Black-box adversarial attacks for quantized image classifiers

Yihao Wang
B.Eng in Electronic and Electrical Engineering(Hons)
University of Birmingham, 2020

A Thesis submitted in fulfilment of requirements for the degree of
Master of Science
Communications and Signal Processing
of Imperial College London

Department of Electrical and Electronic Engineering
Imperial College London
August 31, 2021

Abstract

Deep Neural Networks image classifiers are currently state-of-art models in terms of accuracy, which can also be implemented into hardware systems that require higher computational efficiency standards and memory resources. There are a few quantization techniques to generate a model with fewer parameters and competitive accuracy. When these mobile-size models are implemented into realistic scenes, it is essential to improve the robustness of these systems. Therefore, studies about adversarial attacks towards neural network-based models help us better understand the inner structure of the model and learn how to defend our classifiers against adversarial attacks. This project aims to investigate the robustness of state-of-art EfficientNet-B7 and EfficientNet-B0 when they are full precision and 16-bit block floating-point quantized versions. We discussed the performance of Sparse SimBA and Adaptive Sparse SimBA attacks on these models. Also, we proposed a new attack method called Dense One-pixel attack based on the One-pixel attack and investigated its performance as well. It is found that when attacking EfficientNet-B0, the Dense One-pixel attack achieves better results on both efficiency and visual quality. However, due to the limitation of computation resources, we haven't thoroughly investigated the settings of various attack methods. Also, the Dense One-pixel attack performs similarly for both quantized and non-quantized versions of EfficientNet models. New ideas about improving the mentioned methods are also discussed.

Acknowledgment

First of all, I would like to show my sincere gratitude to my supervisor Dr.Christos-Savvas Bounganis. Your patience, kindness, and clear guidance offered me so much power to move forward during this particular period. I was trapped in a small office room due to COVID-19 and experienced a tough time during the project; you encouraged me to keep fighting and provided a few suggestions about my situation. Therefore, I began to have a positive attitude towards my project and make more effective efforts towards the final experiment of my individual project. Thanks again for your insightful advice on project details and supportive attitude towards my work.

Everyone needs to fight against COVID-19 for the whole academic year, especially for international students. We need to overcome the time difference and communicate with classmates and professors through a cold screen. I would like to congratulate all graduate students who have accomplished the examinations and their individual projects. Also, I want to thank my friends Haoran Zhang, Mengyu Wang, and Yunchen Zhu, my classmate Jiaqi Wu, who offered me so much comfort when I felt isolated and helpless.

Lastly, I would like to show my incredible gratitude to my parents, cousins, and aunt. Without your support and company, I would not go through all of the pressure and challenges. Your encouragement and positive attitude gave me so much energy to embrace every single day.

Abbreviations

SVM: Support Vector Machine

CNNs: Convolutional Neural Networks

DCNNs: Deep Convolutional Neural Networks

DNNs: Deep Neural Networks

DE: Differential Evolution

FLOPS: floating point operations per second

ILSVRC: ImageNet Large Scale Visual Recognition Challenge

SimBA: Simple Black-box Adversarial Attack

ZOO: Zeroth Order Optimization

DL: Deep Learning

ML: Machine Learning

DL: Deep Learning

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Deep Neural Network (DNN)-based algorithms have become the main focus in image recognition over the last decade. These DNN-based approaches have achieved high accuracies in tasks of image classification, object localization, and detection. Some of the networks even outperform human-perceived results. Therefore, many real-scene applications such as autonomous vehicles, human computer interaction, and fraud detection have adopted DL-based techniques both in application and research.

Due to the large volume of parameters and the requirement to process a large scale of data, training and implementing the DL-based model involves much computational cost and memory resources. Therefore, it is not easy to employ the DNN models on embedded systems and power-limited devices. This problem has raised the interest in research about compressing a DNN model. Typically, a state-of-art image classifier using DNN techniques holds from 10 million to 100 million parameters. For example, the models we implemented in our experiment EfficientNet-B0 and EfficientNet-B7 consist of 5.3 million and 66 million parameters respectively [2]. The models store the parameters in full-precision (32-bit), making them inefficient when applying to the consumer-level embedded system or real-time practical applications. Therefore, utilizing a comparatively lightweight model will improve efficiency and save energy and computational resources.

The compression method is normally categorized into two approaches which are pruning and quantisation [3]. As the name "Pruning" suggests, the model is compressed

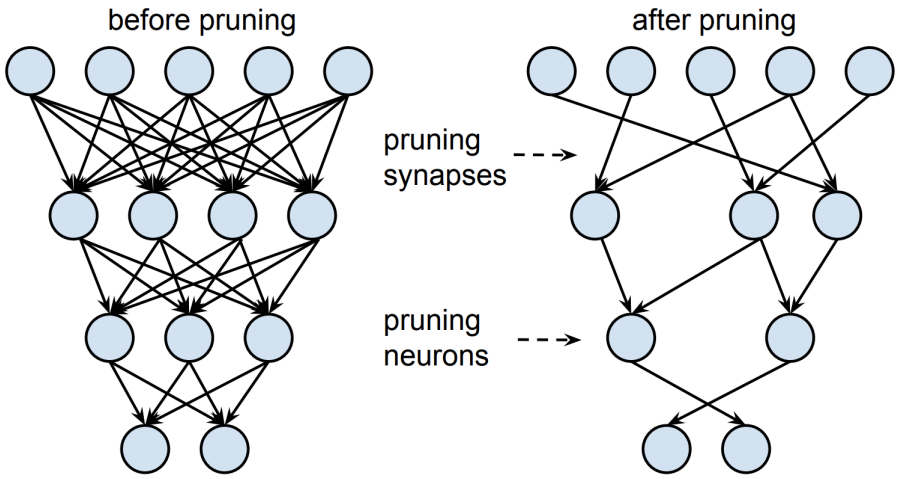

Figure 1.1: Visualisation of Synapses and neurons before and after pruning

through identifying and removing parameters and synapses that are relatively less relevant to a model's performance. Figure 1.1 visualized the original and pruned version of the DNN model [4].

Quantization usually means representing a model's parameters using fewer fits [5]. A common design of quantizer is called Uniform Affine Quantizer [5]. If a floating-point variable with range (x_{min}, x_{max}) that needs to be quantised to the range $(0, N_{levels} - 1)$ where $N_{levels} = 256$ for 8-bits of precision, we consider two parameters Δ and Zero-point(z) which map the floating-point values to integers [5]. The scale determines the step size of the quantizer and floating-point zero maps to zero-point. Zero-point is an integer, ensuring that zero is quantized with no error [5]. Once the scale and zero-point are defined, quantization proceeds as follows:

$$x_{int} = round(\frac{x}{\Delta} + z) \tag{1.1}$$

$$x_Q = clamp(0, N_{levels} - 1, x_{int}) \tag{1.2}$$

where

$$\begin{aligned} clamp(a, b, x) &= a \quad x \leq a \\ &= x \quad a \leq x \leq b \\ &= b \quad x \geq b \end{aligned}$$

The quantized model indeed saves computational resource usage with negligible loss in top-5 accuracy. However, the overall robustness of the model has not been improved, and the DNN-based model is still vulnerable to adversarial attacks. Producing an adversarial example can be summarized as adding imperceptible perturbation to the target image. Such modification can confuse the image classifier to return a wrong label. In many application scenes, the DNN-based model should be robust enough when being exposed to adversarial attacks. Otherwise, it may cause effect the performance of the relative system and bring safety concerns when taking autonomous vehicles even worse. Therefore, it is essential to study how to properly attack an image classifier such that we know how to defend a DNN-based model. Also, studying adversarial examples produced under minimal scenarios might give new insights about the geometrical characteristics and overall behavior of DNN's model in high dimensional space [6]. For instance, the attributes of adversarial images close to the decision boundaries can assist with describing the boundaries' shape.

This project aims to use various attack methods to attack quantized and non-quantized versions of state-of-art image classifiers. We adapted SimBA [7] and One-pixel attack [8] to better comply with the ILSVRC2012-CLS dataset and 16-bit block floating-point quantized version of EfficientNet-B0 and EfficientNet-B7. Unlike the original version of SimBA adding perturbation to the whole perceptive area, Sparse SimBA and Adaptvie SimBA are focusing a random window and adding a well-tuned modification to the specific window. The Dense One-pixel attack is proposed to improve the success rate of adversarial attacks for large-scale images within the data set. We analyzed the performance of these three attack methods in terms of the visual quality of adversarial examples, accumulated success rate, and model queries. The content of this report is shown as the following

structure:

- **Chapter 2–Background:** Introduce the basic knowledge about image classifier, deep learning and important concepts in adversarial attacks.
- **Chapter 3–Literature Review:** The related works about adversarial attacks and the original version of One-pixel and SimBA attack. Also, it includes the information about EffientNet models.
- **Chapter 4–System Architecture and Analysis:** We first introduced the architecture of the attack toolkit and the implementation of differential evolution. Also, the data set and visual quality metrics are included in this chapter. Finally, we detail the modified attack methods: Adaptive and Sparse SimBA and Dense One-pixel attack.
- **Chapter 5–Experiment:** This Chapter is mainly about the experiment to compare these three attack methods targeted at EfficientNet-B0 and EfficientNet-B7. We also analyzed them in terms of the visual quality, accumulated success rate, and model queries.
- **Chapter 6–Evaluation and Result:** We summarized the results of each attack method in this chapter. There are also some unsolved problems we encountered during the project. These problems and difficulties are addressed and discussed.

Chapter 2

Background

It is required to generate a perturbed image and use this adversarial example to attack the quantized model. The aim of Chapter 2 is to introduce basic knowledge of image classifier and adversarial attacks. We firstly introduce the development of image classifiers and a few corresponding quantized techniques. Section 2.2 mainly focuses on the background of adversarial attacks and presents the differences between white-box attacks and black-box attacks. In Section 2.3, some relative terms about adversarial attacks are introduced.

2.1 Image Classifier

Generally, image classification has two phases: interpret pictures with feature learning methods and try to use image classifier to identify images' specific categories [9]. It is important to choose an appropriate way to extract image features; however, the design of feature extraction has been proved to be challenging and arduous.

Image classification techniques have evolved for decades. Traditionally, the process of classification relies on pixels, which means computers comprehend a picture by looking at every single pixel. The support vector machine(SVM), a supervised learning model in the machine learning field, is also used for classification. Previous literature also checks different application scenarios and selects accordingly appropriate classification process. For instance, in remote-sensing land cover classification, the semi-supervised method should

be leveraged to overcome the time-consuming and inadequate training samples [10].

2.2 Deep Learning

In recent years, convolutional neural networks(CNNs) is introduced to image classification and have become a powerful method for image feature extraction. With the development of computing resources and algorithms, deep convolutional neural networks(DCNNs) guarantee their popularity among modern classification. After the breakthrough in the ImageNet Large Visual Recognition Challenge(ILSVRC) achieved by DCNNs, further research improves its performance by advancing nonlinear activation functions, supervision components, optimization techniques, etc. This project also utilizes DCNNs to solve image classification problems.

DCNNs is based on standard CNN architecture, including input images, convolutional layers, pooling layers, fully connected layers, and output class. To be more specific, convolutional layers are used to extract feature from input images, while pooling layers serve to realize spatial invariance by decreasing feature maps' spatial resolution [11]. The rest of fully connected layers, then, help explain abstract feature representations extracted by previous stacked layers [12].

Deep learning improves CNNs' performance and gives birth to DCNNs. It introduces unsupervised pre-training, performed pseudo-tasks and transfer knowledge to DCNNs, and succeed in a series of visual tasks like image classification [13]. DCNNs has been further advanced with modifications on layers, for example. This includes using multilayered perceptron (MLPs) as substitute for convolutional filters [14] and leveraging stochastic pooling as a regularization fashion in pooling layers [15]. Besides, DCNNs also utilized Softmax Loss and L2-SVM Loss [16], which are new supervision components, as loss functions to be minimized.

However, as with other techniques, there are still unsolved problems in DCNNs. For example, there is a lack of robustness when DCNNs encounter adversarial attacks, showing the indispensable gap between human and computer vision capabilities. Therefore, this

project mainly focuses on a few attack methods and trying to improve the performance of attack methods. Having a solid understanding of attacking the model helps us know how to defend the DCNNs.

2.3 Adversarial Attack

As discussed in Section 1.3, deep learning has played an important role in image classification and accomplished the corresponding tasks with high accuracy. However, Szegedy et al. [17] found an apparent weakness of image classifiers utilizing deep neural networks, which is that the prediction results can be altered when the input images are added with small perturbations. These perturbations have a negligible effect on human vision quality. Under certain circumstances, these perturbed images can be predicted into exact categories. These findings have raised researchers' interest in the adversarial attacks and how to defend these attacks for deep learning models.

We then introduce some standard terms used in adversarial attacks and introduce some basic knowledge about attacks in the context of image classification.

- **adversarial example**: An adversarial example represents the perturbed image that is trying to fool the ML-based model. It is expected that the modified image is can successfully confuse the model so that it gives a wrongly predicted class. As the perturbation is added intentionally, the modified image is so regarded as 'adversarial'. Also, the individual who produces this example is called adversary.
- **adversarial perturbation**: it is referred to the modification added on the clean form of input. There are some constraints for the adversarial perturbation when generating adversarial examples. The visual quality is supposed to remain above a certain level so that it is still easy for a human to categorize the original class. We are using some metrics such as absolute value norm, PSNR to qualify the visual quality.
- **query**: Each time when an adversarial example is sent to the model, and a prediction result is returned, we call this process a query to the image classifier. One of the

directions to optimize the attack techniques is to reduce the number of queries, which can accordingly accelerate the production of adversarial examples.

- **Non-targeted/targeted attack**: Targeted adversarial attacks are opposite to non-targeted adversarial attacks. As the name suggested, the predicted results are limited to a specific class. On the contrary, non-targeted attacks alter the predicted results to any other classes besides the correct class.

- **adversarial training**: It is referred to the process where adversarial images are utilized as training data set to train the machine learning model.

- **transferability**: It measures if the adversarial example remains effective when it is generated locally but used to attack other models [18].

- **white-box/black-box attack**: The difference between white-box attacks and black attacks is the knowledge of our targeted model. For black-box attacks, only the prediction results are available. In deep neural networks, the resulted probabilities are the only information we can utilize to generate adversarial examples. For white-box attacks, it is assumed that the information about the mode (parameters, architecture and etc.) is fully accessible to generate adversarial examples.

- **one-shot/iterative attack**: For the one-shot attack, the attacker can only finish one single shot to attack the model, which means that there are no fine-tuning steps when modifying the adversarial example. For iterative attacks, they perform better in terms of visual quality, but more computational resources are also required.

