DiffChaser: Detecting Disagreements for Deep Neural Networks

Xiaofei Xie[1], **Lei Ma**[2*], **Haijun Wang**[1*], **Yuekang Li**[1], **Yang Liu**[1,3] and **Xiaohong Li**[4]

[1]Nanyang Technological University, Singapore
[2]Kyushu University, Japan
[3]Zhejiang Sci-Tech University, China
[4]Tianjin University, China

Abstract

The platform migration and customization have become an indispensable process of deep neural network (DNN) development lifecycle. A high-precision but complex DNN trained in the cloud on massive data and powerful GPUs often goes through an optimization phase (*e.g.*, quantization, compression) before deployment to a target device (*e.g.*, mobile device). A test set that effectively uncovers the disagreements of a DNN and its optimized variant provides certain feedback to debug and further enhance the optimization procedure. However, the minor inconsistency between a DNN and its optimized version is often hard to detect and easily bypasses the original test set. This paper proposes *DiffChaser*, an automated black-box testing framework to detect untargeted/targeted disagreements between version variants of a DNN. We demonstrate 1) its effectiveness by comparing with the state-of-the-art techniques, and 2) its usefulness in real-world DNN product deployment involved with quantization and optimization.

1 Introduction

Deep Learning (DL) has achieved tremendous success in many cutting-edge real-world applications such as image processing [Ciregan *et al.*, 2012], speech recognition [Hinton *et al.*, 2012] and autonomous driving [Huval *et al.*, 2015]. While the advances in system-on-chip (SoC) technologies greatly improved the performance of mobile devices (*e.g.*, smartphones, IoT edge computing device) over the past decade, directly deploying a complex DL model on a high-end mobile device could still lead to huge computational overhead and energy consumption. The current best practices oftentimes leverage model quantization and compression (QC) [Cheng *et al.*, 2017] with the intention to reduce the model complexity, enhance the model runtime execution performance while preserving the prediction accuracy to the best extent.

With currently urgent industrial demands to apply deep neural network (DNN) to mobile devices, quite a few QC techniques [Polino *et al.*, 2018; Wu *et al.*, 2016] were proposed, and the test data quality of QC process is becoming a pressing concern. The QC process relies on a test data set to measure to what extent the performance (*e.g.*, prediction accuracy) is preserved, which makes the quality of the test data set of great importance. Without high-quality test data, it is would still be uncertain whether a DNN that achieved high accuracy after QC indeed functions as expected. High-quality test data that effectively captures the disagreement behaviors of a DNN and its deployment version variant at an early stage could provide valuable feedback to developers for debugging and further enhancing the QC technique.

In practice, it is challenging to capture the disagreement as an optimized DNN variant is often very similar to the original DNN. In this paper, we propose *DiffChaser*, an automated black-box disagreement detection technique for multiple variants of a DNN. Our key observation is that the decision boundaries between a DNN and its QC version variant are often quite similar. Therefore, the inputs near the decision boundary are more likely to capture the differences between the decision boundaries, *i.e.*, the disagreement of the DNN models. *DiffChaser* automatically generates such inputs based on *prediction uncertainty*[1] of the model.

To demonstrate the effectiveness and usefulness of *DiffChaser*, we evaluate and compare it with the state-of-the-art techniques. The results confirm that *DiffChaser* can generate much more disagreements efficiently. We further apply *DiffChaser* on real products, *i.e.*, *TensorFlow Lite* and *CoreML*, the results demonstrate that *DiffChaser* can generate disagreements with a high success rate. Finally, we investigate the effectiveness of our approach in generating targeted disagreements. The results show that *DiffChaser* achieves 85.56% and 100% success rate on LeNet-5 and ResNet-20 models, respectively.

The contributions of the paper are summarized as follows.

- We propose a black-box testing framework for detecting disagreements of multiple models.
- We evaluate the effectiveness of our approach, which

*Lei Ma and Haijun Wang are the corresponding authors, E-mail:malei@ait.kyushu-u.ac.jp, haijun.wang@ntu.edu.sg

[1]Note that, in this paper, the *prediction uncertainty* is different from the *model uncertainty* of Bayesian neural network perspective (e.g., [Gal and Ghahramani, 2016]). In particular, our *prediction uncertainty* refers to that a model is not certain about its decision based on the predictive output vector.

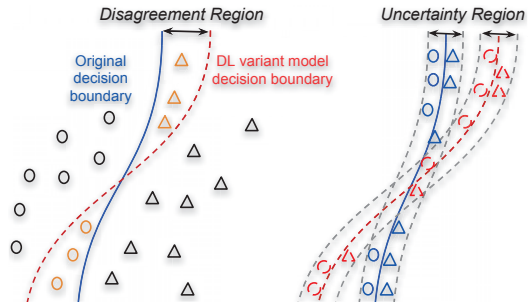

Figure 1: A comparative view on the relationship of disagreement region and uncertainty region.

demonstrates that *DiffChaser* achieves competitive results compared with the state-of-the-art techniques.

