ROBUSTNESS AND RESILIENCE OF DEEP NEURAL NETWORKS

by

ABDULLAH MURAT BULDU

B.S., Computer Engineering, Boğaziçi University, 2019

Submitted to the Institute for Graduate Studies in
Science and Engineering in partial fulfillment of
the requirements for the degree of
Master of Science

Graduate Program in Computer Engineering
Boğaziçi University

2022

ACKNOWLEDGEMENTS

I would like to thank my advisor Prof. Alper Sen for guiding me from the start. I also thank the jury members, Prof. Sıddaka Berna Örs Yalçın and Assist Prof. İnci Meliha Baytaş, for their feedback.

This research was supported in part by Semiconductor Research Corporation under task 2020-AH-2970.001. I would like to thank Brian Kahne and Karthik Swaminathan for their feedback in SRC meetings.

ABSTRACT

ROBUSTNESS AND RESILIENCE OF DEEP NEURAL NETWORKS

Deep Neural Networks (DNN) are used extensively to solve challenging problems in computer vision, natural language processing, and speech recognition. However, recent studies such as adversarial attacks show that high accuracy is not enough to ensure the performance of DNNs. Additionally, deployment of DNN models on edge devices requires high resilience against bit errors in the DNN model. Therefore, robustness and resilience improvement methods are necessary. However, there is no study that discusses these methods together.

In this thesis, we compare and analyze the effect of robustness and resilience improvement methods on resilience and robustness, respectively. We use adversarial training and bit error training as representatives of robustness and resilience improvement methods. We also introduce adversarial and bit error training, a combined training method of adversarial training and bit error training. For robustness, we compare test accuracy and robust accuracy of four trained DNN models. For resilience, we compare the performance against random bit errors with different bit error rates of four trained DNN models. The results show that resilience improvement methods improve the robustness, while the robustness improvement method can cause a decrease in resilience due to the test accuracy drop of models trained with adversarial training.

We propose multiple bit error training (MBET), that utilizes more than 1-bit error rates inside the loss function during the training. We test MBET with four different DNN models on two datasets. The results show that MBET improves resilience and robustness compared to normal training.

ÖZET

DERİN ÖĞRENME AĞLARININ SAĞLAMLIĞI VE DAYANIKLILIĞI

Derin Sinir Ağları (DNN), bilgisayar görüşü, doğal dil işleme ve konuşma tanımadaki zorlu sorunları çözmek için yaygın olarak kullanılmaktadır. Bununla birlikte, rakip saldırılar gibi son araştırmalar, DNN'lerin performansını sağlamak için yüksek doğruluğun yeterli olmadığını göstermektedir. Ek olarak, uç cihazlarda DNN modellerinin kullanılması, DNN modelindeki bit hatalarına karşı yüksek dayanıklılık gerektirir. Bu nedenle, sağlamlık ve esneklik iyileştirme yöntemleri gereklidir. Ancak bu yöntemleri bir arada ele alan bir çalışma bulunmamaktadır.

Bu tezde, sağlamlık ve dayanıklılık iyileştirme yöntemlerinin dayanıklılık ve sağlamlık üzerindeki etkisini analiz ediyoruz. Sağlamlık ve dayanıklılık iyileştirme yöntemlerinin temsilcileri olarak rakip eğitim ve bit hatası eğitimini kullanıyoruz. Ayrıca rakip eğitimi ve bit hatası eğitimini birleştirip yeni bir öğrenme yöntemi, rakip ve bit hatası eğitimi sunuyoruz. Sağlamlık için, eğitilmiş dört DNN modelinin test doğruluğunu ve sağlam doğruluğunu karşılaştırıyoruz. Dayanıklılık için, dört eğitimli DNN modelinin farklı bit hata oranlarıyla rastgele bit hatalarına karşı performansını karşılaştırıyoruz. Sonuçlar, dayanıklılık geliştirme yöntemlerinin sağlamlığı iyileştirdiğini, sağlamlık geliştirme yönteminin ise rakip eğitim ile eğitilen modellerin test doğruluğundaki düşüş nedeniyle dayanıklılığın azalmasına neden olabileceğini göstermektedir.

Eğitim sırasında kayıp fonksiyonu içinde birden fazla bit hata oranı kullanan çoklu bit hata eğitimi (MBET) sunuyoruz. MBET'i normal eğitim ve bit hatası eğitimine karşı iki veri kümesinde dört farklı DNN modeliyle test ediyoruz. Sonuçlar, MBET'in normal eğitime kıyasla dayanıklılığı ve sağlamlığı geliştirdiğini göstermektedir.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS

B	Bit error rate list
BE	Bit error injection function
f	DNN model prediction function
J	Loss function
$\widetilde{J}$	Updated loss function
$\mathcal{L}$	Loss function of DNN model
$\mathcal{L}'$	Updated loss function of DNN model
m	Number of bits in quantization
Q	Quantization function
Q^{-1}	Dequantization function
S	Valid input domain
w	Weights of DNN model
$\widetilde{w}$	Bit error injected weights of DNN model
x	Input of DNN model
x'	Adversarial input
y	Ground truth of input
α	Perturbation step of PGD adversarial attack
ϵ	Perturbation budget of adversarial attack
η	Perturbation of adversarial attack
θ	DNN model parameters
λ	Normalization factor
∇_x	Gradient with respect to x

LIST OF ACRONYMS/ABBREVIATIONS

ADBET	Adversarial and Bit Error Training
ADV	Adversarial Training
BET	Bit Error Training
BIM	Basic Iterative Method
C&W	Carlini & Wagner
DL	Deep Learning
DNN	Deep Neural Network
FGSM	Fast Gradient Sign Method
GAN	Generative Adversarial Network
GPU	Graphical Processing Units
LRP	Layer-wise Relevance Propagation
MBET	Multiple Bit Error Rate Training
PGD	Projected Gradient Descent

1. INTRODUCTION

Deep learning (DL) systems have become more widespread thanks to technological developments. The accessible data has increased significantly with technological development. Graphics Processing Units (GPUs) have become accessible to people. As a result, people can access high computational power easily. These two main reasons set the stage for many researchers to conduct several experiments with deep neural network (DNN) models.

DNN models have become very dominant in many industrial domains. DNN models have high performance on challenging tasks, such as image classification [1–3], natural language processing [4–6], and speech processing [7–9]. These software solutions with DNN models can be applied to various industrial domains, such as autonomous driving [10, 11], fraud detection [12–14] and healthcare [15, 16].

Applications on safety-critical domains require high correctness. Any faults in these applications can lead to a failure in the system. These failures can cause casualties and huge financial losses to companies. DNN models have become widespread in safety-critical domains thanks to the high performance of DNN models. Despite the high performance of DNN models, recent accidents such as autonomous vehicle accidents [17] show that test accuracy is not enough to assess the performance of DNN models.

The robustness and resilience of DNN models aims to assess the performance of DNN model beyond the test accuracy. Figure 1.1 shows the study area of the robustness and resilience of DNN models and the purple circles represent perturbations on the inputs and bit errors on the DNN model weights. The robustness of DNN model studies the effect of perturbations on the input. The resilience of DNN model studies the effect of faults on weights of DNN model.

Adversarial inputs are inputs generated by adding small perturbations on original

inputs to change the prediction of DNN model and the generation methods are called adversarial attacks [18–21]. Although DNN model have high test accuracy, adversarial attacks can find imperceptible perturbations which cause DNN model to make misclassifications [18, 22, 23]. The robustness of DNN model aims to understand and improve the performance of DNN model against adversarial inputs [19, 21, 24, 25].

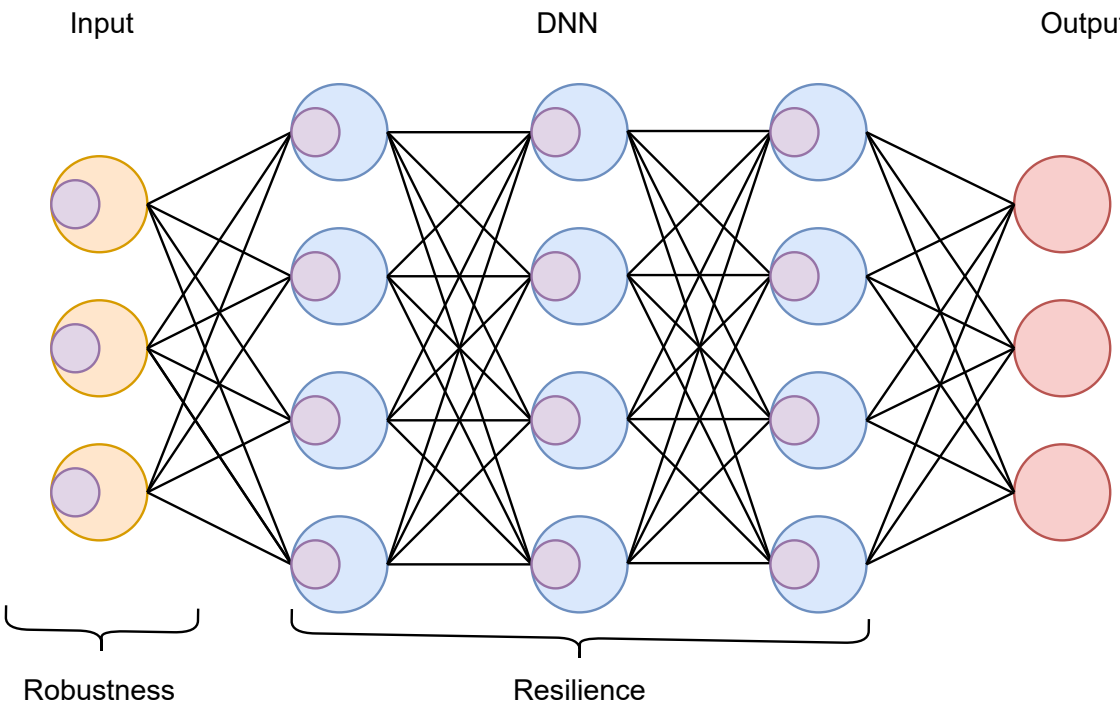


Figure 1.1. Illustration of the robustness and resilience of DNN models.

The resilience of DNN model focuses faults on weights of DNN models and operations on hardware [26], while the robustness of DNN model focuses on perturbations on inputs. The high performance of DNN models has attracted many industrial fields and the DNN models have been deployed on edge devices with additional constraints, such as low precision and low operating voltage [27–31]. These constraints can decrease the performance of DNN model [31]. The resilience of DNN models aims to understand and improve the performance of DNN model against faults caused by these conditions [31–34].

Although adversarial robustness and resilience are widely studied separately, no work considers both approaches together. We use adversarial training as a robustness improvement method and bit error training as a resilience improvement method. We analyze the effect of the robustness improvement method on resilience and the effect of the resilience improvement method on robustness. To understand their effect on each other, we combine both training methods and present adversarial and bit error training. We compare these three training methods, adversarial training, bit error training, and adversarial and bit error training, with normal training as a baseline. We use robust accuracy against an adversarial attack, FGSM for robustness evaluation, and bit error injection with different bit error rates for resilience evaluation.

In addition to the relation between robustness and resilience discussion, we propose a resilient improvement method, multiple bit error rate training (MBET). MBET is an improved method of BET which utilizes more than a 1-bit error rate during the training. We compare MBET with normal training and bit error training to test the performance. The main contributions of the thesis as follows:

- We discuss and analyze the effect of robustness and resilience improvement methods on robustness and resilience of DNN models.
- We present a combined version of adversarial training and bit error training as adversarial and bit error training.
- We compare the results of adversarial training, bit error training, and adversarial and bit error training against normal training, as baseline.
- We also propose a resilience improvement, multiple bit error rate Training (MBET).
- MBET improves bit error training by using more than 1-bit error rate during the training.
- We use four state-of-the-art models, MobileNet, ShuffleNet, ResNet-18 and ResNet-50 on two public image classification datasets, CIFAR10 and CIFAR100.
- Bit error training increases robustness of DNN model significantly.
- Adversarial training can cause a decrease on resilience of DNN model due to initial accuracy drop.

- Adversarial and bit error training can improve resilience at higher bit error rates of the DNN model.
- MBET increases the resilience more than bit error training.

The rest of the thesis is as follows. In Chapter 2, we present related work on the robustness and resilience of DNN models. In Chapter 3, we provide background information related to our methods. In Chapter 4, we describe our methods, adversarial and bit error training, and multiple bit error rate training. In Chapter 5, we discuss the experimental setups and experimental results. In Chapter 6, we conclude this thesis.

2. RELATED WORK

We discuss works related to the robustness and resilience of DNN models in this chapter.

The robustness of DNN models is related to the performance of the DNN model for all the valid inputs. There are different methods to assess the robustness of DNN models. One method is the performance under corner case scenarios. DeepTest [35] proposes a method to generate new inputs for autonomous driving problem. DeepTest adds rain and fog to the test inputs to generate corner case scenarios. TACTIC [36] aims to identify which environmental conditions are more critical for autonomous driving systems. DeepRoad [37] generates driving scenes with different weather conditions using generative adversarial networks (GANs) [38]. In addition to transformations related to real world constraints, adversarial robustness [39] studies to find imperceptible perturbations to change the prediction of DNN models.

