# Boosting Verified Training for Robust Image Classifications via Abstraction

Zhaodi Zhang [1,2,4], Zhiyi Xue [2], Yang Chen [2], Si Liu [3], Yueling Zhang [2], Jing Liu [1], Min Zhang [2]
[1] Shanghai Key Laboratory of Trustworthy Computing
[2] East China Normal University
[3] ETH Zürich [4] Chengdu Education Research Institute

## Abstract

*This paper proposes a novel, abstraction-based, certified training method for robust image classifiers. Via abstraction, all perturbed images are mapped into intervals before feeding into neural networks for training. By training on intervals, all the perturbed images that are mapped to the same interval are classified as the same label, rendering the variance of training sets to be small and the loss landscape of the models to be smooth. Consequently, our approach significantly improves the robustness of trained models. For the abstraction, our training method also enables a sound and complete black-box verification approach, which is orthogonal and scalable to arbitrary types of neural networks regardless of their sizes and architectures. We evaluate our method on a wide range of benchmarks in different scales. The experimental results show that our method outperforms state of the art by (i) reducing the verified errors of trained models up to 95.64%; (ii) totally achieving up to 602.50x speedup; and (iii) scaling up to larger models with up to 138 million trainable parameters. The demo is available at https://github.com/zhangzhaodi233/ABSCERT.git.*

## 1. Introduction

The robustness of image classifications based on neural networks is attracting more attention than ever due to their applications to safety-critical domains such as self-driving [51] and medical diagnosis [43]. There has been a considerable amount of work on training robust neural networks against adversarial perturbations [1, 6–8, 10, 12, 28, 32, 56–59, 62]. Conventional defending approaches augment the training set with adversarial examples [5, 15, 27, 32, 37, 41, 54, 55]. They target only specific adversaries, depending on how the adversarial samples are generated [7], but cannot provide robustness guarantees [2, 11, 26].

Recent approaches attempt to train certifiably robust models with guarantees [2, 11, 14, 24, 26, 29, 49, 63]. They rely on the robustness verification results of neural networks in the training process. Most of the existing verification approaches are based on symbolic interval propagation (SIP) [11, 14, 14, 24, 26], by which intervals are *symbolically input* into neural networks and propagated on a layer basis. There are, however, mainly three obstacles for these approaches to be widely adopted: (i) adding the verification results to the loss function for training brings limited improvement to the robustness of neural networks due to overestimation introduced in the verification phase; (ii) they are time-consuming due to the high complexity of the verification problem *per se*, e.g., NP-complete for the simplest ReLU-based fully connected feedforward neural networks [19]; and (iii) the verification is tied to specific types of neural networks in terms of network architectures and activation functions.

To overcome the above obstacles, we propose a novel, abstraction-based approach for training verified robust image classifications whose inputs are *numerical intervals*. Regarding (i), we abstract each pixel of an image into an interval before inputting it into the neural network. The interval is *numerically* input to the neural network by assigning to two input neurons its lower and upper bounds, respectively. This guarantees that all the perturbations to the pixel in this interval do not alter the classification results, thereby improving the robustness of the network. Moreover, this imposes no overestimation in the training phase. To address the challenge (ii), we use forward propagation and back propagation only to train the network without extra time overhead. Regarding (iii), we treat the neural networks as black boxes since we deal only with the input layer and do not change other layers. Hence, being agnostic to the actual neural network architectures, our approach can scale up to fairly large neural networks. Additionally, we identify a crucial hyper-parameter, namely *abstraction granularity*, which corresponds to the size of intervals used for training the networks. We propose a gradient descent-based algorithm to refine the abstraction granularity for training a more robust neural network.

We implement our method in a tool called AbsCert and assess it, together with existing approaches, on various benchmarks. The experimental results show that our approach reduces the verified errors of the trained neural networks up to 95.64%. Moreover, it totally achieves up to

602.50x speedup. Finally, it can scale up to larger neural networks with up to 138 million trainable parameters and be applied to a wide range of neural networks.

**Contributions.** Overall, we provide (i) a novel, abstraction-based training method for verified robust neural networks; (ii) a companion black-box verification method for certifying the robustness of trained neural networks; (iii) a tool implementing our method; and (iv) an extensive assessment of our method, together with existing approaches, on a wide range of benchmarks, which demonstrates our method's promising achievements.

## 2. Robust Deep Neural Networks

Huber [17] introduces the concept of robustness in a broader sense: (i) the efficiency of the model should be reasonably good, and (ii) the small disturbance applied to the input should only have a slight impact on the result of the model. The problem of robustness verification has been formally defined by Lyu *et al*. [26]:

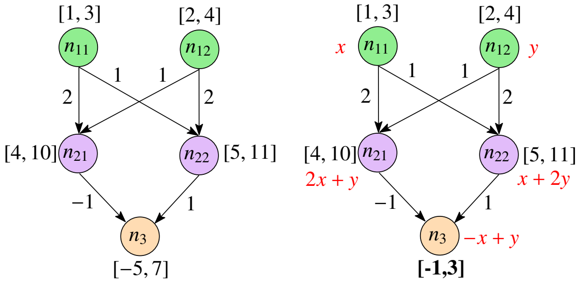
**Definition 1 (Robustness verification)** *Robustness verification aims to guarantee that a neural network outputs consistent classification results for all inputs in a set which is usually represented as an $l_p$ norm ball around the clean image $x$: $\mathbb{B}(x, \epsilon) = \{x' |\ \|x' - x\|_p \leq \epsilon\}$.*

The problem of neural network robustness verification has intrinsically high complexity [65]. Many approaches rely on symbolic interval bound propagation to simplify the problem at the price of sacrificing the completeness [18, 26, 63, 65]. Figure 1 illustratively compares the difference of interval bound propagation (IBP) and SIP. Intuitively, when we know the domain of each input, we can estimate the output range by propagating the inputs symbolically throughout the neural network. According to the output range, we can prove/disprove the robustness of the neural network. See [46] for more details.

To accelerate the propagation, the non-linear activation functions need to over-approximate using linear constraints, therefore rendering the output range overestimated. More hidden layers in a network imply a larger overestimation because the overestimation is accumulated layer by layer. A large overestimation easily causes failure to verification. Therefore, many efforts are being made to tighten the over-approximation [4,53,64]. Unfortunately, it has been proved there is a theoretical barrier [36,44,65].

## 3. The Abstraction-Based Training Approach

In this section, we present our abstraction-based method for training neural networks. We define the training problem in Section 3.1. Section 3.2 and Section 3.3 give the abstraction procedure and the training procedure. Throughout these two sections, we also illustrate how varying network inputs can affect the loss function, which in turn contributes to the robustness of trained models.



(a) IBP (b) SIP

Figure 1. IBP and SIP [46].

