Toward Improving the Robustness of Deep Learning Models via Model Transformation

Yingyi Zhang
State Key Laboratory of Communication Content Cognition, People's Daily Online, Beijing, China 100733; College of Intelligence and Computing, Tianjin University
yingyizhang@tju.edu.cn

Zan Wang
State Key Laboratory of Communication Content Cognition, People's Daily Online, Beijing, China 100733; College of Intelligence and Computing, Tianjin University
wangzan@tju.edu.cn

Jiajun Jiang*
College of Intelligence and Computing, Tianjin University
Tianjin, China
jiangjiajun@tju.edu.cn

Hanmo You
College of Intelligence and Computing, Tianjin University
Tianjin, China
youhanmo@tju.edu.cn

Junjie Chen
College of Intelligence and Computing, Tianjin University
Tianjin, China
junjiechen@tju.edu.cn

ABSTRACT

Deep learning (DL) techniques have attracted much attention in recent years, and have been applied to many application scenarios, including those that are safety-critical. Improving the universal robustness of DL models is vital and many approaches have been proposed in the last decades aiming at such a purpose. Among existing approaches, adversarial training is the most representative. It advocates a post model tuning process via incorporating adversarial samples. Although successful, they still suffer from the challenge of generalizability issues in the face of various attacks with unsatisfactory effectiveness. Targeting this problem, in this paper we propose a novel model training framework, which aims at improving the universal robustness of DL models via model transformation incorporated with a data augmentation strategy in a delta debugging fashion. We have implemented our approach in a tool, called Dare, and conducted an extensive evaluation on 9 DL models. The results show that our approach significantly outperforms existing adversarial training techniques. Specifically, Dare has achieved the highest Empirical Robustness in 29 of 45 testing scenarios under various attacks, while the number drops to 5 of 45 for the best baseline approach.

CCS CONCEPTS

• **Computing methodologies → Neural networks**; • **Software and its engineering → Software testing and debugging**.

*Jiajun Jiang is the corresponding author.

KEYWORDS

Deep Neural Network, Delta Debugging, Model Robustness

ACM Reference Format:
Yingyi Zhang, Zan Wang, Jiajun Jiang, Hanmo You, and Junjie Chen. 2022. Toward Improving the Robustness of Deep Learning Models via Model Transformation. In *37th IEEE/ACM International Conference on Automated Software Engineering (ASE '22), October 10–14, 2022, Rochester, MI, USA.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3551349.3556920

1 INTRODUCTION

In recent years, deep learning (DL) techniques have attracted much attention from researchers, and have been prevalently used in both industrial practice and academic research, such as image processing [83, 85], machine translation [32, 49] and software engineering [5, 42, 65, 76], etc. Particularly, some application scenarios are safety-critical, such as autonomous driving [4, 43, 84, 92] and aircraft collision avoidance [31]. However, as reported by existing studies [9, 58, 63] DL models, in practice, are fragile when facing perturbations and thus easy to be attacked by hackers. For example, researchers from Tencent Keen Security Lab successfully tricked the lane detection system of Tesla Model S with three small adversarial sticker images, making it swerve into the wrong lane without any warnings or precautions [1]. Therefore, it is vital to ensure the safety and enhance the adversarial robustness of DL models in the face of potential adversarial attacks.

Unlike traditional handcrafted programs that are deterministic with a fixed code logic defined by a set of executable machine instructions, deep learning models are built based on a set of input examples. That is, when providing a set of training examples, a model with a set of parameters will be learned according to a predefined neural network structure, which is expected to meet the functionality requirement, such as image classification. However, since the number of input examples is limited and the complete input space is usually enormous or infinite in practice, also known as the incomplete specification issue in traditional software engineering tasks like programming by examples [14–18, 24, 33, 35], the learned model may not work well on unseen inputs, especially those samples that are decorated with crafted attacking features.

In order to improve the robustness of DL models in the face of adversarial inputs, many defensive approaches have been proposed in the last decade. Existing approaches typically can be divided into two categories. The first category aims at enhancing the robustness of the learned model itself with an *offline training* process, such as defensive distillation [53], feature squeezing [75], adversarial training [12, 46, 57], and so on. While the other category aims at designing a detection method that can distinguish adversarial inputs from benign ones [90], then each time a new input is provided, an *online detection* process will be required. In this work, we target the first category that improves the universal robustness of models in an offline fashion.

Among existing approaches, adversarial training is the most representative method in the first category and proved to be effective for strengthening model robustness against adversarial attacks. These kinds of methods take a set of adversarial examples as augmented inputs and proceed with a post-training process with existing models, during which the model architecture keeps unchanged but the parameters are tuned according to the provided examples. The difference of various approaches falls into the generation algorithms of adversarial samples, which takes the responsibility to seed perturbations into legitimate examples. For example, FGSM (Fast Gradient Sign Method) [12] generates adversarial examples directly for given inputs via leveraging the corresponding training gradients, while C&W (Carlini&Wagner) [3] transforms this process to an optimization problem, i.e., adversarial examples are generated via optimizing a set of handcrafted constraints that make the examples close to original inputs but easy to be misclassified. The underlying purpose behind these kinds of approaches is to combat attacks by enlarging the coverage of input features through incorporating adversarial samples in model training.

However, existing adversarial training techniques suffer from the following two major limitations: 1) The training effect is limited as they do not have enough understanding of the model but simply augment the training data, where the crucial input features may be not well captured and strengthened. As a consequence, the robustness improvement of the model is limited. 2) The improvements gained from a particular set of adversarial examples cannot generalize to inputs generated by a different adversarial algorithm, i.e., weak *universal adversarial robustness* [46] to defend against diverse attacking methods. As mentioned above, the different algorithms seed attacking features into inputs from different perspectives, and thus the adversarial examples generated by one algorithm cannot reflect the features from other ones.

With the aim of effectively improving the universal adversarial robustness of DL models combating different attacking methods, in this work we propose **a novel model training framework that constitutes a model transformation method with a customized data augmentation strategy**. Particularly, in order to overcome the first limitation of adversarial training (or rather post-training approaches with data augmentation), we first transform the original classification model into an isomorphic regression model, which extends the underlying network structure of the classification model but incorporates a finer-grained loss function that is more sensitive to small perturbations. In this way, the crucial input features can be better captured and strengthened by suppressing irrelevant input features. On the other hand, to overcome the second limitation, we put forward a novel data augmentation strategy conforming to the transformed regression model, which is inspired by traditional delta debugging techniques [7, 80, 81]. Specifically, this strategy leverages the training historical data for reference to guide the model tuning. It forces the model to strengthen the memory of critical input features and ignore perturbations.

Based on the above process, we designed and implemented our approach into a tool, called Dare. We have conducted an extensive study with Dare on 9 DL models and compared it with 5 representative adversarial training approaches. The results indicate Dare can significantly improve the *universal robustness* of learned models. Specifically, Dare has achieved the highest Empirical Robustness in 29 of 45 testing scenarios under different kinds of attacks, while the best baseline approach perform best merely in 5 testing scenarios.

In summary, we make the following major contributions:

- We propose a novel training framework that improves the universal adversarial robustness of models via model transformation and data augmentation.
- We present a model transformation technique that can transform a classification model into an isomorphic regression model, which preserves the effectiveness of the original model. Especially, the transformed model is more sensitive to small perturbations and suitable for model robustness improvement.
- We propose a novel data augmentation strategy inspired by traditional delta debugging, which conforms to our transformed model without requiring the generation of new samples.
- We have implemented our approach in a tool and conducted an extensive evaluation to demonstrate the effectiveness of the proposed approach.

2 BACKGROUND AND MOTIVATION

2.1 Model Robustness Improvement

As introduced, unlike traditional programs that constitute a sequence of deterministic machine instructions, deep learning models are built over a set of training examples and can be typically viewed as probabilistic decision processes according to the feature distribution among fed training examples. However, since the training data are usually limited and not complete, the learned models may also produce unexpected outputs, especially when facing inputs decorated by unseen features, which makes the learned model easy to be attacked by hackers. We adopt the definition of *Empirical Robustness* from a previous study [68] as the *Model Robustness* in this paper, which denotes the capability of the model to defend against attacking inputs. It has been proved to be practical.

For examples, Figure 1(a) presents a testing image of "dog" from CIFAR10 dataset [36] in our experiment, which can be correctly classified by a well-trained VGG16[61] model $\mathcal{M}$. Particularly, we use the Gradient-weighted Class Activation Map [56] (heatmap for short) to visualize the features that dominate the prediction (i.e., Figure 1(b)). However, when it comes to the inputs decorated by small attacking perturbations (Figure 1(e) and Figure 1(i)), $\mathcal{M}$ easily produces incorrect outputs. To improve the model robustness and

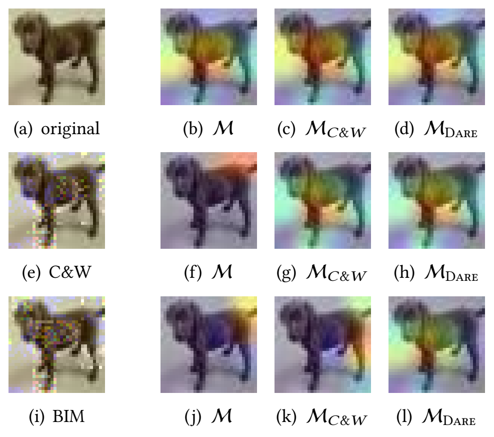


Figure 1: An example image from CIFAR10. (a) is the original image, (e) and (i) are two corresponding adversarial images generated by C&W and BIM, respectively. The heatmaps visualize the prediction results of different models on the three images. $\mathcal{M}$ denotes the original model, while $\mathcal{M}_{C\&W}$ and $\mathcal{M}_{\text{DARE}}$ respectively denote the models fine-tuned by adversarial training method C&W and our approach DARE.

to defend against potential attacks, many approaches have been proposed, among which adversarial training methods stand out due to their high effectiveness. The underlying process is performing post model tuning over a set of samples generated by specially-designed algorithms, such as C&W [3] and BIM [39]. By incorporating new training samples with particular types of unseen attacking features, the model can then identify and ideally become immune to them. As a consequence, the upcoming inputs with similar attacking features will be correctly discriminated and the model will produce the desired results. For instance, $\mathcal{M}_{C\&W}$ can correctly classify the input image shown in Figure 1(e).