Adversarial attacks [18] can generate ill inputs on which the DNN model makes false predictions from original test inputs. These methods show the vulnerability of the DNN models even though the DNN models have high test accuracy. Fast gradient sign method (FGSM) [19] is a fast and efficient adversarial attack method. FGSM uses first gradient of the loss function and L_∞ metric to generate adversarial inputs. C&W attacks [40] minimize the perturbation length and loss function of the perturbed input to generate adversarial samples and target defensive distillation method [24] which is an adversarial defense method. Basic iterative method (BIM) [20] applies FGSM method iteratively with smaller perturbation budget so that we can adversarial inputs with smaller perturbations. Projected gradient descent (PGD) [21] upgrades FGSM with multiple iterations similar to BIM.

In addition to adversarial attacks, software testing methods are adapted to test DNN models to assess the DNN model performance. One workline is coverage met-

rics for DNNs. DeepGauge [41] proposes a set of coverage metrics to understand the performance of DNN model on different inputs. Coverage metrics divide the activated region and count the number of hits for each neuron in the DNN model. This method is similar to code coverage [42] where we count the number of lines we hit during testing. Adversarial attacks are used to verify the quality of coverage metrics. The experimental results show that adversarial samples increase the coverage metrics. Additionally, these coverage metrics can be used to determine the test dataset quality and we can check the similarity between training dataset and test dataset. Importance-driven coverage [43] uses layer-wise relevance propagation (LRP) [44] to determine the important neurons and divides the activation range into different number of clusters for each neuron with the Silhouette index [45]. Reluplex [46] is a verification method which improves Simplex method with relaxing ReLU layers. Reluplex can be used to calculate ϵ-robustness of DNN models locally and globally with calculating the lower bounds and upper bound. Although most work focus on feed forward networks, POPQORN [47] proposes a robustness verification method for LSTM and GRU layers.

The adversarial attacks and testing methods point out the need for improving robustness of DNN models Defensive distillation [24] trains two DNN models with the architecture. The former is trained with the ground truth labels of training dataset while the latter is trained with the predictions of the former DNN model. This method provides additional information from the entropy provided by the former DNN model. Input gradient regularization [25] uses the gradients of loss function with respect to input and aims to improve robustness against modifications on the input. Adversarial training [19] adds adversarial inputs into training to improve robustness of DNN models. Madry et al. (2018) [21] defines the problem as a joint maximization, which aims to minimize the effect of adversarial attacks. Both adversarial training methods require to use strong adversarial attacks to increase the robustness of DNN models.

The resilience of DNN models is how successful the DNN model is against fault on the DNN model weights and operations in the hardware. Aging and process variation can cause faults in DNN models. Additionally, hardware designs with additional

restrictions such as low voltage DNN accelerators [27–30] can lead to faults during inference. These low voltage DNN accelerator designs aim to minimize the accuracy drop due to low operating voltage. Chandramoorthy et al. (2019) [27] proposes a programmable voltage adjustment system for SRAM units so that we can fix the trade off between energy efficiency and accuracy drop. Thundervolt [28] proposes a dynamic voltage scaling per layer so that we can decrease the voltage aggressively for each layer of the DNN model while we take into account the accuracy drop. Minerva [30] proposes a fault mitigation method which round the faulty weights and an iterative method to select quantization granularity.

Fault injection framework is a method to test the resilience of DNN models by simulating the faults on DNN weights and operations in software. Ares [26] proposes a framework to quantify and test the resilience of DNN model against random faults on the DNN model weights. Ares simulates the effects of faults in software and injects bit errors uniformly on the DNN model weights. Ares is also validated on a DNN accelerator with different operation voltages and the results of silicon validation are similar to the results from Ares. BinFI [48] is a framework to determine the critical bits of the operations using monotonicity. If a fault occurs in the bits which is in the higher order than critical bits, DNN model makes false predictions. BinFI uses a binary search to find the critical bits of each operation.

Resilience improvement methods aim to improve the performance of DNN models against faults on DNN weights and operations on hardware [49]. Various training methods are proposed to improve the resilience of DNN models. Bit error training [32] updates the loss function and simulates the effect of bit errors during the training. Therefore, DNN model trained with bit error training becomes more resilient to bit errors. MATIC [31] proposes two methods to improve the resilience of DNN models. The former is memory-adaptive training, which updates the weights with quantization and bit injection masks during training. The latter selects bit-cells which fails at the target operating voltage, and aggressively decreases the operating voltage starting from a high default voltage until a failure occurs on selected bit-cells. In addition to training

methods, range restriction methods are proposed [34, 50] to improve the resilience of DNN models. Ranger [50] is a range restriction method for activation functions. Ranger has two options for range selection. The former uses the natural bound of the activation functions and the latter uses smaller bounds to increase resilience more while causing an accuracy drop. FT-ClipAct [34] updates ReLU activation function with clipping and zeroes a value if the value is higher than the threshold. Schorn et al. (2019) [33] proposes a weight normalization method on CNN layers to reduce the effect of bit errors on DNN model. Wu et al. (2020) [51] proposes a error aware quantization method and weight distribution method to mitigate the effect of faults. The proposed weight distribution method splits a weight into two weights so that the probability of the error decreases. The proposed error aware quantization method selects quantized values with minimum error probabilities.

3. BACKGROUND

We discuss preliminary work related to our thesis in this chapter. First, we describe the robustness of DNN models and robustness improvement methods, where we focus on adversarial training. Then, we discuss the resilience of DNN models and resilience improvement methods, where we focus more on bit error training.

3.1. Robustness of DNN Models

The robustness of the DNN model is how successful the DNN model performs on valid inputs. Although test accuracy is an important metric to assess the DNN model performance, recent work shows that test accuracy is not enough. One can trick the DNN model with small perturbations even if the DNN model has high test accuracy and adversarial robustness studies such perturbations. We focus on the adversarial robustness of DNN models in this work.

3.1.1. Adversarial Samples

Adversarial samples are inputs generated by adding imperceptible perturbations so that the golden class of the generated sample does not change, while the DNN model misclassifies the generated input [19]. Perturbations are limited with ϵ-ball with a distance metric, and the perturbation amount ϵ may vary depending on the dataset. The accuracy of the DNN model against adversarial samples is called robust accuracy.

3.1.2. Adversarial Attacks

Adversarial attacks are the methods to generate adversarial samples. An adversarial attack aims to generate an input that will be classified with the golden class by a human annotator, while the DNN model makes a misclassification. Therefore, the performance of the adversarial attacks is determined by the robust accuracy of the

DNN model. If the robust accuracy is lower, an adversarial attack is more successful.

Adversarial attacks can be categorized into two groups by their access to DNN model parameters. White box adversarial attacks can access the DNN model parameters, while black box adversarial attacks can not access the DNN model parameters. Black box adversarial attacks can use the same DNN model architecture or not but can not use the weights of the target DNN model.

Adversarial attacks can be categorized into two groups according to whether they target a specific class. Non-targeted adversarial attacks aim to make the DNN model wrong predictions, while targeted adversarial attacks aim to change the DNN model prediction into a specific class.

We will focus on Fast Gradient Sign Method (FGSM) and Projected Gradient Descent (PGD), which are gradient based adversarial attacks.

3.1.3. Fast Gradient Sign Method (FGSM)

Fast Gradient Sign Method (FGSM) is a gradient based adversarial attack method. FGSM generates the perturbation using the gradients of the loss function with respect to the input. The perturbation is calculated using the following formula:

$$\eta = \epsilon sgn(\nabla_x J(\theta, x, y)), \tag{3.1}$$

where x is input, y is golden target, θ is the DNN model parameters, $J(\theta, x, y)$ is the loss function and ϵ is the perturbation amount [19]. The adversarial sample is generated by adding the perturbations to the original image, as shown in Equation (3.2):

$$x' = x + \eta, \tag{3.2}$$

FGSM aims to find an effective perturbation using the loss function. Although we minimize the loss function during the training, FGSM aims to fail the DNN model by maximizing the loss function using the gradients with respect to the input. Since FGSM maximizes the loss function, FGSM can generate successful adversarial samples,

which the DNN model will misclassify. In addition to effectiveness, one other advantage of FGSM is speed. Since FGSM calculates the perturbation using only one backward pass on the DNN model, FGSM is a fast adversarial attack.

3.1.4. Projected Gradient Descent (PGD)

PGD is an iterative version of FGSM adversarial attack. Similar to FGSM, PGD is a gradient based adversarial attack. PGD generates adversarial samples with the following formula:

$$x^{t+1} = \prod_{x+S}(x^t + \alpha sign(\nabla_x J(\theta, x, y))), \tag{3.3}$$

where x^t is the generated input at step t, x^{t+1} is the generated input at step $t+1$, $x+S$ is the allowable input domain, $\prod_{x+S}$ is the projection operation to $x+S$ domain, α is the maximum perturbation amount for each step. α is smaller than ϵ where ϵ is the allowable perturbation amount. PGD aims to make small but multiple updates on the input with α amounts so that the total perturbation becomes more effective. Projection operation has two main purposes. The first one is to make sure that we do not perturb the input more than the total perturbation amount ϵ. Since the perturbations are cumulative with the amount α at each step, we can exceed the total perturbation amount ϵ. Therefore, PGD uses the projection operation to limit the maximum perturbation. The second one is the generated input should be a valid input. For instance, the input values range between 0 and 1 in the MNIST dataset. If the generated input contains values higher than 1 or lower than 0, then the generated input becomes invalid. PGD uses projection operation to limit the generated input to stay on the valid input domain.

PGD iteratively generates the adversarial samples. PGD tries to find the best perturbation step by step, similar to gradient descent methods. PGD updates the input with a small amount α in each step, and projects the updated input to the allowable input domain $x+S$. After each step, the perturbation becomes more powerful and effective. Therefore, the effectiveness of PGD is proportional to the number of itera-

tions. One downside is since PGD uses multiple iterations, PGD calculates multiple backward passes on the DNN model, and as a result, PGD is much slower than FGSM.

3.1.5. Adversarial Training

Adversarial training is a training method to improve the robustness of DNN models against adversarial samples. Adversarial training uses adversarial inputs during the training in addition to original inputs. Adversarial training uses the following formula as the loss function [19]:

$$\widetilde{J}(\theta, x, y) = \alpha J(\theta, x, y) + (1 - \alpha) J(\theta, x', y), \tag{3.4}$$

where x is the original input, x' is the adversarial sample generated from the original input x, $\widetilde{J}$ is the total loss function in adversarial training, and α is a coefficient between 0 and 1 and determines the proportion of the adversarial term in the total loss function.

The loss function of adversarial training in Equation (3.4) contains two terms. The first one is the loss value on the original input. The second one is the loss value on the adversarial sample. Two terms are aggregated with α and $1 - \alpha$ coefficients. If α is lower, adversarial samples become more dominant on the loss function. The default value for α is 0.5, where each term contributes equally.

Adversarial training is an effective method to increase robustness and robust accuracy, thanks to utilizing adversarial samples during training. However, since we incorporate inputs from a different distribution in adversarial training, the accuracy on the original inputs can drop slightly. This slight test accuracy drop can be acceptable due to high increase in robust accuracy.

3.2. Resilience of DNN Models

The resilience of DNN models is how successful the DNN model is against faults in the weights of the DNN model and operations in hardware. We focus on bit errors on DNN model parameters in this work.

3.2.1. Bit Errors on DNN models

Bit errors (faults) can be classified into two groups by their permanence [26]. Transient faults are not permanent and can be caused by abnormal conditions or events such as resonant supply voltage noise and particle strikes. On the other hand, static faults are permanent and can be caused by weak SRAM bit cells due to process variation or flash lifetime wear problems.

Bit errors can occur on DNN weights, activities, and hidden states during the inference. We focus on the static errors on the DNN model parameters, and we inject random bit errors to simulate the effect of static errors.

3.2.2. Quantization

Quantization converts the DNN model parameters and operations from floating point representation to a smaller precision. Fixed point quantization represents the original value with 2^m distinct values, where m is the number of bits in the quantization. Quantization divides the range of values into 2^m parts, and each value is mapped to the corresponding parts.

Quantization can be categorized into two parts according to range selection. Asymmetric quantization uses a different value for both minimum and maximum value, while symmetric quantization uses the same value for minimum and maximum values. Symmetric quantization selects the range with the maximum of the absolute of minimum and maximum values.

Quantization can be categorized into two parts according to how the quantization range is applied on the DNN model. Global quantization selects the range of quantization from the whole DNN model layers. However, layerwise (per layer) quantization selects the range of quantization for each layer. Since layerwise quantization uses a different range for each layer, layerwise quantization requires more parameters with

additional memory. However, one important advantage of layerwise quantization is the information loss due to the quantization can be lower in layerwise quantization than in global quantization, since we consider each layer separately in layerwise quantization.

The advantages of quantization are a decrease in inference time and an increase in energy efficiency. Since quantization converts parameters into smaller precision and integers, we use less memory to store the parameters of DNN models, and we can compute faster during the inference on specific hardware. Additionally, quantized DNN models use less memory access and, as a result, consume less computational energy. On the other hand, the performance of the DNN model can decrease due to quantization. Since we convert full precision parameters into smaller precision, we lose some information on the parameters learned during the training. As a result, the test accuracy of the quantized DNN model can decrease. [52] proposes a training method to offset the accuracy drop due to the quantization by simulating the effect of quantization during the training.

Integer quantization [52] allows the inference to be done using integer-only arithmetic. This method allows us to make inferences on integer-only hardware. We used integer quantization in the experiments.

