Robustness of Compressed Convolutional Neural Networks

Arie Wahyu Wijayanto *
Dept. of Computer Science
Tokyo Institute of Technology
Tokyo, Japan
ariewahyu@net.c.titech.ac.jp

Choong Jun Jin *
Dept. of Computer Science
Tokyo Institute of Technology
Tokyo, Japan
junjin.choong@net.c.titech.ac.jp

Kaushalya Madhawa *
Dept. of Computer Science
Tokyo Institute of Technology
Tokyo, Japan
kaushalya@net.c.titech.ac.jp

Tsuyoshi Murata
Dept. of Computer Science
Tokyo Institute of Technology
Tokyo, Japan
murata@c.titech.ac.jp

***Abstract*—Advancements in deep neural networks have revolutionized the way how we conduct our day-to-day activities ranging from how we unlock our phones to self-driving cars. Convolutional Neural Networks (CNN) play the principal role in learning high level feature representations from visual inputs. It is crucial to know how reliable those neural networks are as human lives can be at stake. Recent experiments on the robustness of CNNs show that they are highly susceptible to small adversarial perturbations. Due to the increasing popularity of mobile devices, there is a significant demand for CNN models which are smaller enough to run on a mobile device without sacrificing the accuracy. Although recent researches have been successful at achieving smaller models with comparable accuracy on standard image datasets, their robustness to adversarial attacks has not been studied. However, massive deployment of smaller models on millions of mobile devices stresses importance of their robustness. In this work, we study how robust such models are with respect to state-of-the-art compression techniques such as quantization. Our contributions are summarized as follows: (1) insights to achieve smaller and robust models (2) a compression framework which is adversarial-aware. Our findings reveal that compressed models are naturally more robust than compact models. This provides an incentive to perform compression rather than designing compact models. Additionally, the latter provides benefits of increased accuracy and higher compression rate, up to 90×.**

***Index Terms*—deep learning, compression, robustness**

I. Introduction

Superior performance of Deep Learning has been a major driving force in solving difficult and mundane tasks in our every day life. From language translation to domination in the game of Go, one can only question the potential of Deep Neural Network (DNN) models [1]. Of many contributions, the primary success of Deep Learning could be attributed to the complexity of Convolutional Neural Network (CNN) architectures itself [2]. In exchange of complexity for higher performance, CNN models gained an inevitable increase in file size and slower inferencing speed [3]. Although both complexity and size are negligible for larger devices such as personal computers, a remaining problem exists for smaller devices like smartphones and embedded devices. Smaller devices typically strive for efficient computation power. However, in reality, CNN models can be very large. For instance, AlexNet Caffemodel [4] and VGG-16 Caffemodel [5] are both over 200MB and 500MB respectively. It would be daunting if such large models are loaded into memory on smaller devices with low footprint. Thus, more attention has been paid towards making CNN models smaller [6–9]. Current research is being done along different approaches but with the common goal of achieving smaller models without compromising the accuracy. These approaches comprise of compressing a pre-trained larger CNNs, retraining a smaller architecture from the ground up (distillation) [10] and low-rank factorization [11, 12] of the larger size CNN. However, a crucial question on robustness of small networks has yet to been addressed. We raise the question: "How far can a convolutional neural network be compressed without compromising accuracy and robustness?".

In our experiments we report the robustness of a CNN model as the accuracy drop adversarial examples could cause. An adversarial example is a sample of input data which has been tampered with such that a DNN classifier classifies it incorrectly. Our initial experiment shows that compressed and compact CNNs are equally vulnerable to adversarial examples. In an attempt to study further this issue, we explore the correlation between adversarial examples with respect to compression. To the best of our knowledge, this is the first attempt to investigate large scale adversarial attack on compressed and compact models. Hence, we highlight the contributions of this paper as follows:

- We present an insight on robustness of compressed and compact CNN models. We emphasize that compressed models are more robust than compact CNNs.
- We show that adversarial training via quantization can improve accuracy of compressed networks whilst also being smaller. We achieved better compression rate (90× on AlexNet) than the state-of-the-art method and being the most robust against white-box adversarial attacks compared to other AlexNet-based compression models.

