Detecting and Repairing Deviated Outputs of Compressed Models

Yichen Li YLIPF@CSE.UST.HK
The Hong Kong University of Science and Technology

Qi Pang QPANG@ANDREW.CMU.EDU
Carnegie Mellon University

Dongwei Xiao DXIAOAD@CSE.UST.HK
Zhibo Liu ZLIUDC@CSE.UST.HK
Shuai Wang SHUAIW@CSE.UST.HK
The Hong Kong University of Science and Technology

Editors: Berrin Yanıkoğlu and Wray Buntine

Abstract

With the rapid development of deep learning and its pervasive usage on various low-power and resource-constrained devices, model compression methods are increasingly used to reduce the model size and computation cost. Despite the overall high test accuracy of the compressed models, our observation shows that an original model and its compressed version (e.g., via quantization) can have *deviated prediction outputs* on the same inputs. These behavior deviations on compressed models are undesirable, given that the compressed models may be used in reliability-critical scenarios such as automated manufacturing and robotics systems.

Inspired by software engineering practices, this paper proposes CompD, a differential testing (DT)-based framework for detecting and repairing prediction deviations on compressed models and their plaintext versions. CompD treats original/compressed models as "black-box," thus offering an efficient method orthogonal to specific different compression schemes. Furthermore, CompD can leverage deviation-triggering inputs to fine-tune the compressed models, largely "repairing" its defects. Evaluations show that CompD can effectively test and repair common models compressed by different schemes.

Keywords: Model compression, differential testing, model repairing.

1. Introduction

Modern deep neural networks (DNN) can easily have millions of parameters, requiring high-performance devices like GPUs for inference. With this regard, model compression techniques Cheng et al. (2017) are proposed to reduce the model latency and memory consumption, enabling the execution of DNN models on small devices like edge or low-power mobile devices. In particular, standard compression techniques like pruning Han et al. (2015) strives to remove weights or channels with little contribution, whereas quantization techniques Jacob et al. (2018) aim to encode weights and activations into fewer bits.

To date, it is widely acknowledged that model compression methods have been successfully deployed to reduce computations, decrease power demands, and thus enable the

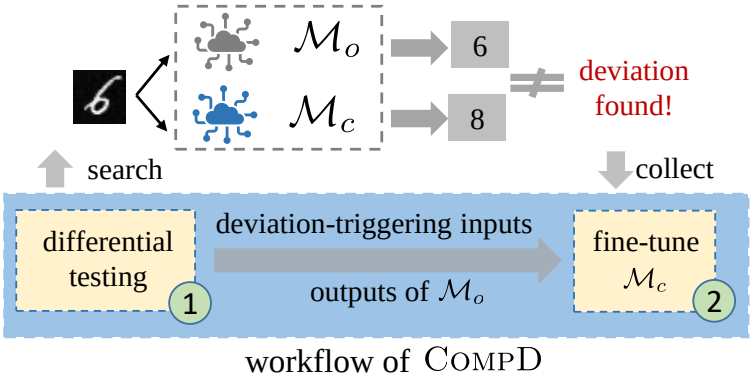

Figure 1: Motivating example and CompD overview. $\mathcal{M}_o$ and $\mathcal{M}_c$ denote the original model and its compressed version.

deployment of DNNs on low-power devices. Furthermore, it is shown that compressed models can often exhibit overall high *test accuracy* compared to the original DNN models Cheng et al. (2017). Nevertheless, our observation shows that the original model $\mathcal{M}_o$ and its compressed version $\mathcal{M}_c$ (e.g., via pruning), though they may both exhibit high test accuracy, can have deviated prediction outputs on the same input, as illustrated in upper Fig. 1. We name such inputs as *deviation-triggering (DT) inputs* in this paper.

Given that compressed models can be involved in supporting safety-critical applications like autonomous vehicles or automated manufacturing processes when deployed in low-cost edge devices and embedded devices. We thus believe that even minor differences between $\mathcal{M}_o$ and $\mathcal{M}_c$ on rare inputs can cause significant security problems, and test accuracy on its own is not sufficient to unveil such hidden behavior deviations between $\mathcal{M}_o$ and $\mathcal{M}_c$. Inspired by principles of software differential testing McKeeman (1998), this research proposes CompD, the first testing and repairing framework to detect prediction deviations between original models and their compressed counterparts. CompD is agnostic to underlying compression methods. As illustrated in ① of Fig. 1, CompD features a unified differential testing approach to effectively search for inputs that result in largely deviant outputs between $\mathcal{M}_o$ and $\mathcal{M}_c$. To enhance the test efficiency, CompD features a novel search-based input generation technique, progressively identifies inputs that maximize the neuron coverage deviations of $\mathcal{M}_o$ and $\mathcal{M}_c$. We show that DT inputs identified by CompD are of high (visual) quality compared with normal inputs, thereby uncovering defects that may cause users of these compressed models substantial confusion in their daily usage and potentially jeopardize the safety of real-world low-cost devices. Moreover, as shown in ② of Fig. 1, CompD collects DT inputs and their corresponding outputs from $\mathcal{M}_o$ as new training data to repair the output deviations, thereby enhancing the robustness and mitigating mispredictions without requiring manual annotations.