- **success rate**: It is a significant metric to evaluate an adversarial attack method. Given a specific data set, the success rate demonstrates the proportion of successful adversarial examples.

2.3.1 White-box Attack against Black-box Attack

In most practical scenarios, black-box attacks are more likely to happen. Take the adversary and defender as an example, the defender is responsible for training and deploying

the model, while the adversary is intended to break the model with a specific goal. The information of the targeted system cannot be fully available for the hack in most cases. Therefore, attack methods which can successfully attack the system with limited knowledge are more worthwhile to go deep for researchers. In our project, we only discuss the black-box attack methods on image classification.

Chapter 3

Literature Review

3.1 Summary of Previous Research

Due to the characteristics of Deep Neural Networks, more and more people are focusing on the security problem of DNN. Using a few gradient-based algorithms based on back-propagation for obtaining gradient information, C. Szegedy [17] firstly proposed the sensitivity to well-tuned artificial perturbation. In 2011, I.J.Goodfelllow et al [19] proposed a "fast gradient sign" algorithm for calculating effective perturbation based on a hypothesis that the linearity and high-dimensions of inputs are the main reason why a broad class of networks are sensitive to small perturbation. Some of the researchers like S.M.Moosavi-Dezfooli et al. [20] proposed a perturbing algorithm based on the greedy searching method and assumed that the linearity of DNN decision boundaries. Also, N.Papernot et al. [21] built "Adversarial Saliency Map" using Jacobian Matrix, which demonstrates how effective when conducting a fixed-length perturbation through the direction of each axis. Not only adding perturbation on the target image, but it is also possible to produce adversarial images through rotating or artificializing the image [22] [23]. It is also common to add adversarial perturbation to serial data like music [24], speech recordings [25] and text information [21].

Corresponding to various adversarial attack methods, researchers have also developed many detection and defense methods to mitigate the vulnerability induced by

adversarial perturbation. Network instillation [26] is one of the methods to increase the robustness of the neural network through squeezing information of a network to a smaller one. However, it reduces the model's sensitivity. Also, adding adversarial images into the training data set, which is called adversarial training [27] is able to improve the robustness to specific types of adversarial examples.

There are some effective methods to detect adversarial attacks. B.Liang et al. [28] show that noise reduction such as scalar quantization and spatial smoothing filter can be selectively utilized for mitigating the effect of adversarial attack. The detection can be accomplished by comparing the label of an image before and after the transformation. Also, W.Xu et al. [29]proposed an algorithm to squeeze color bits and local/non-local spatial smoothing aiming to have a high success rate when detecting adversarial images. However, some recent studies [30] [31] have demonstrated that a mild modification of the attack method can get rid of the detection.

A black-box adversarial attack means that there is nothing we need to know when attacking a model. Except for the SimBA and One-pixel attack methods we are discussing, another black-box attack method called zeroth order optimization (ZOO) [32] performs as effective as the state-of-art white-box attack (C&W attack [33]) based on MNIST and CIFAR10 data set. This attack method utilizes zeroth order stochastic coordinate descent and dimension reduction, hierarchical attack, and importance sampling techniques.

3.2 EfficientNet

EfficientNet is currently one of the best ConVNets models which balances accuracy and efficiency well. To achieve better performance of ConvNets, it is found that scaling the networks in specific direction is an effective method. The networks can be scaled up in terms of depth/width/resolution. For instance, the depth of ResNet grows from ResNet-18 to ResNet-200. The prediction accuracy also improves [34]. Also, WideResNet [35] which scales the network width (#channels) can also boost the performance. Though it is less common to scale up models through image resolution, a few relative researches have

already been conducted (GPipe) [36]. Scaling only one dimension of depth, width and image size does improve the accuracy. However, scaling arbitrarily may require tedious manual tuning and produce sub-optimal accuracy and efficiency if it is possible to scale two or three dimensions simultaneously. In EfficientNet [2], researchers have proposed a method to scale up the networks in a more structured manner, which uniformly scales each dimension with a fixed set of scaling coefficients. Powered by this compound scaling method and AutoML [37], they have developed a family of models called EfficientNetB0-B7. The most powerful mdoel in this famliy is Efficent-B7, which achieves state-of-art 84.3% top-1 accuracy on ImageNet, also being 8.4x smaller and 6.1x faster than GPipe. The details of this compoud scaling method is shown below:

$$\begin{aligned} depth &: d = \alpha^{\phi} \\ width &: w = \beta^{\phi} \\ resolution &: r = \gamma^{\phi} \\ s.t. &\alpha \cdot \beta^2 \cdot \gamma^2 \approx 2 \\ &\alpha \leq 1, \beta \leq 1, \gamma \leq 1 \end{aligned} \tag{3.1}$$

where α, β, γ are constants that can be determined by the small grid search. We firstly conducted a grid search to find the relationship between different scaling dimensions of the baseline network under a fixed resource constraint. In the equations above [2], ϕ is a user-specified coefficient that controls how many more resources are available for model scaling, and α, β, γ determines how to assign these extra resources to network width, depth and resolution, respectively. Notably, the FLOPS of a regular convolution operation is proportional to d, w^2, r^2. More specifically, doubling network depth will double FLOPS, but doubling network width or resolution will increase FLOPS by four times [2]. As convolution operations usually dominate the computation cost in ConvNets, scaling a ConvNet with the above equation will approximately increase total FLOPS by $(\alpha \cdot \beta^2 \cdot \gamma^2)^{\phi}$. In this paper [2], we constraint $\alpha \cdot \beta^2 \cdot \gamma^2 \approx 2$, which means we want to use 2^N times more computational resources (FLOPS). The network is then increased α^{ϕ} in depth, $\beta^{2\phi}$ in width and $\gamma^{2\phi}$ in image size. Figure 3.1 demonstrates the differences between baseline networks,

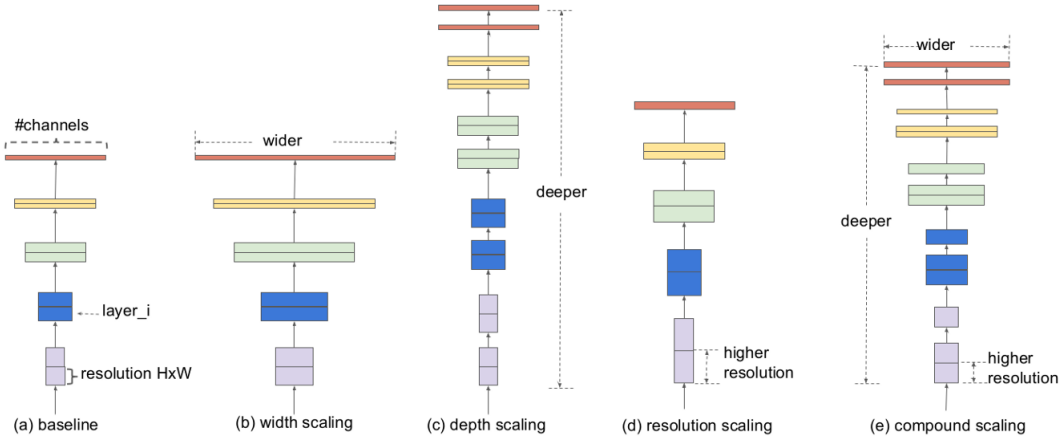

Figure 3.1: (a) is a baseline network example; (b)-(d) are conventional scaling that only increases one dimension of network width, depth, or resolution. (e) is our proposed compound scaling method that uniformly scales all three dimensions with a fixed ratio [2].

networks with only one scaling dimension ,and networks with compound scaling.

In our experiment, we are currently using the Efficient-B7 for image classification. It is the best and most prominent member of EfficientNet models. Tested on a 1000 image subset of the ImageNet ILSVRC2012 validation set, the full precision ,and 8-bit models achieved a top-1 accuracy of 74.2% and 72.7%, respectively.

3.3 One-pixel Attack

The One-pixel attack method is a black-box attack method, which is proposed in 2019. As we introduced in Chapter 2, the only accessible information for a black-box attack is the output probabilities from the DNN model. Compared with other attack methods, the one-pixel attack has the following advantages [8]: 1) The one-pixel attack method performs effectively in the Kaggle CIFAR-10 dataset, the original CIFAR-10 dataset [38] and the BVLC AlexNet model [39]. The corresponding successful rate can be found in [8]. 2) It is strictly categorized to the black-box attack method, which is more likely to be applied in the real world. 3) The one-pixel attack method has the tranferability to attack many types of DNN-based models.

3.3.1 Differential Evolution

It is common to model an optimization problem by creating an objective function with the constraints. In most cases, the objective function defines the optimization problem as a minimization task. The objective function is normally called as "cost" function. Direct search approaches are usually the appropraite options when the cost function is nonlinear and non-differentiable. The greedy decision is utilized in most standard direct search methods. However, it takes the risks of encountering the local minimum. Some researchers have proposed another general category of direct search algorithms that are utilizing inherently parallel search techniques such as genetic algorithms and evolution strategies. Differential Evolution (DE) is a minimization method proposed by R. Storn in 1997 [1], which generally belongs to evolutionary strategies. It has been proved to have the following traits:

- DE is a stochastic direct search method. Therefore, this algorithm can be easily applied to the minimization experiments in the physical world.
- It has the parallelizability to cope with computation-intensive cost functions. It achieves its goal by using a vector population where the stochastic perturbation of population vectors can be done independently [1].
- Only a few inputs are required from the user. It takes the difference vector of two randomly chosen population vectors to perturb an existing vector.
- The DE algorithm has good convergence properties, which is explained in the following descriptions.

DE utilizes NP D-dimensional parameter vectors $x_{i,G}$ to conduct direct search, where NP denotes the population size.

$$x_{i,G}, i = 1, 2, \cdots, NP \tag{3.2}$$

The population size NP remains the same during the process. We randomly selected the initial vector populations, which are supposed to cover the entire parameter space.

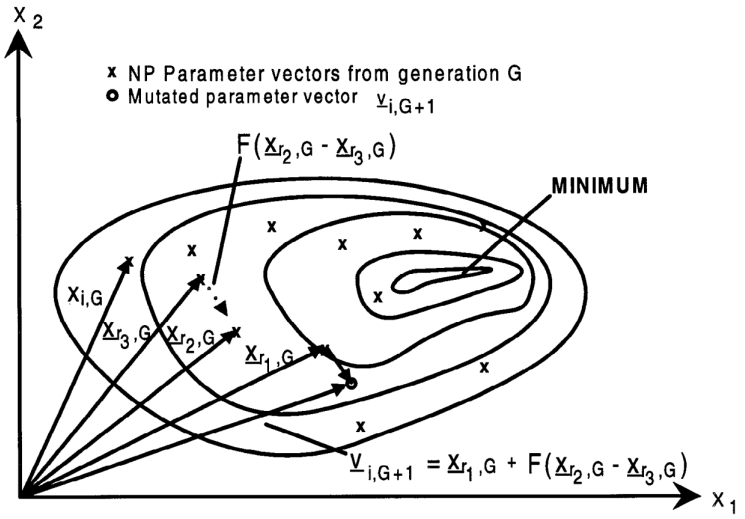

Figure 3.2: An example of a two-dimensional cost function showing its contour lines and the process for generating $v_{i,G+1}$ [1]

Basically, we assume that all of the decisions are distributed with equal probability. If a solution is known in advance, we can add normally distributed random deviations to the nominal solution $x_{nom,0}$. Each time when new parameter vectors are generated, we add the weighted difference between two population vectors to a third vector. This operation is called mutation. For each target vector $x_{i,G}, i = 1, 2, 3, \cdots, NP$, a mutant vector is generated as

$$v_{i,G+1} = x_{r_1,G} + F \cdot (x_{r_2,G} - x_{r_3,G}) \tag{3.3}$$

where $r_1, r_2, r_3 \in \{1, 2, \cdots, NP\}$, are mutually different and $F > 0$. Also, the integers r_1, r_2, r_3 are also chosen to be different from the running index i, so that the population size NP must be greater or equal to four to allow for this condition. $F \in [0, 2]$ is a real and constant factor which controls the amplification of the differential variation $x_{r_2,G-x_{r_3,G}}$. Figure 3.2 demonstrates how the different vectors are evolving to generate the target vector $v_{i,G+1}$.

Another technique to increase the diversity of parameter vectors is called crossover. After the crossover, a trial vector is introduced, which is displayed as

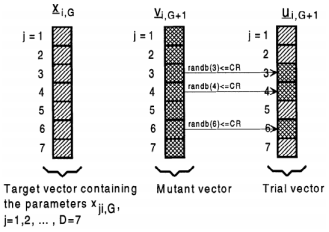

Figure 3.3: An example of a crossover for D = 7 parameters [1]

$$u_{ji,G+1} = \begin{cases} v_{ji,G+1} & \text{if } (randb(j) \leq CR) \text{ or } j = rnbr(i) \\ x_{ji,G+1} & \text{if } (randb(j) > CR) \text{ or } j \neq rnbr(i) \end{cases} \tag{3.4}$$

where $j = 1, 2, \cdots, D$, D is the size of the parameter vector. Also, $randb(j)$ denotes the j_{th} evaluation of a uniform random number generator with outcome $\in [0, 1]$, and CR represents the crossover constant $\in [0, 1]$ which is determined by the user. $rnbr(i)$ is selected randomly from $1, 2, \cdots, D$ which guarantees that $u_{i,G+1}$ gets at least one parameter from $v_{i,G+1}$. Figure 3.3 illustrates how the crossover mechanism for the parameter vectors of length 7.

This final step of DE is called selection. We should decide whether the trial vector should become a member of generation G+1. The trial vector $u_{i,G+1}$ is compared to the target vector $x_{i,G}$ using the greedy criterion. The $x_{i,G+1}$ is set to $u_{i,G+1}$ if $u_{i,G+1}$ yields a smaller cost function value than $x_{i,G+1}$; otherwise, the old value $x_{i,G}$ is retained.

3.3.2 Method

In the One-pixel attack method, the perturbation vector is encoded into an array, which will be optimized through DE. The array is also called the candidate solution. There are a fixed number of perturbation vectors in one candidate solution. The structure of each perturbation vector **x** is a tuple containing five elements:

$$\mathbf{x} = (x, y, R, G, B)$$

which represents the x,y coordinates and the RGB value of the target perturbation pixel. The size of the candidate solution is population size (NP). As we introduced in section 2.3.1, the initial candidate solution is generated, and then the candidate solution will be updated after one iteration. In our experiment, we utilized mutation to generate new candidate solutions, and crossover is currently not incorporated into the scheme. In equation 3.3, the scale factor F is set as 0.5. As the work done by [8], the maximum number of iteration is set to 100. As for non-targeted attacks, when the probability of true label is reduced below 0.05, the algorithm will stop before the limits of maximum iterations. The cost function in our experiment is simply the returned probability of true class.