### 3.1. Formulating the Training Problem

We solve the following training problem:

> *Given a training set and a testing set of images and a perturbation distance $\epsilon$, our goal is to train an image classifier $f$ that is provably robust on $\mathbb{B}(x, \epsilon)$ for each image $x$ in the set while $f$ has a low verified error on the testing set.*

As the target image classifier $f$ must be guaranteed robust on $\mathbb{B}(x, \epsilon)$, $f$ returns the same classification results for all the perturbed images in $\mathbb{B}(x, \epsilon)$. Our idea is to (i) map $\mathbb{B}(x, \epsilon)$ to an interval vector $(I_1, I_2, \ldots I_n)$ by an abstraction function $\phi$ such that $\phi(\mathbb{B}(x, \epsilon)) = (I_1, I_2, \ldots I_n)$, and (ii) train an image classifier $f'$ which takes as input the interval vector and returns a classification result as $x$ is labeled, i.e., $\arg\max_{y \in Y} f'(I_1, I_2, \ldots, I_n) = y_{true}$ with $y_{true}$ the ground truth label of $x$. The target image classifier $f$ is a composition of $f'$ and $\phi$. Apparently, $f$ is provably robust on $\mathbb{B}(x, \epsilon)$ since all the images in $\mathbb{B}(x, \epsilon)$ are mapped to $(I_1, I_2, \ldots, I_n)$ which is classified to $y_{true}$ by $f'$.

To feed the training intervals to the neural network, the number of neurons in the input layer is doubled. As shown in Figure 2, the upper bound and lower bound of each training interval are input to these neurons. Namely, the neurons in the input layer do not correspond to a pixel but the upper and lower bounds of each training interval. Any perturbation interval mapped to the same training interval will be fed into the neural network with the same upper and lower bounds. Other training parameters and settings are the same as the traditional training process.

### 3.2. Abstraction of Perturbed Images

#### 3.2.1 From Perturbed Interval to Training Interval

We first introduce the abstraction function $\phi : \mathcal{I}^n \rightarrow \mathbb{I}^n$, where $\mathcal{I}^n$ is the set of all interval vectors, $\mathbb{I}^n$ is a finite set of

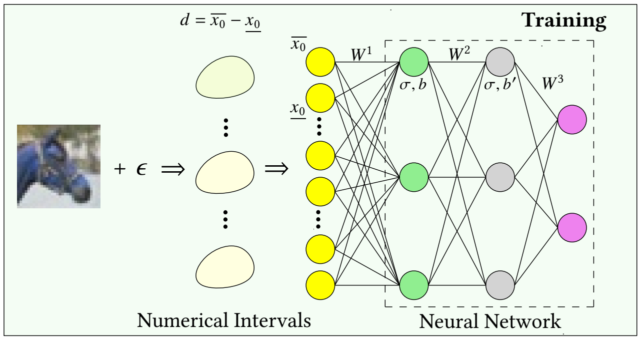


Figure 2. Training on numerical intervals.

interval vectors on which neural networks are trained, and $n$ denotes the size of the interval vectors. The norm ball $\mathbb{B}(x, \epsilon)$ of $x$ under $\epsilon$ is essentially an interval vector in $\mathcal{I}^n$. We call the elements in $\mathcal{I}^n$ *perturbation interval vectors*.

Let $[-1, 1]$ be the range of the complete input space. We divide $[-1, 1]$ evenly into sub-intervals of size $d_i$ and denote the set of all the sub-intervals as $\mathbb{I}_i$. Then, we obtain $\mathbb{I}^n$ as the Cartesian product of all $\mathbb{I}_i$, i.e., $\Pi_{i=1}^n \mathbb{I}_i$. Because we train image classifiers on $\mathbb{I}^n$, we call the elements in $\mathbb{I}^n$ *training interval vectors*. We use $\overline{d}$ to denote the integer vector $(d_1, d_2, \ldots, d_n)$ and call it *abstraction granularity*. In our abstraction process, all values in $\overline{d}$ are the same, so we use a constant $d$ to represent it.

Because perturbation interval vectors are infinite, it is impossible to enumerate all perturbation interval vectors for training. We abstract and map them onto a finite number of training interval vectors in $\mathbb{I}^n$. The purpose of the abstraction is to ease the follow-up robustness verification by restricting the infinite perturbation space to finite training space. In addition, the abstraction function is an element-wise operation. In such cases, we introduce our mapping function with an interval as an example.

In order to ensure that the perturbation interval can only be mapped to a unique training interval, we make the following constraints on the mapping process: We first restrict the abstraction granularity $d$ to be greater than or equal to twice the perturbation size $\epsilon$, for the purpose of guaranteeing that a perturbation interval has an intersection with at most two training intervals. We divide the mapping into three cases: (1) If a training interval contains the perturbation interval, the perturbation interval will be mapped to the training interval; (2) If the perturbation interval has an intersection with two training intervals unequally, the perturbation interval will be mapped to the training interval with the larger coverage area; (3) If the perturbation interval has an intersection with two training intervals equally, the perturbation interval will be mapped to the training interval with a larger value. In this way, we map the perturbation interval on the unique training interval, and then we can train the neural network on these training intervals.

**Algorithm 1:** Abstraction of Perturbations $\phi$.

```
Input  : $\mathcal{I}$: perturbation interval vector;
         $d$: abstraction granularity
Output: $\mathbb{I}$: training interval vector
1  Initialize $I'_{init}$, $\mathbb{I}$;   // $I'_{init}$ is [-1,1], and $\mathbb{I}$ is an empty vector
2  $I' \leftarrow$ Divide($I'_{init}$, $d$);   // Get sub-intervals with size d
3  for i in len($\mathcal{I}$) do
4      for j in len($I'$) do
5          if min($\mathcal{I}_i$) > max($I'_j$) then
6              Continue ;   // No intersection
7          if $\mathcal{I}_i \in I'_j$ then
8              $\mathbb{I}_i = I'_j$;   // $I'_j$ contains $\mathcal{I}_i$
9          else
10             $len_r$ = max($\mathcal{I}_i$) - min($I'_{j+1}$);
11             $len_l$ = max($I'_j$) - min($\mathcal{I}_i$);
12             if $len_r \geq len_l$ then
13                 $\mathbb{I}_i = I'_{j+1}$;   // Map to numerically larger
14             else
15                 $\mathbb{I}_i = I'_j$;   // Map to numerically smaller
16 Return $\mathbb{I}$;
```

Algorithm 1 shows our abstraction function. Firstly, we initialize the complete input interval and an training interval vector (Line 1). Then, we obtain the sub-intervals which each size is $d$ (Line 2). If a training interval contains the perturbation interval, the perturbation interval will be mapped to the training interval (Lines 7-8). The perturbation interval at most intersects with two training intervals at the same time as $d \geq 2 * \epsilon$. Let $len_r$ be the size of the intersection of the perturbation interval and the numerically larger training interval (Line 10). Let $len_l$ be the size of the intersection of the perturbation interval and the numerically smaller training interval (Line 11). If $len_r \geq len_l$, the perturbation interval will be mapped to the numerically larger training interval (Lines 12-13); otherwise, it will be mapped to the numerically smaller training interval (Lines 14-15).

### 3.2.2 Effect of Abstraction on Input

We now explain that the abstraction process results in a smaller variance of training intervals.

In our training method, the value of each pixel is normalized to $[-1, 1]$. Considering the arbitrariness of the input value distribution, we assume that the input values are evenly distributed in $[-1, 1]$. We calculate the variance of input, and in such cases, the value we get is the maximum likelihood estimation of the actual value.