Our evaluation encompasses three widely-used model compression methods, pruning, and quantization, offered by the PyTorch framework. To assess the common defects compression techniques may introduce, we evaluate three widely-used DNN models, VGG11 Simonyan and Zisserman (2014), ResNet18 He et al. (2016), and DenseNet121 Huang et al. (2017). After searching 50,000 mutated inputs in total for testing, CompD detected 1958

inputs that resulted in greatly deviant outputs from zero accuracy gap caused by compression, where the prediction labels are inconsistent between $\mathcal{M}_c$ and $\mathcal{M}_o$. With these 1958 inputs, we are able to fine-tune the compressed models, largely improving their robustness. When re-evaluating the "repaired" model $\mathcal{M}_c^+$, we find that almost all the DT inputs are mitigated. The "repaired" $\mathcal{M}_c$ also shows better robustness against adversarial attacks, regarded as the most widespread threats to machine learning models. Developers can use CompD to benchmark and enhance their compressed models before releasing them to end devices. In sum, we make the following contributions.

- This paper, for the first time, proposes a testing and enhancing framework particularly designed for compressed DNN models. We reveal defects that result in deviant prediction outputs, which can cause great confusions or enable adversarial manipulations in the daily usage of compressed models.
- CompD uses feedback-driven differential testing to effectively search for inputs that maximize behavior deviations of $\mathcal{M}_c$ and $\mathcal{M}_o$. CompD can also "repair" compressed DNN models to enhance their robustness at a moderate cost and without human annotations.
- Our evaluation of standard compression techniques and popular DNN models exposes a substantial number of DT inputs. We further repair the exposed defects, making compressed models considerably more robust.

Artifact Availability.

We provide CompD at https://github.com/winnylyc/ModelCompressionTest to facilitate the community to reproduce our results. We will maintain CompD to benefit the community.

2. Preliminary and Related Works

Pruning. Pruning is an effective and general way to reduce the model size and computations. Exiting pruning works can be divided into two categories: unstructured pruning and structured pruning. Unstructured pruning aims to prune connections in the DNN, leading to unstructured sparsity of models. Han et al. proposed a three-step method to prune redundant connections Han et al. (2015). Dynamic network surgery designed by Guo et al. can integrate connection splicing into the pruning process, significantly reducing network complexity Guo et al. (2016). Tung and Mori proposed CLIP-Q, combining the advantages of weight pruning and weight quantization in a single framework Tung and Mori (2018). Structured DNN pruning aims to reduce the memory footprint and computational workload of DNNs by strategically identifying and removing "unimportant" inter-layer connections and thus simplifying their computation graphs. It has been shown to effectively improve the performance and energy efficiency of DNN models while causing little to no accuracy loss Molchanov et al. (2016); Zhao et al. (2019). For instance, CNNs can be boosted by removing filters with a smaller L1-norm Li et al. (2016). To date, we have seen a variety of metrics quantifying the importance of channels Liu et al. (2017); Kruschke and Movellan (1991), under which the least important channels are pruned.

Quantization. Quantization is another widely used technique to compress models. It largely facilitates reducing the computational and storage complexity of DNN models. Given that floating-point operations are more expensive to evaluate than integer operations, model quantization uses integers as an approximation for the full-precision floating-point numbers in the original model. Quantization can be applied to the model inference Choukroun et al. (2019); Yao et al. (2021). Some works have also designed methods for quantized model training Banner et al. (2018). An issue with model quantization is that the quantized model may give different outputs from its full-precision counterpart. Thus, it is often necessary to adjust the parameters of the quantized models for better accuracy and usability. Quatization-Aware Training (QAT) re-trains quantized models, whereas Post-Training Quantization (PTQ) directly adjusts the parameters of quantized models without fine-tuning.

Robustness of Compressed Models. Model compression techniques generally focus on the accuracy comparison between the original model and the compressed one. However, even though compressed DNN models can achieve high accuracy, they can be susceptible to *adversarial attacks*, i.e., designing inputs with normal appearance as benign ones while leading the attacked DNN models to give incorrect predictions. In fact, Lin et al. (2019) show that compressed models are more vulnerable to adversarial attacks than their original counterparts. To mitigate such issues, some works Ye et al. (2019); Gui et al. (2019); Song et al. (2020) have investigated designing model compression methods that are both *robust* against adversarial attacks and high in accuracy. Also, while we have noticed some parallel works Tian et al. on testing compressed models, we clarify that their methods simply treat the target models as "black-boxes", while this paper explores a "white-box" view such that model gradients can be properly leveraged to guide the discovery of DT inputs. Moreover, this paper for the first time illustrates that the discovered DT inputs can be used for model retraining and effectively improve the robustness of compressed models.

3. Design of CompD

Overview. This work designs CompD, the first automated framework to detect and repair deviation behaviors of compressed DNNs. We have depicted the pipeline of CompD in Fig. 1. Overall, given a target compressed model $\mathcal{M}_c$, developers are also anticipated to possess the original model $\mathcal{M}_o$. These two models serve as inputs of CompD for the following two steps.

① **Differential Testing.** CompD employs feedback-driven differential testing to explore inputs that result in deviant outputs of $\mathcal{M}_c$ and $\mathcal{M}_o$ (Sec. 3.2). That is, CompD aims to find an input i which can result in a considerably large output deviation δ as follows.

$$\underset{i}{\text{maximize: }} \delta = |\mathcal{M}_o(i) - \mathcal{M}_c(i)| \tag{1}$$

In particular, we define δ in accordance with the prediction output, such that when $\mathcal{M}_o$ and $\mathcal{M}_c$ yield distinct prediction labels, we collect the inputs as *deviation-triggering* (DT) input set $\mathcal{I}_{dt}$. We further augment CompD's ability to find DT inputs through neuron

coverage deviation. We have derived the concept of neuron coverage from the work presented in Pei et al. (2017). However, our approach differs from theirs in that we particularly focus on neuron activation changes over pruned neurons instead of activation changes over any neurons. This approach leads to the discovery of more DT inputs. Sec. 3.2 provides technical details on our approach.