- We further demonstrate that *DiffChaser* can generate targeted disagreements effectively.

2 Related Work

2.1 Adversarial Attack

Adversarial attacks generate minor perturbed examples that are close to its original counterpart but wrongly classified by a DNN. A number of adversarial attack techniques have been proposed with representative algorithms, *e.g.*, FGSM [Goodfellow *et al.*, 2015], BIM [Kurakin *et al.*, 2017], JSMA [Papernot *et al.*, 2016], CW [Carlini and Wagner, 2017].

Compared with the adversarial attacks, our method intends to generate test data to uncover the disagreements of multiple DNNs, instead of finding adversarial examples that are misclassified by a DNN. Actually, the disagreement inputs generated by *DiffChaser* are also adversarial examples for one of the models since it is misclassified by at least one model.

2.2 DNN Testing

The current DNN testing mostly focuses on designing testing criteria [Ma *et al.*, 2018a] and test generation techniques [Tian *et al.*, 2018; Xie *et al.*, 2019; Du *et al.*, 2019; Ma *et al.*, 2019; Ma *et al.*, 2018b] for a single version of DNN. DeepXplore [Pei *et al.*, 2017] and TensorFuzz [Odena and Goodfellow, 2018] are two techniques that are most relevant to ours. DeepXplore proposes a white-box differential testing technique to detect behavior inconsistencies among multiple DNNs. In DeepXplore, the behavior difference is encoded as an optimization function, and a gradient-search method is used to detect the behavior difference. However, DeepXplore needs access to the neural network structure and weights of a DNN, which may not always be available. TensorFuzz proposes a coverage-guided testing technique to detect the disagreement inputs between models and the quantized versions. An input is regarded as interesting and is preserved if it makes the DNN in a new 'state'. The state is represented by the logits or the layer outputs before the logits. TensorFuzz mainly considers testing one model, where the 'state' is also coarse in terms of finding the minor disagreement.

3 Disagreements Generation

3.1 Problem Definition

Definition 1 (DNN) *A deep neural network is defined as a classifier $f : \mathcal{X} \rightarrow \mathcal{Y}$, where $\mathcal{X}$ is a set of domain-specific inputs and $\mathcal{Y}$ is a set of classification labels. We refer $|\mathcal{Y}|$ to the number of classification labels.*

Given two DNNs f_1 and f_2, and an input x that is classified as y by both models, *i.e.*, $f_1(x) = f_2(x) = y$, our goal is to generate a disagreement input x_D, based on x, such that $(f_1(x_D) \neq y \wedge f_2(x_D) = y) \vee (f_2(x_D) \neq y \wedge f_1(x_D) = y)$ (untargeted disagreement) or $(f_1(x_D) = t \wedge f_2(x_D) = y) \vee (f_1(x_D) = y \wedge f_2(x_D) = t)$ (targeted disagreement) where t is the targeted class and $t \neq y$. The disagreement input x_D is close to the original input x in terms of L_p-norm, *i.e.*, $||x_D - x||_p < d$, where d is a safe radius.

Fig. 1 shows the basic idea of our approach. The two DNN models are to classify the input as *circle* or *triangle*, and the red solid line and blue dotted line are their decision boundaries, respectively. Black circles and triangles represent the labels that are classified as the same by both DNNs. The intersection of two decision boundaries shows the disagreement region, where two DNNs produce different classification labels (see data in orange color on the left of Fig. 1). Consider DNNs with similar behaviors, *e.g.*, an original model and its quantized version, whose decision boundaries are rather close, therefore, their disagreement region should be small as well.

An input that falls into the small region should be close to either or both boundaries. The predictive probabilities of the two classes tend to be close when the input is close to the decision boundary. Hence, a DNN is confused to classify those inputs that are distributed in the uncertainty region. In other words, the disagreement region and uncertainty region should often overlap. When the uncertainty region is small to a particular level, the inputs in the uncertainty region are more likely to fall into the disagreement region. Based on this observation, the disagreement detection problem could be converted to the generation of inputs that cover the uncertainty region where the decisions of DNNs disagree.

3.2 Prediction Uncertainty

For a DNN f, an input x and a label $y \in \mathcal{Y}$, we use $P_f(x, y)$ to denote the probability that x is classified as y by f.