3.4 SimBA attack

The process of producing an adversarial example can be regarded as an optimization problem with constraints [7]. An input image is represented by an array where each element represents one pixel. We assume f to be the target image classifier which receives n-dimensional inputs, $\mathbf{x} = (x_1, \cdots, x_n)$ be the natural image correctly classified as class t. The predicted probability of $\mathbf{x}$ belonging to the class t is $f_t(\mathbf{x})$. The vector $e(\mathbf{x}) = (e_1, \cdots, e_n)$ is an addictive adversarial perturbation according to $\mathbf{x}$. Our goal is that find a small perturbation so that the prediction $f(\mathbf{x} + e(\mathbf{x})) \neq t$.

SimBA algorithm is intuitive and simple [7]. The inputs are the target image label pair $(\mathbf{x}, y)$, a set of orthonormal candidate vectors Q and a step- $\epsilon > 0$. For any direction $\mathbf{q}$ and some step size ϵ, one of $\mathbf{x} + \epsilon\mathbf{q}$ or $\mathbf{x} - \epsilon\mathbf{q}$ is likely to decrease $p_h(y|\mathbf{x})$. We therefore repeatedly pick random directions $\mathbf{q}$ and either add or subtract them [7]. First of all, $\epsilon\mathbf{q}$ has been added. If this modifications decreases the probability $p_h(y|\mathbf{x})$ we take the step, otherwise we try subtracting $\epsilon\mathbf{q}$. In order to guarantee the maximum query efficiency, it is important to make sure that there are no two directions cancel each other out and diminish progress, or amplify each other and increase the norm of σ disproportionately. Therefore, we selected $\mathbf{q}$ *without replacement* and restrict all vectors in Q to be orthonormal. As shown in [7], this will result in a guaranteed perturbation norm $\|\sigma\|_2 = \sqrt{T}\epsilon$ after T

updates. In simBA, we only need the set of orthogonal search vectors Q and the step size ϵ. The common basis utilized in SimBA includes Cartesian basis, Discrete cosine basis (DCT) and General basis. The pseudocode is presented in algorithm .

Algorithm 1 Original SimBA where $\mathbf{x}$ represents the input image, y is the corresponding class, Q is the orthogonal search vectors and ϵ represents the step size, $p_h(y|\mathbf{x})$ is the probability for class y based on model h, δ represents the perturbation, $y^{'}$ is the targeted class.

$\delta = \mathbf{0}$
$\mathbf{p} = p_h(y|\mathbf{x})$
while $\mathbf{p}_y = \max_{y'} \mathbf{p}_{y'}$ **do**
  Pick randomly without replacement: $\mathbf{q} \in Q$
  for $\alpha \in \{\epsilon, \epsilon\}$ **do**
    $\mathbf{p}^{'} = p_h(y|x + \delta + \alpha q)$
    if $\mathbf{p}^{'}_y < \mathbf{p}_y$ **then**
      $\delta = \delta + \alpha q$
      $\mathbf{p} = \mathbf{p}^{'}$
      break
    end if
  end for
end while
return δ

Chapter 4

System Architecture and Analysis

4.1 System Architecture

The main architecture of the framework is based on the work done by Gasim Ahmed [40]. To make it compatible with new attack methods, image quality metrics, and DNN-based models, Gasim utilized modular design in the framework. The total structure can be divided into three stages: loading and preprocessing the data, generating the adversarial example, attacking the image classifier, and finally collecting the metrics and logging the information. Figure 4.1 demonstrates the structure of the modular framework using a high-level block diagram.

4.1.1 Load Data Module

The data load module is built in the framework to accelerate the process of experimentation. The images in the data set and corresponding ground truth are loaded from the downloaded files and preprocessed based on the requirements of a given model. It is also capable of saving and reloading the preprocessed data. Therefore, the step of preprocessing data is no longer needed when loading saved data. Also, there is a useful function in the load data module to select the specific indexed images from the provided data set. For example, users can specify ’

$$(10, 20), (45, 47), (50, 70)$$

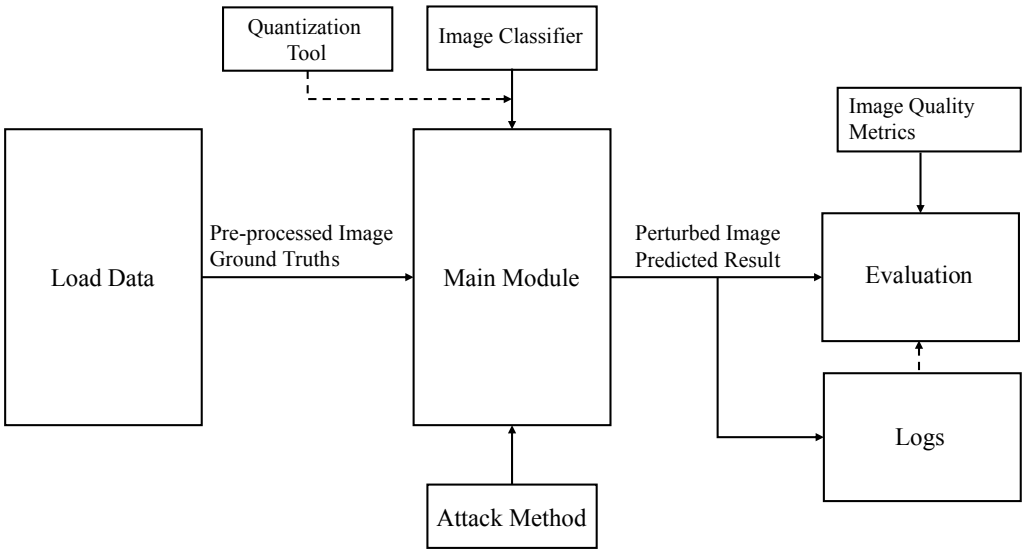

Figure 4.1: The high-level structure of the framework

' to select three different ranges of the required dataset and produce a new data set.

4.1.2 Model Module and Attack Module

A model module consists of an image classifier and quantization tool where the quantization tool is basically the TensorFlow Lite converter. The required input information of this module is the output probabilities for each model query. As there are various abstract functions, the prediction results returned by the model are fully appearing or only showing the top predictions. An important metric in a model module is to track the model query and log the information throughout attacks. The attack success varies under a different situation; the attack module is also responsible for assessing the success and determines when to stop the iteration. For example, the limit of model queries is achieved, or the image quality is dropped below a particular value.

An attack module provides a few choices for the user to attack a given image classifier. It allows users to pass the target model, attack methods, and also the metrics name for evaluation. The module returns the adversarial success rate of specific attack methods.

4.1.3 Metrics Module and Logs module

A metric module assesses a given original image and perturbed image based on specified image quality metrics. As we mentioned in section 4.2.2, we are mainly focusing on the absolute value norm, SSIM, and PSNR. As for implementing the traditional metrics, we utilize the NumPy Linear Algebra and Scikit-image Metrics libraries.

A logger module allows for attack modules to log attack information and uses a metric collector module to add the model query count and results of visual quality assessments using metrics specified by the user to each logged item. The experiment logs are well designed for users to manipulate; an instance of the logger module is created for each attack method in an experiment. Each instance stores a separate log for each sample when its respective attack method is carried out. The logger module stores experiment logs as Pandas DataFrames, a high-level data structure for storing tabular data.

4.1.4 Quantization Toolbox

According to Krishnamoorthi [41], he utilizes the TensorFlow.contrib.quantize toolkit to quantize models for post-training for inference and also provide full-precision models for quantization-aware training. The method of quantization does not quantize the model but simulates the quantization. The algorithm is about inserting fake quantization ops into the classifier's graph [42]. The toolkit provides a stable converter that performs quantization to 16 and 8 bits. There are mainly two functions in the quantization toolbox. The first one is to take the path as the input and load, quantizes and returns the image classifier as a model module in the TensorFlow SavedModel format. Another function is to take a link to a pre-trained TensorFlow Hub classifier, downloads the classifier, quantizes and returns it as a model module. It is important to note that the TensorFlow Lite converter maintains 32-bit precision for the input and output layers of quantized models, allowing the same preprocessed input images to be used for both full-precision and quantized model.

4.1.5 Google Colab Tools

Google Colaboratory(Colab) is a Python development environment that runs on Google Cloud, which can be assessed through a web browser. Google Colab provides access to a virtual machine with a GPU, 12 GB of RAM, and 100 GB of storage space for free [43]. The limited time for the user is restricted to a continuous 12-hour time slot; then the virtual machine is allocated to another user. Though it is possible for the user to reconnect to the virtual machine, the data generated and stored in the local virtual machine will be deleted when switched to another device. In our experiment, we save the experiment file to our local Google drive path.

4.1.6 Implementation of Differential Evolution

The goal of the differential evolution algorithm is to find the global minimum of a multivariate function. Differential Evolution is stochastic in nature and does not involve the gradient method. It can search large areas of candidate space but often requires a larger number of function evaluations than conventional gradient-based methods.

The inputs of the main function are shown as below:

```
def differential_evolution(func, bounds, args=(), strategy='best1bin',
                           maxiter=1000, popsize=15, tol=0.01,
                           mutation=(0.5, 1), recombination=0.7, seed=None,
                           callback=None, disp=False, polish=True,
                           init='latinhypercube', atol=0):
```

The first input is a callable function that is to be minimized. The form of this callable function is restricted as "f(x,*args)", where "x" represents the argument in the form of a 1-D array and "args" denotes a tuple of additional fixed parameters needed to specify the function completely. Also, the second input parameter is the bounds specified for variables. "(min, max)" pairs for each element in "x" defines the lower bound and upper bounds for the optimizing argument of the function to be optimized. It is required to have $len(bounds) == len(x)$, which means $len(bounds)$ is the same as the number of parameters we need to optimize in x. In our experiment, for the original one-pixel

black-box attack, the length of the target tuple is $(xposition, y_position, R, G, B)$ and their corresponding bounds are the size of the input image the pixel value from 0 to 255.

At each pass through the population, the algorithm mutates each candidate solution by mixing with other candidate solutions to create a trial candidate. The input "strategy" refers to the differential evolution strategy to mutate the candidate solutions, which should be one of the strategies below. The default strategy is set as 'best1bin', which is also described in equation 3.3.

The algorithm is started with a randomly chosen i_{th} parameter, and the trial is sequentially filled with parameters from '$b^{'}$' or the original candidate [44]. The choice of whether to use '$b^{'}$' or the original candidate is made with a binomial distribution (the 'best1bin'), which means a random number in $[0, 1)$ is generated. This number is compared with the recombination constant. If it is smaller, the parameter is loaded from the original candidate. '$b^{'}$' always produces the final parameters. Once the trial candidate is built, the algorithm will assess its fitness. If the trial is better than the original candidate, then it replaces the previous one. If it is also better than the best overall candidate, it also takes its place [45].

$$b^{'} = b_0 + mutation * (population[rand0] - population[rand1]) \tag{4.1}$$

The input 'maxiter' is a int-type variable referring to the maximum number of generations over which the entire population is evolved. The maximum number of function evaluations is calculated as:

$$(maxiter + 1) \times popsize \times len(x) \tag{4.2}$$

The input "popsize" is a multiplier for setting the total population size. The population has a length of $popsize \times len(x)$ individuals unless the initial population is supplied via the 'init' keyword. Another parameter, "tol" is a float variable that defines the tolerance for convergence, the solving stops when

```
numpy.std(pop) <= atol + tol * numpy.abs(numpy.mean(population_energies))
```

where 'atol' and 'tol' denote the absolute and relative tolerance respectively.

'Mutation' defines the mutation constant. Based on Equation 3.3, this variable is denoted by F. It should be in the range $[0, 2]$ if it is specified as a float. If it is specified as a tuple "(min,max)", then dithering is employed. The dithering algorithm randomly changes the mutation constant on a generation by generation basis [45], which can help speed convergence significantly. The mutation constant for that generation is taken from $U[min, max)$. Increasing the mutation constant speeds the search radius, but will slow down convergence [45] [44].

The 'recombination' represents the recombination constant, which should be in the range [0,1]. In the literature, this is also called crossover probability, being denoted by CR in Equation 3.4. Although at the risk of population stability, increasing this value allows a larger number of mutants to progress into the next generation.

The 'seed' is used to specify the repeatable minimization. There are three circumstances. If the seed is not specified, the 'numpy.random.RandomState' singleton is used. If 'seed' is an int, a new 'numpy.random.RandomState' instance is used, seeded with seed. If 'seed' is already a 'numpy.random.RandomState' instance, then that 'numpy.random.RandomState' instance is used [45]. The variable 'disp' determines whether to display the status messages.

The 'callback' refers to a callable function which is to follow the progress of the minimization. In callback(xk, convergence = val), 'xk' is the current value of 'x0'. 'val' represents the fractional value of the population convergence. When 'val' is greater than one, the function halts. If the callback returns 'True', then the minimization is halted. In our experiment, we predicted the adversarial example. If the predicted class is not the same as the original class, the callback function is then halted.

The 'Polish' refers as a Boolean variable. If 'Polish' is set as True, then 'scipy.optimize.minimize' with the L-BFGS-B method is used to polish the best population member as the end, which can improve the minimization slightly. The parameter 'init' specifies which type of population initialization is performed which should be one of:

- 'latinhypercube'
- 'random'
- 'array specifying the initial population'

where the third type should have the shape (M,len(x)), and len(x) is the number of parameters. 'init' is clipped to bounds before use. The default setting is 'latinhypercube'. This initialization method tries to maximize coverage of the available parameter space. 'random' initializes the population randomly, which has the drawback that clustering can occur, preventing the whole of parameter space from being covered. We can also use of an array to specify a population subset could be used. For instance, it is possible to create a tight bunch of initial candidates in a location where the solution is known to exist, thereby reducing time for convergence.

The Differential Evolution function returns an optimized result. The optimized result is represented as a 'OptimizeResult' object. Important attributes are: "x" is the solution array, "success" is a Boolean flag indicating if the optimized exited successfully.

