A Post-Training Framework for Improving the Performance of Deep Learning Models via Model Transformation

JIAJUN JIANG, Tianjin University, China
JUNJIE YANG*, Tianjin University, China
YINGYI ZHANG*, Tianjin University, China
ZAN WANG, Tianjin University, China
HANMO YOU, Tianjin University, China
JUNJIE CHEN†, Tianjin University, China

Deep learning (DL) techniques have attracted much attention in recent years and have been applied to many application scenarios. To improve the performance of DL models regarding different properties, many approaches have been proposed in the last decades, such as improving the robustness and fairness of DL models to meet the requirements for practical use. Among existing approaches, post-training is an effective method that has been widely adopted in practice due to its high efficiency and good performance. Nevertheless, its performance is still limited due to the incompleteness of training data. Additionally, existing approaches are always specifically designed for certain tasks, such as improving model robustness, which cannot be used for other purposes.

In this paper, we aim to fill this gap and propose an effective and general post-training framework, which can be adapted to improve the model performance from different aspects. Specifically, it incorporates a novel model transformation technique that transforms a classification model into an isomorphic regression model for fine-tuning, which can effectively overcome the problem of incomplete training data by forcing the model to strengthen the memory of crucial input features and thus improve the model performance eventually. To evaluate the performance of our framework, we have adapted it to two emerging tasks for improving DL models, i.e., robustness and fairness improvement, and conducted extensive studies by comparing it with state-of-the-art approaches. The experimental results demonstrate that our framework is indeed general as it is effective in both tasks. Specifically, in the task of robustness improvement, our approach DARE has achieved the best results on 61.1% cases (*vs.* 11.1% cases achieved by baselines). In the task of fairness improvement, our approach FMT can effectively improve the fairness without sacrificing the accuracy of the models.

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

Additional Key Words and Phrases: Deep Neural Network, Delta Debugging, Model Robustness, Model Fairness

*Both authors contributed equally to this research.
†Corresponding author.

Authors' addresses: Jiajun Jiang, Tianjin University, College of Intelligence and Computing, Tianjin, China, 300350, jiangjiajun@tju.edu.cn; Junjie Yang, jjyang@tju.edu.cn, Tianjin University, College of Intelligence and Computing, Tianjin, China, 300350; Yingyi Zhang, Tianjin University, College of Intelligence and Computing, Tianjin, China, 300350, yingyizhang@tju.edu.cn; Zan Wang, Tianjin University, College of Intelligence and Computing, Tianjin, China, 300350, wangzan@tju.edu.cn; Hanmo You, Tianjin University, College of Intelligence and Computing, Tianjin, China, 300350, youhanmo@tju.edu.cn; Junjie Chen, Tianjin University, College of Intelligence and Computing, Tianjin, China, 300350, junjiechen@tju.edu.cn.

1049-331X/2023/1-ART1 $15.00
https://doi.org/10.1145/3630011

ACM Reference Format:
Jiajun Jiang, Junjie Yang, Yingyi Zhang, Zan Wang, Hanmo You, and Junjie Chen. 2023. A Post-Training Framework for Improving the Performance of Deep Learning Models via Model Transformation. *ACM Trans. Softw. Eng. Methodol.* 1, 1, Article 1 (January 2023), 42 pages. https://doi.org/10.1145/3630011

1 INTRODUCTION

Deep learning (DL) techniques, due to their high effectiveness, have attracted much attention from both academic researchers and industrial developers, and have been widely used in many application scenarios, such as image processing [130, 134], machine translation [54, 76] and software engineering [16, 67, 102, 119], etc. The typical working pipeline of DL techniques consists of three major phases [21]. First, developers design and implement a neural network architecture of desired functionality; then, a DL model will be trained over a set of training data with the implemented network; finally, the trained model can be deployed for practical use. Due to the random nature of model training involved by both training data and training process [144], some properties usually cannot be guaranteed by the trained models even though they may be critically important, which may risk DL models' performance and reliability for practical use, such as causing ethical violations or safety issues. As reported by existing studies [25, 90, 100], 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]. Additionally, AI systems also affect our daily lives with their inherent biases, such as the existence of bias in AI chatbots, employment matching, automated legal aid for immigration algorithms and so on [80]. With the increase in DL models' adoption in more and more ethically sensitive and safety-critical application scenarios, e.g., AI judge [96, 105], hiring [7], loaning [77], autonomous driving [15, 69, 132, 143] and aircraft collision avoidance [52], it is emerging to improve the performance of DL models regarding certain properties (e.g., fairness and safety) for producing reliable results.

However, unlike traditional handcrafted programs that are deterministic with a fixed code logic defined by a set of executable machine instructions, as aforementioned deep learning models are built based on a set of input examples. That is, when providing a set of training data, a model with a set of parameters will be learned according to the implemented neural network architecture, 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 (aka., incomplete specification issue in traditional software engineering tasks like programming by examples [31–35, 45, 55, 60]), the learned model may not work well on unseen inputs. In particular, it tends to produce incorrect results when facing inputs decorated with crafted attacking features.

In order to improve the performance of DL models for practical use, many advanced approaches have been proposed aiming at different properties. Specifically, to improve the safety, researchers put forward a set of defensive approaches to enhance the robustness of DL models to combat adversarial inputs in the last decade, which typically can be classified 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 [82], feature squeezing [118], adversarial training [28, 72, 89], and so on. On the contrary, the other category aims at designing a detection method that can distinguish adversarial inputs from benign ones [140]; then, each time a new input is provided, an online detection process will be required. Similarly, to improve the fairness, many optimization techniques from different perspectives and in different phases have been proposed aiming at reducing the bias in the learned models, including optimizing the training data before model training [8, 20], optimizing the training

process [23, 126], and fine-tuning already learned models [133, 141]. Although many approaches have been proposed, they usually improve the fairness by largely sacrificing the accuracy of the learned model. More importantly, different properties of DL models are always studied separately in the literature, and there is no general technique that can fit them all. In this paper, we aim to fill this gap by proposing a general framework that can be used for improving DL models' performance regarding different properties.

Specifically, our basic intuition is that the unsatisfactory performance of DL models is due to their weak understanding of crucial input features, which makes them easy to be attacked or tricked by less relevant features. However, how to improve the understanding of DL models to such crucial features is indeed challenging since the interpretability of DL models is usually a mystery. Indeed, some approaches try to perform a white-box optimization process during the model training, but they need to change the architecture of the original models and conduct model training from scratch [82], which is actually time-consuming. As a result, most existing approaches typically work in a completely black-box fashion by feeding more training data for model fine-tuning [72, 89], which is orthogonal to the white-box approaches. In other words, the underlying purpose behind the black-box approaches is to improve model performance by enlarging the coverage of input features through incorporating more training samples. The advantage of them is that they can inherit the good performance of the learned models and have higher efficiency. In this paper, we also target the second category that improves DL models by a pluggable post model training, but with a novel finer-grained training strategy in a white-box fashion.

As aforementioned, training samples are limited in practice; as a result the models after fine-tuning may still perform unsatisfactorily with unseen inputs. For example, DL models fine-tuned by FGSM (Fast Gradient Sign Method) [28] (one widely-used fine-tuning method for improving model robustness) can still be easily attacked by a different attacking algorithm, e.g., C&W (Carlini&Wagner) [11]. The reasons of the limited effectiveness of existing black-box fine-tuning approaches are twofold: 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 not be well captured and strengthened. As a consequence, the performance improvement of the model is limited; 2) The improvement gained from a particular set of training examples cannot generalize to other unseen inputs, i.e., weak generalizability. Therefore, the target of our framework is to overcome these two major limitations. Particularly, in order to overcome the first limitation of post-training approaches with data augmentation, we first transform the original classification model into an isomorphic regression model, which inherits the underlying network architecture 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 [19, 124, 125]. It will force the model to strengthen the memory of critical input features and ignore non-relevant features so as to be more generalizable.

In summary, aiming at effectively improving the performance of DL models for meeting the requirements of different properties, we propose a novel post-training framework in this paper, which constitutes a model transformation technique and a delta-debugging-fashion data augmentation strategy. Specifically, the basic idea of our framework was originally proposed in our previous work [138] for improving the *universal robustness* of DL models. Compared with the previous version, in this paper we further generalize it into a post-training framework, and additionally adapt it to a completely new application scenario, i.e., improving the fairness of DL models, for which we design and implement a new data augmentation algorithm under this framework. Finally,

we have conducted an extensive study to evaluate the performance of our framework for improving both robustness and fairness of DL models by comparing it with a set of state-of-the-art approaches. The experimental results demonstrate that our framework is indeed effective and general, where it outperforms the best existing approaches in both application scenarios. Specifically, our approach DARE has achieved the best results on 61.1% cases (*vs* 11.1% cases achieved by the best state-of-the-art baseline approach) in the task of robustness improvement, while in the task of fairness improvement, our approach FMT can effectively improve the fairness without sacrificing the accuracy of the models.

In summary, we make the following major contributions:

- We propose a novel and general post-training framework for improving the performance of DL 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.
- We instantiate our framework for two distinct DL models' performance improvement tasks, i.e., improving robustness and fairness, for which we propose two data augmentation algorithms according to our framework.
- We have implemented our approaches in two tools, named DARE and FMT for robustness and fairness improvement respectively, and conducted extensive evaluations to demonstrate the effectiveness of the proposed approaches.
- We make our implementations and all experimental data open-source to facilitate the replication and comparison in future studies: **https://doi.org/10.5281/zenodo.10009069**

The remainder of the paper is organized as follows. Section 2 introduces the background and motivates our framework; Section 3 presents the overview of our framework; Section 4 and Section 5 present the adaptations and evaluations of our framework for improving the robustness and fairness of deep learning models, respectively; Section 6 introduces the related work. Finally, Section 7 discusses the difference from Knowledge Distillation, limitations and future work; Section 8 discusses the threats to validity while Section 9 concludes the paper.

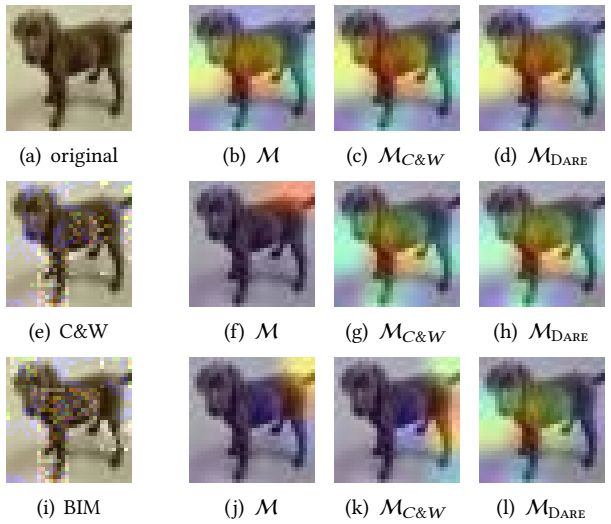
2 BACKGROUND AND MOTIVATION

In this section, we take the robustness improvement task as an example to show the limitation of existing approaches and motivate our research. Then, we introduce the background knowledge of delta debugging [124, 125], which inspires our framework.

2.1 Limited Performance 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 the 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. In this section, we take the model robustness improvement task as the example to present the issues faced by existing approaches. Specifically, we adopt the definition of *Empirical Robustness* from a previous study [107] as the *Model Robustness* in this paper, which denotes the capability of the model to defend against attacking inputs. It has been widely-used by existing studies [10, 73, 108, 109].

For example, Figure 1(a) presents a testing image of "dog" from the CIFAR10 dataset [61] in our experiment, which can be correctly classified by a well-trained VGG16[93] model $\mathcal{M}$. Particularly, we use the Gradient-weighted Class Activation Map [88] (heatmap for short) to visualize the features



In the figure, (a) is the original image from the CIFAR10 dataset, (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.

Fig. 1. An example to show the weak universal robustness of existing approaches.

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 to defend against potential attacks, many approaches have been proposed, among which adversarial training methods stand out due to their high effectiveness [74]. The underlying process is performing post model tuning over a set of samples generated by specifically-designed algorithms, such as C&W [11] and BIM [64]. 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 4.3), 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 the face of this limitation, in this paper we aim at proposing a novel and more effective post-training framework, which improves the performance of DL models by effectively strengthening the memory of crucial input features learned by the models and so that the effects of irrelevant input features will be significantly reduced. 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 our approach (DARE) can effectively defend against different attacks and produce the correct results. As our evaluation

shows (see Section 4.3), DARE can significantly improve the universal robustness of learned models combating various adversarial methods.

Note that the post-training process of our framework is general as it is not specifically designed to improve only the robustness of DL models. It can also be used to improve the other performance of DL models by enhancing their understanding of crucial input features. In particular, new data augmentation algorithms are needed for adapting our framework to new tasks as introduced in Section 3.2. As shown in our final evaluation, it can effectively improve the fairness of DL models as well. The reason is that our framework does not depend on any task specific features but enhancing the DL models' performance from the perspective of fine-grained model optimization (to be presented in Section 3).

2.2 Motivation from Delta Debugging

Delta debugging was originally proposed by Zeller et al. [19, 124, 125]. 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 as a reference that guides the process of finding and finally fixing the bug.

Inspired by this, we borrow the idea of delta debugging for model improvement. It is well known that existing post-training techniques [133, 141] for model optimization rely solely on the final prediction outputs (i.e., the classification labels, which are used for loss calculation). However, these outputs are too coarse-grained and thus may not be sensitive enough to small input perturbations to effectively guide the feature extraction, especially in classification tasks where correct predictions may have a low probability. This indicates that the model failed to accurately discriminate the input from other categories, making it vulnerable to attacks and easily influenced. On the contrary, regression models can be more sensitive to input features since small input perturbations may have a larger possibility to be propagated to and differ from the final output due to their finer-grained ground truth compared with discrete classification labels. Formally, we consider a simple regression model with n input variables $(x_1, x_2, \cdots, x_n)$ and one output variable y. The model can be represented as:

$$y = f(w_1 \times x_1 + w_2 \times x_2 + \cdots + w_n \times x_n + b) \tag{1}$$

where f is a continuous function and w_n and b are the weights and bias of the model. Given a small perturbation in the input variable x_i, denoted as δ_i, the output will be

$$y = f(w_1 \times x_1 + w_2 \times x_2 + \cdots + w_i \times x_i + w_i \times \delta_i + \cdots + w_n \times x_n + b) \tag{2}$$

By using the concept of partial derivatives, the sensitivity of the output variable y with respect to the input variable x_i can be calculated as follows:

$$sensitivity = \left| \frac{\partial y / \partial x_i}{\delta_i / x_i} \right| \tag{3}$$

where $\partial y / \partial x_i$ is the partial derivative of y with respect to x_i. That is

$$\partial y / \partial x_i = f'(w_1 \times x_1 + w_2 \times x_2 + \cdots + w_i \times x_i + \cdots + w_n \times x_n + b) \times w_i \tag{4}$$

However, since f is a continuous function, $f'(*)$ is bounded by a constant M. Therefore, the sensitivity of the output variable y with respect to the input variable x_i can be bounded as follows:

$$sensitivity \leq \frac{M \times w_i}{\delta_i / x_i} \tag{5}$$

This shows that the sensitivity of the output variable y with respect to input variable x_i is directly proportional to the weight w_i and the ratio of the input perturbation δ_i to the original input variable x_i. Similarly, for a classification model with the same n input variables, the sensitivity of y to input x_i can also be calculated by equations 3 and 4. However, since the output variable y is a discrete classification label, the sensitivity will be bounded by some constant M' but not directly proportional to the weight w_i and the input perturbation δ_i. In other words, regression models are generally more sensitive to small input perturbations compared to classification models.