We calculate the variance of the input values with $d$ representing abstraction granularity and $\mathbb{I}$ representing the training intervals of an image. Note that in conventional methods, it is equivalent to $d = 0$, while in our method, $d$

**Algorithm 2:** Abstraction-based Training: AbsTrain

```
Input  : $\mathbf{X}$: training data;
         $\mathbf{Y}$: ground-truth labels of training data;
         $\epsilon$: perturbation radius;
         $d$: abstraction granularity
Output: $f'$: a neural network, $\ell$: training loss
1 Initialize $f'$, $\ell$;
2 for $(X, Y)$ in $(\mathbf{X}, \mathbf{Y})$ do
3     for $(x, y)$ in $(X, Y)$ do
4         $\mathbb{I} \leftarrow \phi(\mathbb{B}(x, \epsilon), d)$ ;      // Get training intervals.
5         $\ell \leftarrow \ell + \mathcal{L}(f'(\mathbb{I}), \mathbf{y})$ ;      // Accumulate the loss.
6     $f' \leftarrow$ Update($f'$, $\ell$) ;      // Update the parameters in $f'$.
7 Return $f'$, $\ell$;
```

is a positive number. In this way, for an input image, the variance of it after the abstraction process is:

$$\begin{aligned} D(\mathbb{I}) &= E((\mathbb{I})^2) - E(\mathbb{I})^2 \\ &= \underbrace{(-1+\frac{d}{2})^2 + (-1+\frac{3d^2}{2}) + \ldots + (1-\frac{d}{2})^2}_{2/d \ \ items} \\ &= \frac{4n^2-1}{6}. \end{aligned} \tag{1}$$

In Equation 1, the variance of training interval is computed by the upper and lower bounds. For an abstraction-based trained neural network, a large abstraction granularity $d$ implies a small $n$, and consequently the variance of the training intervals becomes small. Apparently, intervals have a smaller variance than concrete pixel values.

## 3.3. Training on Intervals

### 3.3.1 The Training Method

When we get the training intervals, hyperparameter settings, such as the number of layers, the number of neurons in hidden layers, the loss function during training, etc., are all the same as the traditional training methods except for the number of neurons in the input layer.

Algorithm 2 shows the pseudo-code of the algorithm for training a neural network. The training dataset $\mathbf{X}$, the ground-truth labels $\mathbf{Y}$ corresponding to the dataset, the perturbation radius $\epsilon$, and the abstraction granularity $d$ are used as inputs. Firstly, a neural network is initialized. That is, the adjustable parameters of the neural network are initialized randomly (Line 1). For each image of training dataset $\mathbf{X}$, the perturbation intervals are mapped to training intervals (Line 4). According to the current adjustable parameters, the cross-entropy loss is calculated (Line 5). Finally, the backward propagation is performed according to the cross-entropy loss, and the values of the adjustable parameters are updated (Line 6). A trained neural network is returned when the loss no longer decreases.

### 3.3.2 Smoothing Loss Landscape

We illustrate that the small variance of input results in a smooth loss landscape during training. Loss landscape is the characterization of loss functions. For example, smooth loss landscape means that the size of the connected region around the minimum where the training loss remains low, showing convex contours in the center, while sharp loss landscape shows not convex but chaotic to that region [25].

We investigate the smoothness of the loss landscape from both theoretical and experimental perspectives. For a classification problem, the loss function is usually cross-entropy loss. We use $y$ to represent the ground-truth label, and $\hat{y}$ to represent the prediction of the neural network. In such cases, we explore the relationship between $\mathbb{I}$ and loss function.

$$\begin{aligned} \mathcal{L}(y, \hat{y}) &= CrossEntropyLoss(y, \hat{y}) \\ &= \sum_{i=1}^{c} y_i \cdot (-log(\hat{y_i})) \\ &= -log(max(A \cdot \mathbb{I} + b)) \end{aligned} \tag{2}$$

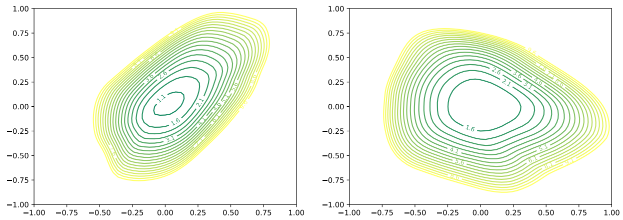
where $c$ represents the number of classes, $y_i$ is the one-hot encoding of labels, $y_{true}$ is the output value corresponding to the correct label, and $A$ and $b$ are the parameters of the neural network. For parameter space, a smooth loss landscape means that when the value of trainable parameters of the neural network gradually deviates from the optimal value, the loss increases slowly with it. In other words, the first-order partial derivative of Equation 2 with respect to $A$ and $b$ should be a constant or a value with less variation. The two partial derivatives are:

$$\begin{aligned} \frac{\partial \mathcal{L}(y, \hat{y})}{\partial A} &= \frac{\partial(-log(max(A \cdot \mathbb{I} + b)))}{\partial A} \\ &= -\frac{\mathbb{I}}{max(A \cdot \mathbb{I} + b)} \end{aligned} \tag{3}$$

$$\begin{aligned} \frac{\partial \mathcal{L}(y, \hat{y})}{\partial b} &= \frac{\partial(-log(max(A \cdot \mathbb{I} + b)))}{\partial b} \\ &= -\frac{1}{max(A \cdot \mathbb{I} + b)} \end{aligned} \tag{4}$$

With fixed $A$ and $b$, we discuss the value of Equation 3 for different training intervals. If $b = 0$, we get a constant $-\frac{1}{A}$. Obviously, the loss landscape is smooth in this case. If $b \neq 0$, we write the derivative as $-\frac{1}{A+b\cdot(\mathbb{I})^{-1}}$. In our method, the variance of $\mathbb{I}$ is small. That is, the value of $\mathbb{I}$ concentrates around a fixed value. Thus, the value of the derivative varies in a small range, and the loss landscape is smoother. For Equation 4, $max(A \cdot \mathbb{I} + b)$ is large as it is the value corresponding to the ground-truth label. Hence, Equation 4 is close to 0, and the loss landscape is smooth.

In summary, the small variance of $\mathbb{I}$ results in a smooth loss landscape. As an example, we utilize the tool in [25]



(a) $d = 0$ (b) $d = 0.025$

Figure 3. Visualization of DM-small's loss landscape [63] trained when $d = 0$ and $d = 0.025$.

to visualize in Figure 3 the loss landscape of two neural networks that are trained with $d = 0$ and $d = 0.025$, respectively. Figure 3 depicts that the neural network trained with $d = 0.025$ has a broader loss landscape than the one trained with $d = 0$, and the loss increases more smoothly in every direction.

### 3.3.3 The Robustness of Trained Neural Networks

Training on numerical intervals guarantees that all the perturbed intervals that are mapped to the same numerical interval will have the same classification result. Intuitively, if a numerical interval represents more concrete values, even if the concrete value is slightly disturbed, these disturbed values will be mapped to the same numerical interval with a high probability. Consequently, the classification of the numerical interval according to the neural network still remains unchanged. In this sense, we say that the perturbation is *dissolved* by the abstraction, and therefore the neural network's robustness is improved.

Theoretically, we show that the smooth loss landscape due to the abstraction in training contributes to the robustness of neural networks. As shown in Figure 3, the loss of DM-small trained by ABSCERT increases slowly and uniformly with the change of parameters, meaning that the growth rate of loss is slow in all directions of parameters' change. However, neural networks trained by conventional methods have a steep slope, which means that there is a direction where the loss increases rapidly. Obviously, smooth loss landscape is helpful for the optimization in training phase to find a global optimum.

Moreover, in the process of our abstraction, an original pixel is first perturbed to a perturbed interval which is then mapped to a training interval for classification. This indicates that the loss is a constant because all the perturbed images of an image are mapped to a fixed set of training intervals, and the $\mathbb{I}^n$ in Equation 2 is never changed.