However, since different attacking algorithms may vary greatly, the improvement of model robustness is actually limited. Specifically, models fine-tuned by a particular adversarial training method hardly generalize to other attacking methods. For example, $\mathcal{M}_{C\&W}$ still misclassifies the input image shown in Figure 1(i). From the figures, we can see that though the attacking features of C&W and BIM are different, they are in fact small and negligible from the perspective of human beings. In other words, the learned model failed to capture the crucial features leading to the desired prediction of the input image, making it easy to be attacked. Actually, it is not a special case, as it will be shown in our evaluation (Section 4.5), model robustness trained by adversarial training approaches will dramatically drop when facing unknown attacks. It is also a common limitation of existing post-training-based model robustness improvement techniques. In this paper, we propose a new method (called DARE) aiming at improving the *universal robustness* of learned models via model transformation which devotes to better suppressing irrelevant perturbations and strengthening crucial input features. In this way, the model will have a larger possibility to produce the desired prediction when facing different kinds of attacking methods. The last three images in Figure 1 show the heatmaps of $\mathcal{M}_{\text{DARE}}$ leading to the prediction results. We can see that DARE can effectively defend against different attacks and produce the correct results. As it will be shown in our evaluation (see Section 4), DARE can significantly improve the universal robustness of learned models combating various adversarial methods.

2.2 Delta Debugging

Delta debugging was originally proposed by Zeller et al. [7, 80, 81]. It aims at finding the root causes of bugs in traditional programs via isolation of potential error-prone states during program running. Specifically, when given two similar test inputs, where one of them passes while the other fails, a typical delta debugging algorithm inspects and compares the program states (e.g., values of variables) during the running of the tests at some particular checkpoints, different states will be reported as latent root causes for subsequent manual inspection. In this procedure, the passing test performs like a tutor that provides the gold standard to guide the process of finding and finally fixing the bug.

Inspired by this, we borrow the idea of delta debugging for model robustness improvement. Specifically, we will find the desired program states (i.e., neuron outputs in the neural network scenario) for each training input and guide the model tuning process by taking them as the reference. A typical training process of models is iterative and the same training samples are usually used more than once (constructed as different training "epochs"). The target of the training process is to search for a group of parameters that best fit the training data. However, since different training inputs may be contradictory, e.g., some inputs make parameter w_i increase while some decrease w_i, the final trained model may not well fit a certain category of inputs. In other words, the learned model, though well balanced for all inputs, may be insufficiently optimized for a certain category of inputs and thus does not have adequate robustness to combat attacks. This is especially true when the training data is limited. Based on this, it is possibly feasible to search the reference program states for different inputs from training histories as they may appear at some point. As it will be introduced in Section 3.2, we assume that those better prediction results in the training history of the model can be leveraged as the "tutor" and provide a gold standard as the reference for model tuning and improving model robustness.

However, how to use those training histories is still challenging. First, the outputs of a typical classification model are discrete category labels, which are too coarse-grained to reflect the difference of model robustness, i.e., an insufficiently trained model can still produce correct classification outputs. Second, the fine-tuning processes of different categories on the original model largely affect each other like the typical training procedure. To overcome the above challenges, we propose a novel model transformation method. It employs *model slicing* to identify a set of crucial neurons per category for tuning and thus can reduce the training effects to irrelevant categories, and employs a *novel loss function* to better discriminate the difference of outputs and guide model tuning.

3 FRAMEWORK

With the aim of improving the universal robustness of deep learning models and thus protecting against unknown attacks, we put forward a novel model training framework, called DARE. Figure 2 presents the overview of DARE. From a high-level perspective, DARE consists of three stages, i.e., model transformation, data augmentation, and model tuning and synchronization. **Model transformation** takes the responsibility to construct an isomorphic regression model to the original classification model via extending its underlying structure. Particularly, we design a finer-grained loss function to perceive small input perturbations. Empowered by this, the transformed regression model can be more sensitive to the difference in inputs and can be more targeted to strengthen the crucial features leading to the correct prediction. In this way, the model robustness can be improved. However, since the labels of the newly constructed model (*continuous* regression values) are different from the original ones (*discrete* category labels), the original training data cannot be directly used for new model training. In order to conform to the transformed model, the second stage, **data augmentation**, performs a novel data collection and transformation strategy via mining model training history inspired by traditional delta debugging. Finally, the transformed model will be **trained** over the collected training data and then the fine-tuned model weights will be **synchronized** to the original classification model by simple weight replacement to obtain a more robust model.

3.1 Model Transformation

In order to improve the universal robustness of deep learning models, the memory of crucial input features should be tremendously strengthened so as to produce the correct prediction regardless of whatever attacking features are seeded as long as the crucial features exist. This requires the model to capture small input perturbations and suppress their impact on the final prediction. To accomplish this, DARE transforms the original classification model into an isomorphic regression model. The reason is that the finer-grained loss function in the regression model (based on continuous values) is naturally more sensitive to input features compared with the coarse-grained loss function in the classification model (based on category labels), which dominates the optimization direction of the model during training (i.e., gradients are calculated according to the loss). More concretely, the model transformation stage consists of two steps, i.e., *model slicing* and *loss function design.*

3.1.1 Model Slicing. As explained in Section 2.2, the optimization direction based on different input examples may contradict each other, making it hard to improve the model robustness for certain classes. To reduce the training effect over different classes, the model slicing process aims at identifying a set of crucial neurons and synapses (i.e., connections between neurons) per each class, which are responsible for reflecting the pivotal input features and determining the prediction, and thus may largely affect model robustness on that class. Therefore, tuning these crucial neurons and synapses is desirable. Compared with tuning all the neurons and synapses, it confines the impact of the post-training on a smaller scale and reduces the performance risk for irrelevant classes.

Specifically, we employed NNSlicer [89], a state-of-the-art dynamic model slicing method. When given a set of interested neurons $\mathcal{N}$ and a set of input samples, it computes a subset of neurons and synapses that may significantly affect the outputs of those neurons in $\mathcal{N}$. In the following, we briefly introduce the process of NNSlicer to make the paper self-contained. It consists of three phases, i.e, profiling, forward analysis and backward analysis.

In the profiling phase, NNSlicer feeds the whole training set $\mathcal{D}$ into the model and calculates the mean activation value per neuron. Specifically, suppose $\sigma \in \mathcal{D}$ is an input sample, by feeding σ into the model, an output value $y^n(\sigma)$ of neuron n can be observed. Formally, $y^n(\sigma) = mean_{i=1}^{m} y_i^n(\sigma)$, where $y_i^n(\sigma)$ is the i^{th} activation value and m is the total number of activations of n. Particularly, when n is a neuron in a fully connected layer, m equals to 1. On the contrary, m is the number of convolution operations performed by the filter if n is in a convolutional layer. Then, the average activation value of neuron n over the complete training set will be computed by $\overline{y^n(\mathcal{D})} = \Sigma_{\sigma \in \mathcal{D}} y^n(\sigma)/|\mathcal{D}|$, which is viewed as the baseline output of neuron n over the training set.

Then, in the forward analysis phase, when given a set of interested input samples $\mathcal{D}'$, NNSlicer computes the reaction difference of each neuron n over the whole training set $\mathcal{D}$ by Formula 1, which is regarded as the relative activation value of neuron n.

$$\Delta y^n = \overline{y^n(\mathcal{D}')} - \overline{y^n(\mathcal{D})} \tag{1}$$

Therefore, Δy^n reflects the sensitivity of neuron n to the particular inputs $\mathcal{D}'$. Finally, in the backward analysis phase, NNSlicer takes a set of interested neurons $\mathcal{N}$ as the target and recursively calculates the contributions of preceding neurons and synapses backward. Specifically, suppose $CONTRIB_n$ is the cumulative contribution of neuron n, the contribution of neuron h can be computed by Formula 2, where Δy^n and Δy^h are the relative activation values of neurons n and h respectively calculated by Formula 1. Besides, we use s to denote the synapse connecting neurons h and n, and use w_{hn} to denote the connection weight.

$$contrib_h = CONCTRIB_n \times \Delta y^n \times w_{hn} \Delta y^h \tag{2}$$

As a consequence, the cumulative contribution of neuron h can be updated by $CONTRIB_h += sign(contrib_h)$, while the cumulative contribution of synapse s can be calculated by $CONTRIB_s += sign(contrib_h)$. Therefore, when given a model $\mathcal{M}$ trained over dataset $\mathcal{D}$, and the interested input samples $\mathcal{D}'$ as well as the output neurons $\mathcal{N}$ as slicing criterion, the cumulative contribution reflecting the importance of each neuron and each synapse to $\mathcal{N}$ can be calculated.

DARE utilizes NNSlicer to identify key neurons. Specifically, in order to obtain the most accurate model slicing for model updating (will be presented in Section 3.3), DARE first takes the $q\%$ synapses with the largest cumulative contributions as the key synapses, and then the neurons they are connecting will be regarded as the key neurons. This process is determined by the following two observations. First, synapses are more fine-grained than neurons since one neuron usually associates with a large number of synapses, while on the contrary, one synapse merely connects two neurons. Second, the larger the cumulative contribution is, the more important the synapse is. Particularly, in order to identify the actual responsible neurons and synapses of particular classes for robustness improvement and reduce the training effects over different classes, DARE identifies key neurons and synapses for different classes separately. Regarding the slicing criterion, DARE simply picks the one neuron corresponding to the desired class in the last classification layer as

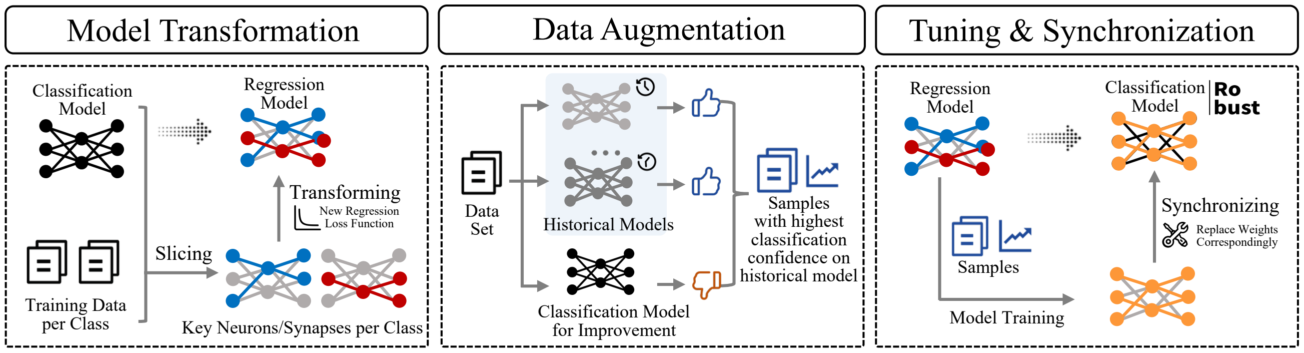


Figure 2: An Overview of Dare Framework.

the interested one, which is targeted and contains comprehensive information to dominate the prediction results.