In Sec. 5.1, we compare the number of effective DT inputs found by CompD on pruning with and without neuron coverage. While recent work Yang et al. (2022) indicates that combining coverage-driven and gradient-based methods may be ineffective to enhance adversarial testing, this work uses empirical results to illustrate the effectiveness of this combination in the differential testing setting. Furthermore, we show that most DT inputs are stealthy and hardly distinguishable compared with normal test inputs. Our evaluation results illustrate stealthy attack vectors toward compression models that are practical in real-world scenarios.

② **Repairing.** Further to the testing campaign, we also "repair" the compressed models and enhance their robustness. In particular, we collect the DT inputs $i \in \mathcal{I}_{dt}$ discovered in ① and form a repairing dataset $\{(i, \mathcal{M}_o(i)) | i \in \mathcal{I}_{dt}\}$ where $\mathcal{M}_o(\cdot)$, the prediction outputs of $\mathcal{M}_o$ over i denotes the expected label of $\mathcal{M}_c(i)$. This formed training repairing dataset will be used to fine-tune the compressed model $\mathcal{M}_c$. The end result would be another compressed model $\mathcal{M}_c^+$ with better robustness while retaining stable accuracy. Note that this step benefits from the DT inputs automatically discovered by CompD and their labels yielded by the original model $\mathcal{M}_o$. Thus, *no* human efforts are required, making the overall repairing effort moderate. Users of CompD, as the model provider, can release $\mathcal{M}_c^+$ for users to use. Before discussing the technical details, we first clarify the application scope of CompD in Sec. 3.1.

3.1. Application Scope

Main Audiences. The main audiences of CompD would be model owners who want to use model compression techniques to process their models before shipping them to low-cost edge or mobile devices. Our work helps model owners to assess and augment the robustness of their compressed models before release. This would eliminate potential attack vectors of compressed models and enhance their reliability. CompD is the first automated framework in this field.

Malicious Model Owners. Following our discussion above, CompD is designed for normal model owners to benchmark the quality of their compressed models in an in-house setting. That is, CompD is *not* intended to be used against an active adversary. For instance, a malicious model owner may want to add a backdoor in his compressed models to control the model predictions regarding inputs with backdoor triggers. Detecting such injected backdoors in compressed models is an orthogonal area, and we leave it for future work.

Distinguishing from Adversarial Examples (AEs). Similar to adversarial examples (AEs), CompD's findings, DT inputs, can manipulate the prediction outputs of compressed models. Real-world black-box AE attacks often denote an online setting Ilyas et al. (2018); Guo et al. (2019); Suya et al. (2020), where they require attackers to iteratively query a remote model (e.g., a cloud service) with mutated inputs to control the model predictions

Algorithm 1 Feedback-driven differential testing.

Input: Corpus of Seed Inputs $\mathcal{S}$, $\mathcal{M}_o$, $\mathcal{M}_c$
$\mathcal{O} \leftarrow \varnothing$
for i_o *in* $\mathcal{S}$ **do**
  $i \leftarrow i_o$
  for *1 ... p* **do**
    LABEL $\leftarrow$ PREDICT($\mathcal{M}_o(i)$)
    $\text{CS}_o \leftarrow$ CONFIDENCESCORE($\mathcal{M}_o(i)$, LABEL)
    $\text{CS}_c \leftarrow$ CONFIDENCESCORE($\mathcal{M}_c(i)$, LABEL)
    LOSS $\leftarrow -(\text{CS}_o - \text{CS}_c)$
    GRAD $\leftarrow \frac{\partial loss}{\partial i}$
    GRAD $\leftarrow$ CONSTRAINT(GRAD)
    $i \leftarrow i -$ GRAD
    add i in $\mathcal{O}$
  end
end
return $\mathcal{O}$

at their will. Nevertheless, we clarify that the findings of CompD are *distinct* from AEs found by prior techniques. CompD uses differential testing to find inputs that maximize the output deviations of original and compressed models. In particular, we find that the DT inputs $\mathcal{I}_{dt}$ will not alter the prediction labels in the original models $\mathcal{M}_o$, whereas they largely alter the predictions of the compressed models $\mathcal{M}_c$. In contrast, conventional AEs change the predictions of $\mathcal{M}_o$. The root cause of DT inputs $\mathcal{I}_{dt}$ is the reduction in information. Pruning reduces the number of neurons and synapses connecting neurons. Quantization reduces the precision of neurons. Based on the result in Sec. 5.1 , we prove that these root causes can be leveraged to augment the ability to find $\mathcal{I}_{dt}$. In contrast, conventional AEs are pervasive in DNNs and are believed to root in inadequate training and unsmooth classification boundaries of the trained model Leino et al. (2021).

3.2. CompD — Differential Testing

Alg. 1 depicts the general workflow of our testing. Function ***DT*** is the main entry point. It accepts a collection of seed inputs $\mathcal{S}$, and a pair of models $\mathcal{M}_o$ and $\mathcal{M}_c$. All detected deviation-triggering inputs would be stored in the output $\mathcal{O}$. For each iteration, we pick one input i from the seed and use gradient back-propagation to mutate it. We will mutate a seed with p times, which is empirically fixed as 10. We first get the predicted label $\mathcal{L}_o$ from $\mathcal{M}_o$. Then, get the confidence score from $\mathcal{M}_o$ and $\mathcal{M}_c$ of i being predicted as $\mathcal{L}_o$.