As aforementioned, the delta debugging process finds incorrect program states by referring to a passed test run. Similarly, if we can find the desired program states (i.e., neuron outputs in the neural network scenario) like delta debugging as the reference for each training input in classification model, the model training process will be more sensitive to input features and thus can be more targeted to enhance the understanding of the model to crucial features. However, since the structure of classification and regression models are different, how to guide the training process of classification models in a finer-grained level like regression models and how to construct the reference program states are still challenging. To achieve this goal, we propose a novel model transformation technique that can automatically transform classification models into regression models for training, and the trained parameters will be synchronized to the original classification models in the end. This process is completely transparent to end users and will also preserve the superiority of the original learned models like existing post-training methods. As for the reference program states, we propose to construct them by mining model training history or taking them from similar test inputs like traditional delta debugging to fit the requirements of certain tasks. More details about the reference output construction process will be introduced when adapting our framework for different tasks later (see Section 4.1 for robustness improvement and Section 5.1 for fairness improvement).

3 FRAMEWORK

In this section, we introduce the details of our post-training framework, which aims at effectively improving the performance of DL models by strengthening the memory of crucial input features. In this way, the irrelevant input features will not affect the prediction results. Figure 2 presents the overview of our framework. From a high-level perspective, it 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 inheriting its underlying structure. Here we borrow the word "isomorphic" to indicate that the structure of the transformed model is the same as the original model. 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. 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 and provide reference program states for inputs during model training, the second stage, **data augmentation**, performs training data construction, which is a plugin component that can be adapted according to the training requirements. Finally, the transformed model will be **trained** over the constructed training data and then the fine-tuned model weights will be **synchronized** to the original classification model by simple weight replacement to obtain a refined model.

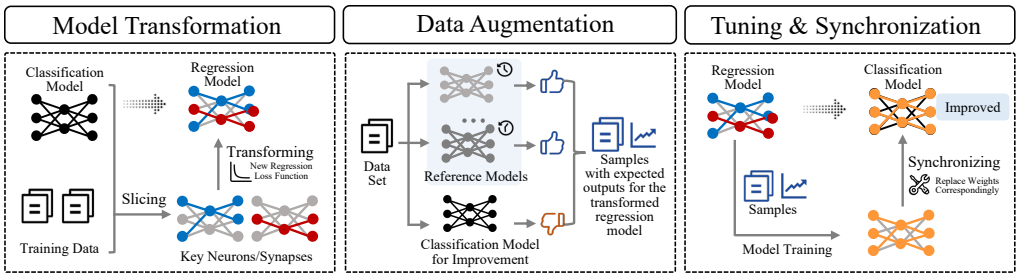


Fig. 2. An overview of our post-training framework for improving DL models' performance.

3.1 Model Transformation

In order to improve the performance 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, we propose to transform the original classification model into an isomorphic regression model in our framework. The reason is that, as aforementioned, 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. In fact, different training inputs may be contradictory, e.g., some inputs make the parameter w_i in the model increase while some decrease w_i. In other words, the optimization direction based on different input examples may contradict each other, making it hard to improve the performance consistently for different inputs. To reduce the training effect over different inputs, the model slicing process aims at identifying a set of crucial neurons and synapses (i.e., connections between neurons) for different inputs, which are responsible for reflecting the pivotal input features and determining the prediction, and thus may largely affect model performance on that input. 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 among different inputs.

Specifically, we employed NNSlicer [139], 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 6, which is regarded as the relative activation value of neuron n.

$$\Delta y^n = \overline{y^n(\mathcal{D}')} - \overline{y^n(\mathcal{D})} \tag{6}$$

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 7, where Δy^n and Δy^h are the relative activation values of neurons n and h respectively calculated by Formula 6. Additionally, we use s to denote the synapse connecting neurons h and n, and use w_{hn} to denote the connection weight.

$$contrib_h = CONTRIB_n \times \Delta y^n \times w_{hn} \Delta y^h \tag{7}$$

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.

According to the above process, when given the input samples and the targeted neurons, the contribution of each neuron and their connections can be calculated. In order to obtain the most accurate model slicing for model updating (will be presented in Section 3.3), we can take 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. Therefore, when instantiating our framework for certain tasks, the input samples and targeted neurons are required to be given, and the hyper-parameter $q\%$ should be configured.

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 in general more sensitive to small perturbations (refer to Section 2.2), and more suitable for fine-grained model tuning. Therefore, to better perceive small input perturbations for suppression and strengthen the memory of crucial input features, our framework inherits the underlying structure of the original classification model and transforms it into an isomorphic regression model with a specifically-designed novel loss function.

Specifically, when given a classification model $\mathcal{M}$ comprising r (convolutional or dense) layers, denoted as $\mathcal{M} := \langle l_1, l_2, \cdots, l_r \rangle$, our framework 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 $0 \leq p < r$. Then, it leverages the outputs of model slicing for loss function building, which are a set of crucial neurons that are responsible for model performance improvement. We use $\mathcal{N}_{l_i}^c$ to denote the crucial neurons from layer l_i for a set of interested inputs c, which can be of the same classification labels or include the same input

features. The selection of the input c is decided according to the application requirements. Then, the regression loss function of transformed model $\mathcal{M}_T$ is defined by Formula 8, where $y^n_{oracle}(\sigma)$ denotes the corresponding gold standard activation value of neuron n under input σ. Particularly, $y^n_{oracle}(\sigma)$ is the reference of expected output, the detailed construction process will be introduced in the data augmentation procedure (Section 3.2).

$$\mathcal{L}oss(\sigma) = \frac{\sum_{n \in \mathcal{N}^c_{l_{r-p}}} (y^n(\sigma) - y^n_{oracle}(\sigma))^2}{|\mathcal{N}^c_{l_{r-p}}|} \tag{8}$$

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, our framework makes $\mathcal{M}_T$ preserve as many layers in the original model as possible. As a consequence, it 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, the framework will take its preceding one until reaching a layer associated with a set of trainable parameters. In fact, it can be flexibly configured on demand in practical use.

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 model $\mathcal{M}_T$ 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, since the losses are calculated based on a subset of neurons that largely affect the prediction results of the training inputs, the parameter tuning process will be more effective. In addition, updating a subset of parameters may also potentially reduce its effects on predictions for different training inputs.

3.2 Data Augmentation

In order to strengthen the memory of crucial input features, the classification model will be transformed into a regression model according to the previous process. However, since the loss function has been changed, the original training data cannot apply anymore due to the different training labels, i.e., from classification labels to regression values. Therefore, the data augmentation process is to fill this gap by constructing new training data for the transformed regression model. In fact, the basic idea of the data augmentation strategy in our framework is simple and intuitive: *finding more effective isomorphic regression models for certain inputs (e.g., history models in the training process) as the reference and then collecting the corresponding prediction outputs of desired neurons.* It defines a data augmentation paradigm to conform to the transformed model in our framework. Note that this paradigm guarantees the collected outputs will make the original model produce correct predictions with high confidence. The reason is straightforward. As introduced in Section 3.1.2, the targeted layer is the last one that contains trainable parameters in the model. In other words, the functionality of the removed layers by our model transformation can be viewed as a constant function f whose output will not change throughout the whole training process as long as the input of f does not change. Therefore, by using the outputs producing higher prediction confidence as the training reference, the trained model is expected to produce the same prediction confidence. However, how to find such regression models can be different according to the requirements of performance improvement. For example, a model achieving good effectiveness in terms of robustness may still perform unsatisfactorily for fairness improvement. Therefore, the

concrete algorithms should be specifically designed and adapted accordingly when instantiating the framework for certain downstream model improvement tasks. Particularly, as they will be presented in Sections 4.1 and 5.1, we have designed two algorithms for robustness and fairness improvement, respectively, by taking the task characteristics into account.

3.3 Model Tuning and Synchronization

In the first two stages of our framework, an isomorphic regression model $\mathcal{M}_T$ to $\mathcal{M}$ and its corresponding training set $\mathcal{T}$ are obtained. The last stage is fine-tuning model $\mathcal{M}_T$ over $\mathcal{T}$, 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 inherit its original superiority (e.g., high accuracy) from previous training and be improved regarding certain properties, such as better robustness 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 [47, 91]. However, compared with the traditional fine-tuning, we would like to highlight two major differences in our framework. First, as introduced in Section 3.1.2, the loss calculation of different inputs can be different due to the discrepancy of the key neuron set $\mathcal{N}^c_{l-p}$ in Formula 8. Second, during the fine-tuning process, 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 our framework in practice. 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 model performance improvement. As it will be shown in our empirical evaluation, these strategies are indeed effective and yield much better results.

After model tuning, some connection weights in model $\mathcal{M}_T$ will be optimized. To make the original model $\mathcal{M}$ inherit the superiority from this process, our framework 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 TASK I: IMPROVING DL MODELS' ROBUSTNESS – DARE

To evaluate the performance of our framework, we first adapt it to the task of DL models' robustness improvement. As discussed in the introduction, improving the robustness of DL models is critically important in practice, especially in safety-critical application scenarios. Therefore, it can show the practical value of our framework to study its performance in this task. Specifically, we propose a novel data augmentation algorithm by following the paradigm proposed in Section 3.2 for adapting the framework to this task, where we find the reference models from the training history for training data construction. We have also implemented it in a tool, named DARE. Next, we will introduce the details of the data augmentation algorithm in Section 4.1, and then evaluate the performance of DARE in Sections 4.2 and 4.3.

4.1 Data Augmentation in DARE

As introduced in Section 3.2, the target of the data augmentation is to provide reference outputs for the transformed regression model as reference during the training process. To meet the requirement of effectively improving the robustness of DL models' robustness, we propose a novel data augmentation algorithm by mining the model training history, which is inspired by traditional delta debugging. The basis of our algorithm 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 gold reference since the higher confidence indicates better understanding of input features of the input, and thus its output can be taken as the gold standard for guiding the model tuning. Note that although this process does not generate completely new test inputs but reuses the original training data, the labels of those collected inputs are actually newly generated based on the historical models. Therefore, we also call this process "data augmentation" to make it consistent with our framework.

Algorithm 1 Data augmentation by mining historical models in DARE

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: $\mathcal{T}$: training data for $\mathcal{M}_T$

1: **for** *each input* σ *in* $\mathcal{D}$ **do**
2: $\mathcal{M}_{best} \leftarrow None$ ▷ Model with best prediction result
3: $gt \leftarrow$ *ground truth label of* σ
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: $\mathcal{T} \leftarrow \mathcal{T} \cup \left\langle \sigma, Y^l(\sigma) \right\rangle$ ▷ Add into training set
19: **end if**
20: **end for**
21: **return** $\mathcal{T}$

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 converge). 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 $conf$. In particular, the confidence $conf$ denotes the predicted probability of the given input to be label $label$, which is usually calculated by a *Softmax* function [36, 62, 93, 99] in classification models. According to these definitions, we present the detailed data augmentation process of DARE in Algorithm 1. When 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

Table 1. Basic information of subjects used to evaluate the performance of DARE

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

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 8.

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 $\mathcal{T}$ by DARE for the transformed regression model $\mathcal{M}_T$. Since the gold standard outputs for training samples in $\mathcal{T}$ 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 $\mathcal{M}_T$. Additionally, the specifically-designed loss function enables a more fine-grained optimization target of feature extraction and promotes the consolidation of the memories of crucial input features.

4.2 Experiment Setup of DARE

In this evaluation, we investigate the performance of DARE by answering the following research questions.

- **RQ1:** *How effective is DARE to improve model robustness?*
- **RQ2:** *How much does the model transformation contribute to the effectiveness of DARE?*
- **RQ3:** *How effective is the data augmentation algorithm in DARE?*
- **RQ4:** *How does DARE perform in terms of efficiency?*

4.2.1 Dataset and Models. To evaluate the performance of DARE extensively, we employed 3 widely-used datasets from prior studies [17, 40, 44, 86, 110], i.e., CIFAR10 [61], SVHN [79] and Fashion-MNIST [116]. Furthermore, to validate the generality of DARE, we employed 3 different neural network architectures in the experiment. They are Alexnet [62], VGG16 [93] and VGG19 [93], all of which are commonly used in previous studies [3, 101]. 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, including 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.

4.2.2 Baseline Approaches. Since 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 four widely-used adversarial training methods as the baselines, i.e., C&W (Carlini&Wagner) [11], FGSM (Fast Gradient Sign Method) [28], JSMA (Jacobian-based Saliency Map Attack) [81] and PGD (Project Gradient Descent) [74], which are state-of-the-art for improving adversarial robustness of DL models [100] through *offline* training on adversarial examples. Note that we also employed BIM

(Basic Iterative Methods) [64] in our previous paper [138] for comparison. In this paper, we omitted it to save space as it is essentially the same as PGD but with a different initialization strategy [89]. It was also evident in our previous paper that BIM is much less effective than PGD. Additionally, Usman et al. [106] recently proposed NNRepair, which leverages constraint solving to improve the robustness of deep neural networks. We also compared our approach with it. However, we failed to adapt it to our subjects due to configuration errors. To avoid implementation bias, we adopted the experimental results of NNRepair from its original paper [106] directly for comparison. Specifically, the model architecture is a 15-layer CNN network with 890k trainable parameters, the training dataset is CIFAR10, and the adversarial technique is FGSM. The reason we choose CIFAR10 and FGSM is that they are both used by NNRepair and our study. In this way, it can ease the comparison and reduce the implementation bias by reusing the results of NNRepair.

In addition, DARE incorporates the superiority of two major components. The first one is the model transformation (including model slicing and a finer-grained loss function) from our framework, while the second one is the data augmentation algorithm specifically designed for robustness improvement. To evaluate their effectiveness, 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 8 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$_{last}$** is another variant for evaluating the effectiveness of our loss function. DARE$_{last}$ selects the last output layer of the deep learning models as the target layer (i.e., $p = 0$ in Formula 8). In other words, DARE$_{last}$ computes the losses based on the predicted probability of the target label for classification models in our experiment.
- **DARE$_{rand}$** is the variant for evaluating the effectiveness of the data augmentation algorithm in DARE. It fully inherits the model transformation process in DARE, and then fine-tunes the transformed model over the same number of randomly selected input samples as DARE. Specifically, the process of lines 9-15 in Algorithm 1 will be replaced by a random selection.