3.2.3. Bit Error Training

Bit error training is a training method to increase the resilience of DNN models against random bit error injections. The idea of bit error training is to infuse bit error injections during the training so that the DNN model becomes more resilient against random bit error injections. Bit error training uses the following formula as the loss function [32]:

$$\mathcal{L}' = \mathcal{L}(f(x; \widetilde{w}), y) + \lambda \mathcal{L}(f(x; w), y), \tag{3.5}$$

where w is the original DNN model parameters, $\widetilde{w}$ is the bit error injected DNN model parameters, λ is the coefficient for the original loss function, $\mathcal{L}$ is the loss function and $\mathcal{L}'$ is the total loss function of bit error training method.

The loss function of bit error training contains two terms. The first term is the loss value of the original input on the bit error injected DNN model. We simulate the effect of a bit error injection and calculate the loss value using this bit error injected DNN model. Since we incorporate bit error injections during the training, the DNN model becomes more resilient against random bit error injections. The second term is the loss value of the original input on the original DNN model. The default value for λ is 1. While bit error training aims to increase the resilience of the DNN model, the performance of the original DNN model should stay satisfying. The second term aims to achieve this goal, not to decrease the test accuracy of the original DNN model. Then, the final loss value is obtained by adding these two loss terms.

The bit error injected DNN model is obtained by injecting random bit errors to the DNN model parameters. Random bit errors are generated from a uniform distribution with a bit error rate. The DNN model becomes more resilient against random bit error injections up to the selected bit error rate. Therefore, the bit error rate used during the training is a crucial hyperparameter of the bit error training method.

The bit error injections during the training are random. The bit error injections used in experiments differ from the bit error injections during the training not to cause any bias.

4. METHODOLOGY

In this chapter, we discuss the relationship between the robustness and resilience of DNNs. Although the robustness and resilience of DNN models are popular topics, there is no work which studies both robustness and resilience together. In this thesis, we want to emphasize the impact of the robustness improvement method on resilience and the resilience improvement method on robustness.

We choose training methods to improve the robustness and resilience of DNN models. We select adversarial training (ADV) as a robustness improvement method. and bit error training (BET) as a resilience improvement method. Additionally, we combine both ADV and BET into a single training method, namely adversarial and bit error training (ADBET) to analyze the relationship between them.

4.1. Adversarial and Bit Error Training (ADBET)

Adversarial and bit error training (ADBET) is a training method to improve robustness and resilience of DNN models. We combine both ADV and BET methods into a single training method. Adversarial training provides a great improvement in robustness. Similarly, bit error training provides a great improvement in resilience. We aim to test the robustness and resilience performance of the unified method. Both ADV and BET methods use an additional loss term. Therefore, we updated the loss function used in training.

The updated loss function is as follows:

$$Loss = \lambda_1 \mathcal{L}(f(x; w), y) + \lambda_2 \mathcal{L}(f(x'; w), y) + \lambda_3 \mathcal{L}(f(x; \widetilde{w}), y), \tag{4.1}$$

where $\mathcal{L}$ is the loss function, f is the model prediction function, x is input, w is parameters of the model, y is the target value, x' is adversarial input, $\widetilde{w}$ is the bit error injected model parameters and λ_1, λ_2 and λ_3 are normalization factors which are set to $1/3$.

The updated loss function in Equation (4.1) has two additional terms representing ADV and BET, respectively. The second term in the loss function is the loss of the model against adversarial input, x', and originates from ADV. Since we simulate the effect of adversarial inputs during the training, we expect to increase the robustness of the DNN model. Similarly, the third term in the loss function is the loss of the bit error injected model on the original input and originates from BET. Since we simulate the effect of bit error injections during the training, we expect to increase the resilience of the DNN model. Hence, we aim to increase the robustness and resilience of the DNN model with ADBET method. Additionally, we use the original loss function, the loss value of the original input on the DNN model, so that we can recover the test accuracy performance from a potential decrease due to having a more complex loss function.

4.2. Multiple Bit Error Rate Training (MBET)

Multiple Bit Error Rate Training (MBET) is a training method for DNNs to increase resilience. MBET is based on bit error training.

MBET utilizes more than 1-bit error rate during the training. The loss function contains an additional loss term for each bit error rate. Therefore, the DNN model can experience the effect of multiple bit error rates during the training and become more resilient against a wider range of bit error rates with MBET.

The updated loss function is as follows:

$$Loss = \lambda(\sum_{b \in B} \mathcal{L}(f(x; \widetilde{w}_b), y) + \mathcal{L}(f(x; w), y)), \tag{4.2}$$

where $\mathcal{L}$ is the loss function, f is the model prediction function, x is input, w is parameters of the model, y is the target value, B is bit error rate list, $\widetilde{w}_b$ is the bit error injected model parameters for bit error rate of b and λ is normalization factor which is set to $1/(|B| + 1)$.

The loss function in Equation (4.2) contains one extra term for the loss value of bit error injected DNN models in addition to the loss value on the original DNN model. A list of bit error rates, B is utilized in the loss function as hyperparameters. For each bit error rate in the bit error rate list B, bit errors are injected into the DNN model, and the loss value on the bit error injected model is calculated separately. Then, the loss values of the bit error injected DNN model are aggregated with a summation. We add the loss on the original model to recover the performance from a potential decrease due to having a more complex loss function, similar to adversarial training and bit error training. Then, the loss value of the original DNN model is added to the losses of the bit error injected DNN models, and the total sum is normalized with λ. λ normalization prevents the gradients of the total loss function to become too large, which can cause divergence of DNN model weights.

MBET uses a complex loss function that contains a prediction for each bit error rate in the bit error rate list, B. This increases the training time of the DNN model compared to normal training and bit error training (BET). However, inference time does not change since MBET is a training method that does not change the prediction procedure, and also, the DNN model structure is not changed. A DNN model trained with MBET can predict with a single forward pass like a DNN model trained with normal training.

Figure 4.1 describes the MBET method. The training takes T epochs. The weights of the DNN model for epoch t, $w^{(t)}$ are quantized with the quantization function Q at Line 2 and $v^{(t)}$ are the quantized weight values. Random bit errors are injected to the quantized values $v^{(t)}$ for each bit error rate b in the bit error rate list B. Bit error injected weights, $BE(v^{(t)}, b)$ are dequantized by Q^{-1} to obtain the corresponding weights in floating point, $\widetilde{w}_b^{(t)}$. Therefore, we have a different bit error injected DNN model for each bit error rate b in B. The DNN model is updated for each input in Training Data between Lines 6 and 12. The gradients of the loss function on the original model, $\nabla^{(t)}$ are calculated separately at Line 7. Then the gradients of the bit error injected DNN models, $\nabla_b^{(t)}$ are calculated at Line 9 for each bit error rate, b in

the bit error rate list B. The weights of the DNN model, $w^{(t)}$ are updated with the gradients of the original model, $\nabla^{(t)}$ and the gradients of the bit error injected models, $\nabla_b^{(t)}$ at Line 11. λ is the normalization factor to prevent the divergence of the weights from high gradient values with the increase in the size of B. This procedure is repeated for each epoch.

The bit errors injected for each bit error rate at Line 4 are different for each epoch during the training. Random numbers are sampled from a uniform distribution for each bit of the weights, and a bit is flipped if the corresponding generated random number is less than the bit error rate. Therefore, different bit errors are injected for each epoch. In this way, the DNN model can undergo a variety of bit errors for each bit error rate and can be more resilient to this bit error rate compared to using only one single bit error mask. This procedure also removes the bias from the selection of a bit error mask and overfitting for a specific bit error mask.

1: **for** $t = 0$, ... , $T - 1$ **do**
2: $v^{(t)} = Q(w^{(t)})$ ▷ Quantize weights
3: **for** $b = 0$, ... , $B - 1$ **do**
4: $\widetilde{w}_b^{(t)} = Q^{-1}(BE(v^{(t)}, b))$ ▷ Inject bit errors and dequantize
5: **end for**
6: **for** (x, y) in Training Data **do** ▷ For each batch in training data
7: $\nabla^{(t)} = \nabla L(f(x; w), y)$ ▷ Gradients of loss value on original weights
8: **for** $b = 0$, ... , $B - 1$ **do**
9: $\nabla_b^{(t)} = \nabla L(f(x; \widetilde{w}_b^{(t)}), y)$ ▷ Gradients of loss value on bit error injected weights
10: **end for**
11: $w^{t+1} = w^{(t)} - \lambda(\nabla^{(t)} + \sum_{b \in B} \nabla_b^{(t)})$ ▷ Update weights
12: **end for**
13: **end for**

Figure 4.1. Multiple Bit Error Rate Training.

4.3. Bit Error Injection

Figure 4.2 describes how we inject bit errors into a trained DNN model. There are three parameters. The first one is the DNN model which bit errors will be injected. The second one is the bit error rate which indicates the probability of injecting bit errors into the model. The last one is the number of bits that will be used in quantization. The number of bins in quantization is assigned to n in Line 1. For each layer in the DNN model, we inject bit errors to the weights between Lines 2 and 18. Since we inject different bit errors to each layer, we initialize $total\_faults$ as zeros in the shape of the layer weights in Line 3. In Line 4, we store the sign of weights to be able to recover them after bit error injection. We quantize the weights between Lines 5 and 7. The minimum and the maximum weights are obtained in Line 5. Weights are normalized using the minimum and the maximum value in Line 6. Then, we obtain the quantized weights, which are the integer representation of weights in Line 7. Between Lines 8 and 14, we inject bit errors randomly for each bit in quantization representation. cb is the current power value for bit b in Line 9. For bit b, we check whether it is active or not in Line 10. Depending on the activity of bit b, we assign the fault value with the current power value and the minus of it in Line 11. We randomly select which weights the bit error will be injected via generating random numbers from a uniform distribution and comparing them with the bit error rate in Line 12. In Line 13, we calculate the fault for bit b by multiplying the fault value, mask and sign and accumulate the fault to $total\_faults$ with addition. We repeat this process for each bit in quantization representation. We update the quantized weights with accumulated $total\_faults$ in Line 15. We dequantize the updated quantized weights using the minimum and maximum weight values. We finally update the DNN model with bit error injected weights. We repeat this process for each layer in the DNN model.

As described in Figure 4.2, the bit error injection process depends on the quantization method because quantization is how the values are stored and represented. We use integer quantization, and Figure 4.2 describes how bit errors can be injected for integer quantization. This method can be extended for any quantization method such

as floating point quantization via replacing the quantization and dequantization parts in Figure 4.2.

We select target bits, which bit errors will be injected, with a uniform distribution. In Line 12, fault mask describes in which bit errors will be injected. A fault mask is constructed with the comparison between random numbers from the uniform distribution and bit error rate. For each bit error injection, we sample different random numbers. Therefore, fault masks and targets of bit errors differ each time. Moreover, different strategies can be used to generate fault masks, instead of using random fault masks. One such idea is to flip the most significant bit in the weights. This method is faster than searching for each bit in random fault masks and causes the most value change with a single bit flip. However, since the faults which may occur in the system are random and can occur at any bit position, we want to simulate this process by utilizing the uniform distribution.

After the training, we inject bit errors on the weights of the DNN model with a selected bit error rate. We use a wide range of bit error rates to capture the resilience better and report the test accuracy under these bit error rates. Additionally, we repeat this process 50 times to decrease potential bias.

4.4. Comparison Metrics

We want to discuss the relationship between the robustness and resilience of DNN models. The robustness and resilience of DNN models are widely studied separately. However, no study focuses on the relationship between them. We want to address the effect of one side on the other, robustness on resilience, and resilience on robustness. For this purpose, we use different training methods which improve robustness and resilience and compare their robustness and resilience performances.

To test and understand the effect of robustness and resilience on each other, we compare the performance of DNN models trained with different training methods.

We use adversarial training as a robustness improvement method. We use bit error training as a resilience improvement method. We use adversarial and bit error training (ADBET) to capture the effect of both robustness and resilience at the same time. We use normal training as a baseline. Then, we check the performance of the trained DNN models on different metrics, such as robust accuracy for robustness and test accuracy under bit error injection with different bit error rates for resilience.

Additionally, we test MBET against normal training and BET. The main target of MBET method is to improve the resilience more than BET while not decreasing the performance of the DNN model on the test data. First, we report the test accuracy of the trained DNN models. Second, we report the test accuracy under bit error injection with different bit error rates to check and compare the resilience performance. We report the performance on robust accuracy to address the question of the relationship between robustness and resilience.

For comparison, we consider the robustness and resilience of DNN models separately. For robustness, we consider test accuracy and robust accuracy. For resilience, we consider test accuracy under bit error injection.

The test accuracy is a metric used in classification problems to check the performance of the DNN model on unseen test data during the training. The test accuracy is calculated by the ratio of the correct predictions over the total number of predictions in test set.

Robust accuracy, on the other hand, is a metric to test the robustness of the DNN model against some adversarial inputs. First, adversarial inputs are generated with a target adversarial attack on the test data. Robust accuracy is the percentage of the correct predictions on the adversarial inputs and is calculated by the ratio of correct predictions over the total number of predictions on the adversarial inputs. Since robust accuracy depends on the adversarial attack which is used to generate adversarial inputs, we state robust accuracy against a specific attack. We consider FGSM and PGD

adversarial attacks in this work. Therefore, we consider robust accuracy against FGSM and robust accuracy against PGD adversarial attacks.