Definition 2 (Prediction Uncertainty) *Given a DNN f and an input x such that $f(x) = y$, the prediction of f against x is c-uncertain if $\exists y' \in \mathcal{Y}$, $|P_f(x, y') - P_f(x, y)| < c$, where $y' \neq y$, $c \in [0, 1]$ is an uncertainty threshold.*

Intuitively, the prediction uncertainty represents that the DNN is uncertain to classify the input x as y or y' because their predictive probabilities are very close. For example, Fig. 1 shows the prediction uncertainty between circles and triangles, denoted by uncertainty region whose size is decided by the threshold c in Definition 2. The smaller the value of c in *c-uncertain* is, the closer the input x is to the decision boundary. The prediction uncertainty is also applicable to multi-classifications: $\exists \mathcal{S} \in 2^{\mathcal{Y}} \backslash \emptyset$ such that $\forall y_i \in \mathcal{S}$, $|P_f(x, y_i) - P_f(x, y)| < c$.

Algorithm 1: Disagreement generation

input : x: an input, f_1 and f_2: two DNN models
output: x_d: a disagreement example
const : m: size of population, r_2: mutation rate

1 Construct an initial population X from input x;
2 $iteration := 0$;
3 **while** *True* **do**
4 **if** *timeout or exceed the maximum iterations* **then**
5 **return** failed;
6 Calculate fitness values for the chromosomes in X;
7 **if** $\exists x_d \in X$ *such that* x_d *is a disagreement example* **then**
8 **return** x_d;
9 **for** $i \in [0, m)$ **do**
10 **if** $X[i]$ *has the best fitness value* **then**
11 **continue**;
12 Select two chromosomes x_1 and x_2 from X;
13 $X[i] := crossover(x_1, x_2)$;
14 $X[i] := mutate(X[i], r_2)$;
15 $iteration+ = 1$;

We emphasize again that the *prediction uncertainty* is different with the *model uncertainty* in previous work [Gal and Ghahramani, 2016] that measures the uncertainty based on the probability distributions over weights (from Bayesian perspective). Differently, in this paper, *prediction uncertainty* directly measures the uncertainty on distinguishing between multiple classes for a specific DNN (from the single point).

3.3 Uncertainty Input Generation

To effectively generate uncertainty inputs, our key idea is to generate tests to satisfy the *c-uncertain* condition, after which we further reduce the value of the uncertainty threshold c (*i.e.*, narrow the uncertainty region) gradually, and continue to generate more fine-grained tests until the generated inputs fall into the disagreement region of DNNs. In this way, the problem is transformed to minimize the uncertainty function $|P_f(x, y') - P_f(x, y)|$.

We adopt the genetic algorithm (GA) [Mitchell, 1998] to solve this optimization problem. Algorithm 1 shows the GA-based procedure that generates the disagreement example. We illustrate the key components of the algorithm as follows.

Population Construction

In this paper, we focus on the image processing domain[2]. For the encoding of GA, we consider a whole image as a chromosome, and each pixel as a gene. Given an image x, we generate a new image x' through random perturbation of x with white noise. The L_∞ norm is used to constrain the difference between original and new images, *i.e.*, $||x - x'||_\infty < d$. As the initial step (Line 1), we randomly generate m images as the initial population.

[2]Our technique is general and can be easily extended to other domains, *e.g.*, audio, video, natural language processing.

Fitness Function

Suppose the DNN f is a n-class classifier. We compute the fitness value (Line 6) based on the *logits* which are the inputs to the softmax layer for final output [Carlini and Wagner, 2017]. In particular, the logits of a DNN on the input x are a n-dimensional vector $(l_0, l_1, \ldots, l_{n-1})$. We use $l_x(i)$ to represent the logit for the i^{th} class (*i.e.*, l_i), where $0 \leq i < n$. We use $T_x^f(i)$ to represent the i^{th} largest value in the logits of x in each iteration of GA. Specifically, $T_x^f(0)$ represents the maximum value of the logits and $T_x^f(n-1)$ is the minimum of the logits. Note that $T_x^f(\cdot)$ is recalculated in each iteration, and a k-th largest value might change in different iterations.

In general, the fitness values are designed based on the optimization objective. In the paper, we provide three different fitness functions towards achieving our goals. Given a DNN f and an input x, we compute the fitness value as follows:

- *Basic Fitness.* The fitness value is computed as $|T_x^f(0) - T_x^f(1)|$.
- *k-Uncertainty Fitness (k-UF).* The fitness value is computed as $|T_x^f(0) - T_x^f(k)|$, where $0 < k < n$ and n is the total number of classes.
- *t-Targeted Fitness.* Given a target class t, the fitness value is computed as $|T_x^f(0) - l_x(t)|$.

The intuition of these fitness function is that the smaller the fitness value is, the higher uncertainty of the prediction is. *Basic Fitness* is a special case of k-UF, *i.e.*, 1-UF. Specifically, *Basic Fitness* is designed to find an input that is close to the boundaries of top two classes. The *k-Uncertainty Fitness* aims to find the input that is very close to boundaries of the top k+1 classes. When $|T_x^f(0) - T_x^f(k)|$ is small, it implies that $\forall 0 < i < k-1$, $|T_x^f(0) - T_x^f(k)|$ is also small. The *t-Targeted Fitness* aims to find the disagreement input that is classified as the targeted class t.