4.2.3 Procedure and Measurement. Following previous studies [51, 101], we apply the four adversarial training (attacking) algorithms explained in Section 4.2.2 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. [107] to measure model robustness, which is defined as the testing accuracy over the attacking inputs. The existing study [113] 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 [140]), 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 four algorithms. Finally, we have repeated all the experiments 9 times to reduce the effects of randomness in model training and increase the reliability of the results. Then, we present the average value for comparison.

4.2.4 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

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

Model	Dataset	C&W					FGSM					JSMA					PGD				
		DARE	C&W	FGSM	JSMA	PGD	DARE	C&W	FGSM	JSMA	PGD	DARE	C&W	FGSM	JSMA	PGD	DARE	C&W	FGSM	JSMA	PGD
VGG16	CIFAR10	**91.9**	76.0	54.8	76.0	71.2	**74.3**	61.4	55.6	56.4	55.7	**84.9**	73.7	52.1	80.6	72.2	**83.3**	65.3	54.2	75.0	81.0
	SVHN	**94.5**	81.6	82.7	77.4	83.6	**81.3**	61.0	74.1	61.6	67.4	**83.7**	65.1	73.1	72.0	73.6	**85.7**	71.2	79.8	74.1	80.5
	FM	81.1	**83.2**	67.4	81.7	75.9	65.5	65.9	70.8	66.8	**83.6**	71.2	70.6	74.5	**85.3**	85.0	65.3	64.3	63.6	64.8	**69.0**
VGG19	CIFAR10	42.7	**67.8**	45.1	58.4	61.4	**73.2**	60.6	63.0	58.3	63.3	72.2	72.7	**83.6**	73.1	72.8	**88.7**	69.8	67.6	73.3	80.9
	SVHN	**83.1**	61.6	58.7	57.0	60.7	**86.3**	61.5	72.6	63.5	67.5	**87.6**	69.2	69.3	68.4	71.1	**89.6**	67.6	79.7	74.1	80.4
	FM	79.3	**86.1**	66.4	66.8	76.2	74.2	71.5	**77.4**	76.5	77.1	76.0	**82.9**	77.5	74.9	76.2	71.4	72.8	69.3	**76.3**	74.1
Alexnet	CIFAR10	**46.5**	34.1	39.5	36.9	36.6	**67.8**	52.7	55.5	55.2	54.4	61.3	56.5	56.1	63.4	**63.9**	**82.4**	60.9	62.8	67.4	69.0
	SVHN	67.6	58.6	49.9	**69.3**	63.4	**71.6**	57.1	65.5	68.2	69.9	80.8	68.3	71.4	**81.1**	77.8	**78.4**	68.2	70.3	73.8	77.8
	FM	76.7	54.6	80.7	86.1	**86.8**	**71.4**	66.2	63.6	61.8	66.1	**85.2**	79.9	55.5	81.0	80.7	**85.0**	80.8	54.7	68.9	80.2

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 our framework in DARE, we perform the model slicing process by classes, i.e., we identify the crucial neurons for different classification categories to reduce the training effects over different classes. 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. 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. Additionally, 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) from the validation set and collect the reference output for each input sample according to Algorithm 1 for model tuning and validation. Finally, we have conducted an extensive model tuning process for each baseline approach by a grid search and take their best configurations for the subsequent experiment, where we set the learning rate $r \in$ {1e-3, 1e-4, 1e-5, 1e-6} and batch size as 128 and training epoch as 20 for balancing effectiveness and efficiency [2, 22, 121]. As an exception, 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. The detailed configurations can be found in our open-source repository.

4.3 Result Analysis of DARE

4.3.1 [Robustness] RQ1: Overall Effectiveness of DARE. As introduced, we evaluated the overall effectiveness of DARE over 9 different models by comparing it with four 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 four 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 four comparing adversarial training methods respectively (i.e., C&W, FGSM, JSMA and PGD) over the testing samples generated by C&W. In particular, to ease the presentation, we use $\mathcal{M}_D^A$ to represent the learned model of architecture A over training set D, such as $\mathcal{M}_{CIFAR10}^{VGG16}$, and call a "*Testing Scenario*" (*TS* for short) as testing a certain model (e.g., $\mathcal{M}_{CIFAR10}^{VGG16}$)

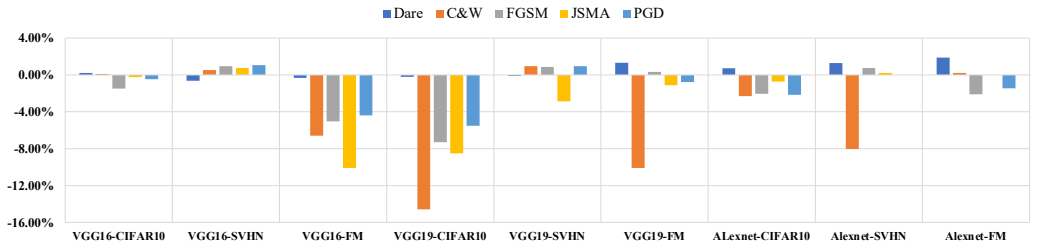


Fig. 3. Model accuracy improvement (%) over the original testing data after fine-tuning by each method.

over a set of testing inputs generated by a particular attacking algorithm (e.g., C&W). As a result, in total there are 36 *TS*s (9 models × 4 attacking algorithms). We also highlighted the best result per each *TS*. As aforementioned, to reduce the effect of randomness involved by trained models, we repeated our experiment 9 times on all 36 *TS*s. Please note that the results presented here are the average values of the repeated experiments. From the table we can see that DARE significantly outperforms all the baseline competitors. Specifically, DARE has achieved the highest accuracy in 22/36 *TS*s, and the improvement ranges from 0.8% to 67.7%, and the average improvement is 22.3% over all the 22 *TS*s. While the second optimal, i.e., C&W, JSMA and PGD, only achieves the highest accuracy in 4/36 *TS*s. Furthermore, we also performed the Wilcoxon signed-rank test [114] at the significance level of 0.05 to investigate whether our results significantly differ from those of the comparative baseline approaches in each *TS*. The results show that in the 22 *TS*s where DARE outperforms all the baselines, the differences between our approach and the baselines are statistically significant in 21/22 cases (except for $\mathcal{M}_{SVHN}^{Alexnet}$ over the attacking inputs generated by PGD since the *p-value* is 0.055.) with all *p-values* much smaller than 0.05, i.e., *p-value*<4e-03 for all cases. However, DARE may also underperform the baselines in some cases. In particular, DARE performs worse than all the baselines in 3/36 *TS*s, where the differences between our approach and the baselines are statistically significant in 2/3 *TS*s (i.e., $\mathcal{M}_{FM}^{VGG16}$ over the attacking inputs generated by FGSM and $\mathcal{M}_{CIFAR10}^{VGG19}$ over the inputs generated by JSMA) while the other one is not (i.e., $\mathcal{M}_{CIFAR10}^{VGG19}$ over the attacking inputs generated by C&W). The detailed *p-values* for all the cases can be found in our open-source repository. The experimental results demonstrate the superiority and significance of our approach.

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 C&W achieved a higher accuracy than DARE (83.2% vs 81.1%) over the attacking inputs generated by C&W. 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 C&W). The results partially confirm the effectiveness of adversarial training methods for model robustness improvement. On the other hand, the results also demonstrate their unsatisfactory *universal robustness* since they tend to achieve much worse robustness when dealing 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. One possible reason may be the simplicity of input samples from FM, where all samples are grayscale article images,

Table 3. Model accuracy (%) over different testing data before and after fine-tuning.

Model		Original Model		Fine-tuned Model			
Test Data		Train	Test	Train	Test	Adv-Train	Adv-Test
NNRepair	Interm.	87.25	81.04	87.30	80.93	34.61	36.30
	Last.	87.25	81.04	87.00	80.77	34.76	36.23
DARE		88.50	81.30	88.87	81.87	66.43	58.71

In the table, we present the results of NNRepair under different repair strategies. "Interm" represents *Intermediate-layer repair* while "Last" represents *Last-layer repair*. Train/Test: represents testing on the original training/testing data. Adv-Train/Adv-Test: represents testing on adversarial samples generated by FGSM based on the original training/testing data.

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. Therefore, we further compare the model performance over the original testing data without seeding attacking features. Figure 3 presents the average relative improvement of test accuracy over the normal testing data after fine-tuning by different approaches (negative number denotes the accuracy after tuning decreases). 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 significantly outperforms the others (p-$value$<0.03 at the significance level of 0.05) and 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 from another perspective, and demonstrates the effectiveness of DARE for practical use.

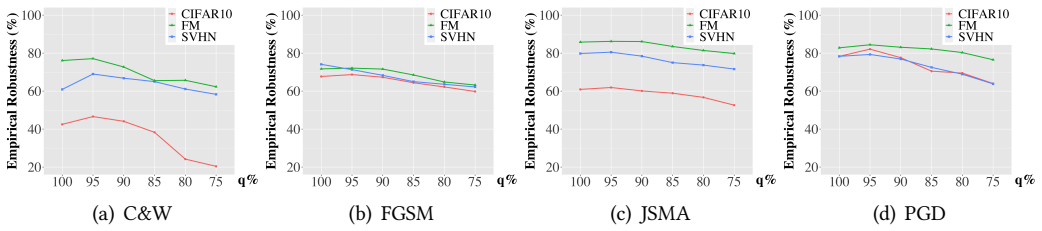
Finally, as introduced in Section 4.2.2, we compare our approach with the latest NNRepair [106] over the commonly used CIFAR10 dataset and reuse the experimental results of NNRepair reported in the corresponding paper. Table 3 presents the details. In particular, we present the model performance (i.e., accuracy) when testing over different datasets. Specifically, when testing over the Adv-Train, DARE achieved model accuracy as 66.43% after fine-tuning, while NNRepair only achieved model accuracies as 34.61% and 34.76% respectively when taking the two different repair strategies, i.e., *Intermediate-layer repair* and *Last-layer repair*. Similarly, over the Adv-Test, DARE achieved model accuracy as 58.71%, while NNRepair only achieved lower than 37%. The results present the effectiveness of our approach. However, the results also show that both NNRepair and DARE can preserve the accuracy of the original model, further demonstrating their effectiveness.

4.3.2 [Robustness] 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 three variants of DARE, i.e., DARE$_{-s}$, DARE$_{-sl}$ and DARE$_{last}$, which have been introduced

(a) C&W (b) FGSM

(c) JSMA (d) PGD

Fig. 4. Average relative improvement of variants DARE$_{-s}$, DARE$_{-sl}$ and DARE$_{last}$ on Empirical Robustness compared with DARE.



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

Fig. 5. Performance of DARE with different configurations of q for model slicing.

in Section 4.2.2. The results are shown in Figure 4. In the figure, we present the average relative improvement 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$_{-s}$

Table 4. Results when employing different data augmentation algorithms by DARE (Accuracy/%).

Model	Dataset	C&W		FGSM		JSMA		PGD		Model Acc After Tuning	
		DARE	$DARE_{rand}$	DARE	$DARE_{rand}$	DARE	$DARE_{rand}$	DARE	$DARE_{rand}$	DARE	$DARE_{rand}$
VGG16	CIFAR10	**91.9**	84.6	**74.3**	69.0	**84.9**	76.0	**83.3**	76.6	**88.9**	85.4
	SVHN	**94.5**	92.0	**81.3**	70.3	**83.7**	74.8	**85.7**	71.4	**93.7**	75.0
	FM	**81.1**	28.1	**65.5**	34.1	**71.2**	31.1	**65.3**	37.5	**90.8**	90.1
VGG19	CIFAR10	**42.7**	-	**73.2**	-	**72.2**	-	**88.7**	-	**90.4**	-
	SVHN	**83.1**	83.8	**86.3**	83.4	**87.6**	87.3	**89.6**	89.5	**93.8**	73.6
	FM	**79.3**	25.2	**74.2**	25.0	**76.0**	27.5	**71.4**	26.7	**91.4**	88.2
Alexnet	CIFAR10	**46.5**	31.1	**67.8**	46.7	**61.3**	44.3	**82.4**	53.7	**83.6**	82.0
	SVHN	**67.6**	18.2	**71.6**	25.4	**80.8**	33.4	**78.4**	38.4	**94.5**	89.7
	FM	**76.7**	26.3	**71.4**	56.5	**85.2**	64.6	**85.0**	76.1	**91.7**	89.8

on average drops 1.7%, and the largest decline is about 17.3% over DARE. By further removing the loss function in $DARE_{-s}$, the decline of testing accuracy over DARE ranges from 3.8% to 63.9%, where the average value is 27.4%. In addition, when we take the output prediction probability at the last layer to compute the losses (i.e., $DARE_{last}$), the decline of testing accuracy ranges from 5.4% to 57.4%, and the average value is 25.6%. The result indicates that the model transformation component in our framework significantly contributed to the effectiveness of DARE. Moreover, the results also prove that the selected layer for loss calculation in DARE is indeed effective.

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 5 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.

4.3.3 [Robustness] RQ3: Contribution of Data Augmentation in DARE. This research question investigates the performance of our data augmentation algorithm in DARE. 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 4.2.2). 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 $\mathcal{M}^{VGG19}_{CIFAR10}$ after fine-tuning dramatically drops, and no model satisfies our constraint regarding the model accuracy (no more than 10% drop). We use "-" to represent the missing data.

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

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

According to the table, DARE always outperforms DARE_{rand} regardless of the attacking algorithm (p-*value*<3e-09 at the significance level of 0.05). More specifically, the data augmentation algorithm in DARE contributes on average 61.2% higher Empirical Robustness by comparing with DARE_{rand}, and the highest improvement is as high as 271.3%. Additionally, the model accuracy can also be better preserved when employing our data augmentation algorithm, indicating the effectiveness of it.

