adversarial training. This claim is verified through a wide range of experiments with the CIFAR-10 dataset under both black-box and white-box adversarial attacks.

An intuition for why a DNN with SPLASH units is more robust than a DNN with ReLUs is provided in Fig. 3. For each of the two networks, we take 100 random samples of frog and ship images and visualize the pre-softmax representations using the tSNE visualization (Maaten & Hinton, 2008) in Fig. 3. The figure shows that the two classes have less overlap for the DNN with SPLASH units than for the DNN with ReLUs.

5.1. Black-box adversarial attacks

For black-box adversarial attacks, we assume the adversary has no information about the parameters of the DNN. The adversary can only observe the inputs to the DNN and outputs of the DNN, similar to that of a cryptographic oracle. We test the robustness of DNNs with SPLASH units using two powerful black box adversarial attacks, namely, the one-pixel attack and the boundary attack. For each attack, we measure the adversarial robustness using the success rate (i.e. the number of successful attacks). An attack on the pair of input and output (x, y) is considered to be a successful attack, once the adversarially modified image x' is classified as $y' \neq y$. In other words, for the pair (x, y) where $f(x) = y$, an attack is successful if $f(x' = x + \epsilon) \neq y$, where $f(\cdot)$ is the neural network and ϵ is the adversarial modification. In the case of a successful adversarial attack, we compare the confidence of the true label to that of the misclassified label. More precisely, we measure the average of $Z(x')_{\text{adversarial_label}} - Z(x')_{\text{true_label}}$ over all adversarial examples where the network is fooled, where $Z(\cdot)$ is the output of the softmax layer and x' is the adversarial sample.

5.1.1. One pixel attack

A successful one pixel attack was proposed by Su, Vargas, and Sakurai (2019), which is based on differential evolution. Using this technique, we can iteratively generate adversarial images to try to minimize the confidence of the true class. The process starts with randomly modifying a few pixels to generate adversarial examples. At each step, several adversarial images are fed to the DNN and the output of the softmax function is observed. Examples that lowered the confidence of the true class will be kept to generate the next generation of adversaries. New adversarial images are then generated through mutations. By repeating these steps for a few iterations, the adversarial modifications generate more and more misleading images. The last step returns the adversarial modification that reduced the confidence of the true

class the most, with the goal being that a class other than the true class has the highest confidence.

In the following experiment, we modify one, three, and five pixels of images to generate adversarial examples. The mutation scheme we used for this experiment is as follows:

$$x_i^{l+1} = x_{r_1}^l + 0.5(x_{r_2}^l + x_{r_3}^l) \quad (4)$$

where r_1 , r_2 , and r_3 are three non-equal random indices of the modifications at step l . x_i^{l+1} will be an element of a new candidate modification.

To evaluate the effect of SPLASH units on the robustness of DNNs, we employ commonly-used architectures, namely, LeNet5, Network-in-Network, All-CNN, and ResNet-20. Each architecture is trained with ReLUs, APL units, Swish units, Tent units, and SPLASH units. The results shown in Table 4 show that SPLASH units significantly improve robustness to adversarial attacks for all architectures and outperform all other activation functions. In particular, for LeNet5 and ResNet-20, SPLASH units improve performance over ReLUs by 31% and 28%, respectively. When adversarial attacks are successful, we found that DNNs with SPLASH units still assign higher confidence to the true labels of the perturbed images than ReLUs and Swish units. For each model, this measurement is included in Table 4.

5.1.2. Boundary attack

We use another black-box adversarial attack to further examine the effect SPLASH units have on the robustness of DNNs to adversarial fooling. Boundary attacks, which were recently introduced by Brendel et al. (2017), are a powerful and commonly used black-box adversarial attack. Considering the original pair of input image and the corresponding target as (x, l_x) , the attack algorithm is initialized from an adversarial pair of (x_{adv}^0, l_{adv}^0) , where $x_{adv}^0 \sim \mathcal{N}(0, 1)$ s.t. $l_{adv}^0 \neq l_x$. Then, a random walk is performed K times along the boundary between the adversarial region, S_{adv} , $\forall x' \in S_{adv}, l_x \neq l_x$, and the region of the true label such that (1) x_{adv} stays in the adversarial region and (2) the distance towards the original image $d(x, x_{adv}^k)$ is reduced. The random walk uses the following three steps: (1) Draw a random sample μ from an i.i.d. Gaussian as the direction of the next move. (2) Project the sampled direction onto the sphere centered at x with a radius of $\|x - x_{adv}^{k-1}\|$ and take a step of size $\epsilon = \frac{\|\mu\|_2}{d(x, x_{adv}^{k-1})}$ in this projected direction. This step guarantees that the perturbed image gets closer to the original image at each step. (3) Make a move of size δ towards the original image, where $\delta = \frac{d(x, x_{adv}^{k-1}) - d(x, x_{adv}^k)}{d(x, x_{adv}^{k-1})}$. Ideally, this algorithm will converge to the adversarial sample x_{adv}^k which is the closest to the original input x . The details and hyper-parameters of the attack are explained in Appendix.

Table 4

Robustness to the one-pixel attack using 1000 images, randomly chosen from the correctly classified images of the CIFAR-10 test set. We attack each architecture five times and report the results in the form of *mean ± standard deviation* of the number of successful attacks. The maximum number of iterations for all attacks is set to 40. $avg(Z_{true} - Z_{adv})$ is computed for the one-pixel attack.