Selection, Crossover and Mutation

For chromosome selection, we adopt the tournament strategy, where chromosome with the best fitness value in each set is selected for crossover. The crossover randomly exchanges the pixel values under the correspondence index. In the next step, the mutator randomly changes each pixel to the value from 0 to 255 with a predefined mutation rate r_2.

Population Maintenance

The uncertainty inputs generated for each of the two DNN models could be the disagreement examples. If both models are accessible, the population is divided into two equal sub-populations. Each sub-population is used to generate uncertainty inputs for a DNN. If only one model is accessible, the population is only constructed for this model. For example, to detect the disagreement between a cloud-trained model and its quantized version for mobile deployment, if the mobile device has privacy constraints that make the DNN model logits unable to easily obtain, we can only generate uncertainty inputs for the original DNN version trained in the cloud.

4 Experiments

To demonstrate the usefulness of our technique, we have implemented *DiffChaser* in Python based on Keras (ver.2.1.3)

with TensorFlow (ver.1.5.0) backend. We aim to investigate the following research questions:

- **Q1**: What are the effects of k-*Uncertainty Fitness* in guiding disagreement input generation under different configurations?
- **Q2**: How effective of *DiffChaser* is in generating disagreement inputs compared with the state-of-the-art techniques?
- **Q3**: Is *DiffChaser* effective in real products?
- **Q4**: Is *DiffChaser* useful for generating targeted disagreement inputs?

All the experiments were run on a server with the Ubuntu 16.04 system with 28-core 2.0GHz Xeon CPU, 196 GB RAM and 4 NVIDIA Tesla V100 16G GPUs.

4.1 Datasets and Models

We select two popular publicly available datasets (MNIST [LeCun and Cortes, 1998] and CIFAR-10 [Krizhevsky *et al.*, 2014]) as the evaluation subjects. For each dataset, we study the popular models used in previous work [Carlini and Wagner, 2017; Ma *et al.*, 2018a], which achieve competitive test accuracy. For MNIST, we select LeNet-1 and LeNet-5 [LeCun *et al.*, 1998], which achieve 97.6% and 99.0% test accuracy, respectively. For CIFAR-10, we select ResNet-20 [He *et al.*, 2016] which achieves 91.2% test accuracy.

For each selected model (originally in 32-bit floating point precision), we perform the quantization with 3 configurations to generate the quantized versions: (1) randomly sampling 1% of weights and truncating 32-bit floating point to 16-bit (1% quantized version), (2) randomly sampling 50% of weights and truncating 32-bit floating point to 16-bit (50% quantized version), and (3) truncating all weights from 32-bit floating point to 16-bit (100% quantized version). The quantized models with smaller quantization rate are often more close to the original model. In the evaluation, the size of the population is set to 1,000.

4.2 Results with Different Uncertainty Configurations (Q1)

For each dataset, we randomly select 50 seed inputs from the test data. Note that all seed inputs are correctly predicted in both of the original model and the quantized version. For each model, we select the original model and 100% quantized version as the subjects. For MNIST and CIFAR-10, the number of classes is 10. Hence, for k-*Uncertainty Fitness*, the value of k ranges from 1 to 9 and we have 9 different configurations. For each configuration, the maximum iteration (see Algorithm 1) is set to 100.

We compare the results from two metrics: 1) the time for generating the first disagreement input (FDI) for each seed, 2) the total number of the unique disagreement inputs (TDI) for each seed.

Fig. 2 shows the detailed box-plot results. The three subgraphs Fig. 2 (a)(c)(e) illustrate the FDI. The thick bars inside the boxes are the medians of time taken on generating the FDI for each seed image under different configurations. A