4.3.4 [Robustness] 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 (**Generate**) and model tuning (**Train**), while for DARE we report the time for data augmentation (**Augment**), model slicing (**Slice**), and tuning. 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 (**Train**), 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 TASK II: IMPROVING DL MODELS' FAIRNESS – FMT

To evaluate the generalizability of our framework for improving DL models' performance from different perspectives, we further adapt our framework to the task of improving the fairness of DL models, which is a very important property especially in those ethically sensitive applications, such as the aforementioned AI judge [96, 105], loaning [77] and so on. Specifically, the target of fairness improvement can be briefly summarized as making the prediction result of the model not to be affected by a certain input feature, which is called a privileged feature (or protected attribute), such as the race and gender of human beings. To fit this target, we also design a novel data augmentation algorithm atop our framework and implement it in a tool named FMT. Particularly, the basic idea of the data augmentation algorithm in FMT is to generate pairs of training inputs having the same features except for the privileged one. By assigning the same prediction results for paired inputs, the trained models are expected to understand that the privileged feature should be ignored when

making decisions. This application is completely new to this paper compared with our previous paper [138]. Before introducing the details of the data augmentation algorithm in FMT, we first define some notions that will be used in the following presentation. If the assigned value of a privileged feature is considered advantageous or beneficial in a particular context, we call it a *privileged value*. For example, in a job application process [7], being *male* of the privileged feature *gender* might be considered a privileged value since men may be preferred over women for certain roles. Similarly, if the predicted outcome is considered desirable or positive in a given situation, the outcome is called a *favourable label*. For instance, in a loan approval process [77], *being approved* for a loan might be considered a favourable label. In real practice, the definitions of *privileged value* and *favourable label* depend on specific situations. In our experiment, we used the default definition in the studied dataset (refer to Section 5.2.1).

Algorithm 2 Data augmentation by mutant in FMT

Require: $\mathcal{D}$: a set of input training samples, $\mathcal{M}$: the source model for fairness improvement, idx: the index of the privileged attribute, l: targeted layer (a.k.a. the output layer of the transformed regression model $\mathcal{M}_T$)
Ensure: $\mathcal{T}_s, \mathcal{T}_d$: training data for fine-tuning model $\mathcal{M}_T$.

$Distr \leftarrow analyzeAttributeDistribution(\mathcal{D})$ ▷ Get the distribution of attributes in training data
$\mathcal{T}_s, \mathcal{T}_d \leftarrow \emptyset$
for i *from* 1 *to* $size(\mathcal{D})$ **do**
  $\sigma \leftarrow generateAttributes(Distr)$ ▷ Randomly assign values for attributes in σ based on $Distr$
  $\sigma' \leftarrow \sigma$
  $\sigma[idx] \leftarrow$ *privileged value* ▷ Assign the privileged value
  $\sigma'[idx] \leftarrow$ *unprivileged value* ▷ Assign the unprivileged value to form a pair with σ
  $\langle label, conf \rangle \leftarrow \mathcal{M}.predict(\sigma)$ ▷ Get the prediction of generated input σ
  $\langle label', conf' \rangle \leftarrow \mathcal{M}.predict(\sigma')$
  if $conf < conf'$ **then**
    $\sigma, \sigma' \leftarrow \sigma', \sigma$ ▷ Make σ store the input producing the higher prediction confidence
  end if
  $Y^l(\sigma) \leftarrow \{y^n(\sigma) | n \in \mathcal{N}_l^{gt}\}$ ▷ Neuron outputs of $\mathcal{M}$ in the layer of l for the input σ
  $\mathcal{T}_s \leftarrow \mathcal{T}_s \cup \left\langle \sigma', Y^l(\sigma) \right\rangle$ ▷ Store input and the desired neuron outputs
  $\mathcal{T}_d \leftarrow \mathcal{T}_d \cup \left\langle \sigma', Y^l(\sigma) \right\rangle \cup \left\langle \sigma, Y^l(\sigma) \right\rangle$ ▷ Store both inputs
end for
return $\mathcal{T}_s, \mathcal{T}_d$

5.1 Data Augmentation in FMT

In order to adapt our framework to the task of improving the fairness of DL models, we also design a new data augmentation algorithm, which takes the responsibility of providing the training data for the transformed regression model. As aforementioned, the target of the fairness improvement task is to make the model produce fair (or the same) prediction results for inputs whose attributes are the same except for the privileged one. According to the characteristics of this task, the basic idea of our algorithm is straightforward: producing pairwise inputs meeting the above condition and forcing the model to produce the same results for them. Additionally, since the original model is naturally well-trained with high accuracy for practical use, to make the improved model inherit this superiority, we set the output with higher confidence as the gold standard (i.e., the reference model is just the one to be improved). That is, for a pair inputs $\langle \sigma, \sigma' \rangle$, we will force the output of σ to be the same as σ' if the model produces higher confidence for σ'. In this way, the accuracy of the original model can be preserved as much as possible while improving fairness.

We present the details of the data augmentation algorithm in Algorithm 2. Similarly, we use $\mathcal{M}.predict(\sigma)$ to obtain the output of model $\mathcal{M}$ for the input σ, which is a pair of $\langle label, conf \rangle$, representing the predicted label *label* and the corresponding confidence *conf*, which is the same as that introduced in Section 4.1. Additionally, we define the function *analyzeAttributeDistribution($\mathcal{D}$)* to return the distribution of all attributes (i.e., feature values) in the given data set $\mathcal{D}$, and the function *generateAttributes(Distr)* to generate an input by randomly assigning a value for each attribute according to its value distribution. Therefore, when given the training data $\mathcal{D}$ of the original model $\mathcal{M}$, FMT first analyzes the distribution of all attributes (line 1) and repeatedly generates a set of new training data according to $\mathcal{D}$ with the expected outputs conforming to the transformed model $\mathcal{M}_T$ (lines 3-16). Specifically, each time FMT generates a pair of inputs with the same attribute values except for the appointed privileged one (i.e., $\sigma[idx]$), whose value will be set oppositely, e.g., *female* for privileged and *male* for unprivileged or vice versa (lines 4-7). Then, according to the prediction results of the paired inputs, FMT will collect the neuron outputs of the input with higher prediction confidence at the targeted layer (lines 10-15), which will play as the ground truth for subsequent model tuning (Slicing details in FMT will be introduced in Section 5.2.4). Particularly, we employ two strategies to construct the training data: (1) using a single one of the pairwise inputs (line 14) and (2) using both of the pairwise inputs (line 15). We use $\mathcal{T}_s$ and $\mathcal{T}_d$ to store the training data accordingly. In our evaluation, we will explore the performance of these two strategies empirically.

In summary, when providing the training data $\mathcal{D}$ for model $\mathcal{M}$, two sets of training data $\mathcal{T}_s$ and $\mathcal{T}_d$ will be automatically generated by FMT with the aim of improving the fairness of model $\mathcal{M}$. According to the construction algorithm, the training data in both $\mathcal{T}_s$ and $\mathcal{T}_d$ will force the model to produce the same prediction results for paired inputs, while the accuracy of the original model will not be largely affected. In this way, the fairness of the model can be improved with the guidance of the finer-grained loss function in the transformed regression model by strengthening the memory of crucial input features and reducing the effects of privileged features on the prediction.

5.2 Experiment Setup of FMT

Similar to the evaluation of Dare, to investigate the performance of FMT, we aim to answer the following research questions in this evaluation.

- **RQ5:** *How effective is FMT to improve DL models' fairness?*
- **RQ6:** *How much does the model transformation contribute to the effectiveness of FMT?*
- **RQ7:** *How effective is the data augmentation algorithm in FMT?*
- **RQ8:** *How does FMT perform in terms of efficiency?*

5.2.1 Benchmark Datasets. In our evaluation, we employ four widely-used datasets with different protected attributes, which have been extensively used in previous research for fairness improvement and integrated into the *AI Fairness 360 (AIF360)* toolkit [4]. We briefly introduce them as follows.

- **Adult** is used for predicting whether the annual income of individuals exceeds $50K. It contains more than 45K instances constituting 14 distinct attributes, two of which are regarded as privileged attributes, i.e., *sex* and *race*.
- **Bank** is used for predicting whether the client will subscribe to a term deposit. It includes more than 41K instances of 20 input attributes, where the attribute of *age* is the privileged one.
- **Compas** is used for assessing the likelihood that a criminal defendant will be an offender again. It includes more than 7K instances with 49 attributes, where *sex* and *race* are privileged.
- **German** is used for predicting the risk of individual credit. It includes 1K instances with 20 attributes. The privileged attributes of this dataset include *sex* and *age*.

Table 6 presents the details of the datasets used in our evaluation. Particularly, we filtered the instances that involve unknown features (i.e., the values are missing) from the first three datasets. Additionally, following previous work [18], we consider one privileged attribute each time for fairness improvement. Therefore, we finally have seven different datasets corresponding to all the privileged attributes, e.g., Adult-Sex and Bank-Age. Then, for each privileged attribute, we build a feed forward DL model for fairness improvement by following previous studies [97, 131]. Table 7 presents the details of the trained models that will be used as the subjects in our experiment. In the table, we listed the sizes, parameter numbers, and the accuracy of the trained models. Also, we listed the model fairness regarding three fairness metrics (i.e., SPD, AAOD, and EOD) that will be introduced in Section 5.2.3.

Table 6. Benchmarks for evaluating the performance of FMT regarding fairness improvement.

Name	Privileged attribute(s)	#Features	Original size	Final size
Adult	Sex, Race	14	48842	45222
Bank	Age	20	41188	30488
Compas	Sex, Race	49	7214	6167
German	Sex, Age	20	1000	1000

5.2.2 Baseline Approaches. To demonstrate the effectiveness of our approach, we compare FMT with four state-of-the-art approaches for fairness improvement, which include pre-processing (i.e., optimizing the training data before model training), in-processing (i.e., optimizing the model training process) and post-processing techniques (i.e., optimizing the learned models). In particular, to investigate whether the simple removal of the privileged attribute for model building can improve the fairness of DL models, we also compare our approach with another baseline method named **SUPP** (i.e., suppressing/ignoring the privileged attribute). The details of the baseline approaches are introduced as follows. In our experiment, we employ the open-source implementation in the *AIF360* toolkit for the first three approaches and the implementation published by the authors of CARE, respectively. Note that we do not compare with the latest NeuronFair [141] because we failed to replicate their results using their open-source implementation.

- **REW** (*Reweighing*) [56] is a pre-processing approach that improves model fairness by pre-computing the weights of training samples in each (group, label) combination to mitigate the effects of bias in training data. The basic idea is to assign higher weights to instances that belong to unprivileged groups and lower weights to instances that belong to privileged groups, which was done by computing the inverse frequency of each group in the training set. Finally, these weights will be used in the training phase to optimize fairness.

Table 7. Basic information of subjects used to evaluate the performance of FMT.

Dataset	Adult-Race	Adult-Sex	Bank-Age	Compas-Race	Compas-Sex	German-Age	German-Sex
Size (KB)	158.1	158.1	127.4	385.4	385.4	128.1	128.1
Parameters	9.1K	9.1K	6.5K	28.5K	28.5K	6.6K	6.6K
Accuracy(%)	85.2	85.2	89.5	65.5	65.5	77.0	77.0
SPD	0.096	0.191	0.086	0.176	0.189	0.071	0.220
AAOD	0.050	0.096	0.028	0.154	0.179	0.023	0.290
EOD	0.051	0.110	0.033	0.107	0.129	0.025	0.175

- **ADV** (*Adversarial Debiasing*) [126] is an in-processing approach that leverages adversarial techniques to reduce the effects of the privileged attribute to the prediction result so as to the change of this attribute will not change the result as expected. In particular, it maintains two independent models, one of which aims at producing fair predictions (named predictor) while the other aims at identifying the sensitive attributes that cause fairness issues in the predictions. By simultaneously competing, the predictor model will be forced to produce fairer predictions and thus reduce bias.
- **ROC** (*Reject Option Classification*) [57] is a post-processing technique that assigns favorable outcomes to unprivileged groups with higher possibility while tends to assign unfavorable outcomes to privileged groups. Specifically, it combines cost-sensitive learning and threshold-moving techniques for minimizing overall classification error but confining fairness constraints by adjusting decision thresholds. In particular, A customizable parameter enables users to balance accuracy and fairness based on their preferences.
- **CARE** [97] is the latest research for improving the model fairness in a post-training way. It leverages causal inference to first identify a set of model parameters that may closely related to the bias on targeted privileged features, and then adopts PSO (Particle Swarm Optimization) algorithm to search the optimal model parameters under the guidance of a fitness function that considers both model accuracy and fairness.
- **SUPP** ignores the privileged attribute during both the model training and testing processes while keeping the other configurations the same as the original model. It aims to investigate whether the removal of the privileged value can improve the fairness of the trained models.

Additionally, to evaluate the performance of the model transformation from our framework and the data augmentation algorithm specifically designed for this task, we further design four variants of FMT for comparison. The details are listed below.

FMT^{-s} removes the model slicing process from FMT and updates all the connection weights between any neurons in the transformed model $\mathcal{M}_T$.

FMT^{-sl} removes the complete model transformation component from FMT, i.e., FMT^{-sl} performs post model training over the original model of $\mathcal{M}$, but employs the same training data obtained by our data augmentation algorithm shown in Algorithm 2.

FMT^{last} denotes the variant that takes the last output layer of the deep learning models as the target layer for loss function design (i.e., $p = 0$ in Formula 8). It is similar to DARE$_{last}$, which computes the losses based on the predicted probability of the target label.

FMT^{rand} is the variant for evaluating the effectiveness of our data augmentation algorithm in FMT, i.e., Algorithm 2. Specifically, FMT^{rand} generates attribute values randomly without considering the distribution of them in the original training data (line 4 in the algorithm).

5.2.3 Fairness Metrics. Following previous work [5, 6, 12, 13, 18, 39, 128], we employ all the three widely-used metrics for measuring the fairness of models, i.e., SPD, AAOD, and EOD, all of which have been integrated into the *AIF360* toolkit. Formally, let A be a privileged (or protected) attribute, and $A = 1$ represents the instance that belongs to the privileged group while $A = 0$ to the unprivileged group. We use Y and $\hat{Y}$ to denote the expected and actual prediction labels, respectively, with 1 as the favorable label and 0 as the unfavorable label. We use P to represent the probability of model prediction. Then, the definitions of the above metrics are defined as follows.

- **SPD** (Statistical Parity Difference) measures the difference of probabilities that the unprivileged and privileged groups receive favorable outcomes:

$$SPD = P[\hat{Y} = 1|A = 0] - P[\hat{Y} = 1|A = 1]. \tag{9}$$

- **AAOD** (Average Absolute Odds Difference) measures the average of absolute difference of false-positive rate and true-positive rate for unprivileged and privileged groups:

$$AAOD = \frac{1}{2}(|P[\hat{Y} = 1|A = 0, Y = 0] - P[\hat{Y} = 1|A = 1, Y = 0]| + |P[\hat{Y} = 1|A = 0, Y = 1] - P[\hat{Y} = 1|A = 1, Y = 1]|). \tag{10}$$

- **EOD** (Equal Opportunity Difference) measures the difference of true-positive rate for unprivileged and privileged groups:

$$EOD = P[\hat{Y} = 1|A = 0, Y = 1] - P[\hat{Y} = 1|A = 1, Y = 1]. \tag{11}$$

According to the definitions of the fairness metrics, the smaller the value of the metric is, the more fair the model will be, i.e., 0 denotes absolutely fair.