3.1.2 Loss Function Design. Loss function is the key to measuring the goodness of predictions during model training. A well-designed loss function can greatly improve the performance of learned models. Typically, different loss functions are appropriate for particular types of models. For example, *Cross-Entropy (CE)* is usually used in classification models, while *Mean Square Error (MSE)* is widely employed by regression models. Compared with classification models, regression models are naturally more sensitive to small perturbations, and more suitable for fine-grained model tuning. Therefore, to better perceive small attacking perturbations for suppression and strengthen the memory of crucial input features, Dare extends the underlying structure of the original classification model and transforms it into an isomorphic regression model with a specially-designed novel loss function.

Specifically, when given a classification model $\mathcal{M}$ comprising r (convolutional or dense) layers, denoted as $\mathcal{M} := \langle l_1, l_2, \cdots, l_r \rangle$, Dare transforms it into a new model $\mathcal{M}_T$ by removing the last p layers in $\mathcal{M}$, i.e., $\mathcal{M}_T := \langle l_1, l_2, \cdots, l_{r-p} \rangle$, where $0 \le p < r$. Then, Dare leverages the outputs of model slicing for loss function building, which are a set of crucial neurons that are responsible for model robustness improvement per each class. We use $\mathcal{N}_{l_i}^c$ to denote the crucial neurons from layer l_i for class c, then the regression loss function of transformed model $\mathcal{M}_T$ is defined by Formula 3, where $y_{oracle}^n(\sigma)$ denotes the corresponding gold standard activation value of neuron n under input σ. Particularly, $y_{oracle}^n(\sigma)$ is collected from the training history, and we will introduce the details in the data augmentation procedure of Dare (Section 3.2).

$$\mathcal{L}oss(\sigma) = \frac{\sum_{n \in \mathcal{N}_{l_{r-p}}^c} (y^n(\sigma) - y_{oracle}^n(\sigma))^2}{|\mathcal{N}_{l_{r-p}}^c|} \tag{3}$$

From the formula, we can find that the loss function is determined by two arguments, the crucial neurons extracted from model slicing and the layer chosen for observation. It is intuitive that the neuron is closer to the output layer, its output value embeds more comprehensive information of input features, which will dominate the classification result. Therefore, with the aim of strengthening the memory of input features, Dare makes $\mathcal{M}_T$ preserve as many layers in the original model as possible. As a consequence, Dare takes the penultimate layer in $\mathcal{M}$ as the output layer (i.e., l_{r-p}) of the transformed model $\mathcal{M}_T$, i.e., we set the default value of p as 1. Particularly, if the layer taken has no trainable parameters, e.g., a dropout layer, Dare will take its preceding one until reaching a layer associated with a set of trainable parameters. Actually, it can be flexibly configured on demand.

Therefore, when providing the gold standard outputs, model $\mathcal{M}_T$ can be tuned according to the calculated loss. Particularly, compared with the original classification model, the transformed mode $\mathcal{M}_T$ will largely benefit from the newly designed loss function in two aspects. First, $\mathcal{M}_T$ is more sensitive to small perturbations under the aid of a regression loss, and thus can be more effectively tuned. Second, the new loss function discriminatively calculates loss values for different classes, which not only is more targeted but also potentially reduces the training effects over different classes.

3.2 Data Augmentation

As a result of the model transformation, model $\mathcal{M}$ has been transformed to model $\mathcal{M}_T$ and the training data for $\mathcal{M}$ is no longer usable for $\mathcal{M}_T$. To provide meaningful and suitable training data for $\mathcal{M}_T$, we propose a novel data augmentation strategy via mining the training history. Specifically, inspired by traditional delta debugging, the basis of our data augmentation strategy is, given an input sample, different historical models possibly produce diverse outputs, and the model producing the correct classification with the highest confidence can be viewed as the "tutor" for others, and thus its output can be taken as the gold standard for model tuning.

Formally, suppose the historical models during the training of model $\mathcal{M}$ are recorded as $\mathbb{M} = \{\mathcal{M}_1, \mathcal{M}_2, \cdots, \mathcal{M}_t\}$, where $\mathcal{M}_i$ represents the historical model in the i^{th} training iteration (e.g., epoch), and t is the total number of training iterations, i.e., $\mathcal{M} = \mathcal{M}_t$. Specifically, to ensure the reliability of the collected gold standard and avoid randomness due to the instability of pre-mature models, Dare adopts the latest $t/2$ historical models in $\mathbb{M}$, which can already make relatively stable predictions (i.e., the accuracy and the loss nearly converges). Particularly, we use $\mathcal{M}_i.predict(\sigma)$ to denote the output of model $\mathcal{M}_i$ when fed with input sample σ. The output is a pair $\langle label, conf \rangle$, representing the predicted label $label$ with confidence $conf$. According to these definitions, we present the detailed data augmentation process of Dare in Algorithm 1. When

Algorithm 1 Data Collection and Transformation

```
Require: $\mathbb{M}$: a set of historical models, $\mathcal{M}$: the model for robustness improvement, $\mathcal{D}$: a set of input samples $l$: targeted layer (a.k.a. the output layer of the transformed regression model $\mathcal{M}_T$).
Ensure: $T$: training data for $\mathcal{M}_T$
1:  for each input $\sigma$ in $\mathcal{D}$ do
2:      $\mathcal{M}_{best} \leftarrow None$                ▷ Model with best prediction result
3:      $gt \leftarrow$ ground truth label of $\sigma$
4:      $N_l^{gt} \leftarrow$ key neurons at layer $l$ for class $gt$
5:      $\langle label, conf \rangle \leftarrow \mathcal{M}.predict(\sigma)$
6:      if not $gt.equals(label)$ then
7:          $conf \leftarrow 0$                         ▷ Minimal confidence if incorrect
8:      end if
9:      for each history model $\mathcal{M}_i$ in $\mathbb{M}$ do
10:         $\langle label_i, conf_i \rangle \leftarrow \mathcal{M}_i.predict(\sigma)$
11:         if $gt.equals(label_i)$ && $conf_i > conf$) then
12:             $\mathcal{M}_{best} \leftarrow \mathcal{M}_i$          ▷ Update model and confidence
13:             $conf \leftarrow conf_i$
14:         end if
15:     end for
16:     if $\mathcal{M}_{best}$ is not $None$ then
17:         $Y^l(\sigma) \leftarrow \{y^n(\sigma) | n \in N_l^{gt}\}$     ▷ Neuron outputs of $\mathcal{M}_{best}$
18:         $T \leftarrow T \cup \langle \sigma, Y^l(\sigma) \rangle$              ▷ Add into training set
19:     end if
20: end for
21: return $T$
```

providing an input sample σ, DARE compares the prediction results of different historical models, and records the historical model if it can produce the desired result with the highest confidence (lines 9-15). If such a best model $\mathcal{M}_{best}$ exists (line 16), a profiling process will be performed and the outputs of crucial neurons in the targeted layer l will be extracted for input σ (line 17). Finally, the input sample σ and corresponding neuron outputs will be collected (line 18) and returned (line 21). Specifically, $Y^l(\sigma)$ denotes the outputs of neurons in layer l when fed with σ, they will play as the gold standard outputs (i.e., $y^n_{oracle}(\sigma)$) used in Formula 3.

In this way, when providing a set of input samples $\mathcal{D}$ for the original classification model $\mathcal{M}$, they will be automatically transformed into a set of training data T by DARE for the transformed regression model $\mathcal{M}_T$. Since the gold standard outputs for training samples in T can make some historical model $\mathcal{M}_i$ produce the correct result with higher confidence, they are reasonably regarded as a better representation of input features, and thus tend to provide good guidance during model tuning for its isomorphic regression model $\mathcal{M}_T$.

Besides, the specially-designed loss function enables a more fine-grained optimization target of feature extraction and promotes the consolidation of key feature memories.

3.3 Model Tuning and Synchronization

In the first two stages of DARE, an isomorphic regression model $\mathcal{M}_T$ to $\mathcal{M}$ and its corresponding training set T are obtained. The last stage of DARE is fine-tuning model $\mathcal{M}_T$ over T, and finally synchronizing the optimized model $\mathcal{M}_T$ back to $\mathcal{M}$. As the optimized model $\mathcal{M}_T$ can better capture the input features, the model $\mathcal{M}$ after synchronization is expected to extend this superiority and have better robustness intrinsically to defend against diverse attacks.

In fact, fine-tuning of well-trained models is widely used in many research areas, such as Natural Language Processing (NLP) and Computer Vision (CV). It has been proved effective in further improving model performance with high efficiency [26, 59]. However, compared with the traditional fine-tuning, we would like to highlight two major differences in DARE. First, as introduced in Section 3.1.2, the loss calculation of different classes of inputs can be different due to the discrepancy of the key neuron set N^c_{l-p} in Formula 3. Second, during the fine-tuning process of DARE, not all synapse (or connection) weights in the model $\mathcal{M}_T$ will be updated in back-propagation. Instead, only a subset of synapses that are chosen in the model slicing is considered. These two optimizations will largely benefit the effectiveness of DARE. On the one hand, they restrain the model tuning to a smaller scale, restraining the negative effects on the original model. On the other hand, the tuning process will be more targeted, satisfying our requirement of strengthening the memory of key input features for universal robustness improvement. As it will be shown in our empirical evaluation, these strategies are indeed effective and yield much better performance.

After model tuning, some connection weights in model $\mathcal{M}_T$ will be optimized. To make the original model $\mathcal{M}$ extend the superiority from this process, DARE will synchronize those optimized weights to $\mathcal{M}$ via simple value replacement. Since model $\mathcal{M}_T$ is an isomorphism to $\mathcal{M}$, this process is clear and straightforward.

4 EVALUATION

In our evaluation, we focus on the following research questions.

- **RQ1:** *How effective is DARE to improve model robustness?*
- **RQ2:** *How much does model transformation (comprising model slicing and loss function) in DARE contribute to its effectiveness?*
- **RQ3:** *How effective is the data augmentation strategy?*
- **RQ4:** *How does DARE perform in terms of efficiency?*

4.1 Dataset and Models

To evaluate the performance of DARE, we have conducted an extensive empirical study. Specifically, we employed 3 widely used datasets from prior studies [6, 19, 23, 55, 69], i.e., CIFAR10 [36], SVHN [51] and Fashion-MNIST [73].

Furthermore, to validate the generality of DARE, we employed 3 different neural network architectures in the experiment. They are Alexnet [37], VGG16 [61] and VGG19 [61], all of which are commonly used in previous studies [2, 64]. Particularly, in our experiment, each network will be trained over the above three datasets, and thus we finally obtain 9 different models for the subsequent study. More concretely, for each dataset, we evenly divide the training data into two parts, one of which is used for model training (actual training data), while the other one is used for model selection (validation data), then a grid search will be performed to obtain the best models, which are the subjects for robustness improvement. We have listed the details of those learned models in Table 1, where we present the size of learned models, the total number of parameters, as well as the testing accuracy over the provided testing data associated with the corresponding dataset.