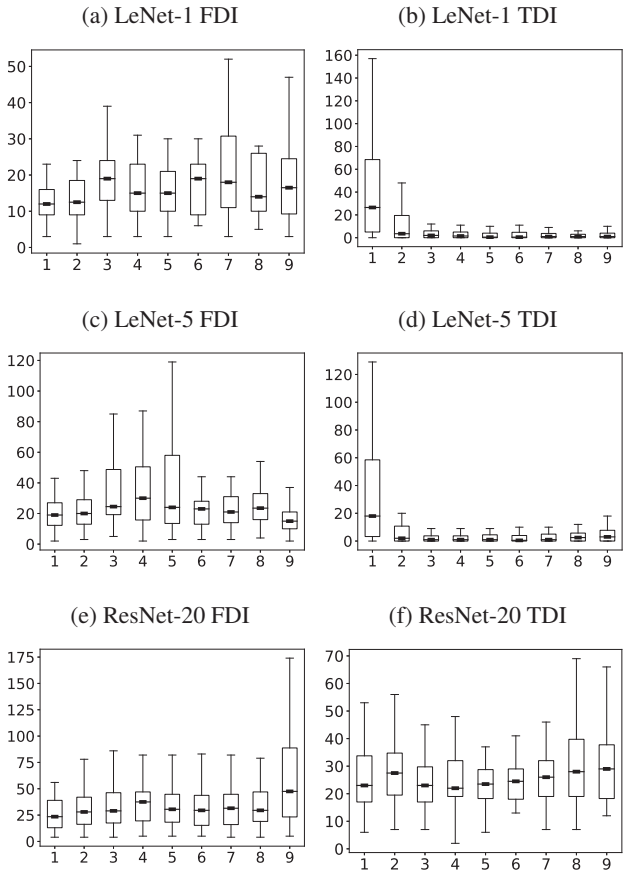

Figure 2: Time to detect the first disagreement (faster the better) and total number of unique disagreements detected (larger the better)

smaller value means that *DiffChaser* spends less time to detect the first disagreement. A smaller box value range means the performance is stable no matter which seed is provided. In general, we can see that *DiffChaser* performs better under the configuration 1-UF, which finds the first disagreement faster with a stable performance. In particular, we can see that for different models, the performance of *DiffChaser* varies. For example, in LeNet-1, the performance of *DiffChaser* is the most unstable under 7-UF. In LeNet-5, the configuration causing the most instability becomes 5-UF.

The three subgraphs Fig. 2 (b)(d)(f) illustrate the total number of unique disagreements generated for each seed. A larger median value means that *DiffChaser* finds more disagreements for each seed. Similarly, a smaller box value range means that a smaller variation for the number of disagreements generated from each seed. In general, *DiffChaser* exhibits an obvious advantage in generating more disagreements under 1-UF for LeNet-1 and LeNet-5. Specifically, in terms of TDI, 1-UF helps to generate much more (2X+) unique disagreement inputs in LeNet models than other configurations. This is because 1-UF considers the uncertainty between the two classes, which is relatively easier than other configurations. However, such advantage disappears when applied to ResNet-20. In fact, the performance of *DiffChaser*

Model	Q.R(%)	Success Rate			Total Time (s)		
		DC	DX	TF	DC	DX	TF
LN-1	1	**92%**	0%	0%	**3,160**	12,000	12,000
	50	**100%**	3%	0%	**1,368**	11,701	12,000
	100	**100%**	1%	0%	**1,203**	11,917	12,000
LN-5	1	**95%**	6%	0%	**3,103**	11,380	12,000
	50	**100%**	4%	1%	**1,381**	11,659	11,987
	100	**100%**	4%	2%	**1,318**	11,585	11,907
RN-20	1	**82%**	0%	5%	**6,755**	12,000	11,855
	50	**100%**	7%	35%	**3,216**	11,514	9,988
	100	**100%**	21%	40%	**2,620**	11,512	9,462

Table 1: Success rate and performance of generating disagreements

Model	Metrics	TensorFlow Lite		CoreML		
		Conv.	Quan.8	Conv.	Quan.16	Quan.8
LN-1	Succ.Rate	15%	98%	89%	84%	90%
	Time(s)	11,181	640	3,414	4,065	3,102
LN-5	Succ.Rate	18%	99%	85%	85%	90%
	Time(s)	11,320	735	4,035	3,890	3,131
RN-20	Succ.Rate	4%	100%	100%	100%	100%
	Time(s)	17,887	3,449	3,580	3,366	2,286

Table 2: Results on models with and without quantization using TensorFlow Lite and CoreML

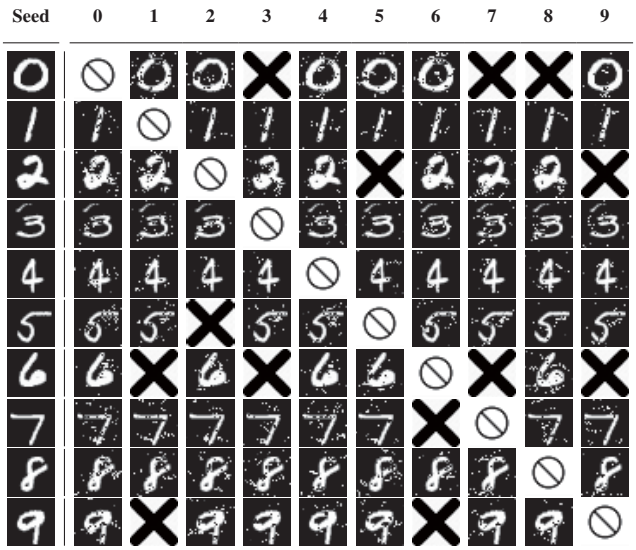

Figure 3: Targeted disagreement examples generated for LeNet-5

in ResNet-20 does not vary much under different configurations. ResNet-20 is a more complex model than LeNet models, on which the disagreements generation is much easier. Thus, all configurations achieve similar results.