5.2.4 Implementation and Configuration. The same as the evaluation of DARE, we also have implemented our approach FMT atop the widely-used deep learning framework Keras and Tensorflow in Python, and conducted this 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 FMT, different from DARE where we perform model slicing by classes, FMT performs model slicing by privileged values. That is, it computes the importance of neurons by feeding two groups of inputs with opposite privileged attribute values, e.g., female *vs* male when considering the attribute of sex. The reason is evident since the target of FMT is to mitigate the effects of the privileged attribute to the model prediction. Therefore, we should identify the most important neurons that are closely correlated to that attribute. Finally, we set the default value of $q\%$ as 80% for model slicing according to a small pilot study. Note that this value is smaller than that (i.e., $q\%$) in DARE. The reason is that the DL models in this study are much smaller than those for DARE (refer to the parameters in Table 1 and Table 7). As a result, to make the model slicing take effects, i.e., identifying actual crucial neurons by filtering sufficient less-relevant ones, the value of $q\%$ should be smaller so that more neurons can be filtered. Furthermore, in our experiment, we perform a grid search for all approaches to obtain the best results of them, where we set the learning rate $r \in$ {1e-5, 1e-4,1e-3} and batch size $d \in$ {16, 32, 64, 128}. Finally, we repeat 10 times of the experiment per each tool for comparison to mitigate the randomness.

5.3 Result Analysis of FMT

5.3.1 [Fairness] RQ5: Overall Effectiveness of FMT. As introduced in Section 5.2, we evaluate the effectiveness of FMT over four datasets involving seven different privileged attributes and compare the results with four state-of-the-art fairness improvement approaches and the simple **SUPP**. Particularly, we employed both the two training data construction strategies for FMT as shown in Algorithm 2. The first set of training data includes only a single one from each paired inputs (i.e., $\mathcal{T}_s$ in line 14), while the second set of training data includes both of the paired inputs (i.e., $\mathcal{T}_d$ in line 15). We call the corresponding implementations of FMT as FMT_s and FMT_d, respectively. The detailed comparison results between our approaches and the baseline approaches are shown in Table 8. We also calculate the *p-values* by leveraging the Wilcoxon signed-rank test at the significance level of 0.05 to investigate whether our approaches statistically differ from the baseline approaches. In particular, We use the **bold** font to highlight the results where our approach and the corresponding baseline approach are statistical different (i.e., *p-value*<0.05). In addition, Figure 6 presents the average relative improvement regarding the three fairness metrics (i.e., SPD, AAOD, and EOD) over the original model among our repeated experiments. That is, a positive value indicates that the method can improve the model fairness, and a negative value indicates that the fairness of the

Table 8. Detailed comparison between our approaches (i.e., FMT_s and FMT_d) and baselines. The value in each cell denotes the average absolute difference (i.e., $|A - B| \times 100$) between our approach (A) and the baseline approach (B) regarding corresponding metrics. Additionally, we use down arrows (↓) to denote that our approach achieved lower values compared with the baseline regarding corresponding metrics, and use up arrows (↑) to denote that our approach achieved higher values. In particular, the arrows of green color (↑↓) represent our approach outperformed the baseline, and the arrows of red color (↓↑) represent our approach underperformed the baseline (Higher Accuracy and lower SPD/AAOD/EOD are preferred). In addition, we use the **bold** font to represent that the results of our approach are statistically different from those of the corresponding baseline under the significance level at 0.05 by using the Wilcoxon signed-rank test [114]. That is, our approach statistically outperforms the baseline if the color of the arrow is green, and vise versa if the color of the arrow is red. For detailed *p-values*, please refer to the open-source repository of our project.

Baseline	Metric	Adult-Race		Adult-Sex		Bank-Age		Compas-Race		Compas-Sex		German-Age		German-Sex	
		FMT_s	FMT_d	FMT_s	FMT_d	FMT_s	FMT_d	FMT_s	FMT_d	FMT_s	FMT_d	FMT_s	FMT_d	FMT_s	FMT_d
REW	Accuracy	**0.51** ↑	0.02 ↑	**0.49** ↑	**0.20** ↑	**0.92** ↑	**0.84** ↑	**1.05** ↑	0.30 ↑	**0.91** ↓	**0.76** ↓	**2.83** ↑	**3.00** ↑	**3.50** ↑	**3.53** ↑
	SPD	**3.16** ↑	**1.44** ↑	**2.19** ↑	**3.26** ↑	**3.82** ↑	**4.00** ↑	**12.44** ↑	**9.47** ↑	0.67 ↑	**3.46** ↑	0.44 ↑	0.20 ↓	**4.29** ↑	**7.14** ↑
	AAOD	0.07 ↑	0.35 ↓	**3.33** ↓	**4.64** ↓	**2.75** ↓	**3.27** ↓	**12.50** ↑	**8.90** ↑	**1.04** ↑	**4.14** ↑	**7.45** ↓	**6.92** ↓	**7.49** ↑	**11.37** ↑
	EOD	**2.65** ↓	**2.30** ↓	**8.55** ↓	**11.99** ↓	**6.08** ↓	**7.43** ↓	**9.27** ↑	**9.94** ↑	0.37 ↓	0.41 ↑	**4.16** ↓	**4.47** ↓	0.48 ↑	1.56 ↑
ADV	Accuracy	**2.06** ↑	**1.57** ↑	**0.32** ↑	0.03 ↑	0.04 ↓	0.12 ↓	**0.62** ↑	0.13 ↓	**0.30** ↓	0.15 ↓	**2.63** ↑	**2.80** ↑	**3.13** ↑	**3.17** ↑
	SPD	**5.62** ↑	**3.90** ↑	**6.22** ↑	**7.29** ↑	2.26 ↑	2.44 ↑	**9.35** ↑	**6.38** ↑	**11.91** ↓	**9.12** ↓	**14.43** ↓	**15.07** ↓	**12.20** ↓	**9.35** ↓
	AAOD	1.08 ↓	1.49 ↓	**7.87** ↓	**9.18** ↓	**4.27** ↓	**4.79** ↓	**8.96** ↑	**5.35** ↑	**12.04** ↓	**8.94** ↓	**15.71** ↓	**15.18** ↓	**12.28** ↓	8.39 ↓
	EOD	**4.97** ↓	**4.63** ↓	**17.41** ↓	**20.84** ↓	**8.00** ↓	**9.35** ↓	**6.77** ↑	**7.44** ↑	**10.52** ↓	**9.73** ↓	**14.39** ↓	**14.69** ↓	**10.54** ↓	**9.46** ↓
ROC	Accuracy	**4.04** ↑	**3.55** ↑	**3.48** ↑	**3.19** ↑	**5.89** ↑	**5.81** ↑	**0.39** ↑	**0.36** ↓	**0.50** ↓	**0.35** ↓	**1.93** ↑	**2.10** ↑	**1.10** ↑	**1.13** ↑
	SPD	**8.02** ↓	**9.75** ↓	**17.18** ↓	**16.11** ↓	**13.57** ↓	**13.39** ↓	**2.34** ↑	0.63 ↓	**7.18** ↓	**4.39** ↓	**0.40** ↓	**1.04** ↓	**0.65** ↑	**3.50** ↑
	AAOD	**5.64** ↓	**6.06** ↓	**9.87** ↓	**11.18** ↓	**8.48** ↓	**9.00** ↓	**2.66** ↑	0.95 ↓	**7.46** ↓	**4.36** ↓	**2.93** ↓	**2.40** ↓	**0.18** ↑	**4.06** ↑
	EOD	**3.60** ↓	**3.26** ↓	**5.04** ↓	**8.48** ↓	**4.73** ↓	**6.08** ↓	**0.32** ↑	0.99 ↑	**4.60** ↓	**3.82** ↓	**2.71** ↓	**3.02** ↓	**1.77** ↑	**2.85** ↑
CARE	Acc.	**0.69** ↑	0.20 ↑	**0.55** ↑	**0.26** ↑	**2.03** ↑	**1.94** ↑	**1.38** ↑	**0.63** ↑	0.20 ↑	**0.35** ↑	**1.81** ↑	**1.98** ↑	**2.18** ↑	**2.22** ↑
	SPD	**2.25** ↑	**0.53** ↑	**4.08** ↑	**5.15** ↑	**3.10** ↑	**3.28** ↑	**10.18** ↑	**7.21** ↑	**0.91** ↓	**1.88** ↑	**1.46** ↑	**0.82** ↑	**8.23** ↑	**11.08** ↑
	AAOD	**0.43** ↑	0.02 ↑	**0.74** ↓	**2.06** ↓	**0.30** ↑	**0.22** ↓	**9.71** ↑	**6.10** ↑	**1.39** ↓	**1.71** ↑	**0.26** ↑	**0.79** ↑	**12.31** ↑	**16.19** ↑
	EOD	**1.29** ↓	**0.95** ↓	0.29 ↓	**3.72** ↓	**0.44** ↑	**0.92** ↓	**6.49** ↑	**7.16** ↑	**1.03** ↓	0.24 ↓	**1.66** ↑	**1.35** ↑	**4.32** ↑	**5.40** ↑
SUPP	Acc.	**0.73** ↑	0.24 ↑	0.11 ↓	**0.40** ↓	0.03 ↑	0.06 ↓	**0.91** ↑	0.16 ↑	0.09 ↑	0.24 ↑	**4.23** ↑	**4.40** ↑	**4.63** ↑	**4.67** ↑
	SPD	**1.19** ↓	**2.92** ↓	**6.97** ↓	**5.90** ↓	**1.18** ↓	**1.01** ↓	**1.38** ↑	**1.59** ↓	**14.36** ↓	**11.57** ↓	**2.21** ↓	**2.84** ↓	2.16 ↓	0.69 ↑
	AAOD	**2.14** ↓	**2.55** ↓	**4.56** ↓	**5.87** ↓	**1.68** ↓	**2.20** ↓	**1.50** ↑	**2.11** ↓	**14.57** ↓	**11.46** ↓	**5.14** ↓	**4.61** ↓	1.31 ↓	**2.57** ↑
	EOD	**4.01** ↓	**3.67** ↓	**4.57** ↓	**8.00** ↓	**1.17** ↓	**2.53** ↓	0.26 ↓	0.40 ↑	**12.79** ↓	**12.01** ↓	**2.20** ↓	**2.51** ↓	1.93 ↓	0.86 ↓

model after applying the corresponding approaches drops. From the table and the figure, we can see that overall our approaches slightly outperform the baseline approaches REW, ADV and ROC while being slightly worse than CARE. Specifically, compared with the first three baselines, FMT_s and FMT_d respectively achieve on average 3.7% and 6.4% improvement in terms of the three fairness metrics. Additionally, FMT_s and FMT_d perform consistently well over the datasets of Adult-Race, Adult-Sex, Compas-Sex, and German-Age, while performing less effectively over the other datasets, i.e., Bank-Age, Compas-Race and German-Sex. From the table and the figure we can also see that the performance of FMT_s and FMT_d is very close to each other, indicating that our framework improves the fairness of DL models indeed by strengthening the memory of crucial features but not simply balancing the number of training inputs of opposite privileged attribute values ($\mathcal{T}_d$ is balanced while $\mathcal{T}_s$ is not). In addition, the unsatisfactory results of SUPP also demonstrate that simply removing the privileged value for model building does not help much for improving the fairness of the trained models. The reason is that the bias may also be implicitly introduced by other attributes even though the privileged one is removed since different attributes may have inherent correlations, which has been proved by prior studies [68]. Particularly, CARE slightly outperforms FMT_s by on average 12.5% but is slightly worse than FMT_d by on average 23.0% over all the datasets. However, as it will be shown in Figure 8, CARE improves fairness by significantly decreasing the accuracy of the models.

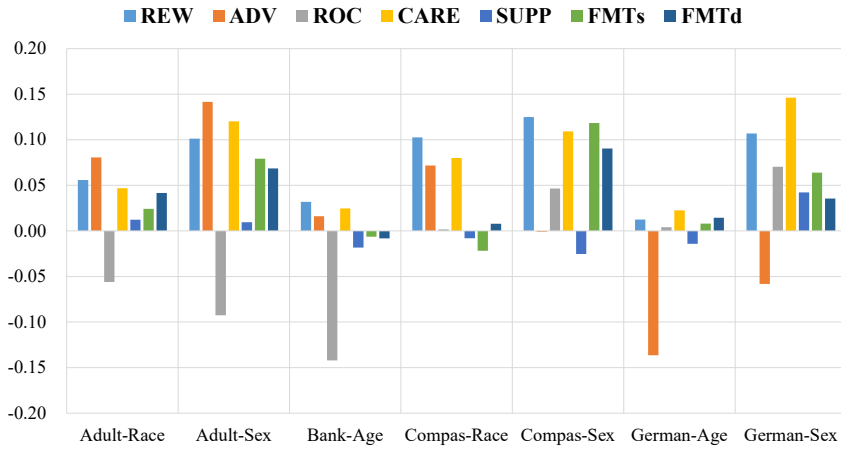
When comparing the results in terms of different fairness metrics, we can see that our approaches are much less effective regarding SPD, which reflects the tendency that the model produces favorable

outcomes for unprivileged groups over the privileged groups (see Formula 9). We argue that this metric is less representative compared with the other two metrics since it only considers the model output but ignores the correctness of the output. As we will present later, most of the baseline approaches improve the fairness of the model by largely sacrificing the model accuracy (see Figure 8). On the contrary, our approaches perform much better than baseline approaches in most cases when comparing the results of AAOD and EOD, both of which take the model accuracy factor into consideration (see Formulas 10 and 11). Specifically, when only comparing AAOD and EOD, FMT_s achieves on overage 14.16% and 12.87%, and FMT_d achieves on averag 16.57% and 19.35% improvement compared with all the baselines over all the datasets. Therefore, although our approaches do not completely outperform the baseline approaches over all datasets regarding all metrics, the overall performance of our approaches is relatively better. Figure 7 shows the value distribution of fairness metrics for different approaches in the repeated experiment (we omit ROC in the figure because its output is deterministic and free from randomness). Note that the smaller value in the figure indicates better results. From the figure, we can see that both our approaches (i.e., FMT_s and FMT_d) and CARE perform more stable than the others, while the performance of our approaches and CARE is comparable, indicating the effectiveness of our approaches even though our framework is not specifically designed for improving model fairness.