Table 1: Basic information of subjects

Model	VGG16			VGG19			Alexnet		
Dataset	CIFAR10	SVHN	FM	CIFAR10	SVHN	FM	CIFAR10	SVHN	FM
Size (MB)	256.9	256.9	245.8	297.7	297.4	256.7	73.6	73.6	63.0
Parameters	33.6M	33.6M	26.6M	39.0M	39.0M	33.6M	9.6M	9.6M	8.1M
Accuracy(%)	88.7	94.3	91.1	90.6	93.9	90.2	83	93.3	90.0

4.2 Baseline Approaches

As explained in the introduction, Dare targets to improve the robustness of DL models via an *offline training* process, which is orthogonal to the *online detection* techniques. Therefore, we take five widely-used adversarial training methods as the baselines, i.e., BIM (Basic Iterative Methods) [39], C&W (Carlini&Wagner) [3], FGSM (Fast Gradient Sign Method) [12], JSMA (Jacobian-based Saliency Map Attack) [52] and PGD (Project Gradient Descent) [47], which are state-of-the-art for improving adversarial robustness of DL models [63] through *offline* training on adversarial examples.

In addition, as explained in Section 3, Dare incorporates the superiority of two major components. The first one is program transformation that constitutes model slicing and a specially designed loss function, while the second one is the data augmentation strategy. To evaluate the effectiveness of each component, we also compare the results with a set of variants of Dare through an ablation study. The details of the variants are listed as follows.

Dare$_{-s}$ removes the model slicing process from Dare and updates all the connection (synapse) weights between any neurons in the transformed model $\mathcal{M}_T$. In this way, the loss function in Formula 3 will degenerate to the traditional *MSE* loss because $\mathcal{N}^c_{l_{r-p}}$ will include all neurons in the layer l_{l-p}.

Dare$_{-sl}$ further replaces the loss function in $\mathcal{M}_T$ with the original loss in $\mathcal{M}$ on the basis of Dare$_{-s}$. Actually, Dare$_{-sl}$ removes the model transformation component completely from Dare, i.e., $\mathcal{M}_T$ is the same as $\mathcal{M}$, but merely employs the same training data for model tuning.

Dare$_{rand}$ is the variant for evaluating the effectiveness of the data augmentation strategy in Dare. It fully extends the model transformation process in Dare, and then fine-tunes the transformed model over a set of randomly selected input samples. Specifically, the process of lines 9-15 in Algorithm 1 will be replaced by a random selection.

4.3 Procedure and Measurement

Following previous studies [30, 64], we apply the five adversarial training (attacking) algorithms explained in Section 4.2 to generate a set of input samples according to the original model to mimic the unknown attacking inputs, and then the performance of fine-tuned models will be evaluated on them. Specifically, each algorithm will generate 5000 input samples per model listed in Table 1 (9 models in total) as the corresponding testing data, and we use the Empirical Robustness proposed by Wang et al. [68] to measure model robustness, which is defined as the testing accuracy over the attacking inputs. The existing study [72] also proposed the metric of CLEVER score for robustness measurement. However, due to its heavy computation cost (taking thousands of seconds to calculate a CIFAR10 example [90]), it is not suitable for such a large-scale study. Particularly, to exhibit the *universal robustness* of DL models, every fine-tuned model will be finally tested on all the attacking inputs generated by the five algorithms.

4.4 Implementation and Configuration

We have implemented our approach atop the widely-used deep learning framework Keras 2.3.1 and Tensorflow 1.1.1 in Python. We conduced our experiment on a server with Ubuntu 18.04, equipped with 128GB RAM and a processor of Intel(R) Xeon(R) E5-2640 that has 10 cores of 2.40GHz.

Regarding the configuration of Dare, we set the default value of $q\%$ as 95% for model slicing via a small pilot study. That is Dare will ignore 5% synapses (corresponding to the parameters of connections) in the original model during parameter updating per each class. Please note that though it is a small percentage, the numbers of parameters involved are relatively large. According to the model details shown in Table 1 the number of parameters ignored by Dare ranges from 0.4M to 1.95M. We will investigate the effect of $q\%$ on the performance of Dare in our evaluation. Besides, Dare collects data for model tuning from the validation set of the original model (i.e., $\mathcal{D}$ in Algorithm 1), while the competitors will generate adversarial samples by corresponding algorithms. Specifically, each adversarial training method will generate the same number of samples as the original training set, 10,000 of which are used for validation and testing (5,000 for each), while the remaining are left for model tuning. Dare will correspondingly collect the same number of samples (if possible) for model tuning and validation. Finally, we have conducted an extensive model tuning process for each baseline approach by grid search and take their best configurations for the subsequent experiment. Exceptionally, during the model tuning process, the model accuracy over the original testing data may dramatically drop, to make the robustness improvement meaningful, we confine the decline of this accuracy to no more than 10%, which is a reasonable constraint for practical use.

To ease the replication and promote future research in this field, we have published our implementation and all experimental data at **https://doi.org/10.5281/zenodo.7018397**. More detailed configurations in our experiment, e.g., confidence of C&W for adversarial sample generation, can be found in our repository.

4.5 Result Analysis

4.5.1 RQ1: Overall Effectiveness of Dare. As introduced, we evaluated the overall effectiveness of Dare over 9 different models by comparing it with five state-of-the-art adversarial training approaches. The results are presented in Table 2. In the table, the first two columns list the architectures of models and datasets for model training, while the subsequent columns are divided into five blocks, each of which represents the testing results corresponding to a certain adversarial testing method for different model tuning approaches. For example, the first block lists the testing accuracies for Dare and the five comparing adversarial training methods respectively (i.e., BIM, C&W, FGSM, JSMA, PGD) over the testing samples generated by BIM. In particular, to ease the presentation, we use $\mathcal{M}^A_D$ to represent the learned model of architecture A over training set D, such as $\mathcal{M}^{VGG16}_{CIFAR10}$, and call a "*Testing Scenario*" (*TS* for short) as testing a certain model (e.g., $\mathcal{M}^{VGG16}_{CIFAR10}$) over a set of

Table 2: Performance comparison regarding Empirical Robustness (i.e., test accuracy/%) when attacking by different methods.

Model	Dataset	BIM						CW						FGSM						JSMA						PGD					
		Dare	BIM	CW	FGSM	JSMA	PGD	Dare	BIM	CW	FGSM	JSMA	PGD	Dare	BIM	CW	FGSM	JSMA	PGD	Dare	BIM	CW	FGSM	JSMA	PGD	Dare	BIM	CW	FGSM	JSMA	PGD
VGG16	CIFAR10	**82.1**	70.6	66.7	55.8	66.5	73.6	**92.3**	66.4	75.5	55.2	75.0	70.9	**74.6**	50.9	59.7	54.8	56.8	52.5	**84.9**	62.7	73.8	51.1	80.9	70.1	**83.7**	75.7	64.9	54.9	74.6	80.6
	SVHN	**88.3**	69.8	67.1	76.5	67.9	77.6	**95.6**	79.1	80.5	83.9	78.2	84.9	**81.8**	64.1	60.4	73.5	60.8	68.4	**85.0**	67.5	66.3	72.7	67.1	74.2	**86.9**	73.7	69.1	78.9	74.6	80.6
	FM	62.0	66.8	65.9	61.4	60.8	**68.8**	81.8	71.3	**83.1**	68.4	81.2	75.8	66.8	73.3	66.8	**78.5**	67.8	77.6	71.8	77.2	70.2	75.4	**86.6**	84.4	63.7	67.1	64.6	62.1	64.3	**69.9**
VGG19	CIFAR10	**90.0**	73.3	72.6	66.6	71.4	79.7	42.6	54.5	**67.9**	45.9	57.2	61.0	**73.0**	60.3	60.4	62.4	57.5	63.5	71.6	68.8	72.7	**84.0**	73.8	73.5	**89.8**	74.8	70.1	68.5	72.2	81.3
	SVHN	**91.5**	71.0	69.0	75.5	70.5	78.4	**83.9**	55.9	62.5	57.7	56.5	60.2	**86.3**	65.0	63.3	71.7	64.1	67.7	**88.5**	68.6	69.2	70.3	68.7	71.3	**90.8**	75.4	68.1	78.4	74.6	80.3
	FM	66.1	69.5	66.8	66.6	69.6	**72.1**	79.6	67.5	**86.9**	66.9	67.2	76.5	73.8	74.7	71.4	**77.3**	74.8	76.1	77.9	75.8	**84.8**	78.5	75.6	77.1	72.5	70.3	72.9	69.7	**75.1**	74.7
Alexnet	CIFAR10	**79.8**	60.9	58.8	61.8	62.4	64.5	**46.6**	38.1	34.9	40.2	37.7	37.7	**68.7**	51.9	52.8	56.5	55.5	54.2	62.0	62.4	56.2	56.6	64.6	**64.8**	**82.1**	65.8	61.4	63.9	67.5	69.4
	SVHN	**78.4**	72.4	62.0	68.3	73.0	74.2	69.0	57.8	58.3	50.6	**69.6**	62.3	**71.3**	61.8	57.8	66.2	68.4	69.9	80.5	69.5	68.0	71.0	**81.9**	76.9	**79.3**	74.9	68.9	70.2	73.1	78.1
	FM	**83.4**	64.9	78.8	52.9	67.3	71.3	77.1	84.1	54.6	81.4	86.1	**87.0**	**72.1**	63.3	66.3	63.0	62.5	65.8	**86.2**	67.3	80.7	55.9	80.2	81.6	**84.4**	76.4	80.1	54.8	69.7	80.2

Table 3: Model accuracy (%) over the original testing data before and after fine-tuning for each method.

Model	Dataset	Original Acc	Fine-tuning Methods					
			Dare	BIM	CW	FGSM	JSMA	PGD
VGG16	CIFAR10	88.7	**88.9**	87.6	88.8	87.4	88.5	88.3
	SVHN	94.3	93.7	95.1	94.8	95.2	95.0	**95.3**
	FM	91.1	**90.8**	86.3	85.1	86.5	81.9	87.1
VGG19	CIFAR10	90.6	**90.4**	84.5	77.4	84.0	82.9	85.6
	SVHN	93.9	93.8	**94.9**	94.8	94.7	91.2	94.8
	FM	90.2	**91.4**	89.7	81.1	90.5	89.2	89.5
Alexnet	CIFAR10	83.0	**83.6**	81.0	81.1	81.3	82.4	81.2
	SVHN	93.3	**94.5**	92.6	85.8	94.0	93.5	93.3
	FM	90.0	**91.7**	86.7	90.2	88.1	90.0	88.7

testing inputs generated by a particular attacking algorithm (e.g., BIM). As a result, in total there are 45 *TS*s (9 models × 5 attacking algorithms). We also highlighted the best result per each *TS*. From the table we can see that Dare significantly outperforms all the baseline competitors. Specifically, Dare has achieved the highest accuracy in 29/45 *TS*s, and the improvements range from 1.0% to 67.3%. While the second optimal, i.e., PGD, only achieves the highest accuracy in 5/45 *TS*s. The results indicate the effectiveness of Dare.