We use gradients from loss to mutate i (lines 9–12). We want to change the predicted label of $\mathcal{M}_c$, and meanwhile keep the predicted label of $\mathcal{M}_o$, so the objective function aims to increase CS_o and decrease CS_c. We use the loss function to calculate the gradient of input. The gradient is constrained to guarantee the stealthiness of mutation. For the current implementation, we limit the mutation over i within the variance in the standard input dataset that was used to train $\mathcal{M}_o$. When a mutation over a pixel in i is beyond the variance, we clip it to variance.

For testing on pruning, we use neuron coverage deviation to augment the ability to find DT input as depicted in Alg. 2. During the process of calculating the confidence score, we need to record the output neurons before each ReLU layer. These neurons are used to calculate the neuron coverage (lines 11–16). We aim to find the neurons that have

Algorithm 2 Neuron coverage augmented feedback-driven differential testing.

Input: Corpus of Seed Inputs $\mathcal{S}$, $\mathcal{M}_o$, $\mathcal{M}_c$
$\mathcal{O} \leftarrow \varnothing$
for i_o *in* $\mathcal{S}$ **do**
  $i \leftarrow i_o$
  for *1 ... p* **do**
    LABEL $\leftarrow$ PREDICT($\mathcal{M}_o(i)$)
    $\text{CS}_{po} \leftarrow$ CONFIDENCESCORE($\mathcal{M}_o(i)$, LABEL)
    $\text{NEURONS}_p \leftarrow$ HOOK($\mathcal{M}_o(i)$, LABEL)
    $\text{CS}_c \leftarrow$ CONFIDENCESCORE($\mathcal{M}_c(i)$, LABEL)
    $\text{NEURONS}_c \leftarrow$ HOOK($\mathcal{M}_c(i)$, LABEL)
    while *True* **do**
      INDEX $\leftarrow$ RANDOMSELECT(NEURONS_o)
      $\text{NEURON}_o \leftarrow \text{NEURONS}_o$[INDEX]
      $\text{NEURON}_c \leftarrow \text{NEURONS}_c$[INDEX]
      if $\text{NEURON}_o > 0$ & $\text{NEURON}_c <= 0$ **then**
        break
      end
    end
    LOSS $\leftarrow -(\text{CS}_o - \text{CS}_c + \lambda * \text{NEURON}_c)$
    GRAD $\leftarrow \frac{\partial loss}{\partial i}$
    GRAD $\leftarrow$ CONSTRAINT(GRAD)
    $i \leftarrow i -$ GRAD
    add i in $\mathcal{O}$
  end
end
return $\mathcal{O}$

been activated by $\mathcal{M}_o$ and have not been activated by $\mathcal{M}_c$. The shape of $Neurons_0$ and $Neurons_c$ are the same since pruning will not influence the structure of the models. Given a randomly chosen neuron from $Neurons_0$, we can easily find the corresponding neuron in $Neurons_c$ through the index. If these two related neurons are different in activation, we add it to the loss calculation. Note that we also tentatively explored to determine the difference in the activation through neurons after ReLU layers. However, the ReLU layers cut off the gradient backpropagation from neurons, so we use the neurons before each ReLU layer instead. We want to increase the difference in the selected neuron, so the objective function aims to increase the value of the neuron (line 17). Compared with $\mathcal{M}_o$, it is harder for $\mathcal{M}_c$ to increase the selected neuron due to the pruned related neuron and synapses. In this situation, the deviation in the selected neuron will be enlarged and propagate the deviation to the prediction during inference. A hyperparameter is applied to weight the neuron in the loss function, which is empirically fixed as 20.

We do not check the validation of deviation-triggering inputs in Alg. 1. The models' inputs are normalized, so the mutated inputs will be changed when denormalized and saved as pictured, including conversion from floating-point numbers to integers. Though the changes are small compared to the mutation conducted on the seed, we found that these changes can influence some predictions from models. We save the $\mathcal{O}$ in picture format for further repair and evaluation.

Availability of Seed $\mathcal{S}$. CompD testing requires a collection of seeds $\mathcal{S}$. We underline that we have no specific requirements for the seeds. In our evaluation, we randomly select the first half of the model's test dataset, which contains 5,000 inputs.

3.3. CompD — Repairing

With the deviation-triggering inputs discovered by CompD, we can repair $\mathcal{M}_c$. We first need to collect $\mathcal{I}_{dt}$ for repairing. We test the deviation-triggering inputs in the picture form and divide them into three classes. The first class is effective deviation-triggering inputs. They will not change the prediction from $\mathcal{M}_o$ through mutation. However, mutation on them will change the prediction from $\mathcal{M}_c$. The second class is invalid deviation-triggering inputs. They will directly change prediction from $\mathcal{M}_o$through mutation. They are similar to AEs. However, their mutations are not strictly constrained as AEs, so many of them are meaningless pictures. There may be doubt about whether the effective deviation-triggering inputs are meaningful. Though they are not constrained as strictly as AEs, they can be recognized successfully by $\mathcal{M}_o$, which guarantees that they are meaningful. The third class is valid deviation-triggering inputs. They are all the remaining inputs. They will be collected with effective deviation-triggering inputs to form $\mathcal{I}_{dt}$ together and join the repairing. Our experiment result shows that repairing only with effective deviation-triggering inputs will reduce the accuracy of $\mathcal{M}_c$.