Neural networks trained by our abstraction-based training method is more robust as the loss landscape is smooth in both parameter space and input space [61]. In particular, for parameter space, the model tends to find a global optimum in a reasonably efficient manner [25, 52]; for input space, the model is insensitive to the input perturbations [30].

**Algorithm 3:** Training with Granularity Tuning.

```
Input  : $\mathbf{X}$: training data; $\mathbf{X_{test}}$: test data;
         $\mathbf{Y}$: ground-truth labels of training data;
         $\mathbf{Y_{test}}$: ground-truth labels of test data;
         $\epsilon$: perturbation radius;
         $d_{step}$: step size of abstraction granularity
Output: $f_{out}$: verified robust neural network;
         $d_{out}$: the best abstraction granularity
Initialize $d, error, e, f_{out}, d_{out}$;        // e : verified error.
while $d \geq 2 * \epsilon$ do
    $(f, \ell) \leftarrow$ ABSTRAIN($\mathbf{X}, \mathbf{Y}, \epsilon, d$); // ℓ: training loss.
    $e \leftarrow$ VERIFY($f, \mathbf{X_{test}}, \mathbf{Y_{test}}, \epsilon, d$); // Verification.
    if $e < error$ then
        $error \leftarrow e$;
        $f_{out} \leftarrow f$;
        $d_{out} \leftarrow d$;
    $\mathbb{I} \leftarrow \Phi(\mathbf{X}, \epsilon, d)$ ;        // Map to training interval.
    $\overline{G}, \underline{G} \leftarrow \ell'(\overline{\mathbb{I}}), \ell'(\underline{\mathbb{I}})$;        // Obtain bounds' gradient,
    if $\overline{G} \leq 0 \wedge \underline{G} \geq 0$ then
        Break;        // Stop when G guides d to increase
    else
        $d \leftarrow d - d_{step}$;        // Decrease d and continue
Return $f_{out}, d_{out}$;
```

## 4. Formal Verification and Granularity Tuning

In this section we introduce a verification-based method for tuning the abstraction granularity to train robust models. Due to the finiteness of $\mathbb{I}^n$, verification procedure can be conducted in a black-box manner, which is both sound and complete. Based on verification results, we can tune the abstraction granularity to obtain finer training intervals.

### 4.1. Black-box Robustness Verification

We propose a black-box verification method VERIFY$(\cdot)$ for neural networks trained by our approach. Given a neural network $f$, a test set $\mathbf{X_{test}}, \mathbf{Y_{test}}$, a perturbation distance $\epsilon$ and an abstraction granularity $d$, VERIFY$(\cdot)$ returns the verified error on the set. The verification procedure is straightforward. First, for each $x \in \mathbf{X_{test}}$, we compute the set $\mathbb{I}$ of training interval vectors of $\mathbb{B}(x, \epsilon)$ using the same abstraction function $\phi$ in Algorithm 1. Then, we feed each interval vector in $\mathbb{I}$ into $f$ and check if the classification result is consistent with the ground-truth label of $x$. The verified error $e$ is the ratio of the inconsistent cases in the test set.

Our verification method is both sound and complete due to the finiteness of $\mathbb{I}$: $f$ is robust on $\mathbb{B}(x, \epsilon)$ if and only if $f$ returns the same label on all the interval vectors in $\mathbb{I}$ as the one of $x$. Another advantage is that it treats $f$ as a black box. Therefore, our verification method is orthogonal and scalable to arbitrary models.

### 4.2. Tuning Abstraction Granularity

When the verified error of a trained neural network is large, we can reduce it by tuning the abstraction granularity and re-train the model on the refined training set of interval vectors. We propose a gradient descent-based algorithm to explore the best $d$ in the abstraction space.

Algorithm 3 shows the tuning and re-training process. First, our algorithm initializes abstraction granularity $d$ and verified error $error$ (Line 1). The tuning process repeats until $d < 2\epsilon$, which means that the size of a training interval is not less than that of the perturbed interval (Lines 2-15). Then a neural network and training loss toward training intervals with $d$ are obtained by calling the Abstraction-based Training function (Algorithm 2) (Line 3). Get the neural network's verified errors on the test dataset (Line 4). If the neural network's verified errors are smaller, we save the neural network and abstraction granularity (Lines 5-8). Next, training intervals are obtained (Line 9). Then the upper and lower bounds' gradient of the training intervals are obtained (Lines 10). If $\overline{G} \leq 0$ and $\underline{G} \geq 0$, the algorithm will terminate (Lines 11-12); otherwise, abstraction granularity will be updated to a smaller granularity (Lines 13-14). When the algorithm terminates, we obtain the neural network with the lowest verified error.

## 5. Experiment

We have implemented both our training and verification approaches in a tool called AbsCert. We evaluate AbsCert, together with existing approaches, on various public benchmarks with respect to both *verified error* and *training and verification time*.

By comparing with the state of the art, our goal is to demonstrate that AbsCert can train neural networks with lower verified errors (**Experiment I**), incurs less computation overhead in both training and verification (**Experiment II**), is applicable to a wide range of neural network architectures (**Experiment III**), and can scale up to larger models (**Experiment IV**).

### 5.1. Benchmarks and Experimental Setup

**Competitors.** We consider three state-of-the-art provably robust training methods: LossLandscapeMatters [24], LBP&Ramp [26], and AdvIBP [11]. All of them rely on linear approximation to train certifiably robust neural networks, which minimizes the upper bound on the worst-case loss against $l_\infty$ perturbations. We use their predefined optimal hyper-parameters to train neural networks on different perturbations $\epsilon$.

**Datasets and Networks.** We conduct our experiments on MNIST [23], CIFAR-10 [21], and ImageNet [9]. For MNIST and CIFAR-10, we train and verify all three convolutional neural networks (CNNs), i.e., DM-small, DM-medium, and DM-large [13, 63], and two fully connected neural networks (FNNs) with three and five layers, respectively. The batch size is set 128. We use cross-entropy loss function and Adam [20] optimizer to update the parameters. The learning rate decreases following the values of the cosine function between 0 and $\pi$ after a warmup period during which it increases linearly between 0 and 1 [48].

For ImageNet, we use the AlexNet [22], VGG11 [38], Inception V1 [42], and ResNet18 [16] architectures, which are winners of the image classification competition ILSVRC[1]. We use the same super-parameters as in their original experiments for training these networks.

**Metrics.** We use two metrics in our comparisons: (i) *verified error*, which is the percentage of images that are not verified to be robust. We quantify the precision improvement by $(e' - e)/e'$, with $e$ and $e'$ the verified errors of neural networks trained by AbsCert and the competitor, respectively; and (ii) *time*, which includes training and/or verification on the same neural network architecture with the same dataset. We compute speedup by $t'/t$, with $t$ and $t'$ the execution time AbsCert and the competitors, respectively. As the time for different perturbations $\epsilon$ is almost the same, we report the average time (in Table 3 and Table 5).

**Experimental Setup.** All experiments on MNIST and CIFAR-10, as well as AlexNet were conducted on a workstation running Ubuntu 18.04 with one NVIDIA GeForce RTX 3090 Ti GPU. All experiments on VGG11, Inception V1 and ResNet18 were conducted on a workstation running Windows11 with one NVIDIA GeForce RTX 3090 Ti GPU.