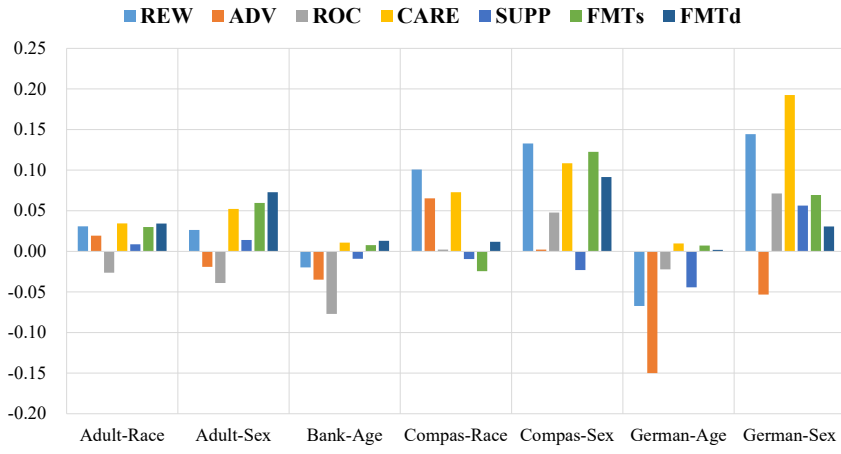
As mentioned above, the fairness improvement should not largely sacrifice the performance of the original model, we further present the results regarding the model accuracy. Figure 8 shows the improvement of accuracy after applying different fairness improvement approaches. The negative value denotes the decline in accuracy. Similarly, Figure 9 shows the distribution of accuracy after improving in our repeated experiment. From the figure, we can see that our approaches FMT_s and FMT_d significantly outperform the baselines as they can still preserve the model accuracy as much as possible after improving the fairness, while the baseline approaches tend to decrease the accuracy. For example, the model accuracy will drop more than 5.75% after applying ROC on the dataset of Bank-Age. Specifically, FMT_s and FMT_d outperform the baseline approaches with higher accuracy over six out of seven datasets (i.e., except Compas-Sex), and the average improvement compared with the baseline approaches is 1.96% and 1.72%, respectively for FMT_s and FMT_d over different datasets. It is worth mentioning that although CARE achieved good fairness performance, it tends to sacrifice accuracy. Its average decline of accuracy is about 1.5%. The results demonstrate that our approaches are relatively stable.

To sum up, although our post-training framework is not specifically designed for improving the fairness of DL models, it is indeed effective in this task according to the experimental results of FMT_s and FMT_d since they improve the fairness without largely sacrificing the model accuracy. Particularly, the performance of our approaches is relatively stable, indicating the reliability of our approaches for practical use.

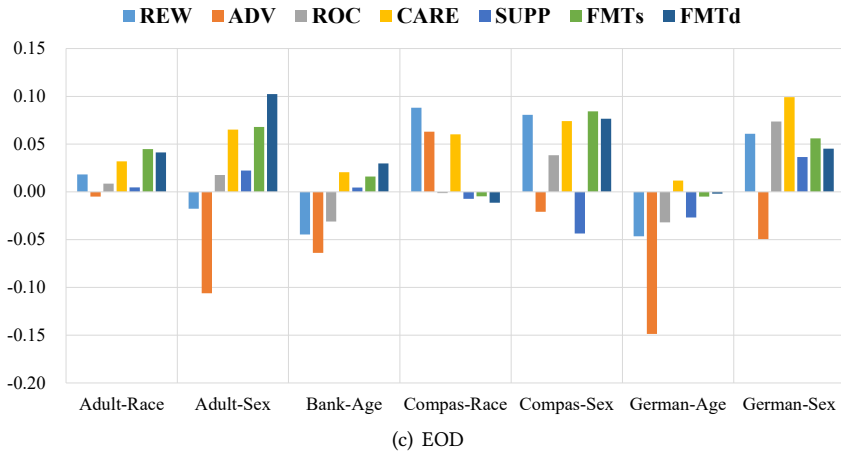
5.3.2 [Fairness] RQ6: Contribution of Model Transformation. In order to investigate how much the model transformation in our framework contributes to the effectiveness of FMT, we conducted an ablation study with the variants of FMT^{-s}, FMT^{-sl} and FMT^{last}, which have been introduced in Section 5.2.2. Specifically, we present the relative improvement over all the datasets for each metric compared with the original FMT in Figure 10 and Figure 11. Similar to the last research question, we conducted this experiment under two different data augmentation strategies, i.e., FMT_s and FMT_d. From the figures, we can conclude that the model transformation component in our framework largely contributed to the effectiveness of FMT (i.e., FMT_s and FMT_d) since the removal of it significantly reduces the performance of both accuracy and fairness. Specifically, compared with FMT_s and FMT_d, the accuracy of ${\text{FMT}_s}^{-s}$ and ${\text{FMT}_d}^{-s}$ drops by 0.96% and 0.33% respectively while the accuracy of ${\text{FMT}_s}^{-sl}$ and ${\text{FMT}_d}^{-sl}$ drops on average by 0.40%. Similarly, the



(a) SPD



(b) AAOD



(c) EOD

Fig. 6. Fairness improvement after applying different approaches.

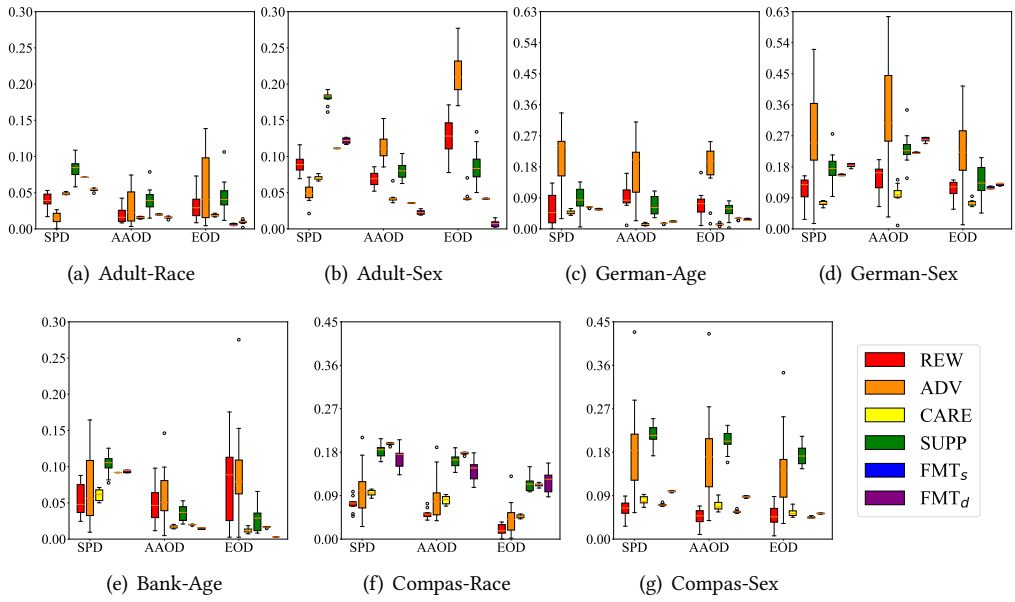


Fig. 7. Value distribution regarding different fairness metrics after applying different approaches.

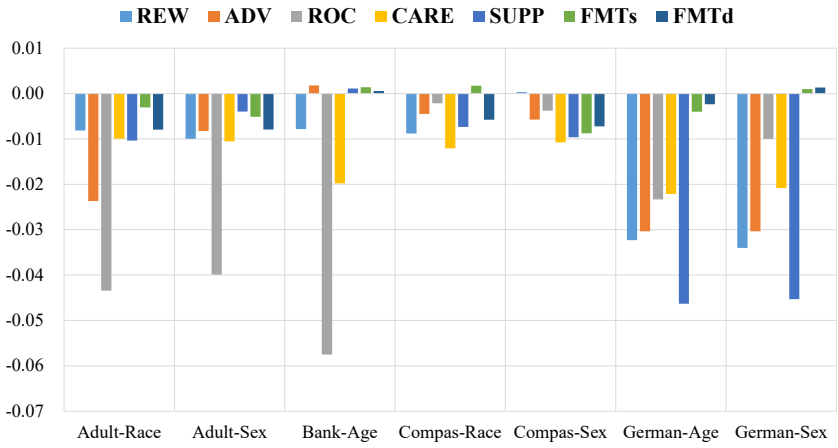
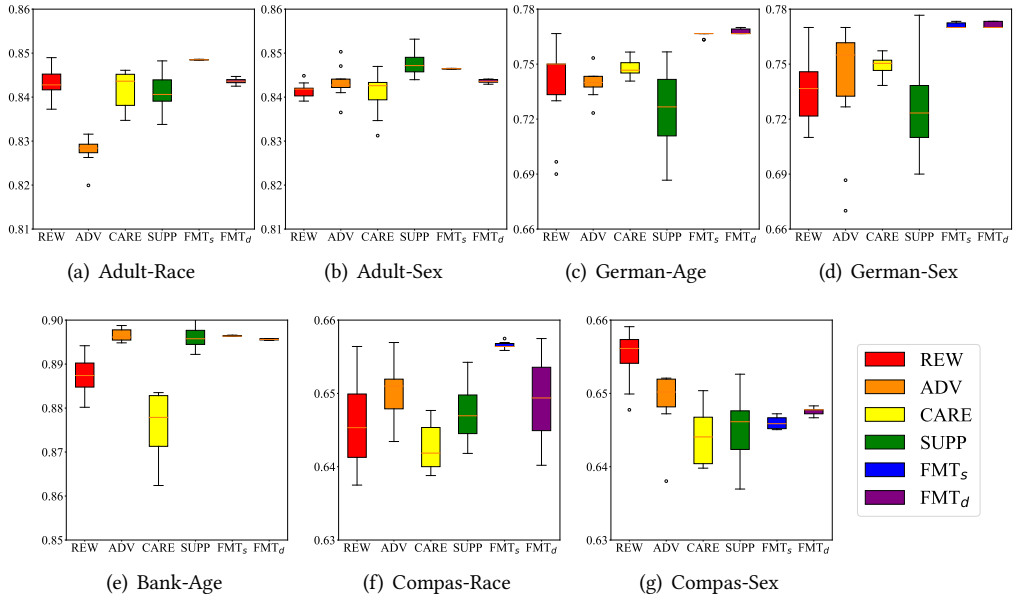


Fig. 8. Accuracy improvement after applying different approaches.

decline of fairness can be up to 101.94% and 99.27% respectively when removing the model slicing and complete model transformation component. In other words, the components in our framework tend to contribute more to the fairness than the accuracy of the models. It is reasonable because our data augmentation algorithm constructs the training data based on the model itself, and thus it is expected that the constructed data should be unlikely to affect the accuracy of the models regardless of the training process. Similarly, compared with FMT$_s$ and FMT$_d$, the fairness of FMT$_s{}^{last}$ and FMT$_d{}^{last}$ also suffer from a large decline, i.e., on average up to 23.32% and 16.11% respectively. However, both FMT$_s{}^{last}$ and FMT$_d{}^{last}$ can slightly improve the model accuracy (about 0.24%). The



(a) Adult-Race (b) Adult-Sex (c) German-Age (d) German-Sex

(e) Bank-Age (f) Compas-Race (g) Compas-Sex

Fig. 9. Accuracy distribution after applying different approaches.

reason is that the output of the last layer directly decides the correctness of the predictions, making the training more targeted and guiding the improvement of accuracy. In summary, the model transformation component in our framework significantly contributed to the effectiveness of FMT for improving the fairness of DL models.

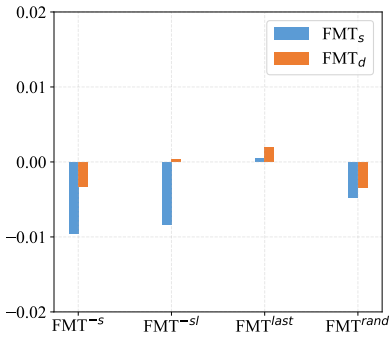


Fig. 10. Relative improvement of accuracy.

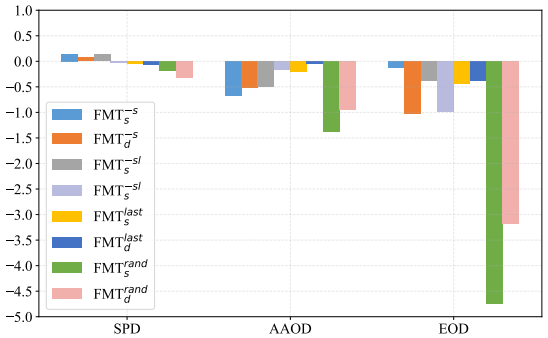


Fig. 11. Relative improvement of fairness.

5.3.3 [Fairness] RQ7: Contribution of Data Augmentation in FMT. To investigate the performance of our specifically designed data augmentation algorithm (i.e., Algorithm 2), we compare the results of FMT with the variant of FMT^{rand}, whose results also are presented in Figures 10 and 11. From the figures, we can see that if we ignore the distribution of the attributes in the training data during training data augmentation, the performance of FMT will be significantly affected. Specifically, our data augmentation algorithm in FMT$_s$ and FMT$_d$ contributed on average 0.41% higher model

Table 9. Time cost of each method for improving model fairness (in seconds).

App.	Adult-Race	Adult-Sex	Bank-Age	Compas-Race	Compas-Sex	German-Age	German-Sex	Average
REW	**68.8**	235.3	152.9	**22.9**	67.9	7.2	34.3	84.2
ADV	116.4	209.2	**27.2**	39.5	**13.4**	**1.9**	**5.4**	**59.0**
ROC	150.6	**149.8**	77.8	33.2	28.6	7.3	7.4	65.0
CARE	395.5	282.3	586.8	426.7	375.3	343.6	108.0	359.7
FMT	216.4	343.3	83.9	57.3	28.9	5.7	11.0	106.6

accuracy and 179.73% higher fairness (i.e., smaller fairness values) in terms of the three fairness metrics, demonstrating the contribution and effectiveness of the algorithm in FMT. Similarly, the reason that FMT^{rand} did not largely affect the model accuracy is that the training data is constructed based on the model itself (i.e., the label is determined by the original model).

5.3.4 [Fairness] RQ8: Efficiency of FMT. Finally, we explore the efficiency of FMT and the baseline approaches. Table 9 presents the time cost of each approach for improving the fairness of DL models. Please note that **SUPP** introduces no extra execution overhead except for the normal model training process, and thus we omit the time cost of it in the Table. Particularly, we use the time of FMT_d to represent that of FMT since it doubles the training inputs compared with FMT_s and thus will consume more time. In the table, we highlight the results of the highest efficiency. From the table, we can observe that FMT is less efficient than the first three baseline approaches, i.e., REW, ADV and ROC, while is much more efficient than the latest CARE. The reason is that CARE depends on the PSO search algorithm, which tends to spend more time compared with the back propagation optimization. However, the reasons for the low efficiency of FMT are twofold. First, FMT incorporates the model slicing process to identify the crucial neurons, which is costly. Second, since FMT depends on a finer-grained loss function in the transformed regression model for optimization, the training process also tends to be less efficient compared with the original classification model. Nevertheless, the overall time cost of all approaches is relatively low, i.e., less than six minutes on average. It can be acceptable in practice, especially for those approaches that work offline, like FMT and CARE.