The structure of this paper is divided into several sections. Section II introduces ways to generate adversarial examples and the methods that we employ in evaluating robustness. Section III introduces related works to recent compression and compacting methods. Consequently, we highlight the strengths and weaknesses of both methods. In Section IV we introduce our strategy to produce adversarial-aware compressed models.

* These authors contribute equally

978-1-5386-5035-6/18/$31.00 ©2018 IEEE

The remaining sections discuss about our experimental setup and findings.

II. Generating Adversarial Examples

A recent finding by [13] describes an intriguing property of neural networks; they are vulnerable to adversarial examples. By slightly tampering input images, one can easily trick a neural network to misclassify data with high confidence [14]. On the contrary, a bolder approach to adversarial examples includes the introduction of a sticker (adversarial patch) in the training image [15]. By including this sticker on various images, Brown *et al.* is able to show that a CNN highly confident classifier performs otherwise. It has been previously hypothesized that such tricks (attacks) are possible due to the nonlinearity property of CNNs. Later research [16, 17] disproved this claim by showing that adversarial examples can be found in larger continuous regions rather than in small pockets due to nonlinearity. The generalization of adversarial examples across different models can be explained as a result of adversarial perturbations being highly aligned with the weight vectors of a model, and different models learning similar functions when trained to perform the same task.

There are two types of attacks depending on how the adversarial examples are generated. *White-box attacks* are crafted for a particular Machine Learning (ML) model using knowledge of its parameters. Such attacks on an ML model can be alleviated by restricting the access to the particular model. The second type of attacks, *black-box attacks* exploit *transferability*, an inherent aspect of ML models. Transferability is the act of using adversarial examples generated by one model to successfully attack a different model. An attacker relying on black-box attacks does not require the knowledge of internal parameters of the model, making such attacks more plausible and harder to defend.

Left unattended, adversarial examples can lead to catastrophic events especially in fields such as autonomous vehicles (self-driving cars). Specifically, cars can be crashed, illicit or illegal content can bypass content filters, resulting in unpredictable behaviors. To combat such security concerns, countermeasures related to knowledge transfer has been proposed [18]. Alternatively, an interesting approach is to introduce an "adversarial detector". Specifically, Xu *et al.* introduced an additional filter known as feature squeezing [19]. This filter acts as a detector which gets trained along the original network. For a comprehensive summary of adversarial defenses, we refer readers to [20] where Yuan *et al.* summarizes a list of adversarial example countermeasures. To this end, several kinds of defenses has been proposed, but none of them concretely addresses the compressed network capacity. In the following sections, we show that smaller model exhibits similar behavior, but at the same time, their resiliency can be improved in a very simple fashion.

In section VI , we report how a diverse set of DNNs perform against both white-box and black-box adversarial attacks. Furthermore, it should be noted that none of these attacks guarantees that generated examples will be misclassified. The number of misclassified adversarial examples is used as a measure of robustness.

Fast gradient sign method (FGSM) [14] is one of the first algorithms used to generate adversarial images. FGSM is an efficient "one-shot" algorithm for generating adversarial examples with a fixed l_∞ norm. Let x be an input image and x^{adv} be the generated adversarial example. We denote $l(\cdot,\cdot)$ be the differentiable loss function that was used to train the classifier $h(\cdot)$, *e.g.*, the cross-entropy loss. The FGSM adversarial example corresponding to a score input x is:

$$x^{adv} = x + \epsilon \cdot \text{sign}(\nabla_x l(x, h(x))), \tag{1}$$

where ϵ is a hyper-parameter to be chosen for some $\epsilon > 0$ that governs the perturbation magnitude.