Model	Activation	One-pixel	Three-pixels	Five-pixels	$avg(Z_{adv} - Z_{true})$
LeNet5	ReLU	736 ± 12.3	803 ± 12.7	868 ± 28.9	0.740
	Swish	701 ± 14.2	780 ± 17.4	840 ± 11.0	0.805
	APL	635 ± 15.9	709 ± 9.8	781 ± 17.7	0.465
	Tent	593 ± 7.4	677 ± 7.4	719 ± 15.9	0.411
	SPLASH	514 ± 17.2	588 ± 7.4	651 ± 21.7	0.540
Net in Net	ReLU	644 ± 16.5	701 ± 20.0	769 ± 18.3	0.621
	Swish	670 ± 28.5	715 ± 33.8	760 ± 26.1	0.419
	APL	521 ± 21.2	661 ± 19.9	703 ± 22.6	0.455
	Tent	491 ± 9.8	588 ± 23.5	649 ± 16.3	0.362
	SPLASH	449 ± 18.6	530 ± 16.3	599 ± 23.5	0.311
All-CNN	ReLU	580 ± 17.2	661 ± 15.0	707 ± 25.7	0.366
	Swish	597 ± 23.5	630 ± 33.9	699 ± 34.6	0.511
	APL	509 ± 25.9	581 ± 21.2	627 ± 24.0	0.295
	Tent	513 ± 21.2	590 ± 21.2	633 ± 24.0	0.223
	SPLASH	471 ± 18.8	515 ± 25.1	570 ± 24.2	0.253
ResNet-20	ReLU	689 ± 28.2	721 ± 28.2	781 ± 25.3	0.551
	Swish	650 ± 17.7	689 ± 17.0	730 ± 29.7	0.601
	APL	579 ± 14.4	631 ± 15.7	692 ± 19.4	0.290
	Tent	551 ± 24.3	633 ± 22.9	669 ± 21.2	0.310
	SPLASH	493 ± 24.3	544 ± 22.9	579 ± 21.2	0.332

Table 5

Robustness to the boundary attack using 1000 images, randomly chosen from the correctly classified images of the CIFAR-10 test set. We attack each architecture five times and report the results in the form of *mean ± standard deviation* of the number of successful attacks.

Model	Activation	# of successful attacks	$avg(Z_{adv} - Z_{true})$
LeNet5	ReLU	801 ± 14.4	0.815
	Swish	779 ± 9.2	0.511
	APL	730 ± 12.0	0.541
	Tent	683 ± 5.4	0.483
	SPLASH	619 ± 15.8	0.401
Net in Net	ReLU	766 ± 9.0	0.502
	Swish	759 ± 5.4	0.391
	APL	654 ± 11.7	0.340
	Tent	632 ± 16.2	0.333
	SPLASH	598 ± 10.1	0.351
All-CNN	ReLU	744 ± 9.0	0.621
	Swish	700 ± 16.2	0.710
	APL	672 ± 6.5	0.480
	Tent	644 ± 11.7	0.399
	SPLASH	611 ± 11.9	0.421
ResNet-20	ReLU	790 ± 6.4	0.548
	Swish	793 ± 11.3	0.566
	APL	711 ± 9.2	0.471
	Tent	677 ± 12.0	0.389
	SPLASH	621 ± 9.4	0.349

In what follows, we employ the same architectures and activation functions that were used in the previous section. The results of this attack are shown in [Table 5](#). We observe that DNNs with SPLASH units are more robust to this adversarial attack than DNNs with APL units, ReLUs, Swish, and Tent units.

5.2. White-box adversarial attacks

For white-box adversarial attacks, the adversary now has information about the parameters of the DNN. To further explore the robustness of DNNs with SPLASH units, in this section, we consider two of the popular benchmarks of white-box adversarial attacks: the fast gradient sign method (FGSM) ([Goodfellow et al., 2014](#)) and Carlini and Wagner (CW) attacks ([Carlini & Wagner, 2017](#)). For both attack methods, we consider four different architectures and compare the rate of successful attacks for each of the networks with ReLUs, Swish units, APL units, Tent units, and

SPLASH units. The dataset and architectures are the same as those used for black-box adversarial attacks.

5.2.1. FGSM

FGSM generates an adversarial image x' from the original image x by maximizing the loss $L(x', y)$, where y is the true label of the image x . This maximization problem is subjected to $\|x - x'\|_{\infty} \leq \epsilon$ where ϵ is considered as the *attack strength*. The loss can be approximated as follows:

$$L(x', y) = L(x, y) + \nabla_x L(x, y)^T \cdot (x - x') \quad (5)$$

So the adversarial image x' would be:

$$x' = x + \epsilon \cdot \text{sign}(\nabla_x L(x, \theta)) \quad (6)$$

The results for different ϵ are summarized in [Table 6](#). The results show that SPLASH units are almost always better than all other activation functions with performance improvements of up to 28.5%.

5.2.2. CW-L2

Another white-box adversarial attack, which is generally more powerful than FGSM, was introduced in [Carlini and Wagner \(2017\)](#). For a given image x and label y , this technique tries to find the minimum perturbation δ , so that the perturbed image x' is classified as $t \neq y$. Using the L_2 norm, this perturbation minimization problem can be formulated as follows:

$$\forall t \neq y, \min \|\delta\|_2^2 \quad \text{subject to} \quad f(x + \delta) = t, \quad x + \delta \in [0, 1]^n \quad (7)$$

To ease the satisfaction of equality, Eq. (7) can be rephrased as $\min \|\delta\|_2^2 + c \cdot g(x + \delta)$ where $g(x) = \max(\max_{t \neq y}(\text{logit}(x)_t - \text{logit}(x)_y))$, c is Lagrange multiplier, and $\text{logit}(x)$ is the pre-softmax vector for the input x .

The robustness performance of ReLUs, Swish units, APL units, Tent units, and SPLASH units for the CW-L2 attack is shown in [Table 7](#). The table is consistent with previous results as it shows that SPLASH units are the most robust to this adversarial attack.

6. ImageNet

In this section, we show the benefits of adding the SPLASH activation function to bigger neural networks, such as those routinely used for the ImageNet ([Deng et al., 2009](#)) benchmark

Table 6

Robustness to the FGSM attack using 1000 images, randomly chosen from the correctly classified images of the CIFAR-10 test set. We attack each architecture five times with random start and report the results in the form of *mean* \pm *standard deviation* of the number of successful attacks. $avg|Z_{true} - Z_{adv}|$ is computed for $\epsilon = 0.04$.