6 RELATED WORK

6.1 Model Repair

Like traditional software [48–50, 66, 111, 112, 117], deep learning (DL) programs also have bugs. Particularly, besides those bugs that are caused by vulnerable source code [42, 43, 46, 71, 78, 98, 135–137]. DL systems have a special type of bugs, called *model bugs* [70], 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 [129]. In order to repair model bugs, many approaches have been proposed [9, 29, 58, 65, 85, 94, 103, 106] and the typical method is optimizing the training data, such as performing data selection [27] or data augmentation [70]. For example, Fahmy et al. [26] proposed to leverage heatmaps to capture the relevance of neurons, and then retained the model for accuracy improvement. Similarly, Ma et al. [70] employed an analogical heatmap to aid the selection of retraining data and proposed MODE. Analogously, Yu et al. [122] 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. Sohn et al. [94] proposed a search-based repair approach named Arachne, which utilized the gradient loss and the forward impact to localize the weights that are responsible for the misclassification behaviors but less related to correct behaviors for fixing. Based on Arachne,

Tokui et al. [103] proposed NEURECOVER, which incorporated the difference between the past and the current model to guide fixing regression faults of deep learning systems. Zhang et al. [127] proposed APRICOT, which divided the training set into subsets to train sub-models, and then dynamically adjusted the weights of the model according to the weights of sub-models. Recently, Usman et al. [106] proposed NNRepair, which leveraged constraint solving for neural network repair. Their experiments demonstrated that NNRepair could be used for improving both model robustness and accuracy. Henriksen et al. [37] proposed I-REPAIR to improve the accuracy of the model with only limited data for fixing. Specifically, I-REPAIR utilizes gradients of each parameter in the deep learning model to measure its influence on the misclassification and then performs back-propagation to adjust the parameters for fixing. Sotoudeh et al. [95] proposed a model structure named Decoupled DNNs to facilitate repairing each layer of deep learning models. Wu et al. [115] proposed GenMuNN to repair models by directly mutating the model weights guided by a genetic algorithm.

However, as studied by Pham et al. [84] 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 proposing a general post-training framework for improving the performance of DL models from different aspects, such as the robustness and fairness. Indeed, our approach can also be used to improve the accuracy of DL models potentially, we leave it as our future work.

6.2 Robustness Improvement

As introduced in the introduction, DL models are fragile in the face of adversarial attacks [81]. In order to improve the adversarial robustness of DL models, many approaches have been put forward in the last decade, which can be classified into two distinct categories: model retraining and adversarial sample detection, where the former is more widely explored. Existing studies [28, 41] 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 [28, 72, 75, 89, 92]. One of the key differences among this kind of approaches lies in the strategies adopted for data augmentation. For example, Zantedeschi et al. [123] proposed to augment training data with examples perturbed using Gaussian noise; Tramèr et al. [104] introduced a technique that augments training data with perturbations transferred from other models; Engstrom et al. [24] proposed a data augmentation method by robust optimization and test-time input aggregation; While Gao et al. [27] proposed to a mutation-based fuzzing technique for such a purpose. Besides simply augmenting or filtering training data, Papernot et al. [82] and Xu et al. [118] 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. [142] 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. [140] proposed to detect adversarial examples based on a predefined *robustness* difference of input examples. Zhang et al. [139] 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. [63] 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. [72] 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. [120] 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.

6.3 Fairness Improvement

Besides the robustness issue, due to the wide application of DL models in practice, especially in ethically sensitive areas, the fairness of DL models has become a critically important property, which can easily cause performance bugs of DL models by producing unfair prediction results. Recently, many approaches have also been proposed with the aim of improving DL models' fairness, including pre-processing, in-processing and post-processing methods, such as the baseline approaches introduced in Section 5.2.2. In the following, we briefly introduce the related research.

The *pre-processing* methods aim at reducing the bias in the training data before model training so as to the learned model can produce fair predictions. For example, besides the Reweighing (REW) [56] compared in our evaluation, Burnaev et al. [8] proposed to leverage imbalance ratios to measure the training data, based on which they optimize the training data by removing or adding new data. Similarly, Cui et al. [20] proposed a data sampling method, which balances different classes in the training data by resampling. Additionally, Chawla et al. [14] proposed SMOTE, which aims at debiasing the training data by generating new data for minority classes of inputs. Different from them, Li et al. [68] proposed to identify the biased features of inputs and remove them during the training of the model, based on which they proposed LTDD. As for the in-processing methods, they modify the models and optimize training algorithms in different ways to mitigate the bias in the model predictions, such as the method of Adversarial Debiasing (ADV) [126] in our evaluation. Additionally, Du et al. [23] proposed CREX, which leverages the model regularization to reduce the effects of privileged (or sensitive) attributes to the final prediction results. Most recently, Chen et al. [18] proposed an ensemble method MAAT, which ensembles an accuracy-based and another fairness-based model. The results show it can effectively trade off the accuracy and fairness. However, different from them, our approach improves the model fairness by post training, which is orthogonal to the above approaches and can be further combined with them.

The *post-processing* methods aim at optimizing the learned models for fairness improvement, e.g., Reject Option Classification (ROC) [57] that has been compared in our evaluation. This category is the most similar approach to ours since FMT improve fairness by post model training. The most latest approach NeuronFair was proposed by Zheng et al. [141], which aims at generating more diverse training data for model optimization according to the interpretability of DL models. Additionally, Sun et al. [97] proposed CARE, which employs PSO (Particle Swarm Optimization) algorithm to optimize the neurons identified by causality analysis under the guidance of a novel fitness function. In our evaluation, we also have compared our approach with it. Different from these approaches that fine tune the learned models without changing the structure of the model, our approach transforms the classification model into an isomorphic regression model for optimization, and the results also demonstrate the stably good performance of our approaches.

Table 10. Performance comparison between DARE and Knowledge Distillation regarding Empirical Robustness (%) when attacking by JSMA and PGD. The model accuracy before and after tuning is also presented.

Model	DataSet	JSMA		PGD		Original Model Acc.		Model Acc. After Tuning	
		Dare	Distill.	Dare	Distill.	Dare	Distill.	Dare	Distill.
VGG16	CIFAR10	84.9	82.7	83.7	79.4	88.7	88.3	88.9	85.9
VGG19		88.5	85.6	89.8	88.3	90.6	90.4	90.4	87.5
Alexnet		62	70.6	82.1	76.9	83	80.1	83.6	79.1

7 DISCUSSION

7.1 Comparison with Knowledge Distillation

Knowledge distillation [38, 59, 87] was originally proposed for reducing the size of deep learning models so that less storage and computing resource will be required for their practical use, e.g., on portable devices like mobile phones [30]. The basic idea of knowledge distillation is to train a lightweight model under the guidance of a well-trained model, where the output probability vectors of the well-trained model can play as the reference for model training. Although our approach is similar to knowledge distillation since they both change the discrete classification labels into finer-grained continuous values, they are still different in several aspects: (1) Knowledge distillation is *model-centric* but our approach is *data-centric*. In other words, the former selects a constant (i.e., the original) model as the training reference. On the contrary, ours aims at finding inputs that can produce a higher prediction probability (i.e., confidence), while the reference models can vary. (2) Our approach incorporates model slicing to identify the crucial neurons for fine-tuning, while knowledge distillation does not. Based on the above discussion, we believe that our approach is significantly different from knowledge distillation.

Additionally, to compare the performance of knowledge distillation and our approach, we further conducted an experimental study. In particular, we compared our approach with knowledge distillation only on the application of robustness improvement since existing distillation-based fairness improvement methods all target image classification tasks [53] and cannot deal with our tabular data. Specifically, we adopted the latest robustness improvement method proposed by Papernot et al. [83] as the representative baseline. Moreover, we used the dataset CIFAR10 and set the attacking method as JSMA for a fair comparison, which was also used by Papernot et al. [83]. Besides, we also used PGD for attacking because it was effective based on the results shown in Table 2. In the experiment, we adopted the same configurations as Papernot et al. [83] did and set the temperature $T = 10$ to balance the effectiveness and efficiency since larger T will cause unaffordable time cost (Distillation took ~9 hours on average per each model in our experiment).

The experimental results are shown in Table 10. The results show that our approach slightly outperformed the distillation method by 1.3% on average regarding Empirical Robustness. However, our approach can better preserve the model accuracy after fine-tuning. In summary, the results further demonstrate that our approach is effective.

7.2 Limitation and Future Work

Limitation: Though our framework was proved to be effective for improving both the robustness and fairness of DL models, it still has limitations. First, DARE is designed for classification models, while it does not conform to regression models due to its underlying model transformation process. However, the data augmentation algorithms 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 [139]. Second, as shown in our experimental results, our framework 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, our framework still outperforms the baselines.

Future Work: In this work, we have evaluated the generalizability of our framework in two emerging tasks, i.e., robustness and fairness improvement. The reason is that they are critically important in practice and have been widely studied by existing research. However, according to the introduction of our framework in Section 3, our framework is general and not designed for specific tasks. In the future, we plan to study its effectiveness in more application scenarios, e.g., accuracy improvement and fairness improvement on image classification models. However, designing effective data augmentation algorithms is still challenging, especially for complex inputs. For example, in the task of improving the fairness in image classification, the data augmentation algorithm in FMT will not apply. In such cases, more advanced data augmentation algorithms may be required and should be developed.

8 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 adapted our framework to two distinct performance improvement tasks, i.e., robustness and fairness improvement. Additionally, we also employed diverse network architectures and 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 are different in multiple aspects, such as different image sizes, different color modes (colored *and* grayscale), and different scenarios (numbers *and* objects) in the evaluation of robustness task, while in the fairness task, the privileged attributes are also different from diverse applications (e.g., *sex* in income prediction and *age* in credit prediction). Consequently, we believe the results are representative. Regarding the experiment, in order to obtain the best performance of baseline approaches, we have conducted an extensive model-tuning process. Finally, our experimental data is publicly accessible for replication and for promoting future studies in this research area.

9 CONCLUSION

In this paper, we have proposed a novel post model training framework that incorporates a novel model transformation process. Specifically, it transforms a classification model into an isomorphic regression model for performance improvement, which can effectively perceive input perturbations for suppression and effectively consolidate the memory of crucial input features. To evaluate the effectiveness of our framework, we have adapted it into two emerging DL tasks by proposing two data augmentation algorithms for training data construction. We have implemented the corresponding approaches in tools DARE and FMT respectively for improving the robustness and fairness of DL models, and conducted extensive studies by comparing them with existing state-of-the-art approaches. The results demonstrate that our framework is indeed effective and general as it can be applied to improve the performance of DL models from different perspectives. In particular, the stable performance of our approaches also reflects the high reliability of our framework for practical use.

ACKNOWLEDGMENTS

We thank the anonymous reviewers for their constructive suggestions to help improve the quality of this paper. This work was supported by the National Natural Science Foundation of China under Grant Nos. 62202324 and 62322208.

REFERENCES

[1] Evan Ackerman. Accessed: 2021. News. https://spectrum.ieee.org/three-small-stickers-on-road-can-steer-tesla-autopilot-into-oncoming-lane

[2] Yang Bai, Xin Yan, Yong Jiang, Shu-Tao Xia, and Yisen Wang. 2021. Clustering Effect of Adversarial Robust Models. In *NeurIPS*. 29590–29601.

[3] 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.

[4] Rachel K. E. Bellamy, Kuntal Dey, Michael Hind, Samuel C. Hoffman, Stephanie Houde, Kalapriya Kannan, Pranay Lohia, Jacquelyn Martino, Sameep Mehta, Aleksandra Mojsilovic, Seema Nagar, Karthikeyan Natesan Ramamurthy, John T. Richards, Diptikalyan Saha, Prasanna Sattigeri, Moninder Singh, Kush R. Varshney, and Yunfeng Zhang. 2019. AI Fairness 360: An extensible toolkit for detecting and mitigating algorithmic bias. *IBM J. Res. Dev.* 63, 4/5 (2019), 4:1–4:15.

[5] Sumon Biswas and Hridesh Rajan. 2020. Do the machine learning models on a crowd sourced platform exhibit bias? an empirical study on model fairness. In *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*, Prem Devanbu, Myra B. Cohen, and Thomas Zimmermann (Eds.). ACM, 642–653.

[6] Sumon Biswas and Hridesh Rajan. 2021. Fair preprocessing: towards understanding compositional fairness of data transformers in machine learning pipeline. In *ESEC/FSE '21: 29th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, Athens, Greece, August 23-28, 2021*, Diomidis Spinellis, Georgios Gousios, Marsha Chechik, and Massimiliano Di Penta (Eds.). ACM, 981–993.

[7] Miranda Bogen and Aaron Rieke. 2018. Help wanted: An examination of hiring algorithms, equity, and bias. *Technical report* (2018).

[8] Evgeny Burnaev, Pavel Erofeev, and Artem Papanov. 2015. Influence of resampling on accuracy of imbalanced classification. In *Eighth International Conference on Machine Vision, ICMV 2015, Barcelona, Spain, 19-20 November 2015 (SPIE Proceedings, Vol. 9875)*, Antanas Verikas, Petia Radeva, and Dmitry P. Nikolaev (Eds.). SPIE, 987521.

[9] Davide Li Calsi, Matias Duran, Thomas Laurent, Xiao-Yi Zhang, Paolo Arcaini, and Fuyuki Ishikawa. 2023. Adaptive Search-based Repair of Deep Neural Networks. In *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2023, Lisbon, Portugal, July 15-19, 2023*, Sara Silva and Luís Paquete (Eds.). ACM, 1527–1536.

[10] Nicholas Carlini, Anish Athalye, Nicolas Papernot, Wieland Brendel, Jonas Rauber, Dimitris Tsipras, Ian J. Goodfellow, Aleksander Madry, and Alexey Kurakin. 2019. On Evaluating Adversarial Robustness. *CoRR* abs/1902.06705 (2019).

[11] Nicholas Carlini and David A. Wagner. 2017. Towards Evaluating the Robustness of Neural Networks. *2017 IEEE Symposium on Security and Privacy, SP*, 39–57.

[12] Joymallya Chakraborty, Suvodeep Majumder, and Tim Menzies. 2021. Bias in machine learning software: why? how? what to do?. In *ESEC/FSE '21: 29th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, Athens, Greece, August 23-28, 2021*, Diomidis Spinellis, Georgios Gousios, Marsha Chechik, and Massimiliano Di Penta (Eds.). ACM, 429–440.