However, some approaches may also produce better results than Dare in some particular *TS*s. For example, the model $\mathcal{M}_{FM}^{VGG16}$ after fine-tuning by the adversarial approach BIM achieved a higher accuracy than Dare (66.8% vs 62.0%) over the attacking inputs generated by BIM. However, it is not hard to find that most of them are cases where the fine-tuning method shares the same algorithm with the attacking method, such as the aforementioned case (both fine-tuning and testing with BIM. Please also note that PGD is essentially the same as BIM but with a different initialization strategy [57]). The results partially confirm the effectiveness of adversarial training methods for model robustness improvement. While on the other hand, the results also demonstrate their unsatisfactory *universal robustness* since they tend to achieve much worse robustness when coming up with unknown attacking inputs (i.e., generated by a different attacking algorithm). On the contrary, Dare performs stably well in the face of different attacking algorithms.

Regarding the performance of Dare in different *TS*s, Dare is not sensitive to different model architectures as it stably performs well on VGG16, VGG19, and Alexnet. However, Dare performs slightly worse with the training data of FM compared with the other datasets. The reasons are mainly due to the simplicity of input samples from FM, where all samples are grayscale article images, while both the other two datasets include color images of complex objects, e.g., animals or street-view numbers. Consequently, the key features are much easier to learn by the model. Under these circumstances, additional seeded noises may have a higher potential to improve the robustness of the model. Nevertheless, the performance achieved by adversarial training approaches on FM is still limited since usually the best performance is achieved when the attacking algorithm is the same as the one used for training data generation, i.e., existing approaches still suffer from weak universal robustness. In other words, Dare complements existing adversarial training methods.

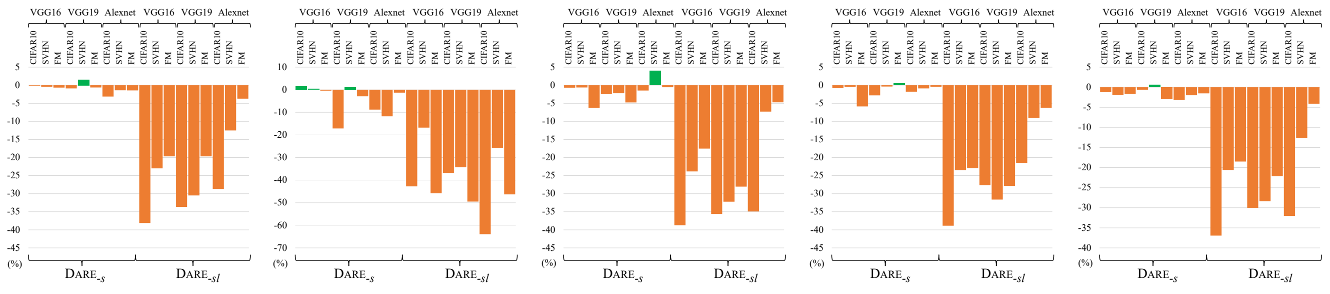
Moreover, improving the robustness of DL models should not largely sacrifice the overall performance of the original models, we further compare the model performance over the original testing data without seeding attacking features. Table 3 shows the results of different approaches. According to the results, Dare still preserves the performance of the original well-trained models. Specifically, Dare slightly improves the accuracy of 5/9 models after fine-tuning, while slightly decreases the accuracy of the others. Compared with the competitors, Dare also achieves the highest accuracy in most circumstances. The reason is Dare aims at improving the universal model robustness by enhancing the memory of crucial features without breaking the normal distribution of input features. On the contrary, traditional adversarial training methods in theory suffer from a higher risk of affecting the feature distribution. This again explains the superiority of Dare against the baseline approaches in another perspective, and demonstrates the effectiveness of Dare for practical use.

4.5.2 *RQ2: Contribution of Model Transformation.*

In this research question, we explore the contribution of model transformation in Dare. Specifically, we conducted an ablation study with two variants of Dare, i.e., Dare$_{-s}$ and Dare$_{-sl}$, which have been introduced in Section 4.2. The results are shown in Figure 3. In the figure, we present the relative improvements of the variants compared with the original Dare in terms of Empirical Robustness. We separately present the results for different attacking methods. According to the figure, without the aid of model transformation (model slicing & loss function), the overall performance of Dare would dramatically drop. More concretely, after removing model slicing, the testing accuracy of Dare$_{-s}$ on average drops 2.0%, and the largest decline is about 17.1% over Dare. By further removing the loss function in Dare$_{-s}$, the decline of testing accuracy over Dare ranges from 3.7% to 63.9%, where the average value is 26.9%. The result indicates that the model transformation component in Dare significantly contributed to its effectiveness.

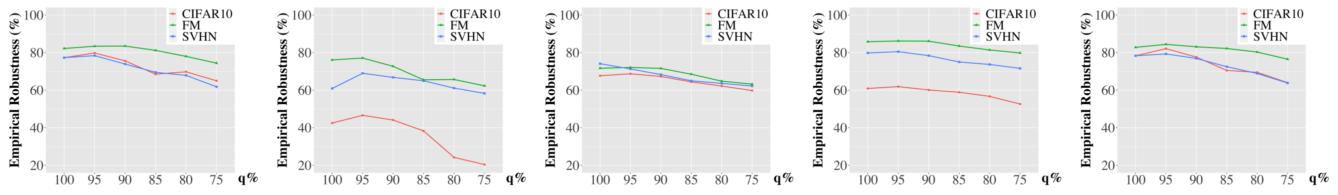
Table 4: Result comparison when employing different data augmentation strategies (Accuracy/%).

Model	Dataset	BIM		C&W		FGSM		JSMA		PGD		Model Acc After Tuning	
		DARE	DARE$_{rand}$	DARE	DARE$_{rand}$	DARE	DARE$_{rand}$	DARE	DARE$_{rand}$	DARE	DARE$_{rand}$	DARE	DARE$_{rand}$
VGG16	CIFAR10	**82.1**	73.7	**92.3**	84.6	**74.6**	69.0	**84.9**	76.0	**83.7**	76.6	**88.9**	85.4
	SVHN	**88.3**	75.6	**95.6**	92.0	**81.8**	70.3	**85.0**	74.8	**86.9**	71.4	**93.7**	75.0
	FM	**62.0**	36.3	**81.8**	28.1	**66.8**	34.1	**71.8**	31.1	**63.7**	37.5	**90.8**	90.1
VGG19	CIFAR10	**90.0**	-	**42.6**	-	**73.0**	-	**71.6**	-	**89.8**	-	**90.4**	-
	SVHN	**91.5**	91.2	**83.9**	83.8	**86.3**	83.4	**88.5**	87.3	**90.8**	89.5	**93.8**	73.6
	FM	**66.1**	28.3	**79.6**	25.2	**73.8**	25.0	**77.9**	27.5	**72.5**	26.7	**91.4**	88.2
Alexnet	CIFAR10	**79.8**	55.2	**46.6**	31.1	**68.7**	46.7	**61.9**	44.3	**82.1**	53.7	**83.6**	82.0
	SVHN	**78.4**	37.1	**69.0**	18.2	**71.3**	25.4	**80.5**	33.4	**79.3**	38.4	**94.5**	89.7
	FM	**83.4**	75.1	**77.1**	26.3	**72.1**	56.5	**86.2**	64.6	**84.4**	76.1	**91.7**	89.8



(a) BIM (b) C&W (c) FGSM (d) JSMA (e) PGD

Figure 3: Relative improvements of variants DARE$_{-s}$ and DARE$_{-sl}$ on Empirical Robustness compared with DARE.



(a) BIM (b) C&W (c) FGSM (d) JSMA (e) PGD

Figure 4: Performance of DARE with different configurations of q for model slicing.

By further investigating the impact of each process in model transformation, we can find that the loss function contributes much more than the model slicing process. The reason is due to the design targets. The model slicing process identifies the crucial neurons that take the most responsibility for reflecting the key input features, and also reduces the training effects over different classes. In this way, the well-trained model performance will not be degraded. However, it provides limited information boosting robustness improvement. On the contrary, the loss function is dedicated to perceiving small perturbations for better compression and consolidating the memory of key input features. In other words, the loss function is expected to be more effective for model robustness improvement.

As explained above, the target of the model slicing is to preserve the superiority of the originally well-trained model, we additionally studied its impact on the performance of DARE with different configurations of $q\%$ ($q \in [75, 100]$ with the interval of 5). Figure 4 visualizes the trend of model robustness over different datasets per each attacking method for Alexnet (due to the experiment cost, we take Alexnet as the representation). From the figure, we can see that the configuration of q may slightly affect the performance of DARE, but the effect is relatively small when $q \in [90, 100]$, where DARE always outperforms the baseline approaches (i.e., achieving the highest Empirical Robustness in the most *TS*s). However, when $q < 90$, the model will suffer a relatively larger performance drop, the major reason is that too many crucial neurons and synapses are eliminated and the model cannot be sufficiently fine-tuned.

4.5.3 RQ3: Contribution of Data Augmentation. This research question investigates the performance of our data augmentation strategy. Specifically, we compare with the variant DARE$_{rand}$ of DARE, which shares the same model transformation component but performs a random sampling of training data for model tuning (see Section 4.2). Table 4 shows the detailed results for DARE and DARE$_{rand}$, including the corresponding Empirical Robustness as well as the model accuracy over the original testing inputs. By randomly sampling training data, the accuracy on the original testing set of $\mathcal{M}^{VGG19}_{CIFAR10}$

Table 5: Time cost for each method (in minutes)

Model	Dataset	BIM		C&W		FGSM		JSMA		PGD		Dare		
		Gen	Trn	Gen	Trn	Gen	Trn	Gen	Trn	Gen	Trn	Aug	Slice	Trn
VGG16	CIFAR10	747	60	3,439	51	103	42	489	54	1,978	52	160	140	40
	SVHN	863	15	4,109	22	63	12	402	19	2,060	18	176	165	60
	FM	689	7	3,165	5	135	8	572	5	1,856	8	188	45	15
VGG19	CIFAR10	898	68	5,928	20	64	28	425	68	2,676	65	160	170	60
	SVHN	1,083	3	5,206	20	65	3	481	3	2,589	3	241	190	80
	FM	1,040	9	5,062	4	171	10	730	9	2,313	10	205	50	35
Alexnet	CIFAR10	87	4	918	<1	13	5	42	4	956	4	43	40	20
	SVHN	339	20	1,997	4	54	20	239	20	896	20	42	45	25
	FM	32	2	1,033	<1	33	1	117	3	451	2	40	20	20

after fine-tuning will dramatically drop, and no model satisfies our constraint regarding the model accuracy (no more than 10% drop). We use "-" to represent the missing data in the table.