A straightforward extension of FGSM attack is the **basic iterative method (BIM)** [17]. BIM is the application of FGSM multiple times with small step size and clip pixel values of intermediate results after each step. This will guarantee that they are in an ϵ-neighborhood of the original input. Let $\text{Clip}_{x,\epsilon}\{x'\}$ denote the function which conducts per-pixel clipping of x' image ensuring the result in L_∞ ϵ-neighborhood of the source image x. The BIM adversarial example of input image x is:

$$x_0^{adv} = x,\ x_{n+1}^{adv} = \text{Clip}_{x,\epsilon}\{x_n^{adv} + \alpha \text{sign}(\nabla_x l(x_n^{adv}, h(x)))\}. \tag{2}$$

III. Compressing Convolutional Neural Networks

Deep neural networks are often over parametrized. Therefore, a fraction of weights in a network can be spared or removed without impacting the original accuracy. The need for compression of networks is driven by the restrictions on the target hardware platform. Reducing energy consumption and reducing model size are common goals in finding smaller sized neural networks. We loosely refer all such techniques as network compression. The research on network compression can be grouped into following categories:

- *Compressing pre-trained networks.* This includes reducing the precision of weights, pruning, non-linear quantization and weight sharing.
- *Training compact models from the ground up.*

The former compression methodologies are network agnostic whereas the latter requires re-engineering of the network structure. In this paper, we focus our attention to the former.

A. Compressing Pre-Trained Networks

Since neural networks often contain redundant weights, a large set of weights can be pruned away (i.e., set to zero). Neural network pruning, termed optimal brain damage was first proposed in 1989 by LeCun *et al.* [21] as a way to reduce model complexity and overfitting. In most cases, proposed methods on network pruning has resorted to trade accuracy for size and performance. However, a recent work by Han *et al.* [22] alleviated the problem by coupling pruning with fine-tuning. Coupling network pruning with quantization and

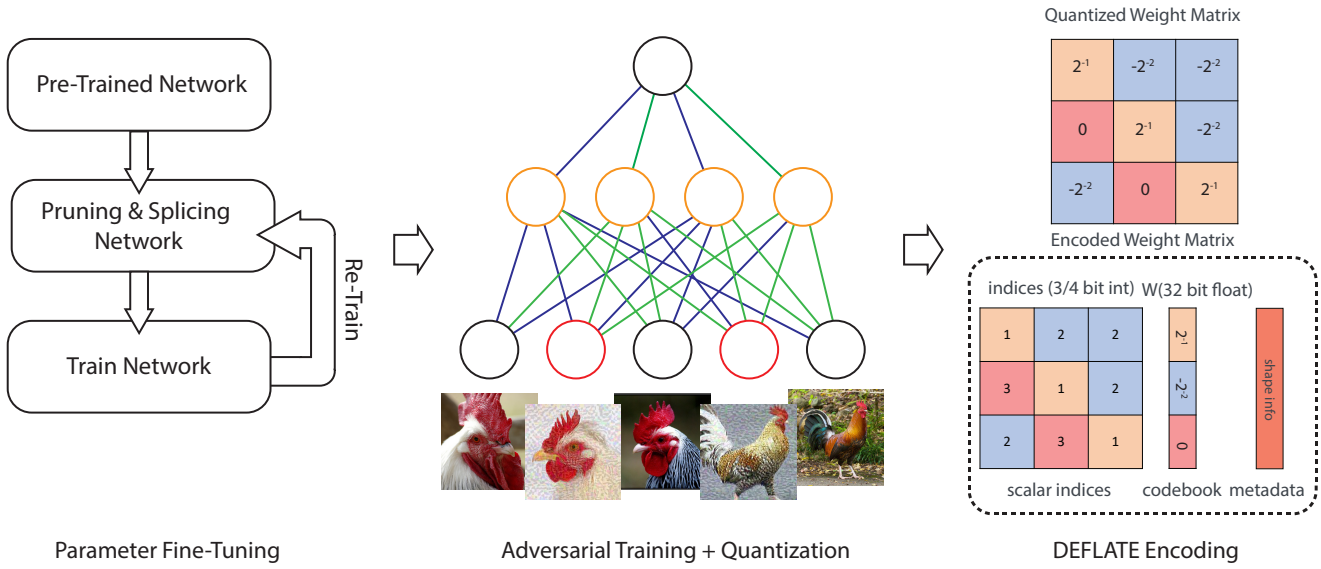


Fig. 1: Pipeline for compressing CNN to increase robustness and compression rate. Firstly, a pre-trained model for fine-tuning and prune unnecessary weights. Secondly, a mixture of adversarial examples with original training datasets are introduced during quantization phase. Finally, codebook is produced based on finalized weights in power of twos or zeroes. For visualization purposes, we use a fully connected layer. Bottom red nodes indicate adversarial examples. Green and blue edges denote INQ quantization phases, where a section of low-weights are first quantized (green) followed by remaining weights.