### 5.2. Experimental Results

**Experiment I: Effectiveness.** Table 1 shows the comparison results of verified errors for DM-small, DM-medium, and DM-large. AbsCert achieves lower verified errors than the competitors. On MNIST, we obtain up to $95.12\%$, $92.65\%$, and $95.64\%$ improvements over LLM, LBP, and AdvIBP on three neural network models (with $\epsilon$ = 0.4), respectively. On CIFAR-10, we achieve a $81.9\%$ improvement to LLM with $\epsilon = 16/255$, and all the improvements are above $49\%$. Note that the publicly available code of LBP does not support CIFAR-10.

Another observation is that the improvement increases as $\epsilon$ becomes larger. This implies that, under larger perturbations, the verified errors of the models trained by AbsCert increase less slowly than those trained by the competing approaches. This reflects that the models trained by AbsCert are more robust than those trained by the three competitors.

As for the accuracy of the trained networks, Table 2 shows that our method achieves *higher accuracy* than the competitors for *all* datasets and models under the same perturbations. Moreover, the decrease speed is much less than

[1] https://image-net.org/challenges/LSVRC/index.php

Table 1. Verified errors (%) of models trained by LossLandscapeMatters (LLM), LBP&Ramp (LBP), AdvIBP (AIBP), and ABSCERT (AC) on MNIST and CIFAR-10 datasets. "–" means that the publicly available code of LBP does not support the CIFAR-10 dataset.

| Dataset | $\epsilon$ | DM-small | | | | | | | DM-medium | | | | | | | DM-large | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | AC | LLM | Impr.(%) | LBP | Impr.(%) | AIBP | Impr.(%) | AC | LLM | Impr.(%) | LBP | Impr.(%) | AIBP | Impr.(%) | AC | LLM | Impr.(%) | LBP | Impr.(%) | AIBP | Impr.(%) |
| MNIST | 0.1 | 0.89 | 3.02 | 70.53 ↑ | 4.09 | 78.24 ↑ | 3.34 | 73.35 ↑ | 0.69 | 2.67 | 74.16 ↑ | 3.47 | 80.12 ↑ | 3.62 | 80.94 ↑ | 0.52 | 2.29 | 77.29 ↑ | 2.93 | 82.25 ↑ | 3.66 | 85.79 ↑ |
| | 0.2 | 0.94 | 6.04 | 84.44 ↑ | 5.54 | 83.03 ↑ | 5.96 | 84.23 ↑ | 0.70 | 5.10 | 86.27 ↑ | 4.73 | 85.20 ↑ | 6.05 | 88.43 ↑ | 0.61 | 4.38 | 86.07 ↑ | 3.96 | 84.60 ↑ | 5.89 | 89.64 ↑ |
| | 0.3 | 1.01 | 12.48 | 91.91 ↑ | 8.11 | 87.55 ↑ | 12.16 | 91.69 ↑ | 0.77 | 11.75 | 93.45 ↑ | 7.02 | 89.03 ↑ | 9.61 | 91.99 ↑ | 0.64 | 10.38 | 93.83 ↑ | 6.14 | 89.58 ↑ | 8.76 | 92.69 ↑ |
| | 0.4 | 1.22 | 20.51 | 94.05 ↑ | 13.03 | 90.64 ↑ | 20.69 | 94.10 ↑ | 0.93 | 19.04 | 95.12 ↑ | 11.59 | 91.98 ↑ | 17.33 | 94.63 ↑ | 0.77 | 15.71 | 95.10 ↑ | 10.48 | 92.65 ↑ | 17.68 | 95.64 ↑ |
| CIFAR-10 | $2/255$ | 25.52 | 50.95 | 49.91 ↑ | – | – | 57.20 | 55.38 ↑ | 16.40 | 49.83 | 67.09 ↑ | – | – | 54.21 | 69.75 ↑ | 13.81 | 48.20 | 71.35 ↑ | – | – | 54.39 | 74.61 ↑ |
| | $4/255$ | 25.52 | 61.90 | 58.77 ↑ | – | – | 65.30 | 60.92 ↑ | 16.40 | 61.46 | 73.32 ↑ | – | – | 62.63 | 73.81 ↑ | 13.81 | 61.22 | 77.44 ↑ | – | – | 61.95 | 77.71 ↑ |
| | $6/255$ | 25.52 | 68.36 | 62.67 ↑ | – | – | 70.20 | 63.65 ↑ | 16.70 | 67.28 | 75.18 ↑ | – | – | 67.69 | 75.33 ↑ | 13.88 | 66.99 | 79.28 ↑ | – | – | 67.56 | 79.46 ↑ |
| | $8/255$ | 25.52 | 71.92 | 64.52 ↑ | – | – | 72.50 | 64.80 ↑ | 16.93 | 70.54 | 76.00 ↑ | – | – | 70.75 | 76.07 ↑ | 13.88 | 70.35 | 80.27 ↑ | – | – | 70.72 | 80.37 ↑ |
| | $16/255$ | 26.61 | 78.13 | 65.94 ↑ | – | – | 78.90 | 66.27 ↑ | 17.16 | 78.27 | 78.08 ↑ | – | – | 78.33 | 78.09 ↑ | 14.12 | 78.03 | 81.90 ↑ | – | – | 78.31 | 81.97 ↑ |

Table 2. Accuracy (%) of models trained by LLM, LBP, AIBP, and AC (our method) on MNIST and CIFAR-10.

| | Dataset | $\epsilon$ | AC | LLM | LBP | AIBP |
|---|---|---|---|---|---|---|
| DM-small | MNIST | 0.1 | 99.11 | 98.43 | 96.63 | 98.36 |
| | | 0.2 | 99.06 | 97.15 | 96.94 | 97.89 |
| | | 0.3 | 98.99 | 94.38 | 96.65 | 96.35 |
| | | 0.4 | 98.78 | 94.46 | 96.65 | 96.14 |
| | CIFAR-10 | $2/255$ | 74.48 | 64.70 | – | 59.20 |
| | | $4/255$ | 74.48 | 55.07 | – | 49.69 |
| | | $6/255$ | 74.48 | 49.29 | – | 41.94 |
| | | $8/255$ | 74.48 | 44.34 | – | 39.52 |
| | | $16/255$ | 73.39 | 32.58 | – | 30.74 |
| DM-medium | MNIST | 0.1 | 99.31 | 98.76 | 97.37 | 98.65 |
| | | 0.2 | 99.30 | 98.13 | 97.36 | 98.42 |
| | | 0.3 | 99.23 | 95.04 | 97.35 | 97.45 |
| | | 0.4 | 99.07 | 93.60 | 97.36 | 97.44 |
| | CIFAR-10 | $2/255$ | 83.60 | 66.00 | – | 62.04 |
| | | $4/255$ | 83.60 | 55.09 | – | 52.37 |
| | | $6/255$ | 83.30 | 48.38 | – | 46.26 |
| | | $8/255$ | 83.07 | 41.19 | – | 40.91 |
| | | $16/255$ | 82.84 | 33.65 | – | 31.32 |
| DM-large | MNIST | 0.1 | 99.48 | 98.95 | 97.79 | 98.96 |
| | | 0.2 | 99.39 | 98.41 | 97.79 | 98.47 |
| | | 0.3 | 99.36 | 95.90 | 97.77 | 98.05 |
| | | 0.4 | 99.23 | 96.14 | 97.79 | 97.78 |
| | CIFAR-10 | $2/255$ | 86.19 | 62.50 | – | 62.33 |
| | | $4/255$ | 86.19 | 56.99 | – | 52.73 |
| | | $6/255$ | 86.12 | 49.58 | – | 45.13 |
| | | $8/255$ | 86.12 | 43.26 | – | 41.54 |
| | | $16/255$ | 85.88 | 33.03 | – | 30.19 |

the one of the networks trained in competitors. Namely, ABSCERT can better resist perturbations, and the models trained by ABSCERT have stronger robustness guarantees.