We check the test accuracy of the DNN model after bit error injections to test the resilience of the DNN model. Bit error is a fault in the weights of the DNN model on bit level. The bit value is updated with its inverse. This procedure is called bit error injection.

Require:

$model$ ▷ DNN model

$bit\_error\_rate > 0$ ▷ Bit error rate

$bit \in \mathbb{Z}^+$ ▷ Quantization parameter

$n = 2^{bit} - 1$ ▷ Number of bins in quantization range

for $weight$ in $model.weights$ **do**

  $total\_faults = zeros(weight.shape)$ ▷ Init total_faults with zeroes in shape of weights

  $sign = get\_sign(weight)$ ▷ Store sign of weights

  $min\_w, max\_w = weight.min(), weight.max()$ ▷ Store minimum and maximum of weights

  $weight\_rescale = (weight - min\_w)/(max\_w - min\_w)$ ▷ Normalize weights with minimum and maximum

  $q\_weight = floor(weight\_rescale * n + 0.5)$ ▷ Quantized weights

  for $b = 0$, ... , $bit - 1$ **do**

    $cb = 2^b$ ▷ Current power value

    $b\_weights = (q\_weights/cb).abs().fmod(2)$ ▷ Check current bit position is active or not

    $fault\_value = where(b\_weights < 1, cb, -cb$ ▷ Assign fault_value according to the active bits on current position

    $fault\_mask = where(rand(b\_weights.shape) < bit\_error\_rate, 1, 0)$ ▷ Check if fault will be injected to each weight with generating random numbers

    $total\_faults+ = fault\_value * fault\_mask * sign$ ▷ Add current fault to total_faults

  end for

  $updated\_q\_weight = q\_weight + total\_faults$ ▷ Add faults to the quantized weights

  $updated\_weight = updated\_q\_weight/n * (max\_w - min\_w) + min\_w$ ▷ Dequantize bit error injected weights

  $model.update(updated\_weight)$ ▷ Update model with bit error injected weights

end for

Figure 4.2. Bit Error Injection.

5. EXPERIMENTS

5.1. Experimental Setup

5.1.1. Datasets

In the experiments, we used CIFAR10 and CIFAR100 datasets [53] which are popular and publicly available image classification datasets. CIFAR10 contains 60000 32x32 color images with 10 classes. For each class, there are 5000 training images and 1000 test images. CIFAR100 contains 60000 32x32 color images with 100 classes. For each class, there are 500 training images and 100 test images.

5.1.2. DNN Models

We used four state-of-the-art DNN models in the experiments. The selected DNN models are MobilenetV2 [54], ShufflenetV2 [55], ResNet-18 [56] and ResNet-50 [56]. We used the same architecture for both CIFAR10 and CIFAR100 datasets with only changing the final output layer according to the class number of the dataset.

5.1.2.1. MobilenetV2. MobileNetV2 [54] is a DNN model which aims to improve the performance on image classification, object detection and segmentation tasks for mobile and resource constrained environments.

5.1.2.2. ShufflenetV2. ShufflenetV2 [55] proposes an efficient network design to tackle speed and accuracy tradeoff.

5.1.2.3. ResNet-18. ResNet-18 [56] is a DNN model with 18 deep layers and belongs to ResNet model family which has residual layers with shortcuts and has high performance on image classification, object detection and segmentation tasks.

5.1.2.4. ResNet-50. ResNet-50 [56] is a DNN model with 50 deep layers and belongs to ResNet model family which has residual layers with shortcuts and has high performance on image classification, object detection and segmentation tasks.

5.1.3. Training Configurations

We used Stochastic Gradient Descent (SGD) method to train DNN models for both CIFAR10 and CIFAR100 datasets. We used initial learning rate of 0.1, momentum of 0.9 and weight decay of 0.0001. We trained each model for 200 epochs with batch size of 64. Additionally, we used a decaying policy on the learning rate. After epochs of 60, 120 and 160, we updated the learning rate with gamma of 0.2 by multiplying the learning rate with gamma.

For the custom loss functions, we used λ coefficients as normalization factors to make the loss function stay in the normal range, otherwise it can cause model to diverge due to high loss values. We set λ values according to the number of terms in the loss function.

- For adversarial training (ADV), we set λ values to $1/2$ since there are two terms in the loss function.
- For bit error training (BET), we set λ values to $1/2$ since there are two terms in the loss function.
- For Adversarial and bit error training (ADBET), we set λ values to $1/3$ since there are three terms in the loss function.
- For multiple bit error rate training (MBET), we set λ values to $1/(|B|+1)$ since there are $|B|+1$ terms in the loss function, where $|B|$ is the number of bit error rates used in the MBET loss function.

5.1.4. Adversarial Attack Parameters

In the experiments, we used FGSM and PGD adversarial attacks to test robust accuracy of DNN models. We used the default parameters of FGSM and PGD adversarial attacks for CIFAR-10 and CIFAR-100 datasets. For FGSM adversarial attack, we set ϵ value to $8/256$ which is the default value for CIFAR-10 and CIFAR-100 datasets. For PGD adversarial attack, we set ϵ value to $8/256$ and α to $2/256$ which is the default value for CIFAR-10 and CIFAR-100 datasets. The number of iterations is set to 10.

5.1.5. Bit Error Injection Parameters

We first checked the test accuracy of the DNN models against different bit error rates to select the parameters. The parameters of bit error injection experiments are as follows:

- We decided to test the DNN models in the range of bit error rates between 10^{-6} and 10^{0} since the effect of bit error injections is small for bit error rates lower than 10^{-6}.
- We decided to use 10^{-4} bit error rate as training bit error rate for BET and ADBET since all 4 DNN models do not lose test accuracy around 10^{-4} bit error rate for both CIFAR-10 and CIFAR-100 datasets.
- Similarly, we decided to use 10^{-4} bit error rate for MBET training method. We expanded the bit error rates in both increasing and decreasing sides. For MBET with 3 bit error rates, we used 10^{-4}, 10^{-3} and 10^{-5} bit error rates during the training and for MBET with 5 bit error rates, we used 10^{-4}, 10^{-3}, 10^{-5}, 10^{-2} and 10^{-6} bit error rates during the training.

To remove the bias from experiments, we repeated the bit error injection process 50 times for each bit error rate. We reported the average of the test accuracies over the 50 trials.

5.2. Experimental Results of Robustness and Resilience Improvement Methods

We investigated the effect of robustness and resilience improvement method on each other. We worked on four training types. We used normal training, which utilize an untouched cross entropy loss function, as baseline. Adversarial training and bit error training are used for robustness and resilience improvement methods, respectively. Additionally, we tested adversarial and bit error training to check their impact on each other directly.

Table 5.1. Accuracy Results of Robustness and Resilience Improvement Methods on CIFAR-10.

Training Type	**Metric (accuracy)**	**Models**			
		mobilenet	**shufflenet**	**resnet18**	**resnet50**
normal	test	0.9161	0.9254	0.9491	0.9495
	robust	0.3073	0.3272	0.4932	0.4325
	resilient	0.7614	0.8057	0.8867	0.6906
adv	test	0.8901	0.8949	0.9268	0.9293
	robust	0.8797	0.8832	0.9163	0.9174
	resilient	0.7955	0.7288	0.8781	0.7671
bet	test	0.9207	0.9272	0.9478	0.9513
	robust	0.7444	0.7782	0.716	0.7244
	resilient	0.8452	0.8147	0.8993	0.9023
adbet	test	0.8848	0.8964	0.9261	0.9262
	robust	0.8765	0.8842	0.9154	0.9128
	resilient	0.7971	0.7536	0.8831	0.8006

5.2.1. Experiments on Robustness

Table 5.1 and Table 5.2 show the robustness experiment results for CIFAR10 and CIFAR100 datasets, respectively. The tables show the results for four DNN models:

Table 5.2. Accuracy Results of Robustness and Resilience Improvement Methods on CIFAR-100.

Training Type	**Metric (accuracy)**	**Models**			
		mobilenet	**shufflenet**	**resnet18**	**resnet50**
normal	test	0.7167	0.7248	0.7507	0.7582
	robust	0.1466	0.2069	0.242	0.2764
	resilient	0.54	0.6369	0.6562	0.6917
adv	test	0.6655	0.6741	0.6952	0.7142
	robust	0.6468	0.6529	0.6739	0.6879
	resilient	0.467	0.5866	0.5989	0.4307
bet	test	0.7099	0.7226	0.7597	0.7658
	robust	0.5124	0.5389	0.5037	0.496
	resilient	0.5206	0.6315	0.697	0.717
adbet	test	0.6635	0.6707	0.7045	0.7273
	robust	0.6475	0.6526	0.6814	0.7026
	resilient	0.465	0.5728	0.6494	0.653

MobileNet, ShuffleNet, ResNet-18, and Resnet-50, on the columns. The models were sorted by increasing complexity. We trained each model with four different training types: normal training, adversarial training, bit error training and adversarial and bit error training. For each training type, three metrics are reported: test accuracy, robust accuracy and resilient accuracy. Test accuracy is the accuracy on the test data of the corresponding dataset. Robust accuracy is the accuracy of the adversarial samples generated by FGSM on the test data of the corresponding dataset. Resilient accuracy is the accuracy for the bit error rate where the performance of the model changes the most on the Figure 5.1 and Figure 5.2, respectively. We discussed the results under three titles: the effect of training type, the effect of model complexity and the effect of dataset complexity.

5.2.1.1. The Effect of Training Type. The DNN models perform above 90% test accuracy on CIFAR-10 and above 70% test accuracy for CIFAR-100 with normal training in Table 5.1 and Table 5.2. This shows the DNN models are successfully trained. ResNet-50 performs the best test accuracy for both datasets. For all training types, we can observe an increase in test accuracy from left to right, which means more complex DNN models perform better as the DNN models are sorted by increasing complexity.

Adversarial training causes a decrease in test accuracy in both CIFAR-10 and CIFAR-100 datasets. We use an additional loss term for adversarially perturbed input in the adversarial training, which becomes a burden on the DNN model during the training and increases the complexity of the problem. Therefore, the DNN model trained with adversarial training performs worse test accuracy than the DNN model trained with normal training.

On the other hand, the DNN model trained with adversarial training performs significantly better robust accuracy than the DNN model trained with normal training in both datasets. Therefore, incorporating adversarial samples during the training makes the DNN model better against them. The increase in robust accuracy is much higher than the decrease in test accuracy. Therefore, adversarial training performs successfully as a robustness improvement method.

The DNN model trained with adversarial training performs worse resilient accuracy than the DNN model trained with normal training in both datasets with six out of eight DNN models. The resilient accuracy decrease can be explained by the test accuracy decrease. Since the resilient accuracy is the test accuracy of the bit error injected DNN model for a specific bit error rate, we can expect a worse resilient accuracy if the DNN model performs worse test accuracy.

The DNN model trained with bit error training performs similar test accuracy to the DNN model trained with normal training in both datasets. Bit error training performs better than normal training for four out of eight DNN models, but the test

accuracy differences in all eight DNN models are less than 1%. Bit error training uses an additional loss term which represents the loss on the bit error injected DNN model. Since we allow the DNN model to adapt the change in weights with bit error injections during the training, the test accuracy change is small. Additionally, the selected bit error rate for the bit error training is 10^{-4} and does not cause a significant test accuracy drop for all eight models. Choosing a high bit error rate could lead the DNN model not to train properly.

The DNN model trained with bit error training performs significantly better robust accuracy than the DNN model trained with normal training in both datasets, even though we do not use adversarial samples directly during the training. The effect of bit error injections on the weights resembles the effect of perturbations on the input. Perturbations on the input propagate through the DNN model, which changes the results of each layer in the DNN model. Similarly, bit error injections affect the results of each layer in the DNN model. Therefore, both perturbations on the input and bit error injection on the weights have similar effects during the training, and training with bit error injections on weights improves the robustness of the DNN model.

The DNN model trained with bit error training performs better resilient accuracy than the DNN model trained with normal training in both datasets with six out of eight DNN models. The resilience of the DNN models increases with simulating the effect of random bit error injections during the training. We will discuss resilience analysis in more detail in the resilience experiments.

Bit error training performs better test accuracy than adversarial training, as bit error training does not cause a decrease in test accuracy. Although both methods use an additional loss term, one difference between perturbations on the input and bit errors on weights is that perturbations in adversarial samples are calculated from the gradients of the input, while bit error injections are random. Therefore, robust accuracy increase is smaller with bit error training than with adversarial training.

The DNN models trained with adversarial and bit error training perform worse test accuracy than the DNN models trained with normal training. The loss function of adversarial and bit error training is the combination of the loss function of adversarial training and the loss function of bit error training. The additional loss term from the adversarial training makes the problem more complex, which results in a decrease in test accuracy.

The DNN model trained with adversarial and bit error training performs significantly better robust accuracy than the DNN models trained with normal training. Since both adversarial training and bit error training increase robust accuracy, we can expect a similar performance boost in robust accuracy with adversarial and bit error training.

The DNN model trained with adversarial and bit error training performs worse resilient accuracy than the DNN model trained with normal training in both datasets with six out of eight DNN models. The decrease in test accuracy can be the main reason for this decrease in resilient accuracy.

Adversarial and bit error training performs similarly to adversarial training. It causes a test accuracy drop, while improving robust accuracy significantly. Therefore, the effect of adversarial training is more dominant than bit error training in adversarial and bit error training. The effect of bit error training remains hidden on accuracy results in Table 5.1 and Table 5.2. We further study the effect of bit error training on adversarial and bit error training with resilient experiments.