Model	Activation	$\epsilon = 0.02$	$\epsilon = 0.04$	$\epsilon = 0.06$	$avg(Z_{adv} - Z_{true})$
LeNet5	ReLU	690 \pm 13.5	755 \pm 16.6	825 \pm 24.1	0.710
	Swish	634 \pm 11.1	740 \pm 15.7	830 \pm 25.7	0.713
	APL	611 \pm 22.5	691 \pm 13.4	807 \pm 19.0	0.419
	Tent	531 \pm 16.6	620 \pm 15.7	758 \pm 21.7	0.434
	SPLASH	493 \pm 15.1	598 \pm 21.4	772 \pm 26.7	0.521
Net in Net	ReLU	590 \pm 12.9	651 \pm 17.5	798 \pm 17.1	0.609
	Swish	577 \pm 12.1	619 \pm 14.6	750 \pm 15.3	0.439
	APL	531 \pm 20.4	607 \pm 19.6	719 \pm 20.3	0.561
	Tent	524 \pm 18.0	586 \pm 20.5	711 \pm 9.4	0.401
	SPLASH	498 \pm 12.6	554 \pm 17.4	689 \pm 11.4	0.499
All-CNN	ReLU	561 \pm 10.5	653 \pm 18.1	741 \pm 24.4	0.590
	Swish	519 \pm 12.1	622 \pm 17.3	740 \pm 16.6	0.576
	APL	522 \pm 18.4	615 \pm 8.5	721 \pm 21.7	0.549
	Tent	501 \pm 17.3	599 \pm 16.6	694 \pm 12.1	0.303
	SPLASH	479 \pm 11.2	588 \pm 14.1	676 \pm 19.6	0.333
ResNet-20	ReLU	651 \pm 18.1	736 \pm 16.1	801 \pm 20.7	0.641
	Swish	639 \pm 18.4	730 \pm 17.3	793 \pm 19.7	0.522
	APL	609 \pm 9.4	701 \pm 18.0	749 \pm 20.5	0.303
	Tent	583 \pm 16.6	684 \pm 16.6	734 \pm 18.4	0.461
	SPLASH	541 \pm 16.4	617 \pm 21.7	711 \pm 21.0	0.411

Table 7

Robustness to the CW-L2 attack using 1000 images, randomly chosen from the correctly classified images of the CIFAR-10 test set. We attack each architecture five times and report the results in the form of *mean* \pm *standard deviation* of the number of successful attacks.

Model	Activation	# of successful attacks	$avg(Z_{adv} - Z_{true})$
LeNet5	ReLU	932 \pm 5.5	0.801
	Swish	919 \pm 6.4	0.713
	APL	922 \pm 7.5	0.609
	Tent	909 \pm 6.1	0.509
	SPLASH	898 \pm 6.4	0.541
Net in Net	ReLU	916 \pm 8.0	0.790
	Swish	919 \pm 5.4	0.724
	APL	915 \pm 6.1	0.653
	Tent	899 \pm 5.4	0.681
	SPLASH	892 \pm 5.5	0.674
All-CNN	ReLU	894 \pm 13.7	0.611
	Swish	887 \pm 8.6	0.631
	APL	876 \pm 12.1	0.509
	Tent	879 \pm 15.1	0.419
	SPLASH	863 \pm 11.7	0.365
ResNet-20	ReLU	903 \pm 11.8	0.603
	Swish	911 \pm 15.1	0.441
	APL	894 \pm 11.5	0.590
	Tent	881 \pm 11.1	0.499
	SPLASH	870 \pm 12.3	0.541

dataset (Howard et al., 2017; Huang, Liu, Van Der Maaten, & Weinberger, 2017; Sandler, Howard, Zhu, Zhmoginov, & Chen, 2018; Tan & Le, 2019). These neural networks can be used for instance for object detection, identification, and localization in real world applications, such as autonomous vehicles. In the following experiments, we show that adding SPLASH units can also improve both the accuracy and adversarial robustness of MobileNet-V1 (Howard et al., 2017), MobileNet-V2 (Sandler et al., 2018), and ResNet-18 (He et al., 2016) when trained on the Imagenet dataset.

Table 8 shows that SPLASH units improve our implementation of MobileNet-V1 and MobileNet-V2 by 0.40% and 0.69% in absolute error rate, and 6.2% and 7.4% in relative error rate, respectively. For our implementation of ResNet-18, adding SPLASH units also improves the performance by 1.10% in absolute error rate and 11.1% in relative error rate. To evaluate adversarial robustness, we use FGSM and the CW attack. Table 9 shows

Table 8

MobileNet-V1, MobileNet-V2, and ResNet-18 architectures trained on the ImageNet dataset. For each network, the first row shows the test accuracies obtained with our implementation of the corresponding references. The second row shows the best accuracies obtained by us by training networks with ReLU, tanh, and Swish activation functions. The third row shows the test accuracies of the same networks with SPLASH activation functions. Each network is trained three times and the numbers are shown as *mean* \pm *standard deviation*.

Activation	Top-1	Top-5
MobileNet-V1 + ReLU	29.53 \pm 0.32	10.58 \pm 0.31
MobileNet-V1 + Swish*	29.33 \pm 0.48	10.19 \pm 0.33
MobileNet-V1 + SPLASH	29.13 \pm 0.42	9.87 \pm 0.47
MobileNet-V2 + ReLU6	29.03 \pm 0.61	9.30 \pm 0.44
MobileNet-V2 + Swish*	28.87 \pm 0.41	8.98 \pm 0.81
MobileNet-V2 + SPLASH	28.44 \pm 0.52	8.61 \pm 0.33
ResNet-18 + ReLU	29.12 \pm 0.39	9.91 \pm 0.71
ResNet-18 + Swish*	28.87 \pm 0.32	9.25 \pm 0.58
ResNet-18 + SPLASH	28.48 \pm 0.66	8.81 \pm 0.61

that SPLASH units are more robust to adversarial attacks when compared to ReLUs and Swish units.