According to the table, Dare always outperforms Dare$_{rand}$ regardless of the attacking algorithms. More specifically, the data augmentation strategy in Dare contributes on average 72.9% higher Empirical Robustness by comparing with Dare$_{rand}$, and the highest improvement is as high as 279.1%. Besides, the model accuracy can also be better preserved when employing our data augmentation strategy, indicating the effectiveness of it.

4.5.4 RQ4: Efficiency of Dare. Table 5 presents the time cost for each method. Specifically, for adversarial training methods, we report the time spent on training data generation (**Gen**) and model tuning (**Trn**), while for Dare we report the time for data augmentation (**Aug**), model slicing (**Slice**), and tuning. Please note that compared with the baseline approaches, Dare requires to record the historical models during the training process of the original model. However, the model saving process is so efficient (i.e., in seconds) that can be ignored in the comparison. According to the table we can see that Dare generally performs efficiently. Dare spent much less time on almost every model compared with all the baselines except for FGSM. However, since the whole process is performed offline, the time costs of all approaches can be acceptable. Furthermore, we can also observe that the time cost of slicing is closely related with the parameter number in the models, where smaller models tend to spend less time (refer to Table 1). Particularly, from the table we can find that Dare tends to spend more time on the model tuning process (**Trn**), the reason is clear as it depends on a finer-grained loss function that can perceive small perturbations and thus requires more time to converge. That also explains the effectiveness of Dare from a different perspective – It performs a more comprehensive model tuning. In general, Dare is not only effective but also efficient.

5 RELATED WORK

5.1 Model Repair

Like traditional software [27–29, 41, 70, 71, 74], deep learning (DL) programs also have bugs. Particularly, besides those bugs that are caused by vulnerable source code [21, 22, 25, 45, 50, 62, 86–88]. DL systems have a special type of bugs, called *model bugs* [44], which cause the learned model to produce unsatisfactory results on certain test inputs, e.g., misclassifying a *car* as a *cat*, thus reducing the accuracy of learned models and thus their usability [82]. In order to repair model bugs, many approaches have been proposed [13, 34, 40, 67] and the typical method is optimizing the training data, such as performing data selection [11] or data augmentation [44]. For example, Fahmy et al. [10] proposed to leverage heatmaps to capture the relevance of neurons, and then retained the model for accuracy improvement. Similarly, Ma et al. [44] employed an analogical heatmap to aid the selection of retraining data and proposed MODE. Analogously, Yu et al. [78] proposed a style-guided data augmentation method for repairing DL models in the operational environment, which employed clustering techniques to guide the generation of failure data for model training.

However, as studied by Pham et al. [54] models may have large overall accuracy differences even among identical training. Therefore, it is still a big challenge to ensure the performance of DL models theoretically. Our work aims at improving the model robustness in face of adversarial attacks but not their original performance, which is similar but orthogonal to the aforementioned approaches.

5.2 Adversarial Robustness Improvement

As introduced in the introduction, DL models are fragile in the face of adversarial attacks [52]. In order to improve the adversarial robustness of DL models, many approaches have been put forward in the last decade, which can be divided into two distinct categories: model retraining and adversarial sample detection, where the former is more widely explored. Existing studies [12, 20] have also shown that injecting adversarial examples into the training set (also called adversarial training) could increase the robustness of DL models combating adversarial examples, and many approaches have been proposed [12, 46, 48, 57, 60]. One of the key differences among this kind of approaches lies in the strategies adopted for data augmentation. For example, Zantedeschi et al. [79] proposed to augment training data with examples perturbed using Gaussian noise; Tramèr et al. [66] introduced a technique that augments training data with perturbations transferred from other models; Engstrom et al. [8] proposed a data augmentation method by robust optimization and test-time input aggregation; While Gao et al. [11] proposed to a mutation-based fuzzing technique for such purpose. Besides simply augmenting or filtering training data, Papernot et al. [53] and Xu et al. [75] further respectively proposed to optimize the labels and input features of training data via model distillation and feature squeezing to improve model robustness. However, different from these existing techniques, our approach improves the adversarial robustness of DL models via model transformation, which introduces an isomorphic neural network for parameter tuning rather than training the original model.

Regarding the latter category, i.e., adversarial sample detection, Zhong et al. [91] proposed two techniques (a black-box and a white-box) via leveraging the properties of *local robustness* of neighbor inputs, which help identify inputs with poor robustness, thereby providing real-time feedback to the end-user. On the contrary, Zhao et al. [90] proposed to detect adversarial examples based on a predefined *robustness* difference of input examples. Zhang et al. [89] proposed slicing deep neural networks based on data flow analysis, and identified adversarial examples by comparing the slices calculated by benign examples and adversarial examples. These approaches are orthogonal to ours and can be combined to further improve the robustness of DL models.

Recent researches also have conducted various studies to investigate the performance of adversarial training in different aspects. Kurakin et al. [38] investigated the performance of adversarial training on large models, and provided guidelines on how to successfully scale adversarial training to large models and datasets, and Madry et al. [46] explored the adversarial robustness of neural networks through the lens of robust optimization, assisting the design of reliable and universal methods for model training. Furthermore, Yokoyama et al. [77] conducted an empirical study on the performance of data-augmentation-based model robustness improvement in real industrial scenarios. The results indicate data-augmentation-based approaches can indeed help improve the robustness of DL models, confirming the effectiveness of our approach.

6 DISCUSSION

Limitation: Though DARE was proved to be effective for improving the robustness of DL models, it still has limitations. First, DARE targets classification models, while it does not conform to regression models due to its underlying model transformation process. However, the data augmentation strategy and model training process can still be applied to improving regression model robustness as long as the model adopts CNN structures due to the limit of NNSlicer [89]. Second, as shown in our experimental results, DARE is more suitable for "hard-to-handle" tasks, where the model inputs originally involve perturbations, e.g., numbers in the natural scene (SVHN). On the contrary, the improvement will be slightly restrained if the crucial input features are apparently distinctive (e.g., FM) for DL models. Nevertheless, DARE still outperforms the baselines. Finally, since the data augmentation strategy in DARE requires the historical models, it will incur more storage compared with adversarial training methods.

Threats to Validity: The threats to validity mainly lie in the model/data selection and experiment construction. To mitigate the selection bias in our evaluation, we have employed three different network architectures and three different datasets, all of which are commonly used by existing studies and cover both complex (e.g., CIFAR10 and VGG19) and simple ones (e.g., FM and Alexnet). Specifically, the datasets adopted in our evaluation are different from multiple aspects, such as different image sizes, different color modes (colored *and* grayscale) and different scenarios (numbers *and* objects), etc. Consequently, we believe the results are representative. Regarding the experiment, in order to obtain the best performance on baseline approaches, we have conducted an extensive model tuning process. Finally, our experimental data is publicly accessible for replication and promoting future research.

Future Work: In this work, we have evaluated our approach over nine image classification models. In the future, we plan to study its effectiveness in more application scenarios, e.g., Natural Language Processing, since DARE does not depend on any image specific features. Also, as presented in our experimental results shown in Table 2, DARE can complement existing adversarial training methods and improve the universal robustness of the model, but sometimes the robustness improvement of DARE for certain adversarial attacks may also be inferior to existing methods. In such cases, incorporating corresponding adversarial samples in the data augmentation process of DARE to further improve its effectiveness can be feasible, we leave it to our future work.

7 CONCLUSION

In this paper, we have proposed a novel model post-tuning framework, called DARE, aiming at effectively improving the *universal robustness* of deep learning models. Specifically, DARE first transforms a classification model into an isomorphic regression model via a novel model transformation process, which can better perceive input perturbations for suppression and effectively consolidate the memory of crucial input features. Then, DARE collects the training data for the transformed model via mining historical models as training guidance. Finally, the fine-tuned regression model can be extended by the original model to accomplish model robustness improvement. We have evaluated DARE over 9 different models. Compared with five state-of-the-art adversarial training approaches, the improvements of DARE can be as large as 67.3%. In particular, DARE performed stably well while combating different attacks, indicating the high reliability of DARE for practical use.

ACKNOWLEDGMENTS

This work is partially supported by State Key Laboratory of Communication Content Cognition Grant No.A32001, the National Natural Science Foundation of China Grant Nos. 61782263 and 62002256.

REFERENCES

[1] Evan Ackerman. Accessed: 2021. News. https://spectrum.ieee.org/three-small-stickers-on-road-can-steer-tesla-autopilot-into-oncoming-lane
[2] Iveta Becková, Stefan Pócos, and Igor Farkas. 2020. Computational Analysis of Robustness in Neural Network Classifiers. In *29th International Conference on Artificial Neural Networks, Bratislava, Slovakia*. Springer, 65–76.
[3] Nicholas Carlini and David A. Wagner. 2017. Towards Evaluating the Robustness of Neural Networks. *2017 IEEE Symposium on Security and Privacy, SP*, 39–57.
[4] Chenyi Chen, Ari Seff, Alain L. Kornhauser, and Jianxiong Xiao. 2015. DeepDriving: Learning Affordance for Direct Perception in Autonomous Driving. In *2015 IEEE International Conference on Computer Vision, ICCV*. IEEE Computer Society, 2722–2730.
[5] Junjie Chen, Xiaoting He, Qingwei Lin, Hongyu Zhang, Dan Hao, Feng Gao, Zhangwei Xu, Yingnong Dang, and Dongmei Zhang. 2019. Continuous Incident Triage for Large-Scale Online Service Systems. In *34th IEEE/ACM International Conference on Automated Software Engineering, ASE 2019*. IEEE, 364–375.
[6] Junjie Chen, Zhuo Wu, Zan Wang, Hanmo You, Lingming Zhang, and Ming Yan. 2020. Practical accuracy estimation for efficient deep neural network testing. *ACM Transactions on Software Engineering and Methodology* 29, 4 (2020), 1–35.
[7] Holger Cleve and Andreas Zeller. 2005. Locating causes of program failures. In *Proceedings. 27th International Conference on Software Engineering, 2005. ICSE 2005*. IEEE, 342–351.
[8] Logan Engstrom, Brandon Tran, Dimitris Tsipras, Ludwig Schmidt, and Aleksander Madry. 2019. Exploring the landscape of spatial robustness. In *International Conference on Machine Learning*. PMLR, 1802–1811.
[9] Kevin Eykholt, Ivan Evtimov, Earlence Fernandes, Bo Li, Amir Rahmati, Chaowei Xiao, Atul Prakash, Tadayoshi Kohno, and Dawn Song. 2018. Robust Physical-World Attacks on Deep Learning Visual Classification. In *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018*. IEEE Computer Society, 1625–1634.
[10] Hazem Fahmy, Fabrizio Pastore, Mojtaba Bagherzadeh, and Lionel Briand. 2021. Supporting Deep Neural Network Safety Analysis and Retraining Through Heatmap-Based Unsupervised Learning. *IEEE Transactions on Reliability* (2021).
[11] Xiang Gao, Ripon K Saha, Mukul R Prasad, and Abhik Roychoudhury. 2020. Fuzz testing based data augmentation to improve robustness of deep neural networks. In *2020 IEEE/ACM 42nd International Conference on Software Engineering (ICSE)*. IEEE, 1147–1158.
[12] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. 2015. Explaining and Harnessing Adversarial Examples. *3rd International Conference on Learning Representations, ICLR 2015*.