$\mathcal{I}_{dt}$ is utilized to fine-tune $\mathcal{M}_c$ by forming data set $\{(i, \mathcal{M}_o(i))|i \in \mathcal{I}_{dt}\}$. It needs to be emphasized that $\mathcal{M}_o(i)$ is used as the training label instead of the true label. Otherwise, fine-tuning with testset data, even mutated, will reduce the models' ability of generalization. Due to this risk, we use another half testset data that does not participate in CompD to test the accuracy to guarantee the ability of generalization. Trainset data also joins the repairing, because fine-tuning only with $\mathcal{I}_{dt}$ will greatly reduce accuracy. In our implementation, trainset data is mixed with $\mathcal{I}_{dt}$ to form the new trainset data, so during the fine-tuning process, $\mathcal{I}_{dt}$ and original trainset data are randomly selected to form a batch. The learning rate needs to be much less than the learning rate applied during the original training process. In this work, the learning rate in the fine-tuning process is 0.01 times the learning rate in the training process.

The process of repairing is orthogonal to all specific different compression schemes and different models mentioned in this paper. They all use the same automatic workflow without human effort, and generally show improved robustness and stable accuracy after repair. A small difference in implementation between pruning and quantization is depicted in Sec. 4.

Table 1: Evaluation setup and statistics.

Scheme	Model	Original Model Accuracy	Compressed Model Accuracy	#Effective Deviation-Triggering Inputs
Pruning	**VGG11**	93.12%	92.78%	1731
	ResNet18	95.80%	95.80%	1959
	DenseNet121	96.16%	95.82%	1417
Quant.	**VGG11**	92.90%	92.78%	767
	ResNet18	95.66%	95.76%	554
	DenseNet121	96.16%	95.16%	2751

4. Implementation & Evaluation Setup

CompD is written primarily in Python. All experiments are launched on a machine with one AMD Ryzen CPU, 256GB RAM, and one Nvidia GeForce RTX 3090 GPU.

Compression Schemes. CompD's testing and repairing technique is orthogonal to the compression schemes. We evaluate CompD using two representative compression schemes: quantization and pruning. We have introduced each scheme in Sec. 2. We implement both compression schemes in PyTorch.

For pruning, we prune the weight parameters of all convolution layers with L1-unstructured pruning. The pruning rate is set to 0.75, meaning that 75% of the parameters will be set to 0. After obtaining the pruned model, it was fine-tuned for 20 epochs with a learning rate of 0.01 during the training process. Based on our observations, the choice of hyperparameters during fine-tuning did not significantly affect the results of the subsequent testing.

As for the setting of quantization, we convert the model from float32 to int8. We use Quantization-Aware Training (QAT) for the quantization. Initially, we perform layer fusion, wherein all combinations of convolutional, batch normalization, and ReLU activation layers are fused together. Subsequently, we retrain the model using quantization-aware regularization terms. In PyTorch, this is done by adding quantization and dequantization layers to the input and output, respectively. Additionally, we need to specify the quantization configuration, which for our experiment is `fbgemm` in PyTorch. After obtaining a float-32 model $\mathcal{M}_{c-float32}$ to mimic the behavior of the quantized model $\mathcal{M}_{c-int8}$ in int-8, we need to convert $\mathcal{M}_{c-float32}$ to quantized integer in PyTorch to obtain our final compressed model. However, conducting gradient backpropagation on $\mathcal{M}_{c-int8}$ is challenging in PyTorch, so we use $\mathcal{M}_{c-float32}$ in the test and repair process. In the DT setting, we find DT inputs to $\mathcal{M}_{c-float32}$ and test DT inputs' quality, including the effect on deviation output and stealthiness. During the repairing process, we first fine-tune $\mathcal{M}_{c-float32}$ to get $\mathcal{M}^{+}_{c-float32}$ using our discovered DT inputs, then quantize it to the corresponding repaired int8-model, $\mathcal{M}^{+}_{c-int8}$. For robustness testing depicted in Sec. 5.2, we use $\mathcal{M}_{c-float32}$ and $\mathcal{M}^{+}_{c-float32}$ to conduct a white-box attack. The mutated inputs will be fed into both $\mathcal{M}_{c-int8}$ and $\mathcal{M}^{+}_{c-int8}$ to compare the robustness of those two models.

Models under Testing. We use three common DNN models, VGG11, ResNet18, and DenseNet121, trained on CIFAR-10, as our tested models. All these models are representative]in the Computer Vision domain and are widely used. The models are slightly adjusted in order to meet the requirements of the compression schemes. The prediction accuracies of the adjusted models are reported in the column "Original Accuracy" of Table 1.The DenseNet121 exhibits a noticeable decrease in accuracy following quantization. This can be attributed to the inability to fuse layers as other models, as half of the convolutional layers in DenseNet121 are connected to multiple batch normalization layers, precluding the possibility of conducting layer fusion on these layers. Although some existing quantization methods for DenseNet have demonstrated superior performance, we opted to employ the same quantization setting on DenseNet121 as on other models to ensure the method's universality across all compression schemes. Nonetheless, this decision has imposed a constraint on our experiment, as it amplifies the divergence between $\mathcal{M}_{c-float32}$ and $\mathcal{M}_{c-int8}$. Despite this limitation, our testing and repairing method has remained highly effective.

Table 2: Effect of neuron coverage. ✕/✓ represents testing without/with neuron coverage.