**Experiment II: Efficiency.** Table 3 shows the average training and verification time. ABSCERT consumes less training time than the competitors for *all* datasets and models; in particular, compared to LLM, our method achieves up to 26.43x speedup on DM-large of MNIST. Additionally, LBP and AdvIBP can hardly be applied to CIFAR-10 (3200 epochs required), while ABSCERT runs smoothly (needs only 30 epochs). This indicates less time (and memory) overhead for ABSCERT.

Regarding the verification overhead, ABSCERT achieves up to 602.5x speedup and is scalable to large models trained on CIFAR-10. That is mainly because our verification approach treats the networks as black boxes thanks to the abstraction-based training method.

**Experiment III: Applicability.** We show that our approach is applicable to both CNNs and FNNs with various activation functions, such as Sigmoid and Tanh. Table 4 shows the verified errors for both types of neural networks trained by our approach. We observe that the verified errors of those neural networks trained on MNIST (resp. CIFAR-10) datasets are all below 14% (resp. 66%), which are smaller than the benchmark counterparts verified in the work [40].

**Experiment IV: Scalability.** We show our method is scalable with respect to training four larger neural network architectures: AlexNet, VGG11, Inception V1, and ResNet18 on ImageNet. Table 5 shows the verified errors and training times. The number of trainable parameters varies from 11 million to 138 million. Compared with the reported errors and training times on those representative large models [16, 22, 38], our approach achieves competitive performance. It is worth mentioning that the reported errors are computed on the testing sets but not verified because those models are too large and cannot be verified by existing verification methods [19,39,40,45,47,64] due to the high computational complexity. In contrast, ABSCERT can verify fairly large networks for its black-box feature.

## 6. Related Work

This work has been inspired by earlier efforts on training and verifying robust models by interval-based abstractions.

**Interval Neural Networks (INNs).** A neural network is called an interval neural network if at least one of its input, output, or weight sets are interval-valued [3]. Interval-valued inputs can capture the uncertainty, inaccuracy, or variability of datasets and thus are used to train prediction models of uncertain systems such as stock markets [35]. Yang and Wu proposed a gradient-based method for smoothing INNs to avoid the oscillation of training [60].

Table 3. Training time (s/epoch) and verification time (s) of LossLandscapeMatters (LLM), LBP&Ramp (LBP), AdvIBP (AIBP), and AbsCert (AC). "–" means that the publicly available code of LBP does not support the CIFAR-10 dataset.

| **Dataset** | **Model** | **Training Time** | | | | | | | **Verification Time** | | | | | | | **Total Speedup** | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | AC | LLM | SpeedUp | LBP | SpeedUp | AIBP | SpeedUp | AC | LLM | SpeedUp | LBP | SpeedUp | AIBP | SpeedUp | LLM | LBP | AIBP |
| MNIST | DM-small | 2.18 | 13.40 | 6.14 ↑ | 6.54 | 2.99 ↑ | 12.46 | 5.72 ↑ | 0.54 | 5.77 | 10.69 ↑ | 66.39 | 122.94 ↑ | 2.35 | 4.35 ↑ | 7.05 ↑ | 26.81 ↑ | 5.44 ↑ |
| | DM-medium | 2.32 | 35.17 | 15.18 ↑ | 11.99 | 5.18 ↑ | 34.72 | 14.97 ↑ | 0.56 | 6.19 | 11.05 ↑ | 129.18 | 230.68 ↑ | 2.63 | 4.70 ↑ | 14.36 ↑ | 49.02 ↑ | 12.97 ↑ |
| | DM-large | 4.28 | 113.14 | 26.43 ↑ | 36.14 | 8.44 ↑ | 99.89 | 23.34 ↑ | 0.56 | 15.11 | 26.98 ↑ | 337.40 | 602.50 ↑ | 3.79 | 6.77 ↑ | 26.50 ↑ | 77.18 ↑ | 21.42 ↑ |
| CIFAR-10 | DM-small | 3.31 | 15.00 | 4.54 ↑ | – | – | 13.76 | 4.16 ↑ | 0.64 | 7.59 | 11.86 ↑ | – | – | 3.53 | 5.52 ↑ | 5.72 ↑ | – | 4.38 ↑ |
| | DM-medium | 3.58 | 31.40 | 8.77 ↑ | – | – | 35.84 | 10.01 ↑ | 0.66 | 7.68 | 11.64 ↑ | – | – | 3.99 | 6.04 ↑ | 9.22 ↑ | – | 9.39 ↑ |
| | DM-large | 5.29 | 123.22 | 23.29 ↑ | – | – | 163.31 | 30.87 ↑ | 0.76 | 18.45 | 24.28 ↑ | – | – | 6.60 | 8.68 ↑ | 23.42 ↑ | – | 28.08 ↑ |

Table 4. Verified errors (%) of the non-ReLU models trained by AbsCert. FC-3 and FC-5 denote FNNs with 3 and 5 hidden layers. DM-s and DM-m refer to DM-small and DM-medium.

| **D.S.** | $\epsilon$ | **Sigmoid** | | | **Tanh** | | | |
|---|---|---|---|---|---|---|---|---|
| | | FC-3 | FC-5 | DM-s | FC-3 | FC-5 | DM-s | DM-m |
| MNIST | 0.1 | 4.23 | 5.73 | 2.28 | 7.95 | 10.50 | 1.56 | 1.04 |
| | 0.2 | 4.23 | 5.87 | 2.66 | 8.06 | 10.50 | 1.65 | 1.14 |
| | 0.3 | 4.23 | 5.87 | 2.84 | 8.29 | 10.50 | 1.75 | 1.14 |
| | 0.4 | 5.09 | 6.23 | 3.12 | 9.39 | 13.16 | 1.97 | 1.24 |
| CIFAR-10 | $2/255$ | 53.49 | 58.95 | 36.22 | 56.62 | 60.07 | 46.69 | 30.02 |
| | $16/255$ | 53.49 | 58.95 | 39.59 | 56.62 | 61.30 | 48.14 | 31.02 |
| | $32/255$ | 54.41 | 60.05 | 41.39 | 57.17 | 61.72 | 49.05 | 32.18 |
| | $64/255$ | 57.65 | 62.77 | 43.09 | 60.99 | 65.73 | 51.22 | 35.62 |

Table 5. Verified errors (%) and training time (s/epoch) of the large models trained by AbsCert on ImageNet.

| $\epsilon$ | **AlexNet** | | **VGG11** | | **Inception V1** | | **ResNet18** | |
|---|---|---|---|---|---|---|---|---|
| | Error | Time | Error | Time | Error | Time | Error | Time |
| $2/255$ | 44.96 | 508.2 | 36.29 | 2530.3 | 41.67 | 2183.7 | 32.15 | 212.6 |
| $4/255$ | 44.96 | | 36.35 | | 41.67 | | 32.25 | |
| $8/255$ | 44.97 | | 36.93 | | 43.12 | | 32.86 | |

Oala et al. recently proposed to train INNs for image reconstruction with theoretically justified uncertainty scores of predictions [31]. All these works demonstrate that training on interval-valued data can improve the prediction accuracy under uncertainties. This is consistent with the robustness improvement for image classifications in this work.