7. Conclusion

SPLASH units are simple and flexible parameterized piecewise linear functions that simultaneously improve both the accuracy and adversarial robustness of DNNs. They accomplish this without the computationally expensive and time consuming task of adversarial training. They had the best classification accuracy across four different datasets and six different architectures when compared to nine other learned and fixed activation functions. When investigating the reason behind their success, we found that the final shape of the learnable SPLASH units did not serve as a good non-learnable (fixed) activation function. Additionally, in our ablation studies, we saw that restricting the flexibility of the activation function hurts performance, even if the restricted activation function can still mimic the final shape of the unrestricted SPLASH units. It could be possible that changes in the activation functions play a particular role in shaping the loss landscape of deep neural networks (Choromanska, Henaff, Mathieu, Arous, & LeCun, 2015; Dauphin et al., 2014; Hochreiter & Schmidhuber, 1997). Future work will use visualization techniques (Craven & Shavlik, 1992; Gallagher & Downs, 2003; Li, Xu, Taylor, Studer,

Table 9

Robustness to the white-box attack using 1000 images, randomly chosen from the correctly classified images of the ImageNet test set. We attack each architecture three times and report the results in the form of *mean ± standard deviation* of the number of successful attacks.

Model	Activation	FGSM			CW	
		$\epsilon = 0.04$	$\epsilon = 0.08$	$avg(Z_{adv} - Z_{true})$		$avg(Z_{adv} - Z_{true})$
MobileNet-V1	ReLU	526 ± 7.1	683 ± 5.5	0.566	948 ± 4.3	0.489
	Swish	533 ± 3.7	671 ± 5.1	0.590	953 ± 3.7	0.391
	SPLASH	488 ± 7.1	622 ± 8.1	0.501	921 ± 5.7	0.411
MobileNet-V2	ReLU	514 ± 8.3	661 ± 9.9	0.499	918 ± 5.5	0.432
	Swish	520 ± 8.5	658 ± 5.5	0.504	914 ± 7.3	0.414
	SPLASH	481 ± 9.4	611 ± 7.1	0.514	907 ± 5.5	0.389
ResNet-18	ReLU	501 ± 3.7	649 ± 5.7	0.603	948 ± 3.2	0.573
	Swish	497 ± 5.1	630 ± 9.9	0.461	944 ± 4.3	0.511
	SPLASH	451 ± 8.1	592 ± 8.3	0.433	929 ± 1.8	0.541

& Goldstein, 2018) to obtain an intuitive understanding of how learnable activation functions affect the optimization process.

Though no adversarial examples are shown during training, SPLASH units still significantly increase the robustness of DNNs to adversarial attacks. Prior research suggests that the reason for this may be related to their final shape, which looks visually similar to that of a symmetric function (Zhao & Griffin, 2016). Given that research has shown that certain activation functions may make deep neural networks susceptible to adversarial attacks (Croce & Hein, 2018), it is possible that adding more inductive biases aimed at reducing these vulnerabilities may increase the robustness of learned activation functions to adversarial attacks. Since our ablation studies have shown the importance of having flexible activation functions during training, these inductive biases may need to allow for flexibility or be applied during the later stages of training, for example, in the form of a regularization penalty.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

Work in part supported by ARO, United States of America grant 76649-CS, National Science Foundation, United States of America grant 1839429, and National Science Foundation, United States of America grant NRT 1633631 to PB. We wish to acknowledge Yuzo Kanomata for computing support.

Appendix

A.1. Initialization of SPLASH weights

In order to optimize the initialization of the SPLASH weights (a_i), we compare the performance of five different LeNet5 architectures trained on CIFAR-10. Each of these architectures uses a different initialization of SPLASH weights. Fig. 4 shows that the leaky ReLU and ReLU initializations perform the best. Leaky ReLU initialization requires us to determine the slope of the line segment on the left side of the x -axis. Adding another parameter that may need tuning. Therefore, for simplicity, we use the ReLU initialization ($a_+^1 = 1$, and all other slope parameters set to 0) in all of our experiments.

A.2. Number of hinges

In this section, we perform a variety of experiments to find the best setting for SPLASH activation in terms of both complexity and performance.

First, we assess the effect of S on the performance of SPLASH. Due to Theorem 3.1, using a greater value of S would increase the expressive power of the SPLASH units, which generally results in better training performance. We tested different values of $S \in \{3, 5, 7, 9, 11\}$, with symmetrically fixed hinges at $x = 0.0, \pm 1.0\sigma, \pm 2.0\sigma, \pm 2.5\sigma, 3.0\sigma$, and $\pm 3.5\sigma$, starting from 0.0 and progressively spreading out in both directions of the x axis. For instance, when $S = 3$, we set the hinges at $x = 0.0$ and ± 1.0 . Note that $\sigma = 1.0$ because batch normalization is used right before each SPLASH activation layer. We also use MNIST (LeCun & Cortes, 2010) and CIFAR-10 (Krizhevsky, Nair, & Hinton) as training datasets. Each network is trained with two types of SPLASH activations; (1) A shared SPLASH: a SPLASH unit with the same set of weights among all neurons of a layer, and (2) An independent SPLASH: a unit with an independent set of parameters for each neuron of a layer. As it is summarized in Table 10, for the majority of cases with independent SPLASH, there is no improvement in the error rate of the DNNs. Additionally, the error rates of networks do not significantly decrease for $S \geq 7$. Therefore, $S = 7$ is a proper choice for the number of hinges. Lastly, based on the experiment explained in Section 4.3, we know that the training time of a network with SPLASH activation of $S \leq 7$ is comparable to the training time of the same network with learnable activations such as maxout and exponential activations such as tanh. As one can conclude from both Tables 10 and 3, there is a trade-off between the complexity of SPLASH units and the performance of DNNs. We believe that $S = 7$ is the best choice for the number of hinges.

A.3. Experiments' details and statistical significance

In this section, we explain the details of each experiment. Also, to better interpret the results of Table 2, we perform a t-test (Kim, 2015) on all the error rates achieved in that experiment.