encoding in a three stage compression pipeline, Han *et al.* [6] proposed Deep Compression which achieved a higher compression ratio. Notably, AlexNet was compressed up to 35× and VGG-16 up to 49× without loss of accuracy. This approach, however, is slow and cumbersome. Iterative pruning itself requires significant amount of training time to ensure no accuracy loss. Moreover, Deep Compression requires specialized hardware [3] for inferencing. A smarter pruning approach called Dynamic Network Surgery (DNS) was proposed by Guo *et al.* [23]. Independently, Ullrich *et al.* [7] unifies the compression pipeline of pruning and quantization by posing it as a mixture model problem. Although the compression ratio is significantly higher, this approach is not scalable for larger models like VGG-16. More recently, Zhou *et al.* [24] suggest quantizing weights in powers of twos or zeroes. Coined as Incremental Network Quantization (INQ), the proposed method achieved an impressive compression ratio up to 89× on AlexNet when coupled with DNS and 3-bit compression. Because the quantized weights are in powers of two or zeroes, it has the potential to speed up inferencing speed without relying on specialized hardware. This is possible with bit-shifting operations widely available on CPU and GPU.

One major disadvantage of the compression techniques mentioned above is that they require special hardware or improvements to existing DNN frameworks to achieve the full potential of smaller model size. Reducing the precision of weights and operations of pre-trained networks [25] is another technique used to achieve speed and reduced model size. Even though low precision network models can be run on commodity hardware easily, the size of such models are often larger than the ones obtained through pruning and quantization.

B. Designing Compact Neural Network Models

Besides compression, another approach to obtain smaller network models with comparable performance to the larger models is designing of compact network architectures. SqueezeNet [26] and MobileNets [27] are such compact architectures designed to be run on hardware platforms with resource restrictions. Even though such models are shown to have accuracy levels only slightly worse than the state-of-art, they are significantly smaller than the latter. In our case study, we utilize both SqueezeNet and MobileNets due to their widespread popularity.

IV. Adversarial-aware Compression Framework

Contrary to previous approaches which accomplish model compression without a robustness objective, we incorporate training with adversarial examples benefiting from three inter-dependent techniques: Dynamic Network Surgery (DNS) on-the-fly connection pruning, Incremental Network Quantization (INQ) low bit-width layer precision weights and DEFLATE encoding. The outcome of this approach is a small yet robust network. It has been presumed that adversarial training can behave like a regularizer [28]. Thus, this approach makes the network much more robust and perform better.

A. Network Pruning with Dynamic Network Surgery

We use Dynamic network surgery (DNS) [23] as the pruning algorithm. DNS employs two operations, pruning and splicing. *Pruning* reduces the number of parameters of the model by removing unimportant parameters as similar to [6]. *Splicing* operation overcomes the problem of incurring accuracy loss due to over-pruning or incorrect pruning by restoring some of the previously pruned parameters based on their current value of importance. Parameter importance is decided by the combined application of pruning and splicing. Furthermore, DNS achieved 7× speed-up (approximately 140 epochs) during retraining of AlexNet model and it achieved an improvement in pruning from 9× to 17.7×.

Specifically, given l^{th} layer of the network, the goal of DNS is optimizing the following loss function:

$$\begin{aligned} &\min_{W_l, C_l} L(W_l \odot C_l) \\ &\text{s.t. } C_l^{(a,b)} = h_l(W_l^{(a,b)}), \forall (a,b) \in I \end{aligned} \tag{3}$$

being $L(\cdot)$ the network loss function, $\odot$ denotes the Hadamard product operator, set I made up of all elements in weight matrix W_l, and h_l is a discriminative function satisfying $h_l(w) = 1$ if parameter w is considered important in the l^{th} layer, and 0 otherwise. We compose function $h_l(\cdot)$ based on some prior knowledge aiming to constrict the possible area of $W_l \odot C_l$ and reduce the initial NP-hard problem.