Prabhakar and Afzal proposed to transform regular neural networks to over-approximated INNs for robustness verification [33]. However, the transformation inevitably introduces overestimation to the models and verification results. Our approach avoids any over-approximation to the trained models by training on interval-valued data.

**Verification-in-the-loop Training**. Many approaches on training neural networks with robustness guarantees have been proposed [11, 13, 29, 34, 49, 50, 63]. Most of them are based on linear relaxation [49, 50, 63] or bound propagation [11, 13, 29, 63]. Linear relaxation-based methods use linear relaxation to obtain a convex outer approximation within a norm-bounded perturbation, which results in high time and memory costs [49, 50]. In contrast, bound propagation methods are more effective. Gowal et al. [13] proposed IBP to train provably robust neural networks on a relatively large scale. However, the bound it produces can be too loose to be put into practice. Zhang et al. [63] improved IBP by combining the fast IBP bounds in a forward propagation and a tight linear relaxation-based bound, CROWN, in a backward propagation. Lee et al. [24] also proposed a training method based on linear approximation but considered another important factor - the smoothness of loss function. However, these methods rely heavily on verification process, so the time complexity is relatively high. AdvIBP [11] computes the adversarial loss using FGSM and random initialization and computes the robust loss using IBP. However, the verified errors obtained by AdvIBP are relatively higher. Thanks to the abstraction, networks trained by our method are more robust.

## 7. Conclusion

We have presented a novel, abstraction-based method, naturally coupled with a black-box verification algorithm, for efficiently training provably robust image classifiers. The experimental results showed that our approach outperformed the state-of-the-art robust training approaches with up to 95.64% improvement in reducing verified errors. Moreover, thanks to its black-box feature, our verification algorithm is more amenable and scalable to large neural networks with up to 138 million trainable parameters and is applicable to a wide range of neural networks.

## ACKNOWLEDGMENTS

This work was supported by the National Key Research and Development (2019YFA0706404), the National Nature Science Foundation of China (61972150), Huawei Technologies, the NSFC-ISF Joint Program (62161146001), the Fundamental Research Funds for Central Universities, Shanghai Trusted Industry Internet Software Collaborative Innovation Center, and Shanghai International Joint Lab of Trust-worthy Intelligent Software (22510750100). Corresponding authors are Jing Liu and Min Zhang ({`jliu, zhangmin`}`@sei.ecnu.edu.cn`).

# References

[1] Anish Athalye, Nicholas Carlini, and David A. Wagner. Obfuscated gradients give a false sense of security: Circumventing defenses to adversarial examples. In *ICML*, volume 80, pages 274–283, 2018. 1

[2] Mislav Balunovic and Martin T. Vechev. Adversarial training and provable defenses: Bridging the gap. In *ICLR*, 2020. 1

[3] Mohsen Beheshti, Ali Berrached, André de Korvin, Chenyi Hu, and Ongard Sirisaengtaksin. On interval weighted three-layer neural networks. In *ANSS*, pages 188–194, 1998. 7

[4] Akhilan Boopathy, Tsui-Wei Weng, Pin-Yu Chen, Sijia Liu, et al. CNN-Cert: An efficient framework for certifying robustness of convolutional neural networks. In *AAAI*, pages 3240–3247, 2019. 2

[5] Jacob Buckman, Aurko Roy, Colin Raffel, and Ian J. Goodfellow. Thermometer encoding: One hot way to resist adversarial examples. In *ICLR*, 2018. 1

[6] Nicholas Carlini and David A. Wagner. Adversarial examples are not easily detected: Bypassing ten detection methods. In *AISec@CCS*, pages 3–14, 2017. 1

[7] Nicholas Carlini and David A. Wagner. Towards evaluating the robustness of neural networks. In *SP*, pages 39–57, 2017. 1

[8] Hongge Chen, Huan Zhang, Pin-Yu Chen, Jinfeng Yi, et al. Attacking visual language grounding with adversarial examples: A case study on neural image captioning. In *ACL*, pages 2587–2597, 2018. 1

[9] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, et al. Imagenet: A large-scale hierarchical image database. In *CVPR*, pages 248–255, 2009. 6

[10] Kevin Eykholt, Ivan Evtimov, Earlence Fernandes, Bo Li, et al. Robust physical-world attacks on deep learning visual classification. In *CVPR*, pages 1625–1634, 2018. 1

[11] Jiameng Fan and Wenchao Li. Adversarial training and provable robustness: A tale of two objectives. In *AAAI*, pages 7367–7376, 2021. 1, 6, 8

[12] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. In *ICLR*, 2015. 1

[13] Sven Gowal, Krishnamurthy Dvijotham, Robert Stanforth, Rudy Bunel, et al. On the effectiveness of interval bound propagation for training verifiably robust models. abs/1810.12715, 2018. 6, 8

[14] Sven Gowal, Krishnamurthy Dvijotham, Robert Stanforth, Rudy Bunel, et al. Scalable verified training for provably robust image classification. In *ICCV*, pages 4841–4850, 2019. 1

[15] Chuan Guo, Mayank Rana, Moustapha Cissé, and Laurens van der Maaten. Countering adversarial images using input transformations. In *ICLR*, 2018. 1

[16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, pages 770–778, 2016. 6, 7

[17] Peter J. Huber. *Robust Statistics*. 1981. 2

[18] Peng Jin, Jiaxu Tian, Dapeng Zhi, Xuejun Wen, and Min Zhang. Trainify: A CEGAR-Driven training and verification framework for safe deep reinforcement learning. In *CAV*, volume 13371, pages 193–218, 2022. 2

[19] Guy Katz, Clark W. Barrett, David L. Dill, Kyle Julian, et al. Reluplex: An efficient SMT solver for verifying deep neural networks. In *CAV*, volume 10426, pages 97–117, 2017. 1, 7

[20] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *ICLR*, 2015. 6

[21] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009. 6

[22] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. 60:84–90, 2017. 6, 7

[23] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. 86:2278–2324. 6

[24] Sungyoon Lee, Woojin Lee, Jinseong Park, and Jaewook Lee. Towards better understanding of training certifiably robust models against adversarial examples. In *NeurIPS*, pages 953–964, 2021. 1, 6, 8

[25] Hao Li, Zheng Xu, Gavin Taylor, Christoph Studer, et al. Visualizing the loss landscape of neural nets. In *NeurIPS*, pages 6391–6401, 2018. 4, 5

[26] Zhaoyang Lyu, Minghao Guo, Tong Wu, Guodong Xu, et al. Towards evaluating and training verifiably robust neural networks. In *CVPR*, pages 4308–4317, 2021. 1, 2, 6

[27] Xingjun Ma, Bo Li, Yisen Wang, Sarah M. Erfani, et al. Characterizing adversarial subspaces using local intrinsic dimensionality. In *ICLR*, 2018. 1

[28] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, et al. Towards deep learning models resistant to adversarial attacks. In *ICLR*, 2018. 1