Model	Neuron Coverage	#Effective Deviation-Triggering Inputs
VGG11	✕	1636
	✓	1731
ResNet18	✕	1955
	✓	1959
DenseNet121	✕	1305
	✓	1417

5. Evaluation

5.1. Finding Deviation-Triggering Inputs

Scheme	DNN Models	Effective Deviation-Triggering Inputs	Avg. L2-Distance
Pruning	VGG11		0.0306
	Resnet18		0.0304
	DenseNet121		0.0303
Quant.	VGG11		0.0300
	Resnet18		0.0304
	DenseNet121		0.0302

Figure 2: Examples of deviation-triggering inputs found by CompD. We report the average L_2 distance of deviation-triggering inputs and the normal inputs to illustrate how close they are. We also present several examples. It can be observed that such deviation-triggering inputs, though stealthy change the prediction outputs of the compressed models, manifest high visual similarity with the normal inputs.

Table 1 reports the statistics of our evaluated two compression schemes and three models. The "Quant." in the table refers to the "Quantization." The accuraciers of uncompressed models are shown in the column "Original Accuracy." Then we compress the three original models using the two compression schemes, and fine-tune the obtained models with the training set. The prediction accuracy of the compressed models is illustrated in the column "Compressed Accuracy". The compressed models exhibit similar or even higher accuracy than their original counterparts, primarily due to the fine-tuning process. In other words, deviation-triggering inputs found by CompD are *not* due to the accuracy loss brought by model compression itself. Instead, they are caused by the decision boundary driftings due to model compression techniques.

We use the first half of testset data as seed and the other quater of testset data as test data for our implementation. We launch CompD to test each compressed model with a maximum of $50,000$ mutation iterations over the seeds. Overall, CompD detects a great number of deviation-triggering inputs, which will not change the prediction results in $\mathcal{M}_o$,

Table 3: Models after applying repairing.

Scheme	Model	Acc. on Test Data Before / After Repairing	#Effective Deviation-Triggering Inputs Before / After Repairing (Test Inputs)
Pruning	**VGG11**	92.78% / 92.82%	1731 (829) / 636 (533)
	ResNet18	95.80% / 95.74%	1959 (991) / 617 (477)
	DenseNet121	95.82% / 95.92%	1417 (617) / 513 (514)
Quant.	**VGG11**	92.78% / 92.86%	767 (389) / 0 (236)
	ResNet18	95.76% / 95.64%	554 (310) / 0 (219)
	DenseNet121	95.16% / 94.74%	2751 (1450)/ 167 (330)

but will lead $\mathcal{M}_c$ to give deviated prediction labels from the corresponding unmutated ones. We report several effective deviation-triggering samples in Fig. 2.

Neuron Coverage Augmentation. At this step, we use neuron coverage to augment CompD's ability to find effective DT inputs. Table 2 compares the number of effective DT inputs found by CompD on pruning with and without neuron coverage. Neuron coverage-guided mutation helps to uncover slightly more DT inputs on all models. We further tune the parameter of λ (described in Sec. 3.2. We observe that when λ is less than 20, increasing λ would increase the number of effective DT inputs on VGG11, illustrating the effectiveness of neuron coverage.

Deviation-Triggering Input Quality. As discussed in Sec. 3.2, CompD bounds the total number of mutations toward each input to ensure that the inputs are meaningful. Moreover, we save the mutated seed in picture format before all the tests on them. This guarantee that the inputs are within a pre-defined value range. In addition, the influence of normalization and data format can be eliminated. In other words, the attack can be applied in practical scenarios. As a common setup, we calculate the $L2$ distance between the mutated deviation-triggering inputs and the original inputs to quantify the distance between these inputs.

Fig. 2 illustrates that the deviation-triggering inputs retain meaningful contents. More importantly, we find that the original model $\mathcal{M}_o$ can indeed give *correct* predictions on all these "deviation-triggering" inputs, which in turn demonstrates that they are perceptually meaningful from the well-trained DNNs' perspective. The average $L2$ distance over the input features also reflects that our mutation is small, which is about 0.03 divergence per pixel on average for the CIFAR-10. In our implementation, the range of data during inference is about [-2.7, 2.7].

5.2. Repairing Compressed Models

With the detected deviation-triggering inputs on hand, we are able to repair the compressed models. We find that following the repairing, all models retain high accuracy on the test dataset and become more robust against the deviation-triggering inputs found by CompD. Moreover, as we ignore all the invalid DT inputs, which directly change prediction from $\mathcal{M}_o$through mutation, the models' robustness against adversarial examples cannot be guaranteed. We use the common adversarial attack method FGSM Goodfellow et al. (2014) to test models. Models after repair show great improvement in robustness.

Testing on Repaired Models. To further demonstrate the effectiveness of the repairing, we use the deviation-triggering inputs found by CompD launched on original models to test

Table 4: Robustness against FGSM with $\epsilon = 2/8/16$.

Scheme	Model	Accuracy on Adversarial Examples Before Repairing	Accuracy on Adversarial Examples After Repairing
Pruning	**VGG11**	57.90%/25.46%/15.00%	61.12%/35.00%/26.20%
	ResNet18	53.40%/35.84%/24.42%	64.44%/53.84%/51.26%
	DenseNet121	60.34%/31.6%/19.84%	65.84%/49.22%/39.86%
Quant.	**VGG11**	59.24%/27.96%/16.64%	61.90%/33.84%/22.48%
	ResNet18	59.92%/40.18%/25.32%	67.06%/54.44%/44.04%
	DenseNet121	57.78%/23.72%/15.10%	68.88%/51.64%/38.14%

the repaired models. Additionally, we produced a set of test inputs that were not involved in the fine-tuning process to demonstrate the repaired model's robustness against DTI. The results are shown in Table 3. We observe that deviation-triggering inputs can influence the compressed models to a much less degree. For quantization, seldom deviation-triggering inputs are still effective on repaired models. Furthermore, all repaired compressed models retain stable accuracy. Most of them show even better accuracy than before, indicating the high effectiveness of our repairing scheme. While some models exhibit a slight reduction in accuracy after repairing, we view the downgrading is negligible, and the overall model accuracy is still sufficiently high. It needs to be noticed that the test data has never participated in any process of testing or repairing as described in Sec. 3.3. The result of accuracy shown here can guarantee the ability of generalization.