5.2.1.2. The Effect of Model Complexity. The performance of the DNN models increases with model complexity in Table 5.1 and Table 5.2. The DNN models are sorted by the increasing DNN model complexity from left to right in Table 5.1 and Table 5.2. Similarly, the performance of DNN models on three metrics for four training types increases from left to right in general. Since we select the state-of-the-art DNN models for the image classification task, test accuracy increases with DNN model complexity.

The limited effect of perturbations on the input can be the main reason for the increase in robust accuracy with the DNN model complexity. Since we restrict the perturbations in ϵ boundary, the complexity of the problem remains the same for each DNN model. The more complex DNN models can handle the same perturbations easier.

Resilient accuracy also increases with the DNN model complexity, even though we inject bit errors uniformly, which means the number of bit errors increases with the number of parameters of the DNN model. The more complex DNN models can recover the effect of the faults on the weights easier, and as a result, have high accuracy against bit errors.

The effect of DNN model complexity on the accuracy results is similar for each training type. While normal training is the simplest method and adversarial and bit error training is the most complex, ResNet-18 and ResNet-50 perform the best for both training methods. Therefore, the complexity of the training method does not change the effect of the DNN model complexity on accuracy.

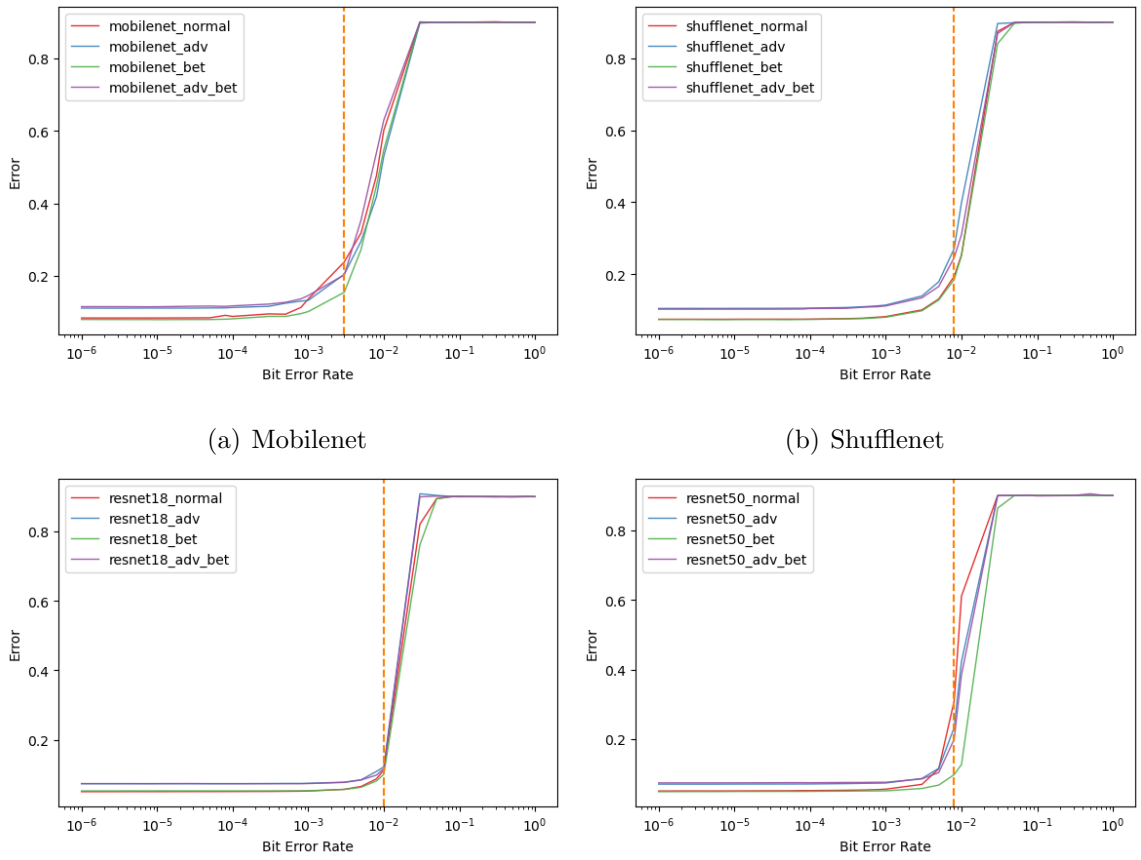
5.2.1.3. The Effect of Dataset Complexity. CIFAR100 is a more complex dataset than CIFAR10. Although the resolution of the images is the same for both datasets, CIFAR100 has 100 classes, while CIFAR10 has 10. Additionally, the number of samples for each class in CIFAR100 is less than in CIFAR10.

The accuracy of the DNN models is lower in CIFAR100. The DNN models can achieve around 90% test accuracy on CIFAR10, while the DNN models can achieve around 70% test accuracy on CIFAR100. Therefore, the complexity of the dataset directly affects the performance of the DNN model.

The performance difference of the DNN models increases with the complexity of the dataset. The test accuracy difference between different DNN models is around 6% for CIFAR100, while 4% for CIFAR10. The test accuracy of the DNN models decreases by around 20% with changing the dataset from CIFAR10 to CIFAR100. The accuracy

of the more complex DNN models decreases less than simpler DNN models. Therefore, the more complex DNN models perform more stably than simpler DNN models.

The performance difference between the training types also increases with the complexity of the dataset. The test accuracy difference between the normal training and adversarial training is around 3% for CIFAR10, while 6% for CIFAR100. Therefore, the effects of different training methods increase with complexity of the dataset.



(a) Mobilenet

(b) Shufflenet

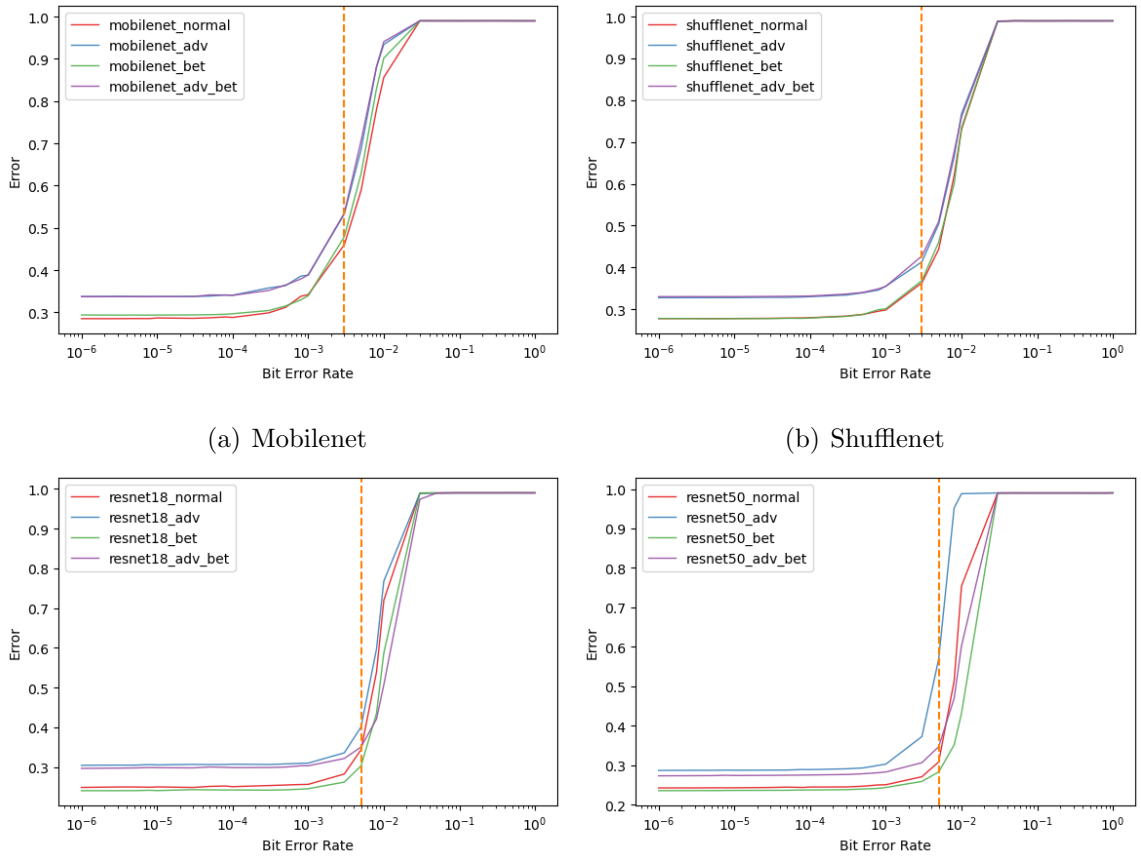
(c) Resnet18

(d) Resnet50

Figure 5.1. Resilience Results of Robustness and Resilience Improvement Methods on CIFAR-10.

5.2.2. Experiments on Resilience

Figure 5.1 and Figure 5.2 show the resilience experiment results for CIFAR10 and CIFAR100 datasets, respectively. The figures show the test accuracy of the bit error injected DNN models for different bit error rates. The x-axis is the bit error rate applied to the DNN model, and the y-axis is the test accuracy of the bit error



(a) Mobilenet

(b) Shufflenet

(c) Resnet18

(d) Resnet50

Figure 5.2. Resilience Results of Robustness and Resilience Improvement Methods on CIFAR-100.

injected DNN model. The figures contain subfigures for four DNN models: MobileNet, ShuffleNet, ResNet-18 and ResNet-50. Each figure shows the results of four training types: normal training, adversarial training, bit error training and adversarial and bit error training.

The figures show the performance of the DNN models trained with different training methods. The performance of the DNN models is similar to the sigmoid function with an "S" shape. The test error is similar at low bit error rates. The test error increases sharply around the inflection point, similar to the exponential function. The test error saturates at high bit error rates. The bit error rates where the test error starts to increase and the test error saturates change for each DNN model and training method.

5.2.2.1. The Effect of Training Type. We can observe a grouping in low bit error rates in Figure 5.1 and Figure 5.2. Adversarial training, and adversarial and bit error training perform similarly up to the inflection point, while normal training and bit error training perform equally. The main reason can be the initial test accuracy of the DNN models in Table 5.1 and Table 5.2. However, the groups disband, and the order of the training types changes after the inflection point.

The performance of the adversarial training against bit errors is the worst in Figure 5.1 and Figure 5.2, in general. The DNN models trained with adversarial training perform high test errors than the DNN models trained with normal training for low bit error rates. The effect of the initial test error of the DNN model is more dominant than the effect of bit error injections on the DNN model weights for low bit error rates. Since the DNN models trained with adversarial training perform low test accuracy, the performance of the DNN models trained with adversarial training is worse for low bit error rates. The test error of adversarial training gets close to the test error of other training types for higher bit error rates and ends up with the same saturation point.

Bit error training performs the best with the minimum test error against random bit error injections in general. Although the performance of the bit error training on resilient accuracy is not as effective as the performance of adversarial training on robust accuracy, the impact of bit error training on resilience is clear in Figure 5.1 and Figure 5.2. For instance, ResNet-50 models trained with bit error training on both CIFAR10 and CIFAR100 datasets clearly outperform other training methods.

The performance of adversarial and bit error training is similar to adversarial training for low bit error rates. Low test accuracy caused by the adversarial term in the loss function of adversarial and bit error training causes high test error for low bit error rates, similar to the adversarial training. However, adversarial and bit error training is better than the other training methods for high bit error rates, especially after the inflection point. For instance, ResNet-50 models trained with adversarial and

bit error training on both CIFAR10 and CIFAR100 datasets perform comparable to ResNet-50 models trained with bit error training. Therefore, high bit error rates can reveal the effect of bit error term in the loss function, and adversarial and bit error training performs better resilience against high bit error rates.

5.2.2.2. The Effect of Model Complexity. The DNN models are sorted by the increasing complexity from left to right for CIFAR10 and CIFAR100 in Figure 5.1 and Figure 5.2, respectively. The test error starts to increase at different bit error rates for each model for both CIFAR10 and CIFAR100 datasets in Figure 5.1 and Figure 5.2. For instance, the test error of Mobilenet on CIFAR10 in Figure 5.1(a) starts to increase around 10^{-3}, while the test error of ResNet-18 on CIFAR10 in Figure 5.1(c) starts to increase around 10^{-2}. Therefore, the more complex DNN models have more internal resilience against random bit errors and the test error of the more complex DNN models increases at higher bit error rates than simpler DNN models.

The bit error at which the test error of the DNN model saturates is similar for each DNN model, even though the bit error where the test error starts to increase changes. The saturation bit error is around $2x10^{-2}$. The main reason for each DNN model saturating around the same bit error rate can be applying bit error injections uniformly. Thus, the number of bit error injections changes with the number of parameters of the DNN model. After this bit error rate, the DNN models make random predictions, and the DNN models become random predictors.

The experimental results in Figure 5.1 and Figure 5.2 show that residual connections in ResNet affect the test error positively in addition to the DNN model complexity. Residual connections allow the DNN model to reuse the input of a layer, which is not affected by bit error injections because we apply bit error injections on the DNN model weights. The DNN model can recover some of the faults caused by the bit errors thanks to these residual connections. As a result, ResNet models perform better than other DNN models.