[13] Divya Gopinath, Mengshi Zhang, Kaiyuan Wang, Ismet Burak Kadron, Corina S. Pasareanu, and Sarfraz Khurshid. 2019. Symbolic Execution for Importance Analysis and Adversarial Generation in Neural Networks. In *30th IEEE International Symposium on Software Reliability Engineering, ISSRE 2019, Berlin, Germany, October 28-31, 2019*. IEEE, 313–322.
[14] Sumit Gulwani. 2011. Automating string processing in spreadsheets using input-output examples. *ACM Sigplan Notices* 46, 1 (2011), 317–330.
[15] Sumit Gulwani. 2016. Programming by Examples: Applications, Algorithms, and Ambiguity Resolution. In *IJCAR*. 9–14.
[16] Sumit Gulwani and Prateek Jain. 2017. Programming by Examples: PL meets ML. In *APLAS*.
[17] Sumit Gulwani, Oleksandr Polozov, and Rishabh Singh. 2017. Program Synthesis. *Foundations and Trends in Programming Languages* 4, 1-2 (2017), 1–119.
[18] Daniel Conrad Halbert. 1984. *Programming by example*. Ph. D. Dissertation. University of California, Berkeley.
[19] Hossein Hosseini, Sreeram Kannan, and Radha Poovendran. 2019. Dropping Pixels for Adversarial Robustness. In *IEEE Conference on Computer Vision and Pattern Recognition Workshops, CVPR Workshops 2019*. Computer Vision Foundation / IEEE, 91–97.
[20] Ruitong Huang, Bing Xu, Dale Schuurmans, and Csaba Szepesvári. 2015. Learning with a strong adversary. *arXiv preprint arXiv:1511.03034* (2015).
[21] Md Johirul Islam, Giang Nguyen, Rangeet Pan, and Hridesh Rajan. 2019. A comprehensive study on deep learning bug characteristics. In *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 510–520.
[22] Md Johirul Islam, Rangeet Pan, Giang Nguyen, and Hridesh Rajan. 2020. Repairing deep neural networks: Fix patterns and challenges. In *2020 IEEE/ACM 42nd International Conference on Software Engineering (ICSE)*. IEEE, 1135–1146.
[23] Adam Ivankay, Ivan Girardi, Chiara Marchiori, and Pascal Frossard. 2020. FAR: A General Framework for Attributional Robustness. *CoRR* abs/2010.07393 (2020).
[24] Ruyi Ji, Yican Sun, Yingfei Xiong, and Zhenjiang Hu. 2020. Guiding dynamic programing via structural probability for accelerating programming by example. *Proceedings of the ACM on Programming Languages* 4, OOPSLA (2020), 1–29.
[25] Li Jia, Hao Zhong, Xiaoyin Wang, Linpeng Huang, and Xuansheng Lu. 2020. An Empirical Study on Bugs Inside TensorFlow. In *International Conference on Database Systems for Advanced Applications*. Springer, 604–620.
[26] Haoming Jiang, Pengcheng He, Weizhu Chen, Xiaodong Liu, Jianfeng Gao, and Tuo Zhao. 2020. SMART: Robust and Efficient Fine-Tuning for Pre-trained Natural Language Models through Principled Regularized Optimization. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. 2177–2190. https://doi.org/10.18653/v1/2020.acl-main.197
[27] Jiajun Jiang, Luyao Ren, Yingfei Xiong, and Lingming Zhang. 2019. Inferring Program Transformations From Singular Examples via Big Code. In *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. 255–266. https://doi.org/10.1109/ASE.2019.00033
[28] Jiajun Jiang, Yingfei Xiong, and Xin Xia. 2019. A manual inspection of Defects4J bugs and its implications for automatic program repair. *Science China Information Sciences* 62 (Sep 2019), 200102. https://doi.org/10.1007/s11432-018-1465-6
[29] Jiajun Jiang, Yingfei Xiong, Hongyu Zhang, Qing Gao, and Xiangqun Chen. 2018. Shaping Program Repair Space with Existing Patches and Similar Code. In *ISSTA*.
[30] Wei Jiang, Zhiyuan He, Jinyu Zhan, and Weijia Pan. 2021. Attack-Aware Detection and Defense to Resist Adversarial Examples. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.* 40, 10 (2021), 2194–2198.
[31] Kyle D. Julian, Jessica Lopez, Jeffrey S. Brush, Michael P. Owen, and Mykel J. Kochenderfer. 2016. Policy compression for aircraft collision avoidance systems. In *2016 IEEE/AIAA 35th Digital Avionics Systems Conference (DASC)*. 1–10.
[32] Nal Kalchbrenner and Phil Blunsom. 2013. Recurrent Continuous Translation Models. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing, EMNLP 2013, A meeting of SIGDAT, a Special Interest Group of the ACL*. ACL, 1700–1709.
[33] Ashwin Kalyan, Abhishek Mohta, Oleksandr Polozov, Dhruv Batra, Prateek Jain, and Sumit Gulwani. 2018. Neural-guided deductive search for real-time program synthesis from examples. In *International Conference on Learning Representations (ICLR)*.
[34] Sungmin Kang, Robert Feldt, and Shin Yoo. 2020. SINVAD: Search-based Image Space Navigation for DNN Image Classifier Test Input Generation. In *ICSE '20: 42nd International Conference on Software Engineering, Workshops, Seoul, Republic of Korea, 27 June - 19 July, 2020*. ACM, 521–528.
[35] Emanuel Kitzelmann. 2009. Inductive programming: A survey of program synthesis techniques. In *International Workshop on Approaches and Applications of Inductive Programming*. Springer, 50–73.
[36] Alex Krizhevsky. 2009. *Learning multiple layers of features from tiny images*. Technical Report.
[37] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. 2012. ImageNet Classification with Deep Convolutional Neural Networks. In *Advances in Neural Information Processing Systems 25: 26th Annual Conference on Neural Information Processing Systems*. 1106–1114.
[38] Alexey Kurakin, Ian Goodfellow, and Samy Bengio. 2017. Adversarial machine learning at scale. In *International Conference on Learning Representations*.
[39] Alexey Kurakin, Ian J. Goodfellow, and Samy Bengio. 2017. Adversarial examples in the physical world. *5th International Conference on Learning Representations, ICLR 2017*.
[40] Seokhyun Lee, Sooyoung Cha, Dain Lee, and Hakjoo Oh. 2020. Effective white-box testing of deep neural networks with adaptive neuron-selection strategy. In *ISSTA '20: 29th ACM SIGSOFT International Symposium on Software Testing and Analysis, Virtual Event, USA, July 18-22, 2020*, Sarfraz Khurshid and Corina S. Pasareanu (Eds.). ACM, 165–176.
[41] Xia Li, Jiajun Jiang, Samuel Benton, Yingfei Xiong, and Lingming Zhang. 2021. A Large-scale Study on API Misuses in the Wild. In *2021 14th IEEE Conference on Software Testing, Verification and Validation (ICST)*. 241–252. https://doi.org/10.1109/ICST49551.2021.00034
[42] Xia Li, Wei Li, Yuqun Zhang, and Lingming Zhang. 2019. DeepFL: integrating multiple fault diagnosis dimensions for deep fault localization. In *Proceedings of the 28th ACM SIGSOFT International Symposium on Software Testing and Analysis, ISSTA 2019*. ACM, 169–180.
[43] Yixing Luo, Xiao-Yi Zhang, Paolo Arcaini, Zhi Jin, Haiyan Zhao, Fuyuki Ishikawa, Rongxin Wu, and Tao Xie. 2021. Targeting Requirements Violations of Autonomous Driving Systems by Dynamic Evolutionary Search. In *36th IEEE/ACM International Conference on Automated Software Engineering, ASE 2021*. IEEE, 279–291. https://doi.org/10.1109/ASE51524.2021.9678883
[44] Shiqing Ma, Yingqi Liu, Wen-Chuan Lee, Xiangyu Zhang, and Ananth Grama. 2018. MODE: automated neural network model debugging via state differential analysis and input selection. In *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 175–186.
[45] Yixuan Ma, Shuang Liu, Jiajun Jiang, Guanhong Chen, and Keqiu Li. 2021. A comprehensive study on learning-based PE malware family classification methods. In *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 1314–1325.
[46] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. 2018. Towards Deep Learning Models Resistant to Adversarial Attacks. In *6th International Conference on Learning Representations, ICLR 2018*. OpenReview.net.
[47] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. 2018. Towards Deep Learning Models Resistant to Adversarial Attacks. *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*.
[48] Ravi Mangal, Kartik Sarangmath, Aditya V. Nori, and Alessandro Orso. 2020. Probabilistic Lipschitz Analysis of Neural Networks. In *Static Analysis - 27th International Symposium, SAS 2020, Virtual Event, November 18-20, 2020, Proceedings (Lecture Notes in Computer Science, Vol. 12389)*. Springer, 274–309.
[49] Haitao Mi, Zhiguo Wang, and Abe Ittycheriah. 2016. Supervised Attentions for Neural Machine Translation. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing, EMNLP 2016*. The Association for Computational Linguistics, 2283–2288.
[50] Mahdi Nejadgholi and Jinqiu Yang. 2019. A study of oracle approximations in testing deep learning libraries. In *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 785–796.
[51] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y Ng. 2011. Reading digits in natural images with unsupervised feature learning. In *NIPS Workshop on Deep Learning and Unsupervised Feature Learning 2011*.
[52] Nicolas Papernot, Patrick McDaniel, Somesh Jha, Matt Fredrikson, Z Berkay Celik, and Ananthram Swami. 2016. The limitations of deep learning in adversarial settings. In *2016 IEEE European symposium on security and privacy (EuroS&P)*. IEEE, 372–387.
[53] Nicolas Papernot, Patrick McDaniel, Xi Wu, Somesh Jha, and Ananthram Swami. 2016. Distillation as a defense to adversarial perturbations against deep neural networks. In *2016 IEEE symposium on security and privacy (SP)*. IEEE, 582–597.
[54] Hung Viet Pham, Shangshu Qian, Jiannan Wang, Thibaud Lutellier, Jonathan Rosenthal, Lin Tan, Yaoliang Yu, and Nachiappan Nagappan. 2020. Problems and opportunities in training deep learning software systems: an analysis of variance. In *Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering*. 771–783.
[55] Sukrut Rao, David Stutz, and Bernt Schiele. 2020. Adversarial Training Against Location-Optimized Adversarial Patches. In *Computer Vision - ECCV 2020 Workshops (Lecture Notes in Computer Science, Vol. 12539)*. Springer, 429–448.
[56] Ramprasaath R. Selvaraju, Michael Cogswell, Abhishek Das, Ramakrishna Vedantam, Devi Parikh, and Dhruv Batra. 2017. Grad-CAM: Visual Explanations from Deep Networks via Gradient-Based Localization. *IEEE International Conference on Computer Vision, ICCV 2017, Venice, Italy, October 22-29, 2017* (2017), 618–626.
[57] Ali Shafahi, Mahyar Najibi, Amin Ghiasi, Zheng Xu, John Dickerson, Christoph Studer, Larry S Davis, Gavin Taylor, and Tom Goldstein. 2019. Adversarial training for free!. In *Advances in Neural Information Processing Systems*, Vol. 32.
[58] Qingchao Shen, Junjie Chen, Jie Zhang, Haoyu Wang, Shuang Liu, and Menghan Tian. 2022. Natural Test Generation for Precise Testing of Question Answering

Software. In *37th IEEE/ACM International Conference on Automated Software Engineering*. to appear.