4.2 Dataset and Metrics

4.2.1 Dataset

The whole experiment is tested on a subset of the ImageNet ILSVRC2012 validation set. The subset contains 1000 images. The full-precision and 16-bit EfficientNet B7 models achieved a top-1 accuracy of 74.2% and 73.4%. Also, for the full-precision and 16-bit EfficientNet B0 models, they achieved 74.3% and 74.2% respectively. The ImageNet Large Scale Visual Recognition Challenge is a benchmark in object category classification and detection on hundreds of object categories and millions of images [46]. It follows the steps of the PASCAL VOC challenge, which is established in 2005 [47]. When creating the ILSVRC dataset, a few challenges have been encountered. They need to scale up from 19,737 images in PASCAL VOC 2010 to 1,461,406 in ILSVRC 2010 and from 20 object classes to 1000 object classes. Therefore, they turn to designing novel crowdsourcing ap-

proaches for collecting large-scale annotations [48]. ILSVRC has consisted of the following three tasks:

1. **Image Classification Task:**

   The CLS (classification) data set consisted of photographs that are labeled with one of 1000 object categories. The task utilizes a top-5 measure of error, which means that each image is predicted five times. As long as the image is correctly predicted, we regard this attempt as a successful prediction.

2. **Single-Object Localization Task:**

   The LOC (localization) task is based on the classification task. The algorithm will produce a few object categories present in each image. At the same time, there is a matching bounding box showing the location and scale of one instance of each object category. They evaluate The quality of labeling is evaluated based on the object category label that best matches the ground truth label. Also, it is important to ensure the accuracy of the location if the predicted instance.

3. **Object Detection Task:**

   The object detection task studies further about single-object localization and deals with the problem of localizing multiple object categories in the image.

   For each image, the algorithms produce bounding boxes indicating the position and scale of all instances of all target object categories. We evaluate the quality of labeling is evaluated by Recall, the number of target object instances detected, Precision, or the number of spurious detection produced by the algorithm.

   More specifically, it is required to find all of the objects with 200 categories, such as humankind, spoon. The criterion is to assess the accuracy of each category and the algorithm achieves the highest accuracy in most categories won this task.

In our experiment, we only utilize the ILSVRC2021-CLS dataset. The detailed information of ILSVRC-CLS is summarized in to Table 4.1.

	Train Images (per class)	Val images (per class)	Test images (per class)
ILSVRC2012	1,281,167 (731-1300)	50,000(50)	100,000 (100)

Table 4.1: Scale of ILSVRC image classification

4.2.2 Metrics

One of the critical criteria to measure the effectiveness of adversarial attacks is the visual quality of the attack example. It is essential to quantify the perturbation in a more direct way. There are some innovative metrics to measure the visual drawbacks, which are also corresponding to human perceived visual quality. The common metrics include the absolute value norm, Euclidean norm, Peak Signal-to-noise ratio (PSNR), Mean Squared Error (MSE), and Structural Similarity Index (SSIM).

Calculating the p-norm distance(l_p) is a common method to measure the magnitude of perturbation when assessing the visual quality of the image. As shown in equation 4.3, $\mathbf{x}$ represents the pixel-wise difference between the original image and perturbed image, x_i is a pixel difference, and N denotes the total number of pixels. There are some normal p-norm distances such as l_0, l_2 and l_∞. In our experiment, l_0 computes the total number of pixels perturbed, l_2 measures the Euclidean distance between the adversarial example and the original image, and l_∞ measures the maximum perturbation added on the image.

$$\|\mathbf{x}\|_p = (\sum_{i=1}^{N} \|x_i\|_p)^{\frac{1}{p}} \tag{4.3}$$

Another metrics MSE is proposed to get rid of negative numbers when calculating the difference of two images.

$$MSE = \frac{1}{MN} \sum_{i=0}^{M-1} \sum_{j=0}^{N-1} [f(x,y) - g(x,y)]^2 \tag{4.4}$$

where M and N represent the dimensions of the image. MSE is a direct method to calculate the difference between two images and is also cost-effective [49]. It satisfies the interpretation of similarity, i.e., non-negativity, identity, symmetry, and triangular

inequality. The lower the MSE is, the more similar the distorted image is to the reference image. It works well when distortion is mainly caused by contamination of additive noise [49].

PSNR is defined as the ratio between the maximum power of a signal to the maximum power of a noise signal. It is measured in terms of peak signal power, whose unit is decibels.

$$PSNR = 20\log_{10}\frac{L^2MN}{\sum_{i=0}^{M-1}\sum_{j=0}^{N-1}[f(x,y)-g(x,y)]^2} = 20\log_{10}\frac{L^2}{MSE} \tag{4.5}$$

In the above equation, $f(x,y)$ is the original reference image and $g(x,y)$ is the distorted image. M and N are dimensions of the image. L denotes the dynamic range of image pixels. When different dynamic ranges are compared, PSNR contains more information than MSE. A higher value of PSNR represents the image has a better quality. When measuring the white noise distortion, PSNR is an excellent measure [50]. PSNR involves simple calculations, has a clear physical meaning, and is convenient in the context of optimization. However, PSNR is not corresponding to the characteristics of the human visual system [50].

As one of the ideal goals is to add imperceptible perturbation to the adversarial example, the inter-dependencies between spatially close pixels must be taken into consideration. The SSIM is a human visual system(HVS) feature-based metric proposed by Wang et al [51], which attempts to measure the structural dissimilarities that are perceptible to humans [52]. The HVS performs many image processing tasks which are superior than other models. SSIM measures the similarity between two images. It is an improvement over methods like MSE and PSNR, which is a weighted combination of luminance, contrast, and structural similarity between a test and reference image. The SSIM is calculated as the equation below.

$$SSIM(x,y) = \frac{(2\mu_x\mu_y + C_1)(2\sigma_{xy} + C_2)}{(\mu_x^2 + \mu_y^2 + C_1)(\sigma_x^2 + \sigma_y^2 + C_2)} \tag{4.6}$$

where μ_x and μ_y are the average of x and y respectively. σ_x^2 and σ_y^2 are the variance of x and y respectively. σ_{xy} is the covariance of x and y.

In our experiment, we mainly utilize the absolute value norm, SSIM and PSNR to measure the difference between the distorted and original images.

4.3 Adaptive SimBA and Sparse SimBA

The SimBA attack method is inspired by the original SimBA method. Instead of selecting random orthonormal candidate vectors and adding perturbations, Sparse SimBA perturbs a $s \times s$ sized window of pixels for every channel in the image per iteration, without replacement. Similar to the original algorithm, we firstly add a positive perturbation of magnitude ϵ to the targeted window and translate the perturbation magnitude ϵ to $\frac{\epsilon}{s}$ per pixel in the window. The pseudo-code is shown in algorithm 2

If the modification reduces the probability of the correct class, the perturbation is then accepted, and a new window is sampled. Otherwise, we will add a negative perturbation to the window and this perturbation is regarded as successful if it causes a drop in the predicted probability.

In order to better correspond to the quantized model, Gasim proposed modified the Sparse SimBA algorithm and call it as adaptive SimBA algorithm. The Adaptive SimBA algorithm is inspired by the idea of decaying learning rate in machine learning algorithms. In the first few iterations, the initial perturbation magnitude is a multiple of ϵ and is decayed periodically at a given rate to ϵ. The details are shown in algorithm 3.

4.4 Dense One-pixel Attack

As the original one-pixel attack method is based on 32×32 pixel image, this attack method is effective due to the limited pixels. However, in our ImageNet data set, the size of required images are 600×600 or 224×224. Attacking only one pixel is no longer a suitable solution for a successful adversarial example. Therefore, we proposed the dense

Algorithm 2 Non-targeted Sparse SimBA where s is the perturbation window size, f_{w*} represents the image classifier, ϵ is the perturbation magnitude, *preds* are the model's labeled top five predictions and $\mathbf{x}$ is initially a correctly classified image with label k and probability p

Require: $\epsilon, s, f_{w*}, \mathbf{x}, k$
$preds \leftarrow f_{w*}(\mathbf{x})$
$p \leftarrow preds(k)$
while argmax($preds$) is k **do**
  $\mathbf{q} \leftarrow$ new $s \times s$ window in x
  $preds \leftarrow f_{w*}(\mathbf{x} + \frac{\epsilon}{s}\mathbf{q})$
  if $preds(k) < p$ **then**
    $p \leftarrow preds(k)$
    $\mathbf{x} \leftarrow \mathbf{x} + \frac{\epsilon}{q}\mathbf{q}$
  else
    $preds \leftarrow f_{w*}(\mathbf{x} - \frac{\epsilon}{s}\mathbf{q})$
    if $preds(k) < p$ **then**
      $p \leftarrow preds(k)$
      $\mathbf{x} \leftarrow \mathbf{x} - \frac{\epsilon}{q}\mathbf{q}$
    end if
  end if
end while
return x

one-pixel attack method. Instead of focusing on one pixel in a large scale, we cut the whole image into a few windows with specified size. Utilizing the differential evolution algorithm, we modified one pixel value of all of the windows. Though it has reduced the visual quality compared with the original method, it has also achieved a much higher success rate. The details of the Dense One-pixel method is shown as the algorithm 4.

Algorithm 3 Non-targeted Adaptive Sparse SimBA where s is the perturbation window size, f_{w*} represents the image classifier, ϵ is the perturbation magnitude, m refers to the magnitude multiplier, r represents the decay rate, T represents the decay period, *preds* are the model's labeled top five predictions and $\mathbf{x}$ is initially a correctly classified image with label k and probability p

Require: $\epsilon, s, f_{w*}, \mathbf{x}, k$
$preds \leftarrow f_{w*}(\mathbf{x})$
$p \leftarrow preds(k)$
For the first decay period $\boldsymbol{T}$
while argmax($preds$) is k **do**
  $\mathbf{q} \leftarrow$ new $s \times s$ window in x
  $preds \leftarrow f_{w*}(\mathbf{x} + m\frac{\epsilon}{s}\mathbf{q})$
  if $preds(k) < p$ **then**
    $p \leftarrow preds(k)$
    $\mathbf{x} \leftarrow \mathbf{x} + m\frac{\epsilon}{q}\mathbf{q}$
  else
    $preds \leftarrow f_{w*}(\mathbf{x} - m\frac{\epsilon}{s}\mathbf{q})$
    if $preds(k) < p$ **then**
      $p \leftarrow preds(k)$
      $\mathbf{x} \leftarrow \mathbf{x} - m\frac{\epsilon}{q}\mathbf{q}$
    end if
  end if
end while
For the following decay periods $\boldsymbol{T}$
while argmax($preds$) is k **do**
  $\mathbf{q} \leftarrow$ new $s \times s$ window in x
  $preds \leftarrow f_{w*}(\mathbf{x} + rm\frac{\epsilon}{s}\mathbf{q})$
  if $preds(k) < p$ **then**
    $p \leftarrow preds(k)$
    $\mathbf{x} \leftarrow \mathbf{x} + rm\frac{\epsilon}{q}\mathbf{q}$
  else
    $preds \leftarrow f_{w*}(\mathbf{x} - rm\frac{\epsilon}{s}\mathbf{q})$
    if $preds(k) < p$ **then**
      $p \leftarrow preds(k)$
      $\mathbf{x} \leftarrow \mathbf{x} - rm\frac{\epsilon}{q}\mathbf{q}$
    end if
  end if
end while
return x

Algorithm 4 Dense one-pixel attack where s is the perturbation window size, f_{w*} represents the image classifier, population size is m, perturbation tuple is l, maximum iteration is I, *preds* represents the prediction result of candidate solutions, $\mathbf{x}$ is initially a correctly classified image with label k and probability p

Require: $s, m, f_{w*}, \mathbf{x}$
randomly selected m candidate solutions l_i $(i = 1, \cdots, m)$
while $iter \leq I$ **do**
  $l_i \rightarrow$ every $s \times s$ window in $\mathbf{x}$
  $preds_i \leftarrow f_{w*}(\mathbf{x}$ *perturbed by* l_i $i = 1, \cdots, m))$
  $l_i^{mutate} = mutate(l_i)$
  $preds_i^{mutate} \leftarrow f_{w*}(x$ *perturbed by* l_i^{mutate} $(i = 1, \cdots, m))$
  if $preds_i^{mutate} < preds_i$ **then**
    $l_i = l_i^{mutate}$
  else
    l_i remains the same
    A new generation of perturbation tuple l is generated
  end if
end while
The last generation of l is produced
$preds_i \leftarrow f_{w*}(x$ *perturbed by* l_i $(i = 1, \cdots, m))$
if $preds_i$ is not k **then**
  l_i **is accepted as the final optimized result**
end if
return x perturbed by l_i

Chapter 5

Experiment

We are mainly focusing on two types of EfficientNet models, which are EfficientNet-B0 and EfficientNet-B7. As we mentioned in section 3.2, the EfficientNet family achieves state-of-art accuracy and also being an order-of-magnitude smaller and faster than previous models. EfficientNet-B0 is a baseline mobile-size model developed by the AutoML MNAS Mobile framework. Then EfficientNet-B7 is scaled up through compound scaling. All of the models are pre-trained based on the ILSVRC-2012-CLS dataset for the classification task. The size of input images is flexible. However, the models perform better if the size of input images are aligned with the training set. As for EfficientNet-B7 and EfficientNet-B0,the image size is 600×600 pixels and 224×224 pixels correspondingly. The color values are within the range of $[0, 1]$. When we are implementing the Dense One-pixel attack method, the color range of the original image is required to be normalized to the range $[0, 255]$.