5.2.2.3. The Effect of Dataset Complexity. The DNN models on CIFAR10 have more resilience against random bit error injections than the DNN models on CIFAR100, as CIFAR100 is a more complex dataset than CIFAR10. The test error starts to increase at lower bit error rates on CIFAR100 than on CIFAR10 for the same DNN model architecture. For instance, the test error starts to increase around 10^{-3} for Mobilenet on CIFAR10, while around $3 * 10^{-4}$ for Mobilenet on CIFAR100. As a result, the DNN model performance decreases on more complex datasets for the same DNN architecture.

The bit error rate where the test error saturates is similar for both CIFAR10 and CIFAR100 datasets even though the bit error rate where the test error starts to increase changes. For instance, both Mobilenet on CIFAR10 and Mobilenet on CIFAR100 have the same saturation bit error rate around $3 * 10^{-2}$. Therefore, the experimental results show that the saturation bit error rate does not change for CIFAR10 and CIFAR100 datasets.

The saturation test error depends on the dataset. The DNN models perform 10% and 1% test accuracy on CIFAR10 and CIFAR100, respectively. Since each class in the datasets has the same number of samples, and there are 10 and 100 classes in CIFAR10 and CIFAR100, respectively, the expected test accuracy of a random classifier is 10% and 1% test accuracy. Therefore, the DNN models become a random classifier due to high bit error rates.

5.3. Experimental Results of MBET

We proposed a new training method to improve resilience of DNN models, called by multiple bit error rate training (MBET). We tested MBET against normal training and bit error training, as baselines. We used two versions of MBET. The former is MBET with 3-bit error rates, and the latter is MBET with 5-bit error rates.

Table 5.3. Accuracy Results of MBET on CIFAR-10.

Training Type	**Metric (accuracy)**	**Models**			
		mobilenet	**shufflenet**	**resnet18**	**resnet50**
normal	test	0.9161	0.9254	0.9491	0.9495
	ra(fgsm)	0.3073	0.3275	0.4932	0.4325
	ra(pgd)	0.0389	0.0477	0.1352	0.0539
bet	test	0.9207	0.9272	0.9478	0.9513
	ra(fgsm)	0.7444	0.7782	0.716	0.7244
	ra(pgd)	0.6908	0.7435	0.57	0.552
mbet_3	test	0.9253	0.9285	0.9492	0.9467
	ra(fgsm)	0.7588	0.7776	0.7199	0.7518
	ra(pgd)	0.7195	0.7435	0.578	0.6293
mbet_5	test	0.9231	0.9298	0.9502	0.9521
	ra(fgsm)	0.7522	0.7791	0.7168	0.7395
	ra(pgd)	0.7108	0.74	0.5809	0.5823

5.3.1. Experiments on Robustness

Table 5.3 and Table 5.4 show the robustness experiment results for CIFAR10 and CIFAR100 datasets, respectively. The tables show the results for four DNN models: MobileNet, ShuffleNet, ResNet-18, and ResNet-50 on the columns. The models are sorted by increasing complexity. We trained each model with four different training methods: normal training, bit error training, MBET with 3-bit error rates, and MBET with 5-bit error rates. For each training type, three metrics are reported: test accuracy, robust accuracy against FGSM (ra(fgsm)), and robust accuracy against PGD (ra(pgd)). Test accuracy is the accuracy on the test data of the corresponding dataset. Robust accuracy against FGSM is the accuracy of the adversarial samples generated by the FGSM on the test data of the corresponding dataset. Robust accuracy against PGD is the accuracy of the adversarial samples generated by the PGD on the test data of the

Table 5.4. Accuracy Results of MBET on CIFAR-100.

Training Type	**Metric (accuracy)**	**Models**			
		mobilenet	**shufflenet**	**resnet18**	**resnet50**
normal	test	0.7167	0.7248	0.7507	0.7582
	ra(fgsm)	0.1466	0.2069	0.242	0.2764
	ra(pgd)	0.0237	0.0619	0.0593	0.0743
bet	test	0.7099	0.7226	0.7597	0.7658
	ra(fgsm)	0.5124	0.5389	0.5037	0.496
	ra(pgd)	0.4842	0.5073	0.4304	0.4107
mbet_3	test	0.7132	0.7222	0.7688	0.7879
	ra(fgsm)	0.5272	0.5333	0.5227	0.5357
	ra(pgd)	0.5001	0.5073	0.451	0.4693
mbet_5	test	0.7199	0.7325	0.7739	0.7946
	ra(fgsm)	0.5437	0.5497	0.5281	0.551
	ra(pgd)	0.5244	0.5125	0.4612	0.482

corresponding dataset. We discussed the results under four titles: the effect of training type, the effect of model complexity, the effect of dataset complexity, and the effect of adversarial attack type.

We used the same results for the DNN models trained with normal training and bit error rate training in Table 5.3, and Table 5.4.

5.3.1.1. The Effect of Training Type. We reported robust accuracy against PGD in addition to test accuracy and robust accuracy against FGSM in Table 5.3 and Table 5.4. The DNN models trained with normal training perform significantly worse results on robust accuracy against PGD than on robust accuracy against FGSM, which can be explained by PGD being an upgraded and more advanced version of FGSM.

The DNN models trained with bit error training perform better robust accuracy against PGD than the DNN models trained with normal training, similar to the robust accuracy against FGSM results. Bit errors injected into the DNN weights have a similar effect with perturbations on the input by PGD adversarial attacks. As a result, robust accuracy against PGD increases with bit error training.

The DNN models trained with MBET with 3-bit error rates perform slightly better test accuracy to the DNN models trained with normal training and bit error training. Although the loss function of MBET method is more complex than normal training and bit error training, and we used higher bit error rates in MBET with 3-bit error rate than in bit error training, we managed to compromise by using low bit error rates. Thus, expanding the bit error rates on both increasing and decreasing sides stabilizes the DNN model performance.

MBET with 3-bit error rate significantly increases robust accuracy against FGSM and PGD, compared to normal training. Utilizing random bit errors on the DNN model weights during the training in MBET provides robustness against perturbations in the input. As a result, DNN models trained with MBET perform higher robust accuracy against FGSM and PGD, similar to bit error training.

MBET with 3-bit error rate performs better robust accuracy against FGSM and PGD than bit error training. Since MBET uses more bit error rates during training than bit error training, the DNN model can utilize more diverse faults on the DNN model weights. Faults on the DNN model weights can be considered as perturbations on the input. Therefore, using more bit error rates with MBET provides the DNN model more robustness against adversarial attacks than bit error training.

The DNN models trained with MBET with 5-bit error rate perform slightly better test accuracy than the DNN models trained with normal training, bit error training, and MBET with 3-bit error rate. Utilizing more bit error rates during the training can be considered as an additional augmentation. Therefore, MBET with 5-bit error rate

can use a larger range of inputs than other training methods. The difference in test accuracy between MBET with 5-bit error rate and other training methods is bigger on CIFAR100 in Table 5.4 than on CIFAR10 in Table 5.3.

The DNN models trained with MBET with 5-bit error rate perform better robust accuracy against FGSM and PGD than other training methods. MBET with 5-bit error rate improves the robustness further than MBET with 3-bit error rate. Using more bit error rates during training provides the DNN model more generalization and the DNN model can perform better against perturbations on the input, similar to test accuracy.

5.3.1.2. The Effect of Model Complexity. The DNN models are sorted by the increasing DNN model complexity from left to right in Table 5.3 and Table 5.4. Test accuracy increases with DNN model complexity for each training type, and ResNet models have the highest test accuracy.

Robust accuracy of MobileNet and ShuffleNet is higher than robust accuracy of ResNet-18 and ResNet-50 models, even though ResNet-18 and ResNet-50 have higher test accuracy than MobileNet and ShuffleNet in Table 5.3 and Table 5.4. Thus using bit errors during training provides less robustness against adversarial samples for more complex DNN models, ResNet models. Since bit errors are injected uniformly, simpler DNN models have a higher chance to hit specific perturbations on the input from adversarial attacks with bit error injections. As a result, simpler DNN models, MobileNet and ShuffleNet, have higher robust accuracy. Using higher bit error rates during training can provide more robustness for more complex DNN models, as shown with MBET with 5-bit error rate in Table 5.4.

ResNet-50 performs better than ResNet-18 in general. Although both DNN models have similar architecture, ResNet-50 model is deeper and more complex than ResNet-18 model. As a result, ResNet-50 has higher test accuracy and robust accuracy than ResNet-18, which supports that the DNN model complexity improves the performance in general.

5.3.1.3. The Effect of Dataset Complexity. Table 5.3 and Table 5.4 show the results of DNN models for CIFAR10 and CIFAR100 datasets, respectively. ResNet-50 models trained with MBET with 5-bit error rate achieved 95.21% and 79.46% test accuracy on CIFAR10 and CIFAR100, respectively. The test accuracy increases more from normal training to MBET with 5-bit error rate in CIFAR100 than CIFAR10 since CIFAR100 is a more complex dataset than CIFAR10.

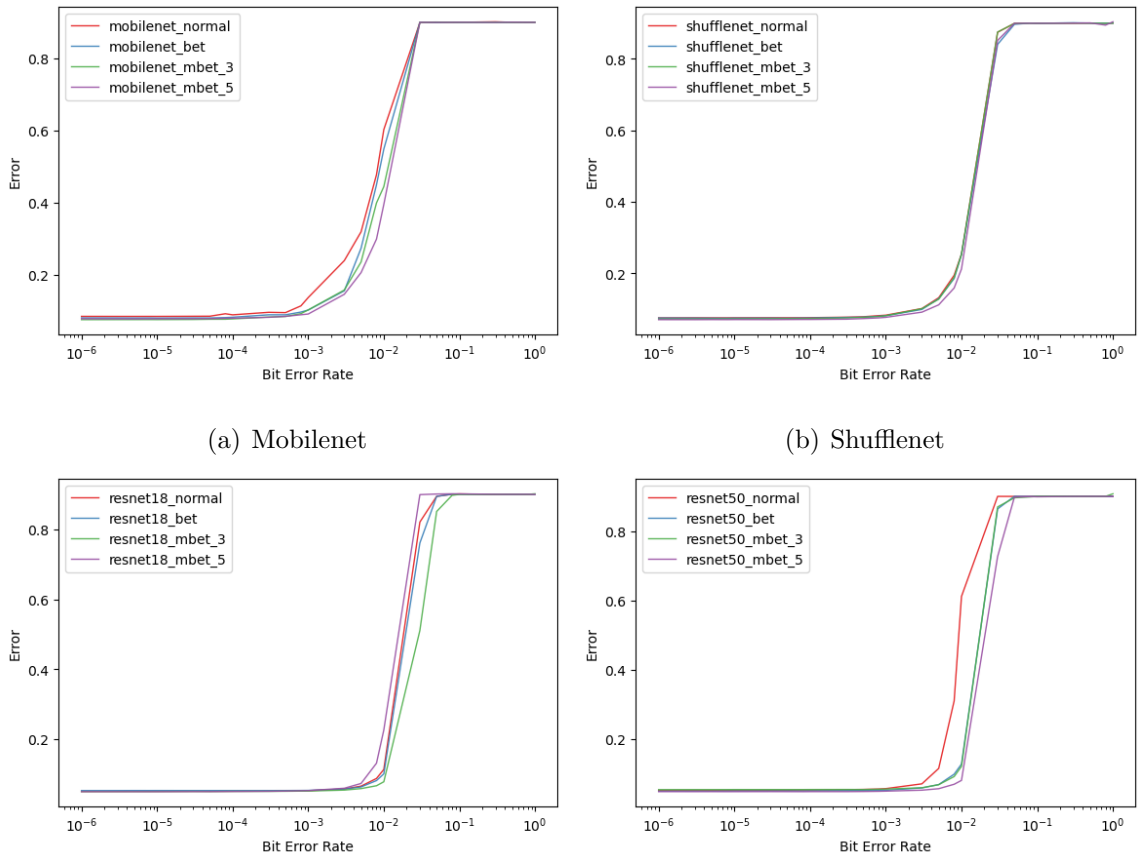
Robust accuracy of ResNet models is worse than robust accuracy of MobileNet and ShuffleNet on CIFAR10 in Table 5.3. However, ResNet models perform similar or better robust accuracy than MobileNet and ShuffleNet on CIFAR100 in Table 5.4. Since CIFAR100 is more complex and adversarial samples increase the complexity of the problem, the DNN models with more capacity can achieve higher robust accuracy.

5.3.1.4. The Effect of Adversarial Attack Type. Table 5.3 and Table 5.4 present robust accuracy against FGSM and PGD of four DNN models trained with four training methods. PGD is an upgraded version of FGSM with multiple iterations. Since PGD is a more complex and powerful adversarial attack than FGSM, robust accuracy against PGD results is lower than robust accuracy against FGSM results.

5.3.2. Experiments on Resilience

Figure 5.3 and Figure 5.4 show the resilience experiment results for CIFAR10 and CIFAR100 datasets, respectively. The figures show the test accuracy of the bit error injected DNN models for different bit error rates. The x-axis is the bit error rate applied to the DNN model, and the y-axis is the test accuracy of the bit error injected DNN model. The figures contain subfigures for four DNN models: MobileNet, ShuffleNet, ResNet-18, and ResNet-50. Each figure shows the results of four training types: normal training, bit error training, MBET with 3-bit error rates, and MBET with 5-bit error rates. We discussed the results under three titles: the effect of training type, the effect of model complexity, and the effect of dataset complexity.