Furthermore, we only need to consider the update scheme of W_l since the binary matrix C_l can be defined under the constraint of Formula 3. With regards to the updating of Lagrange Multipliers and gradient descent, W_l can be updated under the following scheme:

$$\begin{aligned} W_l^{(a,b)} \leftarrow W_l^{(a,b)} - \alpha \frac{\partial}{\partial W_l^{(a,b)} C_l^{(a,b)}} L(W_l \odot C_l), \\ \forall (a,b) \in I \end{aligned} \tag{4}$$

where α denotes a positive learning rate. With respect to that, the entries of C_l, which are assumed to be insignificant and ineffective are given a second chance. This approach is useful to enhance the adaptability of our method because it enables the splicing of inappropriately pruned connections. Consequently, it helps the network to escape local optima which further helps to speed up pruning; which is known to be very time consuming process. To compute the partial derivatives in Formula 4, we use the chain rule with a randomly picked minibatch of samples. When matrix W_l and C_l are updated, they will be applied to re-compute the entire network activations and loss function gradient. By iteratively repeating these steps, the sparse model will have the ability to deliver better accuracy.

Pruning can be performed at whichever point the current connections are considered as unimportant. However, erroneously pruned parameters should be restored if it significantly affect the network's accuracy.

B. Adversarial Training with Incremental Network Quantization

The adversarial training phase corresponds to minimizing an upper bound on the expected cost over noisy examples by adding noise to the inputs.

We employ a well proven network quantization of INQ [24] to efficiently convert pre-trained full-precision CNN model under a low-precision version constrained weights in either powers of two or zero.

$$\begin{aligned} &\min_{W_l} E(W_l) = L(W_l) + \psi R(W_l) \\ &\text{s.t. } W_l(a,b) \in P_l, \text{ when } T_l(a,b) = 0, \ 1 \le l \le L, \end{aligned} \tag{5}$$

being $E(W_l)$, $L(W_l)$ and $R(W_l)$ the expected weights of layer l, network loss and the regularization terms respectively. Regularization term is learned using ψ positive coefficient. With this minimization, subjected to $W_l(a,b)$, the weight element should be derived from the constraint set P_l which consists of fixed values of either powers of twos or zero. $T_l(a,b) = 0$ denotes the weight element, $W_l(a,b)$, that will be quantized in the next step of iteration.

In each re-training iteration, we update the weight set under the following scheme:

$$W_l(a,b) \leftarrow W_l(a,b) - \beta \frac{\partial E}{\partial (W_l(a,b))} T_l(a,b), \tag{6}$$

where β is a positive learning rate.

Algorithm 1 Adversarial Training with DNS + INQ + DEFLATE

Input: training data with adversarial examples $\mathbf{X}$, pre-trained full-precision CNN model $\{\mathbf{W}_l : 1 \le l \le L\}$

Output: quantized weights $\{\widehat{\mathbf{W}}_l : 1 \le l \le L\}$ in values to be either powers of two or zero, codebook $\mathbf{h}_l = \{0, ..., 2^b\}$ mapping to $\widehat{\mathbf{W}}$ in encoded form.

$\widehat{\mathbf{W}} \leftarrow$ DNS($\mathbf{W}_l$)

Initialize $\mathbf{A}_l^{(1)} \leftarrow \emptyset$, $\mathbf{A}_l^{(2)} \leftarrow \{\widehat{\mathbf{W}}_l(i,j)\}$, $\mathbf{T}_l \leftarrow 1$, for $1 \le l \le L$

for $n = 1, ..., N$ **do**

  Reset base learning rate and the learning policy

  Based on σ_n, perform layer-wise weight partition and update $\mathbf{A}_l^{(1)}, \mathbf{A}_l^{(2)}$ and $\mathbf{T}_l$

  Based on $\mathbf{A}_l^{(1)}$, determine $\mathbf{P}_l^{(1)}$

  Quantize weights in $\mathbf{A}_l^{(1)}$ based on [24]