[29] Matthew Mirman, Timon Gehr, and Martin T. Vechev. Differentiable abstract interpretation for provably robust neural networks. In *ICML*, volume 80, pages 3575–3583, 2018. 1, 8

[30] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, Jonathan Uesato, and Pascal Frossard. Robustness via curvature regularization, and vice versa. In *CVPR*, pages 9078–9086, 2019. 5

[31] Luis Oala, Cosmas Heiß, Jan Macdonald, Maximilian März, Wojciech Samek, and Gitta Kutyniok. Interval neural networks: Uncertainty scores. *arXiv preprint arXiv:2003.11566*, 2020. 8

[32] Nicolas Papernot, Patrick D. McDaniel, Xi Wu, Somesh Jha, and Ananthram Swami. Distillation as a defense to adversarial perturbations against deep neural networks. In *SP*, pages 582–597, 2016. 1

[33] Pavithra Prabhakar and Zahra Rahimi Afzal. Abstraction based output range analysis for neural networks. In *NeurIPS*, pages 15788–15798, 2019. 8

[34] Aditi Raghunathan, Jacob Steinhardt, and Percy Liang. Certified defenses against adversarial examples. In *ICLR*, 2018. 8

[35] Antonio Muñoz San Roque, Carlos Maté, Javier Arroyo, and Ángel Sarabia. iMLP: Applying multi-layer perceptrons to interval-valued data. *Neural Processing Letters*, 25(2):157–169, 2007. 7

[36] Hadi Salman, Greg Yang, Huan Zhang, Cho-Jui Hsieh, and Pengchuan Zhang. A convex relaxation barrier to tight robustness verification of neural networks. *NeurIPS'19*, 32, 2019. 2

[37] Pouya Samangouei, Maya Kabkab, and Rama Chellappa. Defense-gan: Protecting classifiers against adversarial attacks using generative models. In *ICLR*, 2018. 1

[38] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *ICLR*, 2015. 6, 7

[39] Gagandeep Singh, Rupanshu Ganvir, Markus Püschel, and Martin T. Vechev. Beyond the single neuron convex barrier for neural network certification. In *NeurIPS*, pages 15072–15083, 2019. 7

[40] Gagandeep Singh, Timon Gehr, Markus Püschel, and Martin T. Vechev. An abstract domain for certifying neural networks. *Proc. ACM Program. Lang.*, 3:41:1–41:30, 2019. 7

[41] Yang Song, Taesup Kim, Sebastian Nowozin, Stefano Ermon, et al. PixelDefend: Leveraging generative models to understand and defend against adversarial examples. In *ICLR*, 2018. 1

[42] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, et al. Going deeper with convolutions. In *CVPR*, pages 1–9, 2015. 6

[43] Joseph J Titano, Marcus Badgeley, Javin Schefflein, Margaret Pain, et al. Automated deep-neural-network surveillance of cranial images for acute neurologic events. *Nature medicine*, 24(9):1337–1341, 2018. 1

[44] Christian Tjandraatmadja, Ross Anderson, Joey Huchette, Will Ma, Krunal Kishor Patel, and Juan Pablo Vielma. The convex relaxation barrier, revisited: Tightened single-neuron relaxations for neural network verification. *NeurIPS'20*, 33:21675–21686, 2020. 2

[45] Shiqi Wang, Kexin Pei, Justin Whitehouse, Junfeng Yang, et al. Efficient formal safety analysis of neural networks. In *NeurIPS*, pages 6369–6379, 2018. 7

[46] Shiqi Wang, Kexin Pei, Justin Whitehouse, Junfeng Yang, et al. Formal security analysis of neural networks using symbolic intervals. In *USENIX Security*, pages 1599–1614, 2018. 2

[47] Shiqi Wang, Huan Zhang, Kaidi Xu, Xue Lin, et al. Beta-CROWN: Efficient bound propagation with per-neuron split constraints for neural network robustness verification. In *NeurIPS*, pages 29909–29921, 2021. 7

[48] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, et al. Transformers: State-of-the-art natural language processing. In *EMNLP*, pages 38–45, 2020. 6

[49] Eric Wong and J. Zico Kolter. Provable defenses against adversarial examples via the convex outer adversarial polytope. In *ICML*, volume 80, pages 5283–5292, 2018. 1, 8

[50] Eric Wong, Frank R. Schmidt, Jan Hendrik Metzen, and J. Zico Kolter. Scaling provable adversarial defenses. In *NeurIPS*, pages 8410–8419, 2018. 8

[51] Bichen Wu, Forrest Iandola, Peter H. Jin, and Kurt Keutzer. Squeezedet: Unified, small, low power fully convolutional neural networks for real-time object detection for autonomous driving. In *CVPR*, 2017. 1

[52] Dongxian Wu, Shu-Tao Xia, and Yisen Wang. Adversarial weight perturbation helps robust generalization. In *NeurIPS*, 2020. 5

[53] Yiting Wu and Min Zhang. Tightening robustness verification of convolutional neural networks with fine-grained linear approximation. In *AAAI*, pages 11674–11681, 2021. 2

[54] Chaowei Xiao, Ruizhi Deng, Bo Li, Taesung Lee, et al. Advit: Adversarial frames identifier based on temporal consistency in videos. In *ICCV*, pages 3967–3976, 2019. 1

[55] Chaowei Xiao, Ruizhi Deng, Bo Li, Fisher Yu, et al. Characterizing adversarial examples based on spatial consistency information for semantic segmentation. In *ECCV*, volume 11214, pages 220–237, 2018. 1

[56] Chaowei Xiao, Bo Li, Jun-Yan Zhu, Warren He, et al. Generating adversarial examples with adversarial networks. In *IJCAI*, pages 3905–3911, 2018. 1

[57] Chaowei Xiao, Dawei Yang, Bo Li, Jia Deng, et al. Meshadv: Adversarial meshes for visual recognition. In *CVPR*, pages 6898–6907, 2019. 1

[58] Chaowei Xiao, Jun-Yan Zhu, Bo Li, Warren He, et al. Spatially transformed adversarial examples. In *ICLR*, 2018. 1

[59] Kaidi Xu, Sijia Liu, Pu Zhao, Pin-Yu Chen, et al. Structured adversarial attack: Towards general implementation and better interpretability. In *ICLR*, 2019. 1

[60] Dakun Yang and Wei Wu. A smoothing interval neural network. *Discrete Dynamics in Nature and Society*, 2012:1–25, 2012. 7

[61] Fuxun Yu, Zhuwei Qin, Chenchen Liu, Liang Zhao, et al. Interpreting and evaluating neural network robustness. In *IJCAI*, 2019. 5

[62] Huan Zhang, Hongge Chen, Zhao Song, Duane S. Boning, et al. The limitations of adversarial training and the blind-spot attack. In *ICLR*, 2019. 1

[63] Huan Zhang, Hongge Chen, Chaowei Xiao, Sven Gowal, et al. Towards stable and efficient training of verifiably robust neural networks. In *ICLR*, 2020. 1, 2, 5, 6, 8

[64] Huan Zhang, Tsui-Wei Weng, Pin-Yu Chen, Cho-Jui Hsieh, et al. Efficient neural network robustness certification with general activation functions. In *NeurIPS*, pages 4944–4953, 2018. 2, 7

[65] Zhaodi Zhang, Yiting Wu, Si Liu, Jing Liu, and Min Zhang. Provably tightest linear approximation for robustness verification of Sigmoid-like neural networks. In *ASE*, 2022. 2