We used the same results for the DNN models trained with normal training and bit error rate training in Figure 5.3 and Figure 5.4.

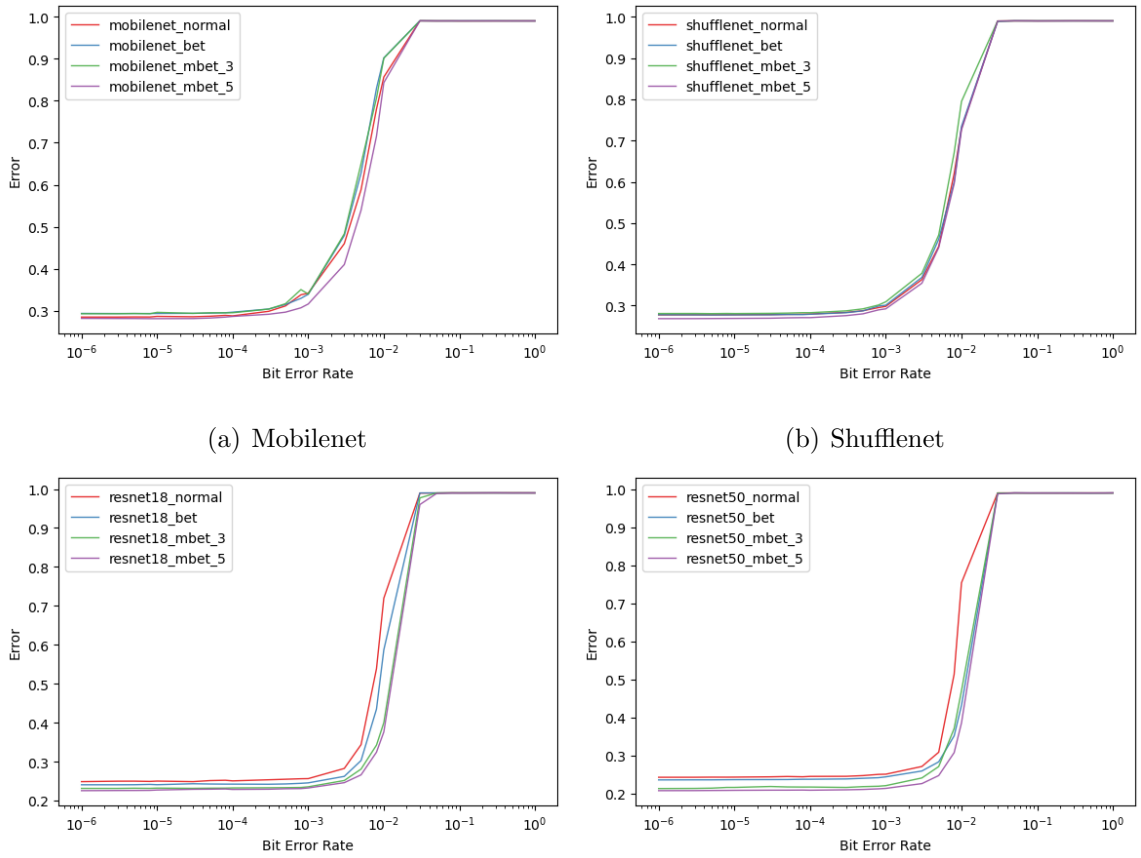


(a) Mobilenet (b) Shufflenet

(c) Resnet18 (d) Resnet50

Figure 5.3. Resilience Results of MBET on CIFAR-10.

5.3.2.1. The Effect of Training Type. The DNN models trained with MBET perform better resilience than the DNN models trained with normal training and bit error training. For instance, the difference between training types are more visible and clear on MobileNet on CIFAR10 in Figure 5.3 and Resnet-18 on CIFAR100 in Figure 5.4. The DNN model trained with normal training performs the worst resilience against random bit errors. The DNN model trained with bit error training performs the second worst resilience against random bit errors. The DNN model trained with MBET with 5-bit error rate performs the best resilience against random bit errors. However, one different example is on Shufflenet models. Shufflenet models perform similar for each training type on both CIFAR10 and CIFAR100 datasets, in Figure 5.3 and Figure 5.4.



(a) Mobilenet

(b) Shufflenet

(c) Resnet18

(d) Resnet50

Figure 5.4. Resilience Results of MBET on CIFAR-100.

MBET aims to improve the resilience of DNN models by using multiple bit error rates during training. Figure 5.3 and Figure 5.4 show how effective MBET is against random bit error injections on DNN model weights. The DNN models trained with MBET outperform the DNN models trained with bit error training, where bit error training is an effective method to increase resilience against random bit error injections.

Figure 5.3 and Figure 5.4 show the results of two MBET versions, MBET with 3-bit error rate and MBET with 5-bit error rate. We expanded the target bit error rates of MBET with 5-bit error rate increasing and decreasing sides. As a result, MBET with 5-bit error rate can achieve better resilience against random bit errors than bit error training and MBET with 3-bit error rate.

The saturation bit error rates of different training methods of the same DNN models are similar in Figure 5.3 and Figure 5.4. The experimental results show that

training methods do not affect the saturation bit error rate, while model architecture determines the saturation bit error rate. Similarly, the inflection bit error rates, where the test error starts to increase are similar for bit error training and MBET. However, the test error starts to increase at early bit error rates for normal training, such as MobileNet on CIFAR10 in Figure 5.3.

ResNet-18 trained with MBET with 5-bit error rate on CIFAR10 in Figure 5.3(a) performs the worst resilience against bit error injections, even tough MBET with 5-bit error rate performs the best test accuracy with 95.02% in Table 5.3. MBET with 5-bit error rate can be a complex training method for ResNet-18, and as a result, the resilience of the DNN model decreases. Increasing the DNN model complexity can resolve this issue, as ResNet-50 trained with MBET with 5-bit error rate performs the best resilience against random errors on DNN model weights on CIFAR10 in Figure 5.3. This example shows that we can not rely solely on the initial test accuracy of the DNN model, and we need bit error injection experiments to understand the resilience against random bit errors.

The test errors at lower bit error rates are similar and it is hard to distinguish the lines for each training methods on CIFAR10 in Figure 5.3. However, since the initial test accuracy increases on CIFAR100, the performance lines of training methods become more distinguishable for lower bit error rates on CIFAR100 in Figure 5.4.

5.3.2.2. The Effect of Model Complexity. The DNN models are sorted by the increasing complexity from left to right for CIFAR10 and CIFAR100 in Figure 5.3 and Figure 5.4, respectively.

The test error starts to increase at different bit error rates for each DNN model. For instance, the test error starts to increase between 10^{-4} and 10^{-3} for MobileNet, while 10^{-3} and 10^{-2} for ResNet-50 on CIFAR100 in Figure 5.4. Therefore, the DNN model complexity positively affects the resilience of the DNN model.

The saturation bit error rate where the test error stabilizes at the highest is similar for each DNN model on both CIFAR10 and CIFAR100 in Figure 5.3 and Figure 5.4, respectively and saturation bit error rate is around $2 * 10^{-2}$.

5.3.2.3. The Effect of Dataset Complexity. The test error starts to increase at higher bit error rates for the same DNN model on CIFAR10 than on CIFAR100, in 5.3 and Figure 5.4, respectively. CIFAR100 is a more complex dataset with more classes and fewer samples for each class than CIFAR10. As the dataset complexity increases, the resilience of the DNN models decreases. For instance, the test error starts to increase around 10^{-2} for ResNet-50 on CIFAR10 in Figure 5.3(d), while $4 * 10^{-3}$ for ResNet-50 on CIFAR10 in Figure 5.4(d). Therefore, the resilience of the same DNN architecture decreases with a more complex dataset.

The performance difference between different training methods becomes more distinguishable on CIFAR100 than on CIFAR10 in Figure 5.4 and Figure 5.3, respectively. The main reason can be the initial test accuracy difference between different training types is higher on CIFAR100 than on CIFAR10 in Table 5.4 and Table 5.3.

Saturation test errors differ for CIFAR10 and CIFAR100 in Figure 5.3 and Figure 5.4, respectively. The saturation test error on CIFAR10 is 0.1, and the saturation test error on CIFAR100 is 0.01. Saturation test error depends on the expected accuracy of a random predictor, which is related to the number of classes in the dataset. Therefore, the saturation test error increases with dataset complexity. However, the saturation bit error rate is similar on both CIFAR10 and CIFAR100 datasets, and the saturation bit error rate is around $3 * 10^{-2}$.

6. CONCLUSION

The performance of DNN models is measured with test accuracy in general. However, the test accuracy of the DNN model is not enough to assess its performance. Adversarial samples have shown a crucial shortage of DNN models and robustness of DNN targets such adversarial samples. Moreover, the faults on DNN weights can lead the DNN model to misclassification and resilience of DNN targets such faults on DNN weights. Therefore, it is necessary to study the robustness and resilience of DNN models.

To understand the relationship between the robustness and resilience of DNN models, we use four training methods, normal training, adversarial training, bit error training, and adversarial and bit error training. Moreover, we propose MBET, which is a training method to improve resilience. We tested our methods with four state-of-the-art DNN models, MobileNet, ShuffleNet, ResNet-18, and ResNet-50, on two image classification datasets, CIFAR10 and CIFAR100.

The experimental results show that bit error training increases robust accuracy significantly. Therefore, a resilience improvement method supports robustness. However, adversarial training decreases the resilience of the DNN model due to the initial test accuracy drop. We can compromise the resilience drop with adversarial and bit error rate training at higher bit error rates. Additionally, we tested MBET with two versions, with 3-bit error rate and 5-bit error rate. Both MBET methods increase the resilience of the DNN model further than bit error training.

For the future work, we plan to investigate the effect of coefficient in the loss functions of adversarial and bit error training, and MBET. Additionally, we are planning to test adversarial and bit error training, and MBET on different datasets and model architectures. Moreover, we can test the resilience with deploying DNNs on edge devices via different operating voltages instead of bit error injection simulations.

REFERENCES

1. Yu, J., Z. Wang, V. Vasudevan, L. Yeung, M. Seyedhosseini and Y. Wu, "CoCa: Contrastive Captioners are Image-Text Foundation Models", arXiv, 2022.

2. Wortsman, M., G. Ilharco, S. Y. Gadre, R. Roelofs, R. Gontijo-Lopes, A. S. Morcos, H. Namkoong, A. Farhadi, Y. Carmon, S. Kornblith and L. Schmidt, "Model soups: averaging weights of multiple fine-tuned models improves accuracy without increasing inference time", K. Chaudhuri, S. Jegelka, L. Song, C. Szepesvari, G. Niu and S. Sabato (Editors), *Proceedings of the 39th International Conference on Machine Learning*, Vol. 162 of *Proceedings of Machine Learning Research*, pp. 23965–23998, PMLR, Baltimore, Maryland, USA, 17–23 Jul 2022.

3. Dai, Z., H. Liu, Q. V. Le and M. Tan, "CoAtNet: Marrying Convolution and Attention for All Data Sizes", *CoRR*, Vol. abs/2106.04803, 2021.

4. Jun, C., H. Jang, M. Sim, H. Kim, J. Choi, K. Min and K. Bae, "ANNA: Enhanced Language Representation for Question Answering", arXiv, 2022.

5. Edunov, S., M. Ott, M. Auli and D. Grangier, "Understanding Back-Translation at Scale", *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pp. 489–500, Association for Computational Linguistics, Brussels, Belgium, Oct.-Nov. 2018.

6. Wang, X., Y. Jiang, N. Bach, T. Wang, Z. Huang, F. Huang and K. Tu, "Automated Concatenation of Embeddings for Structured Prediction", *the Joint Conference of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing*, Association for Computational Linguistics, Bangkok, Thailand, Aug. 2021.

7. Jung, J., S. Jung, H. Seo, H. Namgung and S. Kim, "Sequence Alignment Ensemble

With a Single Neural Network for Sequence Labeling", *IEEE Access*, Vol. 10, pp. 73562–73570, 2022.

8. Lian, Z., J. Tao, B. Liu, J. Huang, Z. Yang and R. Li, "Context-Dependent Domain Adversarial Neural Network for Multimodal Emotion Recognition", *Proc. Interspeech 2020*, pp. 394–398, Shanghai, China, 2020.

9. Zhang, Y., J. Qin, D. S. Park, W. Han, C.-C. Chiu, R. Pang, Q. V. Le and Y. Wu, "Pushing the Limits of Semi-Supervised Learning for Automatic Speech Recognition", arXiv, 2020.

10. Shao, H., L. Wang, R. Chen, H. Li and Y. Liu, "Safety-Enhanced Autonomous Driving Using Interpretable Sensor Fusion Transformer", arXiv, 2022.

11. Wu, P., X. Jia, L. Chen, J. Yan, H. Li and Y. Qiao, "Trajectory-guided Control Prediction for End-to-end Autonomous Driving: A Simple yet Strong Baseline", arXiv, 2022.

12. Lim, D., F. Hohne, X. Li, S. L. Huang, V. Gupta, O. Bhalerao and S. Lim, "Large Scale Learning on Non-Homophilous Graphs: New Benchmarks and Strong Simple Methods", *CoRR*, Vol. abs/2110.14446, 2021.

13. Zeng, Y. and J. Tang, "RLC-GNN: An Improved Deep Architecture for Spatial-Based Graph Neural Network with Application to Fraud Detection", *Applied Sciences*, Vol. 11, No. 12, p. 5656, Jun 2021.