  Retrain and update weights $\mathbf{A}_l^{(2)} : 1 \le l \le L$

  Update $\widehat{\mathbf{W}}_l$ in-place w.r.t $\mathbf{A}_l^{(1)}$, $\mathbf{A}_l^{(2)}$

end for

for $n = 1, ..., L$ **do**

  Generate codebook index for quantized $\widehat{\mathbf{W}}_l$

end for

return DEFLATE($\widehat{\mathbf{W}}, \mathbf{h}$)

We present the adversarial training with DNS, INQ and DEFLATE in Algorithm 1. Several parameters in the algorithm

has to be adjusted to obtain satisfactory results. Notably, we adjusted the amount of adversarial examples contained in $\mathbf{X}_{batch}$ base on λ. In our experiment, we use λ as suggested in [28]. For instance, with a batch size of 256 and $\lambda = 0.5$, 128 adversarial examples and 128 clean examples are used.

C. DEFLATE Encoding

Huffman encoding was adopted by Han *et al.* [6] as an attempt to further compress the quantized network, pushing compression rate from 27× to 35× for AlexNet and 31× to 49× for VGG-16. The drawback of this approach is the need to decompress the model before being potentially useful to any software. In [6]'s case, a specialized hardware was proposed to perform the inferencing [3]. Although the inference speed is remarkable, it is not suited for standard GPU or CPU processors. In our method, we adopt the DEFLATE (LZ77 + Huffman) algorithm for compression instead of standard Huffman encoding. The DEFLATE algorithm is a widely supported algorithm for lossless compression. In practice, this algorithm has been integrated into standard consumer hardware such as System on Chips (SoC) for fast compression and decompression.

In essence, the DEFLATE algorithm comprises of two phases: duplicate string elimination and bit reduction. Duplicated series of bytes spotted are back-referenced, linking the current series of bytes to the previous location. This is done using a sliding window. As for bit reduction, Huffman coding is used. Essentially, a series of bytes that appear more often are represented with the shorter sequence of bits. In our case, the higher the frequency weights are represented with a shorter bit length. This representation can be constructed using a Huffman tree which predetermines the sequence of codes when reconstructed.

V. Experimental Setup

In this section, we describe the models and datasets that we use in our experiments. We use ImageNet Scale Visual Recognition Challenge 2012 (ILSVRC 2012) [30] as the dataset for training and validating all our models. Presently this is one of the largest publicly available dataset with over 1,000 object classes, 1.2 million training images and 50 thousand validation images. The classes are annotated and verified via Amazon Mechanical Turk workers. Using center crops of validation and training images as suggested in [4, 5], we train the models from the ground up if a pre-trained model is not publicly available. The CNNs used in this experiment are:

- Compressed models: Two pre-trained AlexNet models, one compressed using DeepCompression and the other compressed using Iterative Quantization Method (INQ)
- Compact models: Squeezenet and MobileNets-V1
- Low precision model: Inception-V3, both weights and computations converted to 8-bit integer values

All experiments are conducted using Caffe [31] and Tensorflow. Caffe models are obtained from "Caffe model-zoo"[1]. All adversarial images are generated using TensorFlow framework using the adversarial attack library 'Cleverhans' [32]. For models which do not have publicly available pre-trained Tensorflow models (eg: AlexNet), the model and weights are converted from Caffe to Tensorflow using a free and open-source tool [2]. Prior to use in experiments, we verified the converted Tensorflow models with ImageNet validation dataset. We report the results in two standard measures, namely top-1 and top-5 accuracy. We perform our experiments on a Ubuntu 16.06 LTS PC with an Intel(R) Core(TM) i9-7900X CPU @ 3.30GHz CPU and NVIDIA GTX 1080 Ti SLI GPU.

VI. Results and Discussion

In this section we demonstrate how different network models performed on adversarial images. All models are pre-trained with ImageNet dataset. The accuracy is reported with respect to the ImageNet validation dataset consisting of 50,000 images. We set the ϵ value of FGSM method to 2. An interesting issue in constructing the adversarial examples is known as *label leaking effect* [28]. If the true labels of images are used in the process of adversarial image generation with a single-step attack method such as FGSM, an adversarially trained model can learn to exploit regularities in the adversarial example generation process. This provides incentives to construct adversarial examples without the use of ground truth label. To alleviate this issue, we use the class corresponding to the maximum prediction probability instead.