[13] Joymallya Chakraborty, Suvodeep Majumder, Zhe Yu, and Tim Menzies. 2020. Fairway: a way to build fair ML software. In *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*, Prem Devanbu, Myra B. Cohen, and Thomas Zimmermann (Eds.). ACM, 654–665.

[14] Nitesh V. Chawla, Kevin W. Bowyer, Lawrence O. Hall, and W. Philip Kegelmeyer. 2002. SMOTE: Synthetic Minority Over-sampling Technique. *J. Artif. Intell. Res.* 16 (2002), 321–357. https://doi.org/10.1613/jair.953

[15] 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.

[16] 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.

[17] 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.

[18] Zhenpeng Chen, Jie M. Zhang, Federica Sarro, and Mark Harman. 2022. MAAT: a novel ensemble approach to addressing fairness and performance bugs for machine learning software. In *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/FSE 2022, Singapore, Singapore, November 14-18, 2022*, Abhik Roychoudhury, Cristian Cadar, and Miryung Kim (Eds.). ACM, 1122–1134.

[19] 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.

[20] Yin Cui, Menglin Jia, Tsung-Yi Lin, Yang Song, and Serge J. Belongie. 2019. Class-Balanced Loss Based on Effective Number of Samples. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019, Long Beach, CA, USA, June 16-20, 2019*. Computer Vision Foundation / IEEE, 9268–9277.

[21] Hao Dong, Akara Supratak, Luo Mai, Fangde Liu, Axel Oehmichen, Simiao Yu, and Yike Guo. 2017. TensorLayer: A Versatile Library for Efficient Deep Learning Development. In *ACM Multimedia*. ACM, 1201–1204.

[22] Li Dong, Nan Yang, Wenhui Wang, Furu Wei, Xiaodong Liu, Yu Wang, Jianfeng Gao, Ming Zhou, and Hsiao-Wuen Hon. 2019. Unified Language Model Pre-training for Natural Language Understanding and Generation. In *NeurIPS*. 13042–13054.

[23] Mengnan Du, Ninghao Liu, Fan Yang, and Xia Hu. 2019. Learning Credible Deep Neural Networks with Rationale Regularization. In *2019 IEEE International Conference on Data Mining, ICDM 2019, Beijing, China, November 8-11, 2019*, Jianyong Wang, Kyuseok Shim, and Xindong Wu (Eds.). IEEE, 150–159.

[24] 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.

[25] 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.

[26] 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).

[27] 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.

[28] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. 2015. Explaining and Harnessing Adversarial Examples. *3rd International Conference on Learning Representations, ICLR 2015*.

[29] 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.

[30] Jianping Gou, Baosheng Yu, Stephen J. Maybank, and Dacheng Tao. 2021. Knowledge Distillation: A Survey. *Int. J. Comput. Vis.* 129, 6 (2021), 1789–1819.

[31] Sumit Gulwani. 2011. Automating string processing in spreadsheets using input-output examples. *ACM Sigplan Notices* 46, 1 (2011), 317–330.

[32] Sumit Gulwani. 2016. Programming by Examples: Applications, Algorithms, and Ambiguity Resolution. In *IJCAR*. 9–14.

[33] Sumit Gulwani and Prateek Jain. 2017. Programming by Examples: PL meets ML. In *APLAS*.

[34] Sumit Gulwani, Oleksandr Polozov, and Rishabh Singh. 2017. Program Synthesis. *Foundations and Trends in Programming Languages* 4, 1-2 (2017), 1–119.

[35] Daniel Conrad Halbert. 1984. *Programming by example*. Ph. D. Dissertation. University of California, Berkeley.

[36] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep Residual Learning for Image Recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*. IEEE Computer Society, 770–778.

[37] Patrick Henriksen, Francesco Leofante, and Alessio Lomuscio. 2022. Repairing misclassifications in neural networks using limited data. In *SAC*. ACM, 1031–1038.

[38] Geoffrey E. Hinton, Oriol Vinyals, and Jeffrey Dean. 2015. Distilling the Knowledge in a Neural Network. *CoRR* abs/1503.02531 (2015).

[39] Max Hort, Jie M. Zhang, Federica Sarro, and Mark Harman. 2021. Fairea: a model behaviour mutation approach to benchmarking bias mitigation methods. In *ESEC/FSE '21: 29th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, Athens, Greece, August 23-28, 2021*. ACM, 994–1006.

[40] 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.
[41] Ruitong Huang, Bing Xu, Dale Schuurmans, and Csaba Szepesvári. 2015. Learning with a strong adversary. *arXiv preprint arXiv:1511.03034* (2015).
[42] 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.
[43] 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.
[44] Adam Ivankay, Ivan Girardi, Chiara Marchiori, and Pascal Frossard. 2020. FAR: A General Framework for Attributional Robustness. *CoRR* abs/2010.07393 (2020).
[45] 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.
[46] 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.
[47] 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
[48] 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
[49] 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
[50] Jiajun Jiang, Yingfei Xiong, Hongyu Zhang, Qing Gao, and Xiangqun Chen. 2018. Shaping Program Repair Space with Existing Patches and Similar Code. In *ISSTA*.
[51] 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.
[52] 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.
[53] Sangwon Jung, Donggyu Lee, Taeeon Park, and Taesup Moon. 2021. Fair Feature Distillation for Visual Recognition. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2021, virtual, June 19-25, 2021*. Computer Vision Foundation / IEEE, 12115–12124.
[54] 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.
[55] 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)*.
[56] Faisal Kamiran and Toon Calders. 2011. Data preprocessing techniques for classification without discrimination. *Knowl. Inf. Syst.* 33, 1 (2011), 1–33.
[57] Faisal Kamiran, Asim Karim, and Xiangliang Zhang. 2012. Decision Theory for Discrimination-Aware Classification. In *12th IEEE International Conference on Data Mining, ICDM 2012, Brussels, Belgium, December 10-13, 2012*, Mohammed Javeed Zaki, Arno Siebes, Jeffrey Xu Yu, Bart Goethals, Geoffrey I. Webb, and Xindong Wu (Eds.). IEEE Computer Society, 924–929.
[58] 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.
[59] Kyungyul Kim, Byeongmoon Ji, Doyoung Yoon, and Sangheum Hwang. 2021. Self-Knowledge Distillation with Progressive Refinement of Targets. In *ICCV*. IEEE, 6547–6556.
[60] Emanuel Kitzelmann. 2009. Inductive programming: A survey of program synthesis techniques. In *International Workshop on Approaches and Applications of Inductive Programming*. Springer, 50–73.
[61] Alex Krizhevsky. 2009. *Learning multiple layers of features from tiny images*. Technical Report.
[62] 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.
[63] Alexey Kurakin, Ian Goodfellow, and Samy Bengio. 2017. Adversarial machine learning at scale. In *International Conference on Learning Representations*.

[64] Alexey Kurakin, Ian J. Goodfellow, and Samy Bengio. 2017. Adversarial examples in the physical world. *5th International Conference on Learning Representations, ICLR 2017*.

[65] 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.

[66] 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

[67] 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.

[68] Yanhui Li, Linghan Meng, Lin Chen, Li Yu, Di Wu, Yuming Zhou, and Baowen Xu. 2022. Training Data Debugging for the Fairness of Machine Learning Software. In *44th IEEE/ACM 44th International Conference on Software Engineering, ICSE 2022, Pittsburgh, PA, USA, May 25-27, 2022*. ACM, 2215–2227.

[69] 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

[70] 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.

[71] 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.

[72] 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.

[73] 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, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net.

[74] 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*.

[75] 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.

[76] 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.

[77] Amitabha Mukerjee, Rita Biswas, Kalyanmoy Deb, and Amrit P Mathur. 2002. Multi–objective evolutionary algorithms for the risk–return trade–off in bank loan management. *International Transactions in operational research* 9, 5 (2002), 583–597.

[78] 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.

[79] 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*.

[80] Osonde A Osoba and William Welser IV. 2017. *An intelligence in our image: The risks of bias and errors in artificial intelligence*. Rand Corporation.

[81] 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.

[82] 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.

[83] Nicolas Papernot, Patrick D. McDaniel, Xi Wu, Somesh Jha, and Ananthram Swami. 2016. Distillation as a Defense to Adversarial Perturbations Against Deep Neural Networks. In *IEEE Symposium on Security and Privacy, SP 2016, San Jose, CA, USA, May 22-26, 2016*. IEEE Computer Society, 582–597.

[84] 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.

[85] Hua Qi, Zhijie Wang, Qing Guo, Jianlang Chen, Felix Juefei-Xu, Lei Ma, and Jianjun Zhao. 2021. ArchRepair: Block-Level Architecture-Oriented Repairing for Deep Neural Networks. *arXiv preprint arXiv:2111.13330* (2021).

[86] 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.

[87] Adriana Romero, Nicolas Ballas, Samira Ebrahimi Kahou, Antoine Chassang, Carlo Gatta, and Yoshua Bengio. 2015. FitNets: Hints for Thin Deep Nets. In *ICLR (Poster)*.

[88] 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.

[89] 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.

[90] 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.

[91] 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.

[92] 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.

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

[94] Jeongju Sohn, Sungmin Kang, and Shin Yoo. 2022. Arachne: Search Based Repair of Deep Neural Networks. *ACM Transactions on Software Engineering and Methodology* (2022).

[95] Matthew Sotoudeh and Aditya V. Thakur. 2021. Provable repair of deep neural networks. In *PLDI*. ACM, 588–603.

[96] Tania Sourdin. 2018. Judge v Robot?: Artificial intelligence and judicial decision-making. *University of New South Wales Law Journal, The* 41, 4 (2018), 1114–1133.

[97] Bing Sun, Jun Sun, Long H. Pham, and Tie Shi. 2022. Causality-Based Neural Network Repair. In *44th IEEE/ACM 44th International Conference on Software Engineering, ICSE 2022, Pittsburgh, PA, USA, May 25-27, 2022*. ACM, 338–349.

[98] 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.

[99] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott E. Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. 2015. Going deeper with convolutions. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2015, Boston, MA, USA, June 7-12, 2015*. IEEE Computer Society, 1–9.

[100] 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*.

[101] 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.

[102] 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.

[103] Shogo Tokui, Susumu Tokumoto, Akihito Yoshii, Fuyuki Ishikawa, Takao Nakagawa, Kazuki Munakata, and Shinji Kikuchi. 2022. NeuRecover: Regression-Controlled Repair of Deep Neural Networks with Training History. In *SANER*. IEEE, 1111–1121.

[104] 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*.

[105] Jasper Ulenaers. 2020. The impact of artificial intelligence on the right to a fair trial: Towards a robot judge? *Asian Journal of Law and Economics* 11, 2 (2020).

[106] 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.

[107] 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.

[108] Yisen Wang, Xingjun Ma, James Bailey, Jinfeng Yi, Bowen Zhou, and Quanquan Gu. 2019. On the Convergence and Robustness of Adversarial Training. In *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA (Proceedings of Machine Learning Research, Vol. 97)*, Kamalika Chaudhuri and Ruslan Salakhutdinov (Eds.). PMLR, 6586–6595.

[109] Yisen Wang, Difan Zou, Jinfeng Yi, James Bailey, Xingjun Ma, and Quanquan Gu. 2020. Improving Adversarial Robustness Requires Revisiting Misclassified Examples. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net.

[110] 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.

[111] 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

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

[113] 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).

[114] Robert F Woolson. 2007. Wilcoxon signed-rank test. *Wiley encyclopedia of clinical trials* (2007), 1–3.

[115] Huanhuan Wu, Zheng Li, Zhanqi Cui, and Jianbin Liu. 2022. GenMuNN: A mutation-based approach to repair deep neural network models. *Int. J. Model. Simul. Sci. Comput.* 13, 2 (2022), 2341008:1–2341008:17.

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

[117] 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

[118] 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).

[119] 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.

[120] 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.

[121] Hanmo You, Zan Wang, Junjie Chen, Shuang Liu, and Shuochuan Li. 2023. Regression Fuzzing for Deep Learning Systems. In *ICSE*. to appear.

[122] 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.

[123] 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.

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

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

[126] Brian Hu Zhang, Blake Lemoine, and Margaret Mitchell. 2018. Mitigating Unwanted Biases with Adversarial Learning. In *Proceedings of the 2018 AAAI/ACM Conference on AI, Ethics, and Society, AIES 2018, New Orleans, LA, USA, February 02-03, 2018*, Jason Furman, Gary E. Marchant, Huw Price, and Francesca Rossi (Eds.). ACM, 335–340.

[127] Hao Zhang and W. K. Chan. 2019. Apricot: A Weight-Adaptation Approach to Fixing Deep Learning Models. In *ASE*. IEEE, 376–387.

[128] Jie M. Zhang and Mark Harman. 2021. "Ignorance and Prejudice" in Software Fairness. In *43rd IEEE/ACM International Conference on Software Engineering, ICSE 2021, Madrid, Spain, 22-30 May 2021*. IEEE, 1436–1447.

[129] 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

[130] 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.

[131] Mengdi Zhang and Jun Sun. 2022. Adaptive fairness improvement based on causality analysis. In *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/FSE 2022, Singapore, Singapore, November 14-18, 2022*, Abhik Roychoudhury, Cristian Cadar, and Miryung Kim (Eds.). ACM, 6–17.

[132] 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.

[133] Peixin Zhang, Jingyi Wang, Jun Sun, Guoliang Dong, Xinyu Wang, Xingen Wang, Jin Song Dong, and Ting Dai. 2020. White-box fairness testing through adversarial sampling. In *ICSE '20: 42nd International Conference on Software Engineering, Seoul, South Korea, 27 June - 19 July, 2020*, Gregg Rothermel and Doo-Hwan Bae (Eds.). ACM, 949–960.

[134] 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.

[135] 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.

[136] 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.

[137] 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.

[138] Yingyi Zhang, Zan Wang, Jiajun Jiang, Hanmo You, and Junjie Chen. 2022. Toward Improving the Robustness of Deep Learning Models via Model Transformation. (2022).

[139] 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.

[140] 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.

[141] Haibin Zheng, Zhiqing Chen, Tianyu Du, Xuhong Zhang, Yao Cheng, Shouling Ji, Jingyi Wang, Yue Yu, and Jinyin Chen. 2022. NeuronFair: Interpretable White-Box Fairness Testing through Biased Neuron Identification. In *44th IEEE/ACM 44th International Conference on Software Engineering, ICSE 2022, Pittsburgh, PA, USA, May 25-27, 2022*. ACM, 1519–1531.

[142] 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.

[143] 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.

[144] Donglin Zhuang, Xingyao Zhang, Shuaiwen Song, and Sara Hooker. 2022. Randomness in Neural Network Training: Characterizing the Impact of Tooling. In *MLSys*. mlsys.org.