14. Peng, H., R. Zhang, Y. Dou, R. Yang, J. Zhang and P. S. Yu, "Reinforced Neighborhood Selection Guided Multi-Relational Graph Neural Networks", *ACM Trans. Inf. Syst.*, Vol. 40, No. 4, Dec 2021.

15. Cardoso, M. J., W. Li, R. Brown, N. Ma, E. Kerfoot, Y. Wang, B. Murrey, A. Myronenko, C. Zhao, D. Yang, V. Nath, Y. He, Z. Xu, A. Hatamizadeh, A. Myronenko, W. Zhu, Y. Liu, M. Zheng, Y. Tang, I. Yang, M. Zephyr, B. Hashemian, S. Alle,

M. Z. Darestani, C. Budd, M. Modat, T. Vercauteren, G. Wang, Y. Li, Y. Hu, Y. Fu, B. Gorman, H. Johnson, B. Genereaux, B. S. Erdal, V. Gupta, A. Diaz-Pinto, A. Dourson, L. Maier-Hein, P. F. Jaeger, M. Baumgartner, J. Kalpathy-Cramer, M. Flores, J. Kirby, L. A. D. Cooper, H. R. Roth, D. Xu, D. Bericat, R. Floca, S. K. Zhou, H. Shuaib, K. Farahani, K. H. Maier-Hein, S. Aylward, P. Dogra, S. Ourselin and A. Feng, "MONAI: An open-source framework for deep learning in healthcare", arXiv, 2022.

16. Choi, E., M. T. Bahadori, L. Song, W. F. Stewart and J. Sun, "GRAM: Graph-Based Attention Model for Healthcare Representation Learning", *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '17, p. 787–795, Association for Computing Machinery, New York, NY, USA, 2017.

17. Teoh, E. R. and D. G. Kidd, "Rage against the machine? Google's self-driving cars versus human drivers", *Journal of Safety Research*, Vol. 63, pp. 57–60, 2017, `https://www.sciencedirect.com/science/article/pii/S002243751730381X`.

18. Szegedy, C., W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. J. Goodfellow and R. Fergus, "Intriguing properties of neural networks", Y. Bengio and Y. LeCun (Editors), *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*, Banff National Park, Canada, 2014.

19. Goodfellow, I. J., J. Shlens and C. Szegedy, "Explaining and Harnessing Adversarial Examples", Y. Bengio and Y. LeCun (Editors), *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, San Diego, California, United States, 2015.

20. Kurakin, A., I. J. Goodfellow and S. Bengio, "Adversarial examples in the physical world", *CoRR*, Vol. abs/1607.02533, 2016.

21. Madry, A., A. Makelov, L. Schmidt, D. Tsipras and A. Vladu, "Towards Deep Learning Models Resistant to Adversarial Attacks", arXiv, 2017.

22. Biggio, B., I. Corona, D. Maiorca, B. Nelson, N. Šrndić, P. Laskov, G. Giacinto and F. Roli, "Evasion Attacks against Machine Learning at Test Time", H. Blockeel, K. Kersting, S. Nijssen and F. Železný (Editors), *Machine Learning and Knowledge Discovery in Databases*, pp. 387–402, Springer Berlin Heidelberg, Berlin, Heidelberg, 2013.

23. Nguyen, A., J. Yosinski and J. Clune, "Deep neural networks are easily fooled: High confidence predictions for unrecognizable images", *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 427–436, Boston, Massachusetts, 2015.

24. Papernot, N., P. McDaniel, X. Wu, S. Jha and A. Swami, "Distillation as a Defense to Adversarial Perturbations Against Deep Neural Networks", *2016 IEEE Symposium on Security and Privacy (SP)*, pp. 582–597, San Jose, California, 2016.

25. Ros, A. S. and F. Doshi-Velez, "Improving the Adversarial Robustness and Interpretability of Deep Neural Networks by Regularizing Their Input Gradients", *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence and Thirtieth Innovative Applications of Artificial Intelligence Conference and Eighth AAAI Symposium on Educational Advances in Artificial Intelligence*, AAAI'18/IAAI'18/EAAI'18, AAAI Press, New Orleans, Louisiana, USA, 2018.

26. Reagen, B., U. Gupta, L. Pentecost, P. Whatmough, S. K. Lee, N. Mulholland, D. Brooks and G.-Y. Wei, "Ares: A framework for quantifying the resilience of deep neural networks", *2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC)*, pp. 1–6, San Francisco, California, USA, 2018.

27. Chandramoorthy, N., K. Swaminathan, M. Cochet, A. Paidimarri, S. Eldridge, R. V. Joshi, M. M. Ziegler, A. Buyuktosunoglu and P. Bose, "Resilient Low Voltage

Accelerators for High Energy Efficiency", *2019 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pp. 147–158, Washington, DC, USA, 2019.

28. Zhang, J., K. Rangineni, Z. Ghodsi and S. Garg, "Thundervolt: Enabling Aggressive Voltage Underscaling and Timing Error Resilience for Energy Efficient Deep Learning Accelerators", *Proceedings of the 55th Annual Design Automation Conference*, DAC '18, Association for Computing Machinery, New York, NY, USA, 2018.

29. Denkinger, B. W., F. Ponzina, S. S. Basu, A. Bonetti, S. Balási, M. Ruggiero, M. Peón-Quirós, D. Rossi, A. Burg and D. Atienza, "Impact of Memory Voltage Scaling on Accuracy and Resilience of Deep Learning Based Edge Devices", *IEEE Design & Test*, Vol. 37, No. 2, pp. 84–92, 2020.

30. Reagen, B., P. Whatmough, R. Adolf, S. Rama, H. Lee, S. K. Lee, J. M. Hernández-Lobato, G.-Y. Wei and D. Brooks, "Minerva: Enabling Low-Power, Highly-Accurate Deep Neural Network Accelerators", *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, pp. 267–278, Niagara Falls, Canada, 2016.

31. Kim, S., P. Howe, T. Moreau, A. Alaghi, L. Ceze and V. S. Sathe, "Energy-Efficient Neural Network Acceleration in the Presence of Bit-Level Memory Errors", *IEEE Transactions on Circuits and Systems I: Regular Papers*, Vol. 65, No. 12, pp. 4285–4298, 2018.

32. Stutz, D., N. Chandramoorthy, M. Hein and B. Schiele, "On Mitigating Random and Adversarial Bit Errors", *CoRR*, Vol. abs/2006.13977, 2020.

33. Schorn, C., A. Guntoro and G. Ascheid, "An Efficient Bit-Flip Resilience Optimization Method for Deep Neural Networks", *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 1507–1512, Florence, Italy, 2019.

34. Hoang, L.-H., M. A. Hanif and M. Shafique, "FT-ClipAct: Resilience Analysis of Deep Neural Networks and Improving Their Fault Tolerance Using Clipped Activation", *Proceedings of the 23rd Conference on Design, Automation and Test in Europe*, DATE '20, p. 1241–1246, EDA Consortium, San Jose, CA, USA, 2020.

35. Tian, Y., K. Pei, S. Jana and B. Ray, "DeepTest: Automated Testing of Deep-Neural-Network-Driven Autonomous Cars", *Proceedings of the 40th International Conference on Software Engineering*, ICSE '18, p. 303–314, Association for Computing Machinery, New York, NY, USA, 2018.

36. Li, Z., M. Pan, T. Zhang and X. Li, "Testing DNN-based Autonomous Driving Systems under Critical Environmental Conditions", M. Meila and T. Zhang (Editors), *Proceedings of the 38th International Conference on Machine Learning*, Vol. 139 of *Proceedings of Machine Learning Research*, pp. 6471–6482, PMLR, 18–24 Jul 2021.

37. Zhang, M., Y. Zhang, L. Zhang, C. Liu and S. Khurshid, "DeepRoad: GAN-Based Metamorphic Testing and Input Validation Framework for Autonomous Driving Systems", *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*, ASE 2018, p. 132–142, Association for Computing Machinery, New York, NY, USA, 2018.

38. Goodfellow, I., J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville and Y. Bengio, "Generative Adversarial Nets", Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence and K. Weinberger (Editors), *Advances in Neural Information Processing Systems*, Vol. 27, Curran Associates, Inc., Lake Tahoe, Nevada, USA, 2014.

39. Khamaiseh, S. Y., D. Bagagem, A. Al-Alaj, M. Mancino and H. W. Alomari, "Adversarial Deep Learning: A Survey on Adversarial Attacks and Defense Mechanisms on Image Classification", *IEEE Access*, Vol. 10, pp. 102266–102291, 2022.

40. Carlini, N. and D. Wagner, "Towards Evaluating the Robustness of Neural Networks", *2017 IEEE Symposium on Security and Privacy (SP)*, pp. 39–57, IEEE Computer Society, Los Alamitos, CA, USA, May 2017.

41. Ma, L., F. Juefei-Xu, F. Zhang, J. Sun, M. Xue, B. Li, C. Chen, T. Su, L. Li, Y. Liu, J. Zhao and Y. Wang, "DeepGauge: Multi-Granularity Testing Criteria for Deep Learning Systems", *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*, ASE 2018, p. 120–131, Association for Computing Machinery, Montpellier, France, 2018.

42. Elmendorf, W. R., "Controlling the Functional Testing of an Operating System", *IEEE Transactions on Systems Science and Cybernetics*, Vol. 5, No. 4, pp. 284–290, 1969.

43. Gerasimou, S., H. F. Eniser, A. Sen and A. Cakan, "Importance-Driven Deep Learning System Testing", *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*, ICSE '20, p. 702–713, Association for Computing Machinery, Seoul, South Korea, 2020.

44. Lapuschkin, S., A. Binder, G. Montavon, F. Klauschen, K.-R. Müller and W. Samek, "On Pixel-Wise Explanations for Non-Linear Classifier Decisions by Layer-Wise Relevance Propagation", *PLoS ONE*, Vol. 10, p. e0130140, 07 2015.

45. Rousseeuw, P. J., "Silhouettes: A graphical aid to the interpretation and validation of cluster analysis", *Journal of Computational and Applied Mathematics*, Vol. 20, pp. 53–65, 1987.

46. Katz, G., C. Barrett, D. L. Dill, K. Julian and M. J. Kochenderfer, "Reluplex: An Efficient SMT Solver for Verifying Deep Neural Networks", R. Majumdar and V. Kunčak (Editors), *Computer Aided Verification*, pp. 97–117, Springer International Publishing, Heidelberg, Germany, 2017.

47. Ko, C.-Y., Z. Lyu, L. Weng, L. Daniel, N. Wong and D. Lin, "POPQORN: Quantifying Robustness of Recurrent Neural Networks", K. Chaudhuri and R. Salakhutdinov (Editors), *Proceedings of the 36th International Conference on Machine Learning*, Vol. 97 of *Proceedings of Machine Learning Research*, pp. 3468–3477, PMLR, Long Beach, CA, USA, 09–15 Jun 2019.

48. Chen, Z., G. Li, K. Pattabiraman and N. DeBardeleben, "BinFI: An Efficient Fault Injector for Safety-Critical Machine Learning Systems", *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '19, Association for Computing Machinery, New York, NY, USA, 2019.

49. Hanif, M. A. and M. Shafique, "Dependable Deep Learning: Towards Cost-Efficient Resilience of Deep Neural Network Accelerators against Soft Errors and Permanent Faults", *2020 IEEE 26th International Symposium on On-Line Testing and Robust System Design (IOLTS)*, pp. 1–4, Naples, Italy, 2020.

50. Chen, Z., G. Li and K. Pattabiraman, "Ranger: Boosting Error Resilience of Deep Neural Networks through Range Restriction", *CoRR*, Vol. abs/2003.13874, 2020.

51. Wu, X.-X., Y.-W. Hung, Y.-C. Chen and S.-C. Chang, "Accuracy Tolerant Neural Networks Under Aggressive Power Optimization", *2020 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 774–779, Grenoble, France, 2020.

52. Jacob, B., S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam and D. Kalenichenko, "Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference", *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Salt Lake City, UT, USA, June 2018.

53. Krizhevsky, A., "Learning Multiple Layers of Features from Tiny Images",

https://www.cs.toronto.edu/ kriz/learning-features-2009-TR.pdf, 2009, accessed on December 27, 2022.

54. Sandler, M., A. G. Howard, M. Zhu, A. Zhmoginov and L. Chen, "Inverted Residuals and Linear Bottlenecks: Mobile Networks for Classification, Detection and Segmentation", *CoRR*, Vol. abs/1801.04381, 2018.

55. Ma, N., X. Zhang, H. Zheng and J. Sun, "ShuffleNet V2: Practical Guidelines for Efficient CNN Architecture Design", *CoRR*, Vol. abs/1807.11164, 2018.

56. He, K., X. Zhang, S. Ren and J. Sun, "Deep Residual Learning for Image Recognition", *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778, Las Vegas, NV, USA, 2016.

57. Papernot, N., P. McDaniel, S. Jha, M. Fredrikson, Z. B. Celik and A. Swami, "The Limitations of Deep Learning in Adversarial Settings", *2016 IEEE European Symposium on Security and Privacy (EuroS&P)*, pp. 372–387, Congress Center Saar, Saarbrücken, Germany, 2016.

58. Moosavi-Dezfooli, S., A. Fawzi and P. Frossard, "DeepFool: A Simple and Accurate Method to Fool Deep Neural Networks", *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2574–2582, IEEE Computer Society, Las Vegas, NV, USA, 2016.