5.1 Attacks on EfficientNet-B0

5.1.1 Sparse SimBA and Adaptive SimBA Attack

One of the important parameters we need to determine is size of the perturbed window. Intuitively, increasing the size of perturbed windows will reduce the visual quality when attacking target images. As Robledo [53] suggests in Sparse SimBA algorithm implemen-

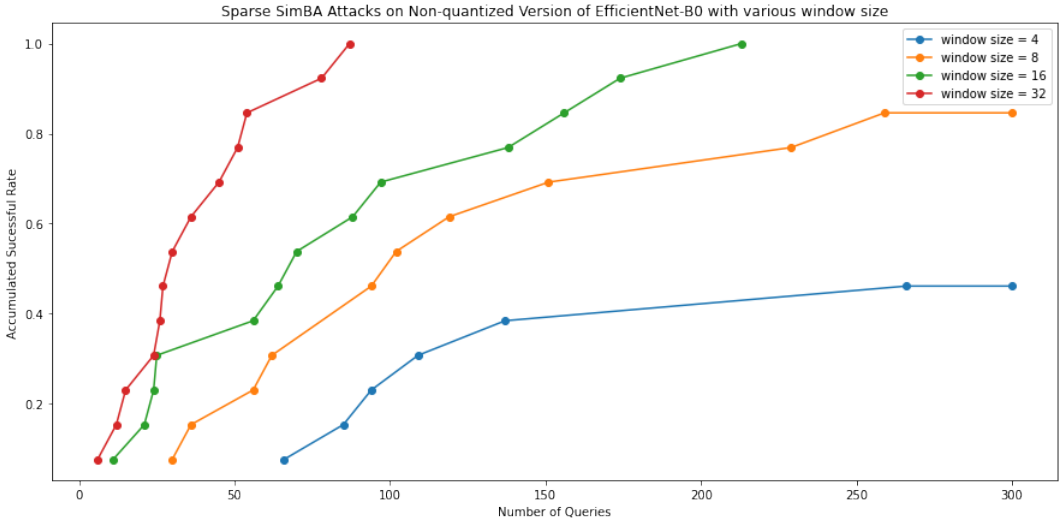

Figure 5.1: Attacking 32-bit EfficientNet-B0 using Sparse SimBA with various window size

window size	model queries	absolute value norm	PSNR	SSIM
4	266	1345	25.598	0.978
8	119	2621	22.933	0.974
16	156	8224	18.998	0.944
32	54	11710	16.644	0.927

Table 5.1: Visual Quality measurement for adversarial examples (various window size)

tation, the recommended use of window size is $s = 8$. We firstly experimented with the Sparse SimBA algorithm on a full-precision Efficient-Net B0 model with various window size, and the maximum model queries is set as 300. Also, the perturbation magnitude added on the window ϵ is set as 1000 temporarily to save the computational cost. The comparison result is shown in Figure 5.1.

When the window size is scaled up to 32×32 and 16×16 pixels, fewer model queries are needed to produce a successful adversarial example. Also, the maximum accumulated success rate is improved under the limitation of model queries. The maximum accumulated success rate is reduced to 38.4% when the window size is 4×4 pixels. The successful adversarial examples under such conditions are shown in Figure 5.2. We can clearly see that with the increasing window size the target image becomes more and more annoying. The detailed PSNR, SSIM, and absolute norm values for each adversarial example are shown in Table 5.1.

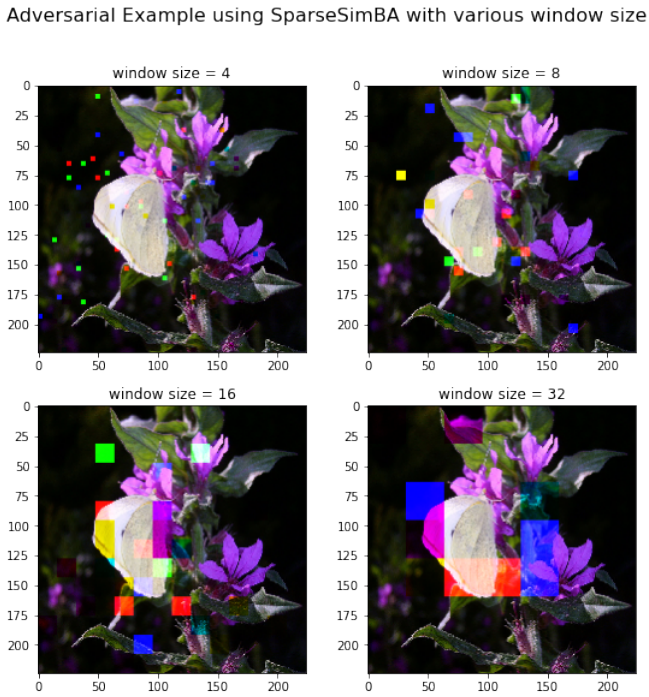

Figure 5.2: Adversarial Examples using Sparse SimBA to attack EfficientNet-B0(various s)

To balance the visual quality and accumulated success rate, we select the perturbed window size as $s = 8$. We then tested the perturbation magnitude ϵ to better visualize the successful adversarial example.

When the perturbation value ϵ decreases to 200, the maximum accumulated success rate achieves 92.3%, which means nearly all images are successfully attacked when model queries are limited to 300. The corresponding measurement of visual quality is summarized in Table 5.3. The perturbation magnitude determines the step when adding noise to the target image. It is found that a higher perturbation value does not guarantee a higher accumulated success rate. However, in terms of the calculated visual quality, when $\epsilon = 800$, it achieves the lowers absolute value norm compared with the original image. Also, the algorithm requires relatives fewer model queries saving the computational cost. The successful adversarial examples are shown in Figure 5.4. The parameters when attacking EfficientNet-B0 are determined as window size $s = 8$, perturbation magnitude $\epsilon = 800$. We further utilized these parameters into the Adaptive SimBA algorithm when attacking quantized EfficicentNet-B0.

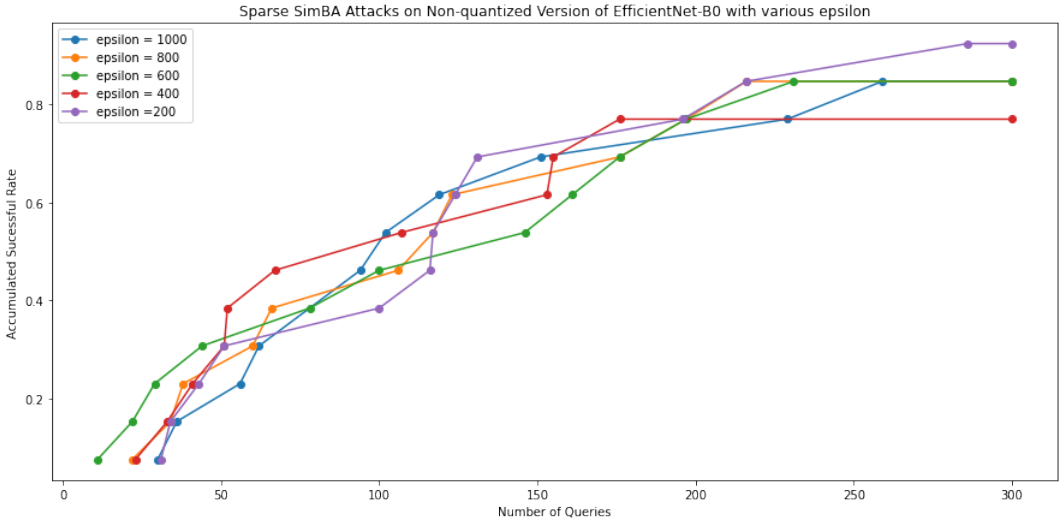

Figure 5.3: Attacking 32-bit EfficientNet-B0 using Sparse SimBA with various epsilon

epsilon	model queries	absolute value norm	PSNR	SSIM
200	131	3047	20.281	0.960
400	153	2796	22.678	0.973
600	197	3852	21.112	0.964
800	123	2592	22.746	0.973
1000	119	2621	22.933	0.974

Table 5.2: Visual Quality measurement for adversarial examples (various epsilon)

We then tested both the Sparse SimBA and Adaptive SimBA algorithms on quantized and non-quantized versions of EfficientNet B0. We plotted the corresponding figure illustrating the relationship between the accumulated success rate and model queries. In Figure 5.5, it is found that Sparse SimBA performs similarly when attacking full-precision and 16-bit EfficientNet-B0, and Adaptive SimBA is more suitable to attack the 16-bit version of EfficientNet-B0. When using Adaptive SimBA to attack the full-precision model, the maximum accumulated success rate is the lowest among these four experiments. It may be attributed to the degradation in visual quality, which also supports the design of adaptive SimBA using decaying perturbation magnitude and backtracking features. Also, for each data point, it is known that the 32-bit model requires more model queries to produce a successful adversarial example when adopting the adaptive method. Figure 5.6 demonstrates the final adversarial examples for EfficientNet-Bo using the SimBA algorithm. Due to the limited computation resources, we increase perturbation magnitude ϵ

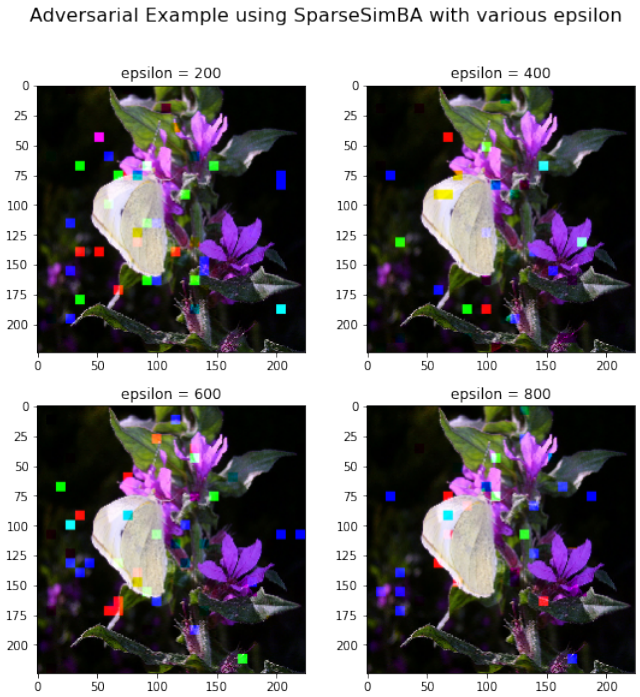

Figure 5.4: Adversarial Examples using Sparse SimBA to attack EfficientNet-B0(various ϵ)

to save running time. However, the visual quality drops to a relatively annoying degree. As the input image size is restricted to 224×224 pixels, an 8×8 window inevitably covers an area that cannot be ignored. The main body within the image is still available in terms of the human vision visual system.

5.1.2 Dense One-pixel Attack

As we mentioned in section 4.1.6, the Dense One-pixel attack is crafted based on the one-pixel black attack method and mainly focusing on ILSVRC-2021-CLS dataset. We again use 8×8 window size to select a key pixel to change its pixel value thtough the differential evolution algorithm. Due to the characteristics of differential evolution algorithm, we need to query the model m times for each iteration, where m represents the size of the generation. Therefore, the total model queries is calculated as *number of iteration* $\times$ *population size*. The bound of our differential evolution is set as $[(0, 8), (0, 8), (0, 255), (0, 255), (0, 255)]$. We again tested our new algorithm on the same

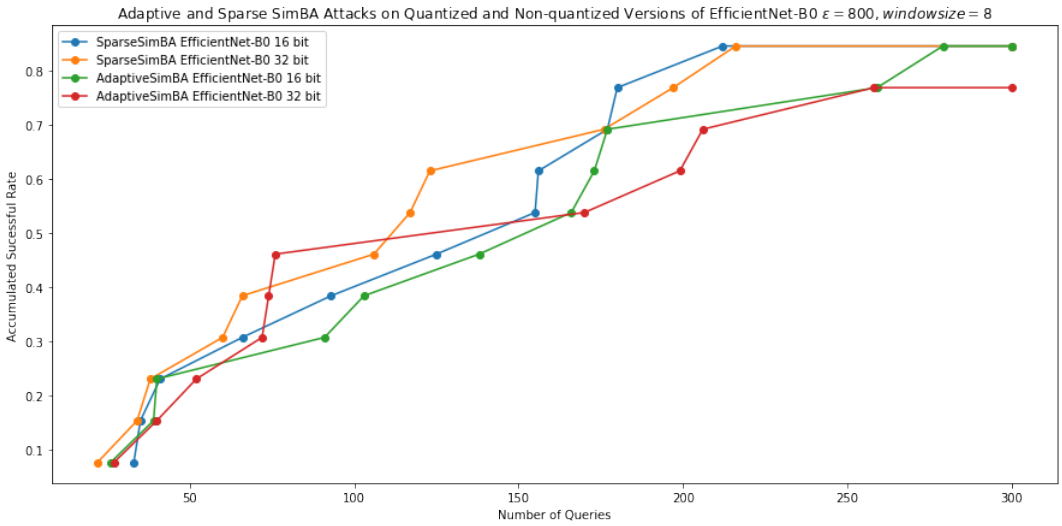

Figure 5.5: Attacking 32-bit and 16-bit EfficientNet-B0 using Adaptive and Sparse SimBA

attack method and quantised version	model queries	absolute value norm	PSNR	SSIM
SparseSimBA 32 bit version	210	3891	20.897	0.958
SparseSimBA 16 bit version	123	2592	22.746	0.973
AdaptiveSimBA 32 bit version	293	4995	21.149	0.958
AdaptiveSimBA 16 bit version	237	3912	21.219	0.961

Table 5.3: Visual Quality measurement for adversarial examples using various attack method

image used in Section 5.1.1. The successful adversarial example is shown in Figure 5.7, and the visual measurement is demonstrated in Table 5.4.

We also tested the Dense One-pixel on the 16-bit version of the EfficientNet-B0 model. It performs similarly on these two models. As it does not require the gradient information of the model, the quantized version of the model does not affect the performance of the Dense One-pixel algorithm. From Table 5.4, in terms of the PSNR and absolute value norm, it performs much better than Sparse SimBA and Adaptive SimBA algorithm. Also, as the perturbed pixel remains at the same position in every window, humans' visual quality is also better than SimBA algorithms.

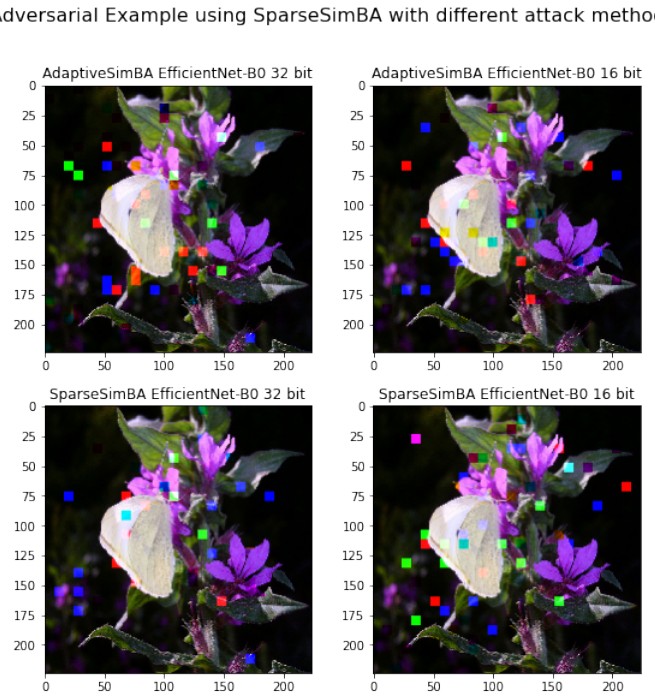

Figure 5.6: Adversarial Examples using Sparse and Adaptive SimBA to attack EfficientNet-B0 16 and 32 bit version

attack method	model queries	original class	predicted class	absolute value norm	PSNR	SSIM
Dense One-Pixel	50	324	325	2170	25.899	0.731

Table 5.4: Visual Quality measurement for adversarial examples using Dense One-pixel attack

We haven't plotted the figure for the following reasons in terms of the model queries and accumulated success rate. Firstly, it is because of the limitation of computation resources. As the usage of Google Colab is restricted to 12 hours, if the maximum iterations is set as 100 and population size is set as 400, an unsuccessful adversarial example requires $100 \times 400 = 40,000$ model queries before generating the next adversarial image. For the EfficientNet-B0 model, the corresponding running time exceeded 12 hours. Also, as the initial candidate solution is produced randomly and the optimized solution is updated through mutation, the accumulated success rate and the model queries may vary for different experiments. Therefore, we cannot guarantee the relationship between the model queries and the accumulated success rate.