Robustness on Repaired Models. Evaluation of effective deviation-triggering inputs does not include input that misleads the prediction of the original model. These inputs are similar to adversarial examples, except there is no strict constraint on mutation. An Adversarial attack is easier to be realized on the compressed model than deviation-triggering inputs, since the deviation-triggering inputs also need the information of the original model. Thus, the robustness against adversarial examples is significant. Referring to the setting in Lin et al. (2019), we use a common adversarial attack method FGSM, with $\epsilon = 2/8/16$ and infinite norm, which means the infinite norm of mutation needs to be smaller than ϵ. ϵ=1 means 1/255. It is the smallest meaningful mutation on pictures.

The results are shown in Table 3. The repaired compressed model shows improvement in all the experiment sets. It performs extremely effectively on the ResNet18 model, both on the pruning scheme and quantization scheme. Fine-tuning with deviation-triggering inputs works similarly to adversarial training. Compared with adversarial training, deviation-triggering inputs are bounded by the decision boundaries of original models, so the accuracy of the compressed model will not be reduced as adversarial training.

The robustness of the original models is shown in Table 5. It matches our results on the robustness of repaired models. The original ResNet18 shows much better robustness compared with the other two models. The corresponding repaired compressed ResNet18 shows the best robustness on both pruning and quantization. This high correlation proves that repairing compressed models leverages the original models' decision boundary.

Table 5: Original models' Robustness against FGSM with $\epsilon = 2/8/16$.

Model	Original Accuracy on Adversarial Examples
VGG11	59.26%/26.30%/15.34%
ResNet18	58.96%/41.24%/27.24%
DenseNet121	59.5%/32.36%/20.74%

6. Discussion

Limitations. CompD is a feedback-driven differential testing tool. The $\mathcal{I}_{dt}$ collected from CompD can be utilized to repair $\mathcal{M}_c$. The end result $\mathcal{M}_c^+$ shows better robustness while retaining stable accuracy. However, using CompD to test $\mathcal{M}_c^+$ will not find obviously less deviation-triggering inputs than $\mathcal{M}_c$. We regard robustness against adversarial attack as a more critical issue than robustness against CompD. Both the compressed model and original model are necessary for CompD. It is impractical to meet the requirements of attack. However, as described in Sec. 3.1, the deviation-triggering inputs differ from adversarial examples. They reveal potential threats to the compressed model. We assume robustness against CompD will contribute to the general robustness. Our future work will focus on improving robustness against CompD while retaining stable robustness against adversarial training.

More DataSet. While we acknowledge the significance of conducting experiments on additional datasets, comprehensive testing on multiple datasets was not feasible within the given time constraints. Our preliminary experiments demonstrate notable results. DT found in CNN for MNIST decreases from 348 to 70 for pruning and from 64 to 4 for quantization, indicating a expected effect. However, DT found in VGG11 for CIFAR100 decreases from 985 to 929 for pruning and 980 to 387 for quantization, indicating not sufficient effect for pruning. We hypothesize that the increasing number of labels diminishes the effectiveness of our tool CompD. We leave more experiments as future work.

Alternative Feedbacks. As depicted in Table 5.1, we apply neuron coverage to compression and observe the increasing effect over reducing parameters. We also test neuron coverage on quantization and observe the negligible effect. Since the neuron coverage is implemented based on root causes of DT inputs from pruning, it is reasonable to perform poorly on quantization. Neuron coverage is the main direction for the more powerful differential test because it can leverage the root cause of deviation between the original model and the compressed model. In the future, We will further explore the neuron coverage moderated for quantization and models with massive parameters. The key point is to expand the single neuron deviation to channel-wise or even layer-wise.

Other Compression Methods. In addition to the pruning and quantization, we also notice other compression methods, such as knowledge distillation (KD). KD is a technique that distills knowledge from a well-trained, often redundant model into a smaller model. Hinton et al. (2015); Romero et al. (2014); Yim et al. (2017). Our workflow is mostly orthogonal to the specific implementation of compression schemes, including KD. The process on KD will be similar to pruning. We leave it as one future work to explore the feasibility of testing and repairing models compressed via other schemes like KD.

7. Conclusion

We present CompD, a feedback-driven differential testing tool to detect deviation behaviors of compressed DNNs. The detected deviation-triggering inputs appear to be highly meaningful and visually consistent with regular model inputs. However, they stealthily change the model prediction outputs of compressed models. This illustrates that defects found by CompD are practical, yet overlooked by existing works. We further demonstrate techniques to enhance the robustness of compressed models. We show that the repaired models have high robustness without incurring much extra overhead. We envision that fixing these deviation-triggering inputs can effectively enhance the robustness of compressed models, making their adoption in reliability-critical scenarios more feasible. We have also released the source code of CompD for the community to use and benefit follow-up research.

Acknowledgement

This work was supported in part by the research fund provided by HSBC and the HKUST-VPRDO 30 for 30 Research Initiative Scheme under the the contract Z1283. We are grateful to the anonymous reviewers for their valuable comments.

References

Ron Banner, Itay Hubara, Elad Hoffer, and Daniel Soudry. Scalable methods for 8-bit training of neural networks. *Advances in neural information processing systems*, 31, 2018.