In summary, *DiffChaser* can generally detect the disagreement efficiently under 1-UF at the cost of finding fewer variations of disagreements (*i.e.*, low diversity).

4.3 Comparison with the Baselines (Q2)

We compare *DiffChaser* with the state-of-the-art techniques, *i.e.*, *DeepXplore* and *TensorFuzz*, which adopt white-box and black-box approach, respectively. We slightly changed the *TensorFuzz* to add support for Keras models. We follow the configurations [Odena and Goodfellow, 2018; Pei *et al.*, 2017] to run the selected tools. For *DiffChaser*, we select the 1-UF as the fitness function. For each model, we select three quantized models (*i.e.*, 1%, 50%, and 100% quantized version) as the subjects. For each dataset, we randomly select 100 seed inputs from the test data. Each seed is predicted with the same result by the original model and quantized models. To perform the fair comparison to the best extent, each seed was run with a timeout (*i.e.*, 120 seconds). If a tool generates a disagreement for the seed, it returns the success status and processes the next seed. We use the success rate (*i.e.*, the number of seeds based on which the disagreements are found) to compare the results.

Table 1 summarizes the overall results. Column *Q.R.* refers to the quantized version with different quantization rates. In the columns *Success Rate* and *Total Time*, we list the success rates and the total time cost using *DiffChaser*, *DeepXplore* and *TensorFuzz*, represented by *DC*, *DX* and *TF*, respectively.

The results show that *DiffChaser* generates disagreements successfully for all seeds (100%) under the 50% and 100% quantized version. For 1% quantized version, where only 1% weights of the original model are truncated, *DiffChaser* can still generate disagreements for most of the seeds. Specifically, the success rates are 92%, 95% and 82% for LeNet-1, LeNet-5 and ResNet-20, respectively. For the white-box technique *DeepXplore*, it fails to generate disagreements for most seeds. For example, *DeepXplore* cannot generate any disagreements under 1% quantized versions of LeNet-1 and ResNet-20. The highest success rate is only 21% on the model ResNet-20 with quantization rate of 100%.

For the black-box technique*TensorFuzz*, it fails to generate any disagreements for all quantized versions of LeNet-1 and the 1% quantized version of LeNet-5. For the other two quantized versions (*i.e.*, 50% and 100%) of LeNet-5, the success rate is 1% and 2%. For larger model ResNet-20, it generates more disagreements. However, the best case is 40% in 100% quantized version. The results of *TensorFuzz* and *DiffChaser* also show that when the quantization rate increases, the difference between the original model and the quantized model tends to become larger. Thus the disagreements could be found more easily by the black-box techniques.

In addition, we evaluate the time cost in generating such disagreements. In general, *DiffChaser* is more efficient to generate disagreements than other tools. The time cost in *DeepXplore* and *TensorFuzz* is rather expensive because there are many timeout cases. Consider the successful cases only, we find that *DiffChaser* is also more efficient. For example, in the 100% quantized version of ResNet-20, on average, *DiffChaser* takes about 26.2 seconds to generate the first disagreement for each successful seed, while *DeepXplore* and *TensorFuzz* take about 98.85 and 56.55 s, correspondingly.

In summary, *DiffChaser* substantially outperforms the state-of-the-art techniques. In particular, it can generate more disagreements with higher efficiency.

4.4 Results on Real Products (Q3)

We also apply *DiffChaser* to two real products *TensorFlow-Lite* and *CoreML*, which are among the most popular tools for DNN model migration to Android and iOS platforms, re-

spectively. *TensorFlow Lite* and *CoreML* provide different quantization options to quantize models. Since the *CoreML* models are currently only supported in iOS platforms, therefore, the *CoreML* experiments were conducted on a MacBook Pro with 2.7GHz Intel Core i7 CPU with 16GB RAM. We select 100 seed inputs with 180 seconds as the timeout.

Table 2 shows the overall experiment results in terms of success rate and time cost. Column *Conv.* represents that the original model is converted to the platform-specific model without quantization. Column *Quan.8* represents the original model quantized from 32-bits to 8-bits of floating precision. Similarly, Column *Quan.16* corresponds to the original model quantized from 32-bits to 16-bits of floating precision. Ideally, the model conversion without quantization should preserve the behavior of the original model.

To our surprise, for the models converted without quantization, *DiffChaser* can still find disagreements. For example, for *TensorFlow Lite*, *DiffChaser* produces the disagreements for 4% to 15% of the seed inputs. For *CoreML*, *DiffChaser* finds disagreements for 85% to 100%. The possible reason is that the conversion still introduces some differences due to implementation issues or platform differences, *e.g.*, the different floating precision between iOS and Ubuntu.

For the quantization models, *DiffChaser* is highly effective in generating disagreements. For example, in the case of quantized models of *TensorFlow Lite*, *DiffChaser* achieves 98% to 100% success rate in the selected models. For *CoreML* quantized models, *DiffChaser* generates more disagreements (90% upto 100% success rate) for the quantized models from 32bits to 8-bits.