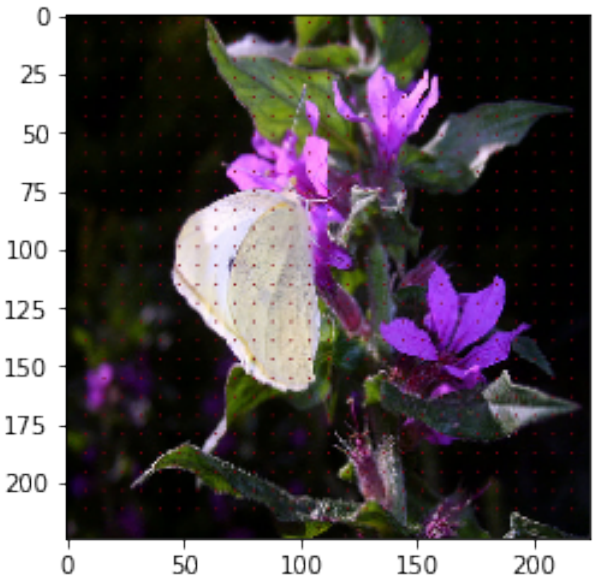

Figure 5.7: Adversarial Example on 16-bit precision EfficientNet-B0 using Dense One-pixel attack

5.2 Attacks on EfficientNet-B7

5.2.1 Sparse SimBA and Adaptive SimBA

The experiment based on EfficientNet-B7 is quite consuming, as for each model query, it may take 5 seconds when running the experiment on Google Colab. Therefore, when we set the maximum model queries as 300, it may take 15 minutes to finish a circle if the image has not been successfully attacked. We firstly adopted the Sparse SimBA algorithm on both the full-precision and 16-bit EfficientNet-B7 model. The window size is set as 16×16 pixels and the epsilon is set as 1000 temporarily. The accumulated success rate and the model queries are demonstrated in Figure 5.8. It is clear to conclude that Sparse SimBA performs poorly when the model is quantized to the 16-bit version. The degradation is attributed to vanishing gradients and local minima in the space of the algorithm's heuristic's space due to quantization. Similarly, as we mentioned in Section 5.1.1, Adaptive SimBA performs poorly when targeting at the full-precision model. Therefore, with the limitation of computation resources, we only apply Sparse SimBA to the full-precision model and Adaptive SimBA to quantized version model.

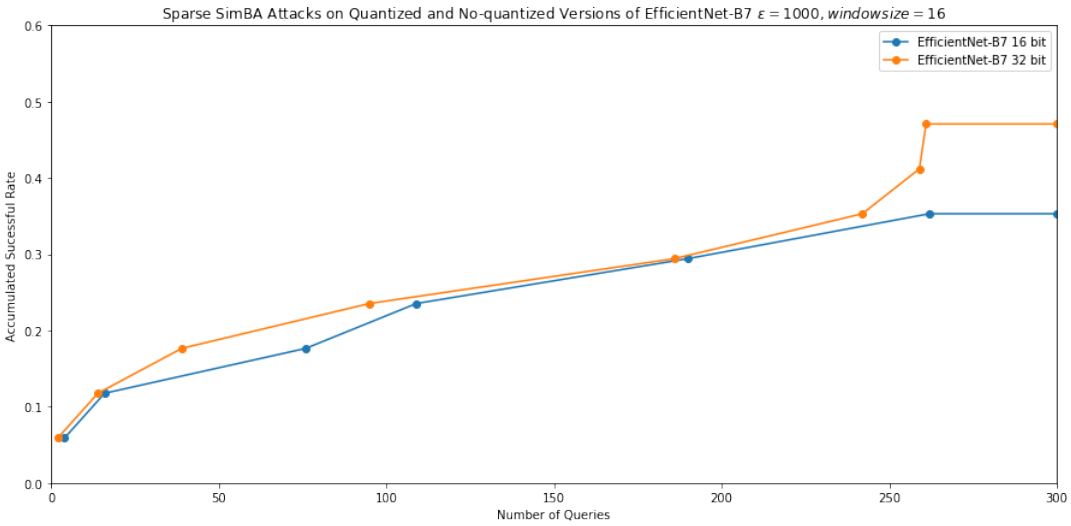

Figure 5.8: Attacking 32-bit and 16-bit EfficientNet-By using Sparse SimBA (windowsize = 16, $\epsilon = 1000$)

In Figure 5.9, we plotted the accumulated success rate versus model queries for both Sparse SimBA and Adaptive SimBA attack. Both of the attack methods are tested on a subset of ILSVRC2012-CLS of size 20. 17 images of the subset are correctly predicted and ready to be attacked. From Figure 5.10 and Figure 5.11, only four images are successfully attacked. The total success rate is 23.5% for these two methods. Image 13 within the data set is attacked successfully for both Sparse SimBA and Adaptive Sparse SimBA. The adversarial example are plotted in Figure 5.12. We selected it as the target image for the Dense One-pixel attack method.

5.2.2 Dense One-pixel attack

As the time for accomplishing one prediction using EfficientNet-B7 is around 10 seconds, it is calculated that it takes about 9 minutes for each iteration if the population size is set as 50. The Dense One Pixel attack is not suitable for the model that requires much time to run a prediction in terms of time efficiency. The successful adversarial example is shown in Figure 5.13. The population success is set as 50, and the maximum iteration is set as 30.

To compare the performance of these three attack methods, we summarised the

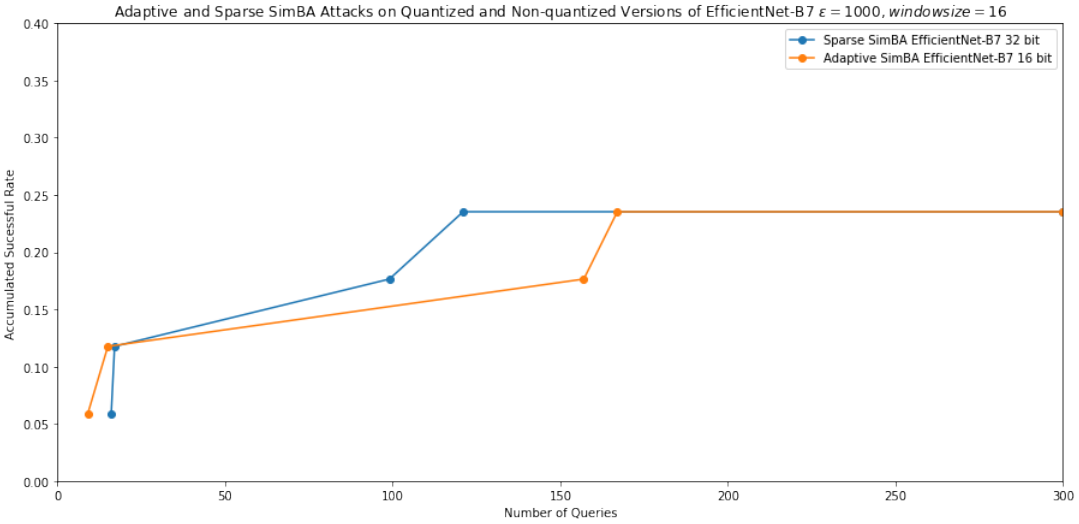

Figure 5.9: Attacking 32-bit and 16-bit EfficientNet-B7 using Sparse SimBA and Adaptive SimBA(window size = 8, $\epsilon = 800$, query limit=300)

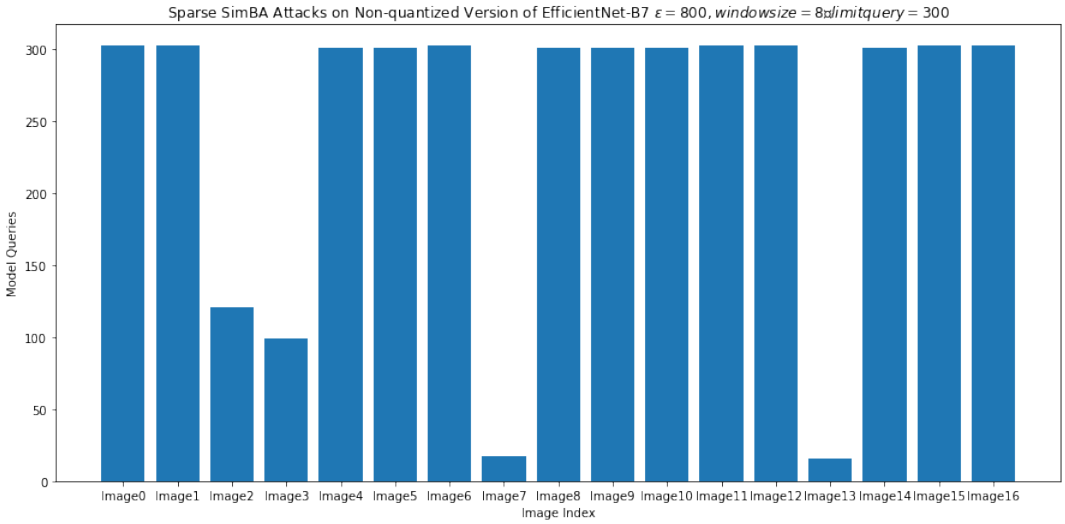

Figure 5.10: Model Queries for images attacked by Sparse SimBA

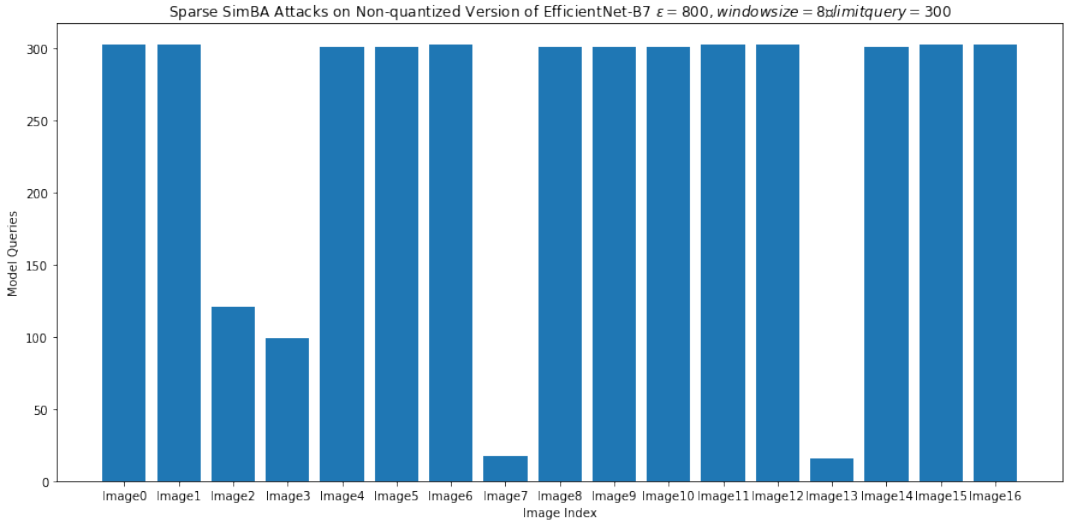

Figure 5.11: Model Queries for images attacked by Adaptive SimBA

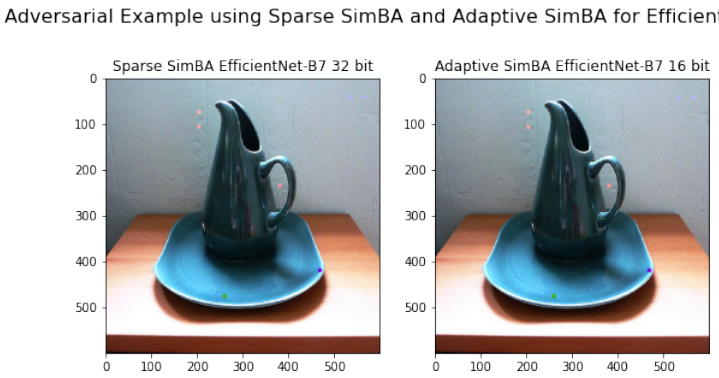

Figure 5.12: Successful Adversarial Examples Produced by Sparse SimBA and Adaptive SimBA (EfficientNet-B7)

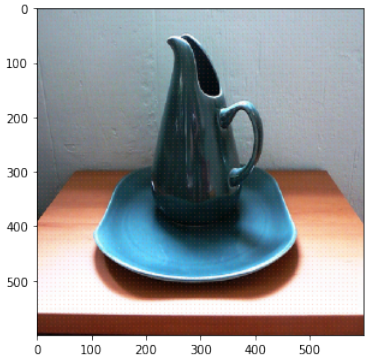

Figure 5.13: Successful Adversarial Examples Produced by Dense One-pixel Attack (EfficientNet-B7)

attack method	model queries	absolute value norm	PSNR	SSIM
Dense One-Pixel	50	16380	25.551704	0.551564
Sparse SimBA	16	448	38.856	0.998947
Adaptive SimBA	9	448	39.846	0.999

Table 5.5: Visual Quality measurement for adversarial examples using various attack method (EfficientNet-B7)

details of these attacks targeted on the same image in Table 5.5.

It is found that both of the SimBA algorithms perform better than the Dense One-pixel attack method in terms of model queries and visual quality. This may attribute to the large-scale image used for EfficientNet-B7. A small target window deems the number of perturbed pixels to increase. As SimBA algorithm randomly selects a window to add perturbations, there may be fewer pixels perturbed, especially for those vulnerable to attack. Also, in terms of human opinions, the noise added on the image through Dense One-pixel is more organized than the SimBA algorithms.