Yu Cheng, Duo Wang, Pan Zhou, and Tao Zhang. A survey of model compression and acceleration for deep neural networks. *arXiv preprint arXiv:1710.09282*, 2017.

Yoni Choukroun, Eli Kravchik, Fan Yang, and Pavel Kisilev. Low-bit quantization of neural networks for efficient inference. In *2019 IEEE/CVF International Conference on Computer Vision Workshop (ICCVW)*, pages 3009–3018. IEEE, 2019.

Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014.

Shupeng Gui, Haotao Wang, Haichuan Yang, Chen Yu, Zhangyang Wang, and Ji Liu. Model compression with adversarial robustness: A unified optimization framework. *Advances in Neural Information Processing Systems*, 32, 2019.

Chuan Guo, Jacob Gardner, Yurong You, Andrew Gordon Wilson, and Kilian Weinberger. Simple black-box adversarial attacks. ICML, 2019.

Yiwen Guo, Anbang Yao, and Yurong Chen. Dynamic network surgery for efficient dnns. *Advances in neural information processing systems*, 29, 2016.

Song Han, Jeff Pool, John Tran, and William Dally. Learning both weights and connections for efficient neural network. *Advances in neural information processing systems*, 28, 2015.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016.

Geoffrey Hinton, Oriol Vinyals, Jeff Dean, et al. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2(7), 2015.

Gao Huang, Zhuang Liu, Laurens van der Maaten, and Kilian Q. Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.

Andrew Ilyas, Logan Engstrom, Anish Athalye, and Jessy Lin. Black-box adversarial attacks with limited queries and information. In *International Conference on Machine Learning*, pages 2137–2146. PMLR, 2018.

Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2704–2713, 2018.

John K Kruschke and Javier R Movellan. Benefits of gain: Speeded learning and minimal hidden layers in back-propagation networks. *IEEE Transactions on systems, Man, and Cybernetics*, 21(1):273–280, 1991.

Klas Leino, Zifan Wang, and Matt Fredrikson. Globally-robust neural networks. In *International Conference on Machine Learning*, pages 6212–6222. PMLR, 2021.

Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning filters for efficient convnets. *arXiv preprint arXiv:1608.08710*, 2016.

Ji Lin, Chuang Gan, and Song Han. Defensive quantization: When efficiency meets robustness. *arXiv preprint arXiv:1904.08444*, 2019.

Zhuang Liu, Jianguo Li, Zhiqiang Shen, Gao Huang, Shoumeng Yan, and Changshui Zhang. Learning efficient convolutional networks through network slimming. In *Proceedings of the IEEE international conference on computer vision*, pages 2736–2744, 2017.

William M McKeeman. Differential testing for software. *Digital Technical Journal*, 10(1): 100–107, 1998.

Pavlo Molchanov, Stephen Tyree, Tero Karras, Timo Aila, and Jan Kautz. Pruning convolutional neural networks for resource efficient inference. *arXiv preprint arXiv:1611.06440*, 2016.

Kexin Pei, Yinzhi Cao, Junfeng Yang, and Suman Jana. DeepXplore: Automated whitebox testing of deep learning systems. In *Proceedings of the 26th Symposium on Operating Systems Principles*, SOSP '17, 2017.

Adriana Romero, Nicolas Ballas, Samira Ebrahimi Kahou, Antoine Chassang, Carlo Gatta, and Yoshua Bengio. Fitnets: Hints for thin deep nets. *arXiv preprint arXiv:1412.6550*, 2014.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

Chang Song, Elias Fallon, and Hai Li. Improving adversarial robustness in weight-quantized neural networks. *arXiv preprint arXiv:2012.14965*, 2020.

Fnu Suya, Jianfeng Chi, David Evans, and Yuan Tian. Hybrid batch attacks: Finding black-box adversarial examples with limited queries. {USENIX} Security, 2020.

Yongqiang Tian, Wuqi Zhang, Ming Wen, Shing-Chi Cheung, Chengnian Sun, Shiqing Ma, and Yu Jiang. Finding deviated behaviors of the compressed dnn models for image classifications. *ACM Transactions on Software Engineering and Methodology*.

Frederick Tung and Greg Mori. Clip-q: Deep network compression learning by in-parallel pruning-quantization. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7873–7882, 2018.

Z. Yang, J. Shi, M. Asyrofi, and D. Lo. Revisiting neuron coverage metrics and quality of deep neural networks. In *2022 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*, 2022.

Zhewei Yao, Zhen Dong, Zhangcheng Zheng, Amir Gholami, Jiali Yu, Eric Tan, Leyuan Wang, Qijing Huang, Yida Wang, Michael Mahoney, et al. Hawq-v3: Dyadic neural network quantization. In *International Conference on Machine Learning*, pages 11875–11886. PMLR, 2021.

Shaokai Ye, Kaidi Xu, Sijia Liu, Hao Cheng, Jan-Henrik Lambrechts, Huan Zhang, Aojun Zhou, Kaisheng Ma, Yanzhi Wang, and Xue Lin. Adversarial robustness vs. model compression, or both? In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, October 2019.

Junho Yim, Donggyu Joo, Jihoon Bae, and Junmo Kim. A gift from knowledge distillation: Fast optimization, network minimization and transfer learning. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4133–4141, 2017.

Chenglong Zhao, Bingbing Ni, Jian Zhang, Qiwei Zhao, Wenjun Zhang, and Qi Tian. Variational convolutional neural network pruning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2780–2789, 2019.