Table 2 shows the the total time cost for each model. Consider the successful cases of quantized models of *TensorFlow Lite*, *DiffChaser* takes about 2.87, 5.61, 34.49 seconds for LeNet-1, LeNet-5 and ResNet-20. On *CoreML*, *DiffChaser* takes 14.67, 32.79 and 22.85 seconds for LeNet-1, LeNet-5 and ResNet-20, respectively.

In summary, the overall results confirm the usefulness of *DiffChaser* for disagreement detection in the real-world DNN model deployment.

4.5 Targeted Disagreements Generation (Q4)

In addition to untargeted disagreements generation, we also evaluate *DiffChaser*'s capability to generate targeted disagreement inputs (see Section 3.1). In some cases, it could be more difficult to generate targeted disagreements of a specific class than the untargeted disagreements.

We randomly select 10 seed inputs from the 10 different classes of the test data. For each seed, we generate the targeted disagreement for each of the other 9 labels except its original label. We set the maximum iterations to 300, and select the LeNet-5, ResNet-20, as well as their corresponding 100% quantized versions as the subjects.

Fig. 3 and Fig. 4 give the specific disagreement examples generated for LeNet-5 and ResNet-20, respectively. The first column shows the selected seed inputs. Each row lists the resulting targeted disagreement examples from the original seed to each of the corresponding labels. The symbol $\times$ represents that *DiffChaser* fails to generate disagreements for the target class and seed. In total, *DiffChaser* generates 77/90 (85.56%) and 90/90 (100%) disagreements for LeNet-5 and ResNet-20, respectively. We also collect the number of iterations (Algorithm 1) for generating each disagreement. On average, *DiffChaser* takes 75.9 iterations to generate a disagreement for the small model LeNet-5. For ResNet-20, it takes about 7.28 iterations for each disagreement. The results further confirm that it could easier to generate disagreements for larger models because the quantization on large models tends to introduce larger differences between the original model and the quantized version.

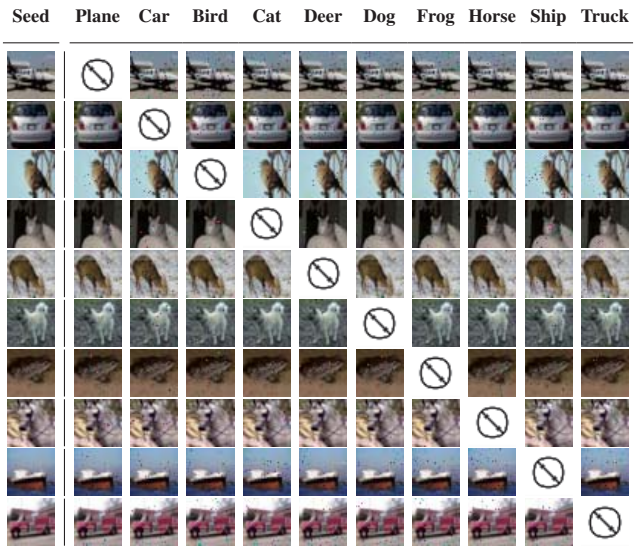

Figure 4: Targeted disagreement examples generated for ResNet-20

5 Conclusion

Platform migration and the existence of multiple DNN version variants (*e.g.*, introduced by model evolution) have become common during DNN development life-cycle, especially with the recent trends to deploy DNNs to diverse mobile devices, edge computing devices, *etc*. In this paper, we propose an automated genetic algorithm based testing technique, *DiffChaser*, to systematically generate tests towards covering disagreement of DNNs. Our in-depth evaluation and comparison with two state-of-the-art technique demonstrate the effectiveness of our technique in generating more disagreements efficiently. Its usefulness is also demonstrated for disagreement detection in quantization process of real-world DNN products.

Acknowledgments

This research was supported (in part) by the National Research Foundation, Prime Ministers Office Singapore under its National Cybersecurity R&D Program (Award No. NRF2018NCR-NCR005-0001), National Satellite of Excellence in Trustworthy Software System (Award No. NRF2018NCR-NSOE003-0001) administered by the National Cybersecurity R&D Directorate, and JSPS KAKENHI Grant 19H04086.

References

[Carlini and Wagner, 2017] Nicholas Carlini and David Wagner. Towards evaluating the robustness of neural networks. In *IEEE Symposium on Security and Privacy (S&P)*, pages 39–57, May 2017.

[Cheng *et al.*, 2017] 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.

[Ciregan *et al.*, 2012] Dan Ciregan, Ueli Meier, and Jürgen Schmidhuber. Multi-column deep neural networks for image classification. In *CVPR*, pages 3642–3649, 2012.