Chapter 6

Conclusion and Reflection

6.1 Conclusion

As DNN has become the primary interest in image classifications and other classification problems, researchers have made efforts to reduce the corresponding computational cost and memory requirements. After a few quantization approaches were proposed, the state-of-art quantized models have achieved similar accuracy, but they still suffer the vulnerability of adversarial attacks. Also, quantization techniques influence the practical implementation of deep learning algorithms into hardware systems. Studies into the adversarial not only help us understand how to attack and defend an image classification system and provide us a better understanding of the decision boundary, which is an important characteristic of neural networks. In our project, we investigated the robustness of state-of-art DNN-based image classifiers EfficientNet-B0 and EfficientNet-B7. We studied the performance of Sparse SimBA, Adaptive SimBA on quantized and full-precision EfficientNet-B0 and EfficientNet-B7. Also, we proposed a new attack method called Dense One-pixel attack to correspond to the large scale image in the ILSVRC-2012 data set.

The whole project was based on the previous work of Gasim Gasim, who has proposed the Adaptive SimBA algorithm, which achieved better performance when attacking 8-bit block floating-point quantized EfficientNet-B7. His research has not covered the relationship between the accumulated success rate and model queries, which gives a more

direct view of success rate and robustness. Also, we tested these two attack methods on full-precision and quantized version of EfficientNet-B0, a lightweight model with competitive prediction accuracy. Another innovative attack method–Dense One-pixel attack performs better when the target EfficientNet-B0 produces the 224 pixels image in terms of model queries and visual quality. However, due to the higher volume of pixels needed to be perturbed when the image size increases to 600×600, the Dense One-pixel performs poorly than Sparse SimBA and adaptive SimBA, though it has acceptable human-perceived visual quality. It is essential to mention that the Dense One-pixel attack performs similarly for quantized and full-precision models because it utilizes the differential evolution algorithm. DE starts at random candidate solutions and finds the optimum results totally based on the mutated algorithm. This global-search method heavily depends on computational resources. The large population size and high limit of iterations will assist the algorithm with looking for the perfect pixel and corresponding pixel value within the window. However, the whole experiment is running on Google Colab, which has restricted resources for users. In conclusion, we fully investigated the performance of Sparse and Adaptive SimBA attacks on EfficientNet-B0 and EfficientNet-B7 for both quantized and non-quantized versions and also proposed a new idea called Dense One-pixel attack and studied its performance on the mentioned models as well.

6.2 Reflection

There are a few problems we need to address. First, we haven't fully studied the parameters of three attack methods such as window size and step length in SimBA and window size and population size in the Dense One-pixel attack. To save time for running the experiment, we sacrifice the visual image quality to achieve a quicker attacking process. When conducting the Dense One-pixel attack, EfficientNet requires plenty of time to predict one image. Therefore, we have to limit the size of candidate solutions and select the most vulnerable image to attack. The result may involve inevitable randomness, especially for the Dense One-pixel attack. In the future, we need to investigate the settings of attack methods more thoroughly. Secondly, the Dense One-pixel algorithm can be improved by not splitting the

whole image into specific windows but randomly selecting a few windows. The selection of windows' positions can be optimized through the differential evolution algorithm. It is necessary to note the switching to a smaller image classifier can improve the thoroughness of our experiment. In terms of the SimBA algorithm, we can also test their performance on fewer bits quantized model.

Bibliography

[1] R. Storn and K. V. Price, "Differential evolution - a simple and efficient heuristic for global optimization over continuous spaces." *J. Glob. Optim.*, vol. 11, no. 4, pp. 341–359, 1997. [Online]. Available: http://dblp.uni-trier.de/db/journals/jgo/jgo11.html#StornP97

[2] M. Tan and Q. V. Le, "Efficientnet: Rethinking model scaling for convolutional neural networks," 2020.

[3] Y. Zhao, I. Shumailov, R. Mullins, and R. Anderson, "To compress or not to compress: Understanding the interactions between adversarial attacks and neural network compression," 2020.

[4] S. Han, J. Pool, J. Tran, and W. J. Dally, "Learning both weights and connections for efficient neural networks," 2015.

[5] R. Krishnamoorthi, "Quantizing deep convolutional networks for efficient inference: A whitepaper," 2018.

[6] A. Fawzi, S.-M. Moosavi-Dezfooli, and P. Frossard, "The robustness of deep networks: A geometrical perspective," *IEEE Signal Processing Magazine*, vol. 34, no. 6, pp. 50–62, 2017.

[7] C. Guo, J. R. Gardner, Y. You, A. G. Wilson, and K. Q. Weinberger, "Simple black-box adversarial attacks," 2019.

[8] J. Su, D. V. Vargas, and K. Sakurai, "One pixel attack for fooling deep neural networks," *IEEE Transactions on Evolutionary Computation*, vol. 23, no. 5, p.

828–841, Oct 2019. [Online]. Available: http://dx.doi.org/10.1109/TEVC.2019.2890858

[9] W. Rawat and Z. Wang, "Deep Convolutional Neural Networks for Image Classification: A Comprehensive Review," *Neural Computation*, vol. 29, no. 9, pp. 2352–2449, 09 2017. [Online]. Available: https://doi.org/10.1162/neco_a_00990

[10] S. S. Nath, G. Mishra, J. Kar, S. Chakraborty, and N. Dey, "A survey of image classification methods and techniques," in *2014 International Conference on Control, Instrumentation, Communication and Computational Technologies (ICCICCT)*, 2014, pp. 554–557.

[11] M. Ranzato, F. J. Huang, Y.-L. Boureau, and Y. LeCun, "Unsupervised learning of invariant feature hierarchies with applications to object recognition," in *2007 IEEE Conference on Computer Vision and Pattern Recognition*, 2007, pp. 1–8.

[12] M. D. Zeiler and R. Fergus, "Visualizing and understanding convolutional networks," 2013.

[13] A. S. Razavian, H. Azizpour, J. Sullivan, and S. Carlsson, "Cnn features off-the-shelf: an astounding baseline for recognition," 2014.

[14] M. Lin, Q. Chen, and S. Yan, "Network in network," 2013, cite arxiv:1312.4400Comment: 10 pages, 4 figures, for iclr2014. [Online]. Available: http://arxiv.org/abs/1312.4400

[15] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *Journal of Machine Learning Research*, vol. 15, no. 56, pp. 1929–1958, 2014. [Online]. Available: http://jmlr.org/papers/v15/srivastava14a.html

[16] Y. Tang, "Deep learning using linear support vector machines," 2015.

[17] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus, "Intriguing properties of neural networks," 2014.

[18] N. Akhtar and A. Mian, "Threat of adversarial attacks on deep learning in computer vision: A survey," 2018.

[19] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," 2015.

[20] S.-M. Moosavi-Dezfooli, A. Fawzi, and P. Frossard, "Deepfool: a simple and accurate method to fool deep neural networks," 2016.

[21] N. Papernot, P. McDaniel, S. Jha, M. Fredrikson, Z. B. Celik, and A. Swami, "The limitations of deep learning in adversarial settings," 2015.

[22] A. Nguyen, J. Yosinski, and J. Clune, "Deep neural networks are easily fooled: High confidence predictions for unrecognizable images," 2015.

[23] L. Engstrom, B. Tran, D. Tsipras, L. Schmidt, and A. Madry, "Exploring the landscape of spatial robustness," 2019.

[24] J. B. Li, S. Qu, X. Li, J. Szurley, J. Z. Kolter, and F. Metze, "Adversarial music: Real world audio adversary against wake-word detection system," 2019.

[25] M. Alzantot, B. Balaji, and M. Srivastava, "Did you hear that? adversarial examples against automatic speech recognition," 2018.

[26] N. Papernot, P. McDaniel, A. Swami, and R. Harang, "Crafting adversarial input sequences for recurrent neural networks," 2016.

[27] R. Huang, B. Xu, D. Schuurmans, and C. Szepesvari, "Learning with a strong adversary," 2016.

[28] B. Liang, H. Li, M. Su, X. Li, W. Shi, and X. Wang, "Detecting adversarial image examples in deep neural networks with adaptive noise reduction," *IEEE Transactions on Dependable and Secure Computing*, vol. 18, no. 1, p. 72–85, Jan 2021. [Online]. Available: http://dx.doi.org/10.1109/TDSC.2018.2874243

[29] W. Xu, D. Evans, and Y. Qi, "Feature squeezing: Detecting adversarial examples in deep neural networks," *Proceedings 2018 Network and Distributed System Security Symposium*, 2018. [Online]. Available: http://dx.doi.org/10.14722/ndss.2018.23198

[30] N. Carlini and D. Wagner, "Adversarial examples are not easily detected: Bypassing ten detection methods," 2017.

[31] ——, "Defensive distillation is not robust to adversarial examples," 2016.

[32] P.-Y. Chen, H. Zhang, Y. Sharma, J. Yi, and C.-J. Hsieh, "Zoo," *Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security*, Nov 2017. [Online]. Available: http://dx.doi.org/10.1145/3128572.3140448

[33] N. Carlini and D. Wagner, "Towards evaluating the robustness of neural networks," 2017.

[34] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," 2015.

[35] S. Zagoruyko and N. Komodakis, "Wide residual networks," 2017.

[36] Y. Huang, Y. Cheng, A. Bapna, O. Firat, M. X. Chen, D. Chen, H. Lee, J. Ngiam, Q. V. Le, Y. Wu, and Z. Chen, "Gpipe: Efficient training of giant neural networks using pipeline parallelism," 2019.

[37] I. Guyon, L. Sun-Hosoya, M. Boullé, H. J. Escalante, S. Escalera, Z. Liu, D. Jajetic, B. Ray, M. Saeed, M. Sebag, A. Statnikov, W. Tu, and E. Viegas, "Analysis of the automl challenge series 2015-2018," in *AutoML*, ser. Springer series on Challenges in Machine Learning, 2019. [Online]. Available: https://www.automl.org/wp-content/uploads/2018/09/chapter10-challenge.pdf

[38] A. Krizhevsky, V. Nair, and G. Hinton, "Cifar-10 (canadian institute for advanced research)." [Online]. Available: http://www.cs.toronto.edu/~kriz/cifar.html

[39] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing*

Systems, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, Eds., vol. 25. Curran Associates, Inc., 2012. [Online]. Available: https://proceedings.neurips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf

[40] G. Ahmed, "Effects of quantisation on the vulnerability of image classifiers to black-box adversarial attacks," 2020.

[41] R. Krishnamoorthi, "Quantizing deep convolutional networks for efficient inference: A whitepaper," *CoRR*, vol. abs/1806.08342, 2018. [Online]. Available: http://arxiv.org/abs/1806.08342

[42] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. G. Howard, H. Adam, and D. Kalenichenko, "Quantization and training of neural networks for efficient integer-arithmetic-only inference," *CoRR*, vol. abs/1712.05877, 2017. [Online]. Available: http://arxiv.org/abs/1712.05877

[43] B. E., "This is how you cite a website in latex," https://colab.research.google.com/notebooks/intro.ipynb?utm_source=scs-index, May 2019.

[44] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. J. Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. J. Carey, İ. Polat, Y. Feng, E. W. Moore, J. VanderPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa, P. van Mulbregt, and SciPy 1.0 Contributors, "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python," *Nature Methods*, vol. 17, pp. 261–272, 2020.

[45] A. Nelson, "Differential evolution implementation," https://github.com/scipy/scipy/blob/70e61dee181de23fdd8d893eaa9491100e2218d7/scipy/optimize/_differentialevolution.py, Dec. 2017.

[46] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "ImageNet Large Scale

Visual Recognition Challenge," *International Journal of Computer Vision (IJCV)*, vol. 115, no. 3, pp. 211–252, 2015.

[47] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman, "The pascal visual object classes (voc) challenge," *International Journal of Computer Vision*, vol. 88, no. 2, pp. 303–338, Jun. 2010.

[48] H. Su, J. Deng, and L. Fei-Fei, "Crowdsourcing annotations for visual object detection," in *Human Computation - Papers from the 2012 AAAI Workshop, Technical Report*, ser. AAAI Workshop - Technical Report, Dec. 2012, pp. 40–46, 2012 AAAI Workshop ; Conference date: 23-07-2012 Through 23-07-2012.

[49] Z. Wang and A. C. Bovik, "Mean squared error: Love it or leave it? a new look at signal fidelity measures," *IEEE Signal Processing Magazine*, vol. 26, no. 1, pp. 98–117, 2009.

[50] H. Sheikh, M. Sabir, and A. Bovik, "A statistical evaluation of recent full reference image quality assessment algorithms," *IEEE Transactions on Image Processing*, vol. 15, no. 11, pp. 3440–3451, 2006.

[51] Z. Wang and A. Bovik, "A universal image quality index," *IEEE Signal Processing Letters*, vol. 9, no. 3, pp. 81–84, 2002.

[52] Z. Liu and R. Laganiere, "Phase congruence measurement for image similarity assessment," *Pattern Recognition Letters*, vol. 28, pp. 166–172, 01 2007.

[53] A. Robledo, "Black-box adversarial attacks for commercial api image classifiers," 2029.

Appendix A

Appendix Title

The whole subset of ILSVRC2012-CLS data set is shown as the figures below. For EfficientNet-B0, 13 of 20 images are correctly predicted and 17 of 20 images are correctly predicted by EfficientNet-B7. Image 7, 11, 18, 20 are not correctlt predicted by EfficientNet-B7 and Image 3, 7, 9, 10, 11 ,16, 18 are not correctly predicted by EfficientNet-B0.