The LeNet5 (LeCun et al., 1998) consists of two convolution layers followed by two MLPs that are connected to a softmax layer. We use our implementation of LeNet5 with all the hyperparameters as in Li (2017), except for the number of epochs. We train all the LeNet5 networks for 100 epochs. All-CNN architecture (Springenberg et al., 2014) contains only convolutional layers. Since we could not reproduce the same top-1 accuracy on the CIFAR-10 dataset using the hyperparameters specified in the main article, we used our implementation instead. In this implementation, we use a learning rate of 0.1, with a decay rate of $1e-6$, and a momentum of 0.9. The batch size is set to 64 and we trained these networks for 300 epochs. All other hyperparameters have the same setting as in Springenberg et al.

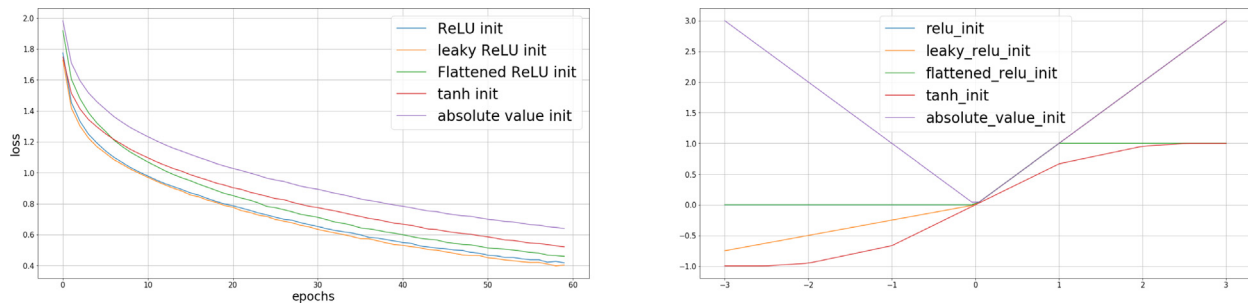


Fig. 4. Left: The loss trajectory of training LeNet5 architecture on CIFAR-10 using different initializations of SPLASH units. Right: Visualizations of the initializations.

Table 10

Two classification tasks are performed using neural networks with five different numbers of hinges S for SPLASH activation. For each value of S , train two neural networks with independent SPLASH units and shared SPLASH units. For each network, the number of additional parameters due to the use of SPLASH and the test error are shown below. LeNet5 architecture is used for both MNIST and CIFAR-10 datasets.

	S	3	5	7	9	11
Error rate	MNIST	1.57–1.61	1.33–1.39	1.13–1.17	1.10–1.08	1.12–1.08
	CIFAR-10	30.79–30.55	30.57–30.29	30.20–30.18	30.14–30.22	30.11–30.19
# of additional params	MNIST	12–1408	18–2112	24–2816	30–3520	36–4224
	CIFAR-10	16–>75k	24->120k	32->150k	40->180k	48->225k

Table 11

For each architecture, the best activation among ReLU, leaky-ReLU, PReLU, tanh, sigmoid, ELU, maxout (nine features), Swish: $f(x) = x * (1 + e^{-\beta x})^{-1}$ corresponding to the minimum average of error rate is selected. Then the significance of the comparison between each network with the best activation function vs. the network with SPLASH activation is calculated through a t-test. The p-values for each comparison are provided below.

Activation	MNIST	CIFAR-10		CIFAR-100	
	–	–	D-A	–	D-A
LeNet5 (PReLU vs. SPLASH)	.057	.043	.055		
Net in Net (ReLU vs. SPLASH)		.042	.039	.038	.055
All-CNN (maxout vs. SPLASH)		.041	.050	.066	.061
ResNet-20 (PReLU vs. SPLASH)		.033	.044	.046	.044

(2014). For the ResNet architectures, we use a popular variant, ResNet-20 (He et al., 2016) which has 0.27M parameters. Our implementation of ResNet-20 is taken from Chollet et al. (2015). All the hyperparameters including batch size, number of epochs, weight initializations, learning rate and its decay, and the choice of optimizer are set to the default values described in Li (2017). Lastly, for the Net in Net architecture, we use the implementation from Li (2017) with the same hyperparameters including batch size, weight initialization, learning rate, and the choice of the optimizer.

In Table 11 we also show the p-values associated with the statistical significance of the experiments described in Table 2. Since each number in Table 2 corresponds to the average of five experiments, we are able to perform an independent one-tailed t-test and provide p-values for each individual experiment. As one can see, all p-values are below 0.1 and most of them are smaller than 0.05 suggesting that the networks trained with SPLASH activation functions are outperforming all the networks trained with all other activation functions.

In Section 4.2, we use ResNet-20 architecture to visualize SPLASH shapes at different stages of the training process in Fig. 1. Here we include two more plots showing the evolution of SPLASH units during training. Figs. 5 and 6 are showing the evolution of SPLASH units during training an MLP architecture using MNIST dataset and LeNet5 architecture using CIFAR-10 dataset, respectively. Both architectures are explained within the caption of the corresponding figures.

In Section 4.3, we compare the training time of different models with different activation functions. For the models using the SPLASH units, we use the same setting as described in Appendix A.2

In Section 5, we start with a tSNE visualization of 100 random samples of frogs and ships images from the CIFAR-10 test set. The tSNE mapping is performed using a learning rate of 30 and a perplexity of 40. Within the same section, for the black-box adversarial attack experiments, each network is attacked five times and the reported numbers in Tables 4 and 5 are the average of the success rate of attacks. One-pixel-attacks are done with the maximum number of iteration set to be 40 and the pop size of 400. For the boundary attack, we use the implementation in Rauber, Brendel, and Bethge (2017). To reduce the rate of successful attacks, the step hyper-parameter is set to 6000. All other hyper-parameters are left as the default from the mentioned implementation. As for the white-box attacks, for both FGSM and CW-L2 attacks, we employ the implementation and default hyper-parameters in Rauber et al. (2017). However, to reduce the success rate for the CW technique, we use a binary search step of 7 for 1000 steps. The network architectures used for experiments in this section are identical to the architectures used in Section 4. Lastly, The activation functions use to trained the networks of Section 5 are as follows: ReLU ($y = x$ for $x > 0$, 0 otherwise), APL ($S = 5$, with fixed hinges on 0, ± 1 , and ± 2), Swish, SPLASH (with the configurations mentioned in Section 4) are used. We also use Tent units, which are designed to improve the adversarial robustness. Tent units can be formulated as $y = \max(0, \delta - |x|)$ where δ is a learnable parameter, initialized at 1.0 with no decay during training.