[59] Hoo-Chang Shin, Le Lu, Lauren Kim, Ari Seff, Jianhua Yao, and Ronald M. Summers. 2016. Interleaved Text/Image Deep Mining on a Large-Scale Radiology Database for Automated Image Interpretation. *J. Mach. Learn. Res.* 17, 1 (2016), 3729–3759.

[60] David Shriver, Sebastian G. Elbaum, and Matthew B. Dwyer. 2021. Reducing DNN Properties to Enable Falsification with Adversarial Attacks. In *43rd IEEE/ACM International Conference on Software Engineering, ICSE 2021, Madrid, Spain, 22-30 May 2021*. IEEE, 275–287.

[61] Karen Simonyan and Andrew Zisserman. 2015. Very Deep Convolutional Networks for Large-Scale Image Recognition. In *3rd International Conference on Learning Representations, ICLR 2015*.

[62] Xiaobing Sun, Tianchi Zhou, Gengjie Li, Jiajun Hu, Hui Yang, and Bin Li. 2017. An empirical study on real bugs for machine learning programs. In *2017 24th Asia-Pacific Software Engineering Conference (APSEC)*. IEEE, 348–357.

[63] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian J. Goodfellow, and Rob Fergus. 2014. Intriguing properties of neural networks. *2nd International Conference on Learning Representations, ICLR 2014*.

[64] Rajkumar Theagarajan, Ming Chen, Bir Bhanu, and Jing Zhang. 2019. ShieldNets: Defending Against Adversarial Attacks Using Probabilistic Adversarial Robustness. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019, Long Beach, CA, USA, June 16-20, 2019*. Computer Vision Foundation / IEEE, 6988–6996.

[65] Zhao Tian, Junjie Chen, Qihao Zhu, Junjie Yang, and Lingming Zhang. 2022. Learning to Construct Better Mutation Faults. In *37th IEEE/ACM International Conference on Automated Software Engineering*. to appear.

[66] Florian Tramèr, Alexey Kurakin, Nicolas Papernot, Ian Goodfellow, Dan Boneh, and Patrick McDaniel. 2018. Ensemble Adversarial Training: Attacks and Defenses. In *International Conference on Learning Representations*.

[67] Muhammad Usman, Divya Gopinath, Youcheng Sun, Yannic Noller, and Corina S. Pasareanu. 2021. NNrepair: Constraint-Based Repair of Neural Network Classifiers. In *Computer Aided Verification - 33rd International Conference, CAV 2021, Virtual Event, July 20-23, 2021, Proceedings, Part I (Lecture Notes in Computer Science, Vol. 12759)*. Springer, 3–25.

[68] Jingyi Wang, Jialuo Chen, Youcheng Sun, Xingjun Ma, Dongxia Wang, Jun Sun, and Peng Cheng. 2021. RobOT: Robustness-Oriented Testing for Deep Learning Systems. *43rd IEEE/ACM International Conference on Software Engineering, ICSE 2021, Madrid, Spain, 22-30 May 2021* (2021), 300–311.

[69] Zan Wang, Hanmo You, Junjie Chen, Yingyi Zhang, Xuyuan Dong, and Wenbin Zhang. 2021. Prioritizing test inputs for deep neural networks via mutation analysis. In *2021 IEEE/ACM 43rd International Conference on Software Engineering*. IEEE, 397–409.

[70] Westley Weimer, ThanhVu Nguyen, Claire Le Goues, and Stephanie Forrest. 2009. Automatically finding patches using genetic programming. In *ICSE*. 364–374. https://doi.org/10.1109/ICSE.2009.5070536

[71] Ming Wen, Junjie Chen, Rongxin Wu, Dan Hao, and Shing-Chi Cheung. 2018. Context-Aware Patch Generation for Better Automated Program Repair. In *ICSE*.

[72] Tsui-Wei Weng, Huan Zhang, Pin-Yu Chen, Jinfeng Yi, Dong Su, Yupeng Gao, Cho-Jui Hsieh, and Luca Daniel. 2018. Evaluating the Robustness of Neural Networks: An Extreme Value Theory Approach. *The 6th International Conference on Learning Representations, ICLR 2018* (2018).

[73] Han Xiao, Kashif Rasul, and Roland Vollgraf. 2017. Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms. *CoRR* abs/1708.07747 (2017).

[74] Yingfei Xiong, Jie Wang, Runfa Yan, Jiachen Zhang, Shi Han, Gang Huang, and Lu Zhang. 2017. Precise Condition Synthesis for Program Repair. In *ICSE*. https://doi.org/10.1109/ICSE.2017.45

[75] Weilin Xu, David Evans, and Yanjun Qi. 2018. Feature squeezing: Detecting adversarial examples in deep neural networks. *2018 Network and Distributed System Security Symposium* (2018).

[76] Lin Yang, Junjie Chen, Zan Wang, Weijing Wang, Jiajun Jiang, Xuyuan Dong, and Wenbin Zhang. 2021. Semi-supervised log-based anomaly detection via probabilistic label estimation. In *2021 IEEE/ACM 43rd International Conference on Software Engineering*. IEEE, 1448–1460.

[77] Haruki Yokoyama, Satoshi Onoue, and Shinji Kikuchi. 2020. Towards building robust DNN applications: an industrial case study of evolutionary data augmentation. In *2020 35th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 1184–1188.

[78] Bing Yu, Hua Qi, Qing Guo, Felix Juefei-Xu, Xiaofei Xie, Lei Ma, and Jianjun Zhao. 2021. DeepRepair: Style-Guided Repairing for Deep Neural Networks in the Real-World Operational Environment. *IEEE Transactions on Reliability* (2021), 1–16.

[79] Valentina Zantedeschi, Maria-Irina Nicolae, and Ambrish Rawat. 2017. Efficient defenses against adversarial attacks. In *Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security*. 39–49.

[80] Andreas Zeller. 2002. Isolating cause-effect chains from computer programs. *ACM SIGSOFT Software Engineering Notes* 27, 6 (2002), 1–10.

[81] Andreas Zeller and Ralf Hildebrandt. 2002. Simplifying and isolating failure-inducing input. *IEEE Transactions on Software Engineering* 28, 2 (2002), 183–200.

[82] Jie M. Zhang, Mark Harman, Lei Ma, and Yang Liu. 2022. Machine Learning Testing: Survey, Landscapes and Horizons. *IEEE Transactions on Software Engineering* 48, 1 (2022), 1–36. https://doi.org/10.1109/TSE.2019.2962027

[83] Kai Zhang, Wangmeng Zuo, Yunjin Chen, Deyu Meng, and Lei Zhang. 2017. Beyond a Gaussian Denoiser: Residual Learning of Deep CNN for Image Denoising. *IEEE Trans. Image Process.* 26, 7 (2017), 3142–3155.

[84] Mengshi Zhang, Yuqun Zhang, Lingming Zhang, Cong Liu, and Sarfraz Khurshid. 2018. DeepRoad: GAN-based metamorphic testing and input validation framework for autonomous driving systems. In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering, ASE 2018*. ACM, 132–142.

[85] Qiao Zhang, Zhipeng Cui, Xiaoguang Niu, Shijie Geng, and Yu Qiao. 2017. Image Segmentation with Pyramid Dilated Convolution Based on ResNet and U-Net. In *Neural Information Processing - 24th International Conference, ICONIP 2017 (Lecture Notes in Computer Science, Vol. 10635)*. Springer, 364–372.

[86] Ru Zhang, Wencong Xiao, Hongyu Zhang, Yu Liu, Haoxiang Lin, and Mao Yang. 2020. An empirical study on program failures of deep learning jobs. In *2020 IEEE/ACM 42nd International Conference on Software Engineering (ICSE)*. IEEE, 1159–1170.

[87] Tianyi Zhang, Cuiyun Gao, Lei Ma, Michael Lyu, and Miryung Kim. 2019. An empirical study of common challenges in developing deep learning applications. In *2019 IEEE 30th International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 104–115.

[88] Yuhao Zhang, Yifan Chen, Shing-Chi Cheung, Yingfei Xiong, and Lu Zhang. 2018. An empirical study on TensorFlow program bugs. In *Proceedings of the 27th ACM SIGSOFT International Symposium on Software Testing and Analysis*. 129–140.

[89] Ziqi Zhang, Yuanchun Li, Yao Guo, Xiangqun Chen, and Yunxin Liu. 2020. Dynamic slicing for deep neural networks. *ESEC/FSE '20: 28th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, Virtual Event, USA, November 8-13, 2020* (2020), 838–850.

[90] Zhe Zhao, Guangke Chen, Jingyi Wang, Yiwei Yang, Fu Song, and Jun Sun. 2021. Attack as Defense: Characterizing Adversarial Examples Using Robustness. *Proceedings of the 30th ACM SIGSOFT International Symposium on Software Testing and Analysis* (2021), 42–55.

[91] Ziyuan Zhong, Yuchi Tian, and Baishakhi Ray. 2021. Understanding Local Robustness of Deep Neural Networks under Natural Variations. *Fundamental Approaches to Software Engineering* 12649 (2021), 313.

[92] Husheng Zhou, Wei Li, Zelun Kong, Junfeng Guo, Yuqun Zhang, Bei Yu, Lingming Zhang, and Cong Liu. 2020. DeepBillboard: systematic physical-world testing of autonomous driving systems. In *ICSE '20: 42nd International Conference on Software Engineering*. ACM, 347–358.