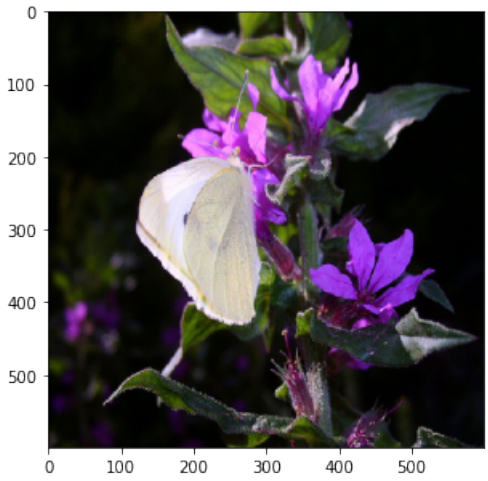

Figure A.1: Image 1

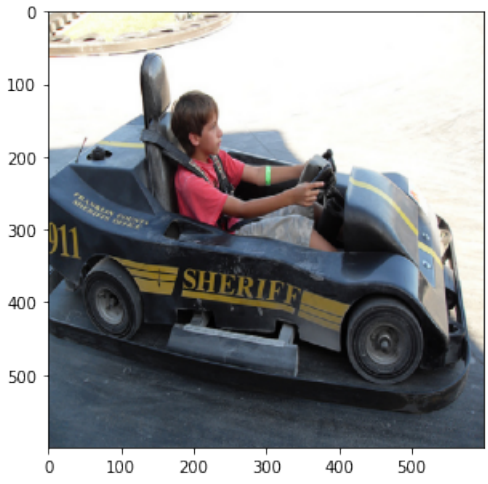

Figure A.2: Image 2

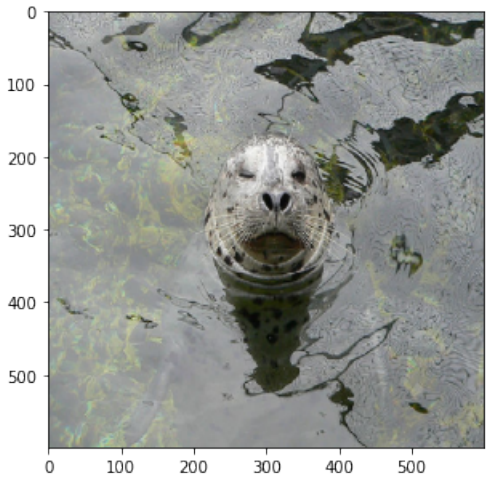

Figure A.3: Image 3

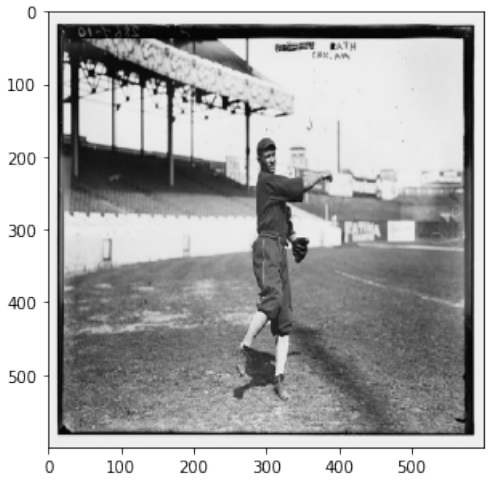

Figure A.4: Image 4

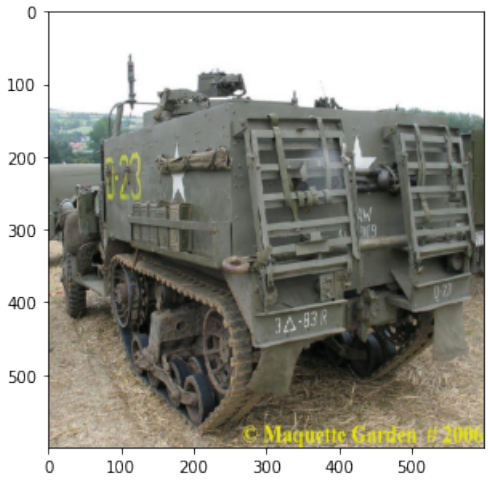

Figure A.5: Image 5

Figure A.6: Image 6

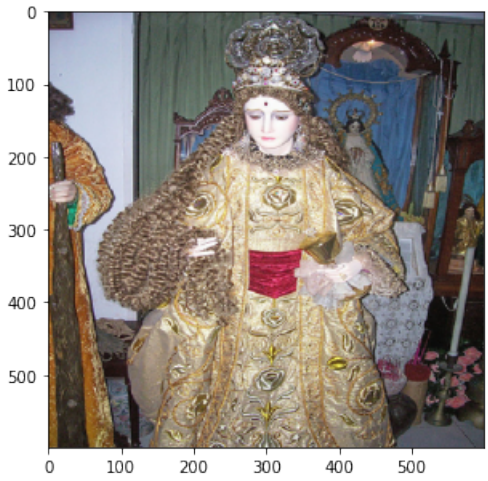

Figure A.7: Image 7

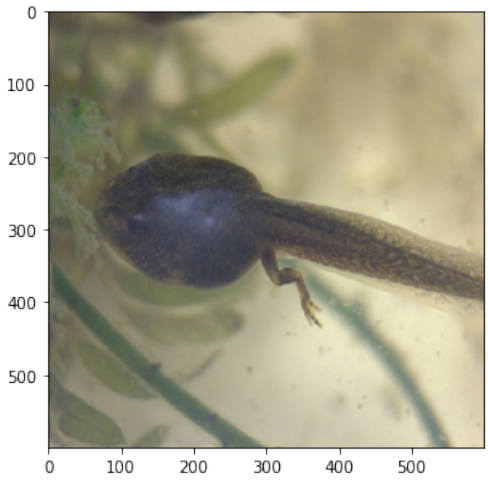

Figure A.8: Image 8

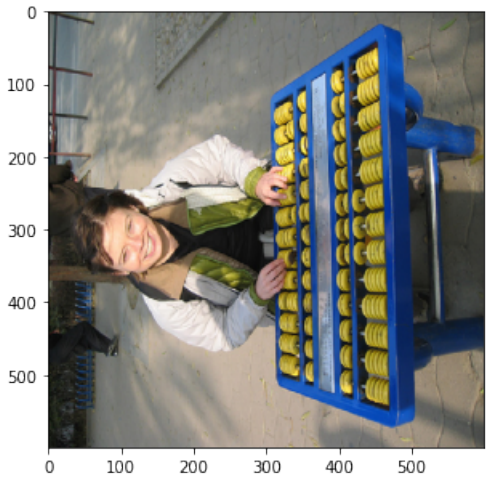

Figure A.9: Image 9

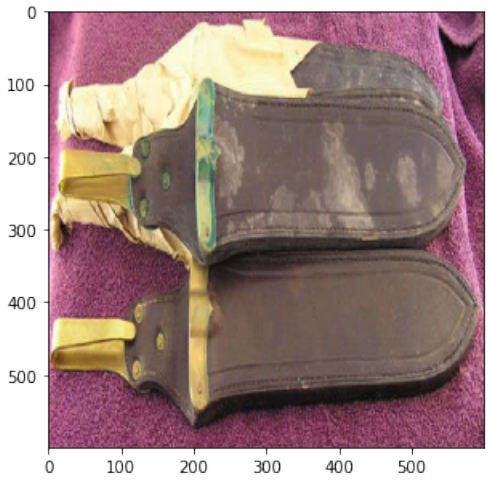

Figure A.10: Image 10

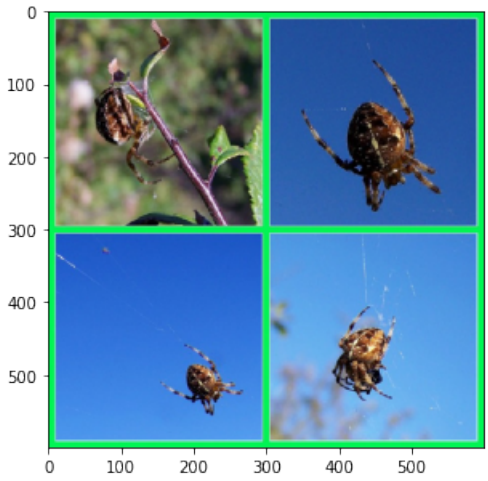

Figure A.11: Image 11

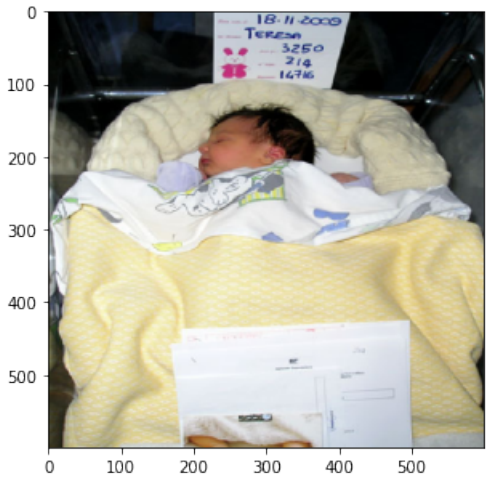

Figure A.12: Image 12

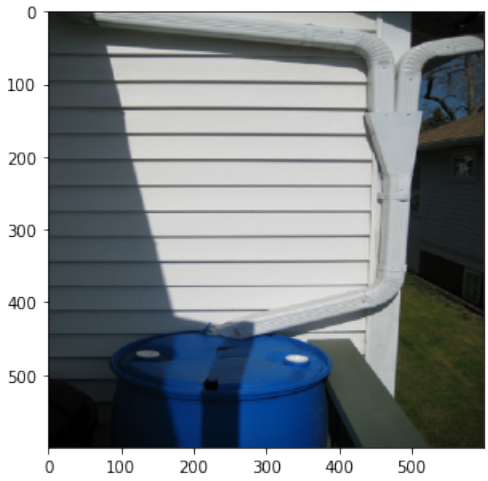

Figure A.13: Image 13

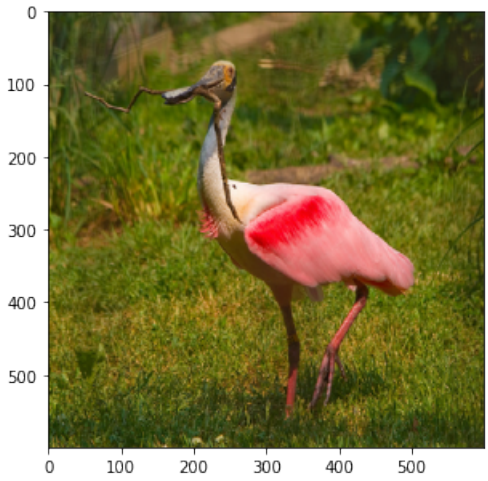

Figure A.14: Image 14

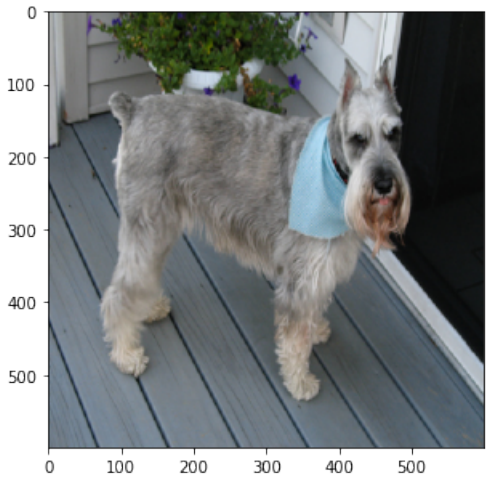

Figure A.15: Image 15

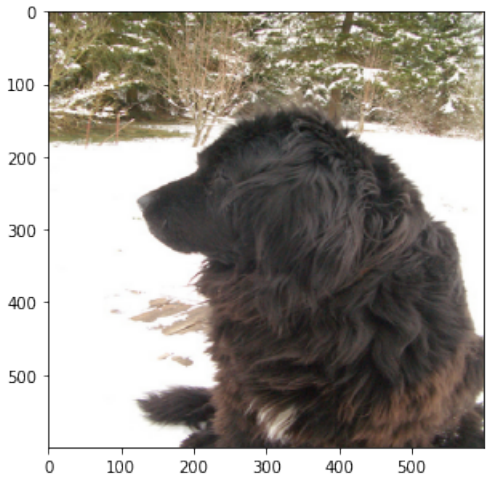

Figure A.16: Image 16

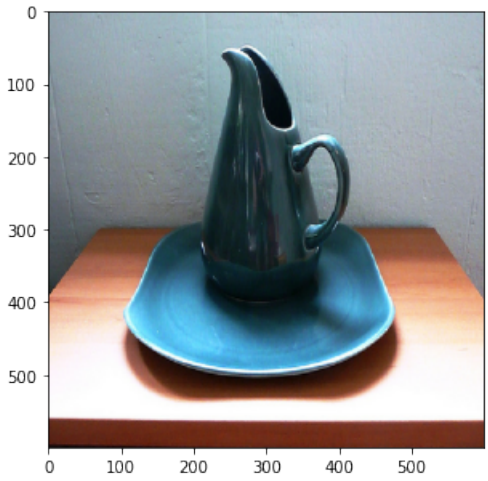

Figure A.17: Image 17

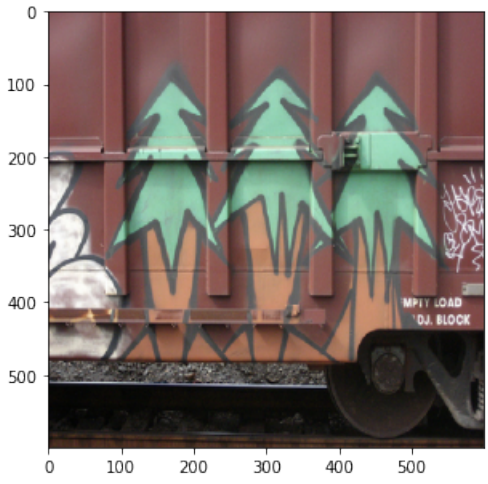

Figure A.18: Image 18

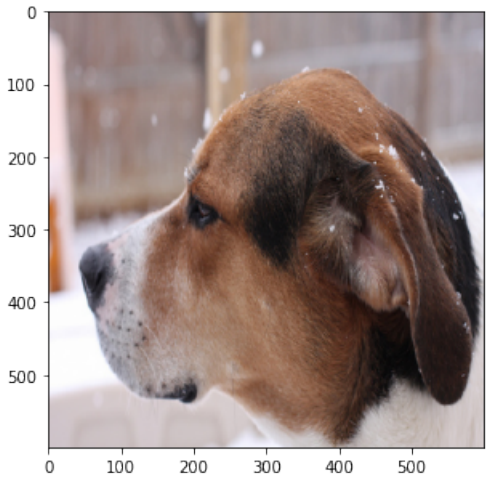

Figure A.19: Image 19

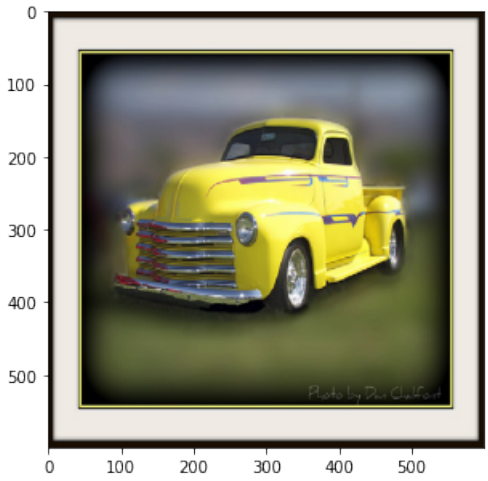

Figure A.20: Image 20