Section 6 is dedicated to the experiments on ImageNet dataset. In our experiments using the MobileNet-V1, we train the networks with batches of size 32, the initial learning rate of 0.001, and a weight decay of 0.00004. The RMSProp optimizer with a decay of 0.95 and a momentum of 0.9 is used to optimize the loss function. For the MobileNet-V2 architecture, we use the implementation from Sandler et al. (2018). To train these networks, we use the RMSProp optimizer with decay and momentum set to 0.9. The weight decay is set to 0.00004. The learning rate is initialized at 0.045 and decays with a rate of 0.98 per epoch. The batch size is set to 96. All the networks are trained on random crops of size 224 by 224 and the standard color augmentation is performed as explained in Krizhevsky et al. (2009). For testing

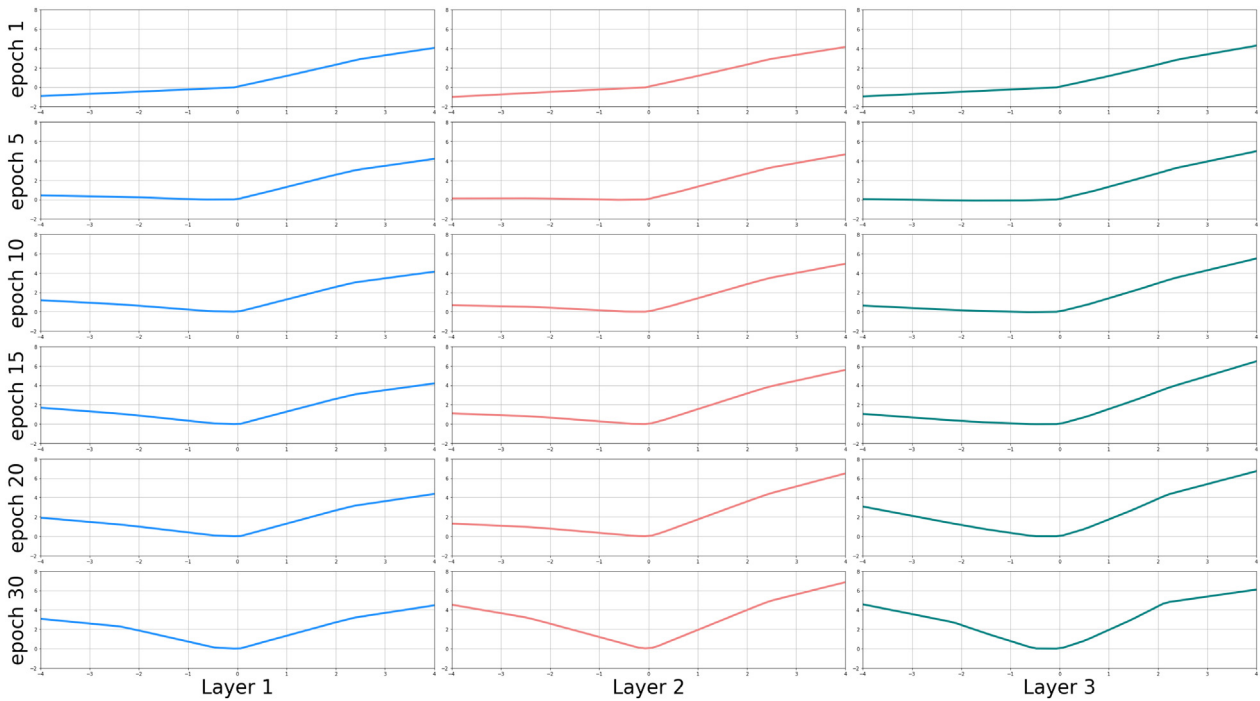


Fig. 5. The shape of SPLASH activation during training a simple network of MLPs on the MNIST dataset. The MLP architecture consists of three layers with 256, 128, and 64 neurons. No dropout was employed. The batch size is 64 and the SGD optimizer is used with momentum 0.9 and learning the fixed learning rate of 0.1.