[Du *et al.*, 2019] Xiaoning Du, Xiaofei Xie, Yi Li, Lei Ma, Yang Liu, and Jianjun Zhao. Deepstellar: Model-based quantitative analysis of stateful deep learning systems. In *The 27th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2019.

[Gal and Ghahramani, 2016] Yarin Gal and Zoubin Ghahramani. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *international conference on machine learning*, pages 1050–1059, 2016.

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

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

[Hinton *et al.*, 2012] Geoffrey Hinton, Li Deng, Dong Yu, George E Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara N Sainath, et al. Deep Neural Networks for Acoustic Modeling in Speech Recognition: The Shared Views of Four Research Groups. *IEEE Signal Processing Magazine*, 29(6):82–97, 2012.

[Huval *et al.*, 2015] Brody Huval, Tao Wang, Sameep Tandon, Jeff Kiske, Will Song, Joel Pazhayampallil, Mykhaylo Andriluka, Pranav Rajpurkar, Toki Migimatsu, Royce Cheng-Yue, Fernando Mujica, Adam Coates, and Andrew Y. Ng. An empirical evaluation of deep learning on highway driving. *CoRR*, abs/1504.01716, 2015.

[Krizhevsky *et al.*, 2014] Nair Krizhevsky, Hinton Vinod, Christopher Geoffrey, Mike Papadakis, and Anthony Ventresque. The cifar-10 dataset. http://www.cs.toronto.edu/kriz/cifar.html, 2014.

[Kurakin *et al.*, 2017] Alexey Kurakin, Ian Goodfellow, and Samy Bengio. Adversarial examples in the physical world. *ICLR*, 2017.

[LeCun and Cortes, 1998] Yann LeCun and Corrina Cortes. The MNIST database of handwritten digits, 1998.

[LeCun *et al.*, 1998] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proc. of the IEEE*, 86(11):2278–2324, 1998.

[Ma *et al.*, 2018a] Lei Ma, Felix Juefei-Xu, Jiyuan Sun, Chunyang Chen, Ting Su, Fuyuan Zhang, Minhui Xue, Bo Li, Li Li, Yang Liu, et al. Deepgauge: Multi-granularity testing criteria for deep learning systems. *The 33rd IEEE/ACM International Conference on Automated Software Engineering (ASE 2018)*, 2018.

[Ma *et al.*, 2018b] Lei Ma, Fuyuan Zhang, Jiyuan Sun, Minhui Xue, Bo Li, Felix Juefei-Xu, Chao Xie, Li Li, Yang Liu, Jianjun Zhao, et al. Deepmutation: Mutation testing of deep learning systems. In *2018 IEEE 29th International Symposium on Software Reliability Engineering (ISSRE)*, pages 100–111. IEEE, 2018.

[Ma *et al.*, 2019] Lei Ma, Felix Juefei-Xu, Minhui Xue, Bo Li, Li Li, Yang Liu, and Jianjun Zhao. Deepct: Tomographic combinatorial testing for deep learning systems. In *2019 IEEE 26th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pages 614–618. IEEE, 2019.

[Mitchell, 1998] Melanie Mitchell. *An introduction to genetic algorithms*. MIT press, 1998.

[Odena and Goodfellow, 2018] Augustus Odena and Ian Goodfellow. Tensorfuzz: Debugging neural networks with coverage-guided fuzzing. *arXiv preprint arXiv:1807.10875*, 2018.

[Papernot *et al.*, 2016] Nicolas Papernot, Patrick McDaniel, Somesh Jha, Matt Fredrikson, Z Berkay Celik, and Ananthram Swami. The limitations of deep learning in adversarial settings. In *Security and Privacy (EuroS&P), 2016 IEEE European Symposium on*, pages 372–387. IEEE, 2016.

[Pei *et al.*, 2017] 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*, pages 1–18, 2017.

[Polino *et al.*, 2018] Antonio Polino, Razvan Pascanu, and Dan Alistarh. Model compression via distillation and quantization. *arXiv preprint arXiv:1802.05668*, 2018.

[Tian *et al.*, 2018] Yuchi Tian, Kexin Pei, Suman Jana, and Baishakhi Ray. Deeptest: Automated testing of deep-neural-network-driven autonomous cars. In *Proceedings of the 40th International Conference on Software Engineering*, pages 303–314. ACM, 2018.

[Wu *et al.*, 2016] Jiaxiang Wu, Cong Leng, Yuhang Wang, Qinghao Hu, and Jian Cheng. Quantized convolutional neural networks for mobile devices. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4820–4828, 2016.

[Xie *et al.*, 2019] Xiaofei Xie, Lei Ma, Felix Juefei-Xu, Minhui Xue, Hongxu Chen, Yang Liu, Jianjun Zhao, Bo Li, Jianxiong Yin, and Simon See. Deephunter: A coverage-guided fuzz testing framework for deep neural networks. In *28th ACM SIGSOFT International Symposium on Software Testing and Analysis*, 2019.