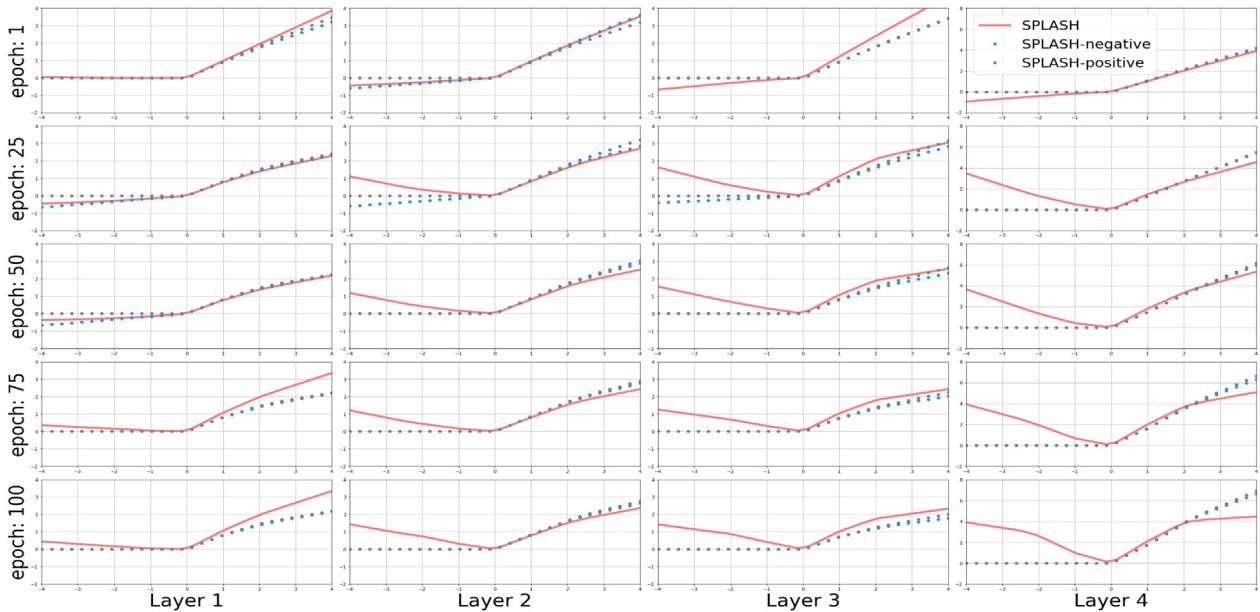


Fig. 6. The shape of SPLASH during training a LeNet5 architecture on the CIFAR-10 dataset. This is the same architecture used in Section 4.

the trained networks, we use one-crop testing for both top-1 and top-5 error rates. Our implementation and training of the MobileNet-V2, with ReLU6 activation functions $f(x) = 0$ if $x \in \mathbb{R} - [0, 6]$ else $f(x) = x$, yields an error rate that is 0.95% higher than the rate reported in Sandler et al. (2018). This could be due to a number of implementation details differences. Nevertheless, within the consistency provided by our own implementation, the MobileNet-V2 architecture with SPLASH activation functions outperforms all other MobileNet-V2 architectures using ReLU, tanh, or Swish activation functions. For the ResNet-18 architecture, we followed the same implementation and hyperparameters as in He et al. (2016). We use the SGD optimizer with a momentum

of 0.9. The learning rate is initialized at 0.1 and divided by 10 when the error plateaus. We apply the weight decay of 0.0001 to the convolutional layers only. We use no drop-out and the batch size is set to be 256. The results in Table 8 show that the SPLASH activation is outperforming ReLU, Swish, and tanh activation functions. For testing, we use the standard ten-crop method to be able to reproduce the error rates in He et al. (2016). We did not use any resizing for the testing phase as there was no clear guideline for that in He et al. (2016). This is most likely the reason for why our implementation results in a slightly higher error rate than what is reported in He et al. (2016).

These trained networks with the aforementioned hyperparameters configurations are used in the adversarial attack experiments. For both the FGSM and CW attacks, we use the same hyperparameter as described in Section 5.

References

- Agostinelli, F., Hoffman, M., Sadowski, P., & Baldi, P. (2015). Learning activation functions to improve deep neural networks. In *International Conference on Learning Representations (Workshop Track)*.
- Baldi, P. (2021). *Deep Learning in Science*. Cambridge, UK: Cambridge University Press, in press.
- Brendel, W., Rauber, J., & Bethge, M. (2017). Decision-based adversarial attacks: Reliable attacks against black-box machine learning models. arXiv preprint arXiv:1712.04248.
- Carlini, N., & Wagner, D. (2017). Towards evaluating the robustness of neural networks. In *2017 IEEE Symposium on Security and Privacy (SP)* (pp. 39–57). IEEE.
- Chollet, F., et al. (2015). Keras. <https://keras.io>.
- Choromanska, A., Henaff, M., Mathieu, M., Arous, G. B., & LeCun, Y. (2015). The loss surfaces of multilayer networks. In *Artificial Intelligence and Statistics* (pp. 192–204).
- Clevert, D.-A., Unterthiner, T., & Hochreiter, S. (2015). Fast and accurate deep network learning by exponential linear units (ELUs).
- Craven, M. W., & Shavlik, J. W. (1992). Visualizing learning and computation in artificial neural networks. *International Journal on Artificial Intelligence Tools*, 1(03), 399–425.
- Croce, F., & Hein, M. (2018). A randomized gradient-free attack on relu networks. In *German Conference on Pattern Recognition* (pp. 215–227). Springer.
- Csáji, B. C., et al. (2001). Approximation with artificial neural networks. *Faculty of Sciences, Eötvös Loránd University, Hungary*, 24(48), 7.
- Dauphin, Y. N., Pascanu, R., Gulcehre, C., Cho, K., Ganguli, S., & Bengio, Y. (2014). Identifying and attacking the saddle point problem in high-dimensional non-convex optimization. In *Advances in Neural Information Processing Systems* (pp. 2933–2941).
- Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., & Fei-Fei, L. (2009). Imagenet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition* (pp. 248–255). Ieee.
- Dhillon, G. S., Azzadenesheli, K., Lipton, Z. C., Bernstein, J., Kossai, J., Khanna, A., et al. (2018). Stochastic activation pruning for robust adversarial defense.
- Gallagher, M., & Downs, T. (2003). Visualization of learning in multilayer perceptron networks using principal component analysis. *IEEE Transactions on Systems, Man and Cybernetics, Part B (Cybernetics)*, 33(1), 28–34.
- Garvin, W., Crandall, H., John, J., & Spellman, R. (1957). Applications of linear programming in the oil industry. *Management Science*, 3(4), 407–430.
- Goodfellow, I. J., Shlens, J., & Szegedy, C. (2014). Explaining and harnessing adversarial examples. arXiv preprint arXiv:1412.6572.
- Goodfellow, I. J., Warde-Farley, D., Mirza, M., Courville, A., & Bengio, Y. (2013). Maxout networks. arXiv preprint arXiv:1302.4389.
- He, K., Zhang, X., Ren, S., & Sun, J. (2015). Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *2015 IEEE International Conference on Computer Vision (ICCV)*. IEEE.
- He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (pp. 770–778).
- Hochreiter, S., & Schmidhuber, J. (1997). Flat minima. *Neural Computation*, 9(1), 1–42.
- Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., et al. (2017). Mobilenets: Efficient convolutional neural networks for mobile vision applications. arXiv preprint arXiv:1704.04861.
- Huang, G., Liu, Z., Van Der Maaten, L., & Weinberger, K. Q. (2017). Densely connected convolutional networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (pp. 4700–4708).
- Ioffe, S., & Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. arXiv preprint arXiv:1502.03167.
- Jin, X., Xu, C., Feng, J., Wei, Y., Xiong, J., & Yan, S. (2016). Deep learning with s-shaped rectified linear activation units. In *Thirtieth AAAI Conference on Artificial Intelligence*.
- Khan, M. M., Ahmad, A. M., Khan, G. M., & Miller, J. F. (2013). Fast learning neural networks using cartesian genetic programming. *Neurocomputing*, 121, 274–289.
- Kim, T. K. (2015). T test as a parametric statistic. *Korean Journal of Anesthesiology*, 68(6), 540.
- Klambauer, G., Unterthiner, T., Mayr, A., & Hochreiter, S. (2017). Self-normalizing neural networks. In *Advances in Neural Information Processing Systems* (pp. 971–980).
- Krizhevsky, A., Hinton, G., et al. (2009). *Learning multiple layers of features from tiny images*. Citeseer.
- Krizhevsky, A., Nair, V., & Hinton, G. CIFAR-10 (Canadian Institute for Advanced Research).
- LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11), 2278–2324.
- LeCun, Y., & Cortes, C. (2010). MNIST handwritten digit database.
- Li, W. (2017). Cifar-10-cnn: Play deep learning with CIFAR datasets. <https://github.com/BIGBALLON/cifar-10-cnn>.
- Li, H., Ouyang, W., & Wang, X. (2016). Multi-bias non-linear activation in deep neural networks. In *International Conference on Machine Learning* (pp. 221–229).
- Li, H., Xu, Z., Taylor, G., Studer, C., & Goldstein, T. (2018). Visualizing the loss landscape of neural nets. In *Advances in Neural Information Processing Systems* (pp. 6389–6399).
- Lin, M., Chen, Q., & Yan, S. (2013). Network in network. arXiv preprint arXiv:1312.4400.
- Maas, A. L., Hannun, A. Y., & Ng, A. Y. (2013). Rectifier nonlinearities improve neural network acoustic models. In *Proc. Icml, Vol. 30* (pp. 3).
- Maaten, L. v. d., & Hinton, G. (2008). Visualizing data using t-SNE. *Journal of Machine Learning Research*, 9(Nov), 2579–2605.
- Nair, V., & Hinton, G. E. (2010). Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10)* (pp. 807–814).
- Nguyen, A., Yosinski, J., & Clune, J. (2015). Deep neural networks are easily fooled: High confidence predictions for unrecognizable images. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (pp. 427–436).
- Nwankpa, C., Ijomah, W., Gachagan, A., & Marshall, S. (2018). Activation functions: Comparison of trends in practice and research for deep learning. arXiv preprint arXiv:1811.03378.
- Pedamonti, D. (2018). Comparison of non-linear activation functions for deep neural networks on MNIST classification task. arXiv preprint arXiv:1804.02763.
- Poli, R. (1996). *Parallel distributed genetic programming*. University of Birmingham, Cognitive Science Research Centre.
- Rakin, A. S., Yi, J., Gong, B., & Fan, D. (2018). Defend deep neural networks against adversarial examples via fixed and dynamic quantized activation functions.
- Ramachandran, P., Zoph, B., & Le, Q. V. (2017). Searching for activation functions. arXiv preprint arXiv:1710.05941.
- Rauber, J., Brendel, W., & Bethge, M. (2017). Foolbox: A python toolbox to benchmark the robustness of machine learning models. arXiv:1707.04131.
- Rozsa, A., & Boulton, T. E. (2019). Improved adversarial robustness by reducing open space risk via tent activations. arXiv preprint arXiv:1908.02435.
- Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., & Chen, L.-C. (2018). Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (pp. 4510–4520).
- Song, S., Chen, Y., Cheung, N.-M., & Kuo, C. C. J. (2018). Defense against adversarial attacks with saak transform.
- Springenberg, J. T., Dosovitskiy, A., Brox, T., & Riedmiller, M. (2014). Striving for simplicity: The all convolutional net. arXiv preprint arXiv:1412.6806.
- Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1), 1929–1958.
- Stone, H. (1961). Approximation of curves by line segments. *Mathematics of Computation*, 40–47.
- Su, J., Vargas, D. V., & Sakurai, K. (2019). One pixel attack for fooling deep neural networks. *IEEE Transactions on Evolutionary Computation*, 1.
- Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I., et al. (2013). Intriguing properties of neural networks. arXiv preprint arXiv:1312.6199.
- Tan, M., & Le, Q. V. (2019). Efficientnet: Rethinking model scaling for convolutional neural networks. arXiv preprint arXiv:1905.11946.
- Wang, B., Lin, A. T., Shi, Z., Zhu, W., Yin, P., Bertozzi, A. L., et al. (2018). Adversarial defense via data dependent activation function and total variation minimization.
- Weingaertner, D., Tatai, V. K., Gudwin, R. R., & Von Zuben, F. J. (2002). Hierarchical evolution of heterogeneous neural networks. In *Proceedings of the 2002 Congress on Evolutionary Computation. CEC'02 (Cat. No. 02TH8600)*, Vol. 2 (pp. 1775–1780). IEEE.
- Zantedeschi, V., Nicolae, M.-I., & Rawat, A. (2017). Efficient defenses against adversarial attacks. In *Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security* (pp. 39–49). ACM.
- Zhang, H., Weng, T.-W., Chen, P.-Y., Hsieh, C.-J., & Daniel, L. (2018). Efficient neural network robustness certification with general activation functions.
- Zhao, Q., & Griffin, L. D. (2016). Suppressing the unusual: towards robust CNNs using symmetric activation functions.