Table I lists the models we used in our experiments. Int8 network is obtained by converting both weight matrices and computations of a pre-trained Inception-v3 model to 8 bit integers using Tensorflow[3] framework.

A. White-box attacks

In white-box attacks, the model which is being attacked is used to generate the adversarial examples. For instance, we generate adversarial images with an Inception-v3 model for ImageNet validation dataset and we report the accuracy of the same model for the generated images. Table II illustrates how each model fared against white-box attacks.

We observed that among all AlexNet-based model, our proposed model is more robust against white-box adversarial attacks. It achieves better accuracy on clean images as well as adversarial examples generated using FGSM and BIM methods compared to the original and other compressed models of AlexNet.

B. Black-box attacks

We generated the adversarial examples using one model and test all other models using the adversarial images generated. Similarly, we used BIM method as well. But we observed that the effectiveness of black-box attacks with BIM method is low as reported by [28]. Hence, we limited our study on transferability only to FGSM method. Because both weights and matrix multiplications of Inception-V3 model are converted to

[1]https://github.com/BVLC/caffe/wiki/Model-Zoo

[2]https://github.com/ethereon/caffe-tensorflow

[3]https://www.tensorflow.org/performance/quantization

TABLE I: Models used in experiments

Model	Type	Size (MB)	Parameters (in Million)
Inception-v3 [29]	Original architecture	108.8	25
AlexNet [4]	Original architecture	240	61
MobileNets-v1 [27]	Compact architecture	68	4.2
SqueezeNet [26]	Compact architecture	4.8	1.2
DeepCompression [6]	Compression (AlexNet)	6.9	6.7
INQ [24]	Compression (AlexNet)	2.69	6.7
Int8	Low precision (Inception-v3)	24.7	25

TABLE II: Accuracy on white-box attacks using FGSM and BIM

Model		Clean images	FGSM	BIM
Inception-v3	Top-1	0.7717	0.4279	0.3225
	Top-5	0.9351	0.7220	0.6917
MobileNets-v1	Top-1	0.7048	0.1604	0.0016
	Top-5	0.8941	0.4084	0.0057
SqueezeNet	Top-1	0.5750	0.2184	0.2984
	Top-5	0.8030	0.3928	0.5224
AlexNet-based Model				
Original	Top-1	0.5724	0.2912	0.2492
	Top-5	0.8023	0.5076	0.4540
DeepComp	Top-1	0.5624	0.2684	0.2580
	Top-5	0.7968	0.5040	0.4708
INQ	Top-1	0.5739	0.2460	0.2880
	Top-5	0.8046	0.4496	0.5104
Proposed	Top-1	**0.5818**	**0.3364**	**0.3620**
	Top-5	**0.8135**	**0.5648**	**0.6008**

8-bit integers, it is not possible to compute gradients of the int8 model. Thus, we use this model only to test its robustness against black-box attacks.

As shown in Table III, we observed high transferability of adversarial images for FGSM method. One noticeable exception is that the sparse model obtained using incremental quantization (INQ) exhibiting higher accuracy than the original AlexNet model on both adversarial attacks. Each diagonal cell in Table III represents the accuracy of a model on adversarial images generated with itself, hence the white-box attacks. Rest of the cells show the accuracy of black-box attacks. Our results exhibit that all the networks we used are vulnerable to both white-box and black-box attacks.

C. Robustness of Smaller Models

We quantize vulnerability of a model by calculating following quantity.

$$Vulnerability = \frac{\text{(accuracy on ImageNet)} - \text{(accuracy on adversarial examples)}}{\text{(accuracy on ImageNet)}} \tag{7}$$

Figures 2a and 2b show the vulnerability of each model as a heat-map. Each diagonal cell represents the accuracy of a model on adversarial images generated with itself, hence the white-box attacks. Rest of the cells show the accuracy on images generated by black-box attacks.

Two networks which exhibit the maximum deviation of accuracy for white-box attacks are MobileNets and SqueezeNet. According to the results, MobileNets model is not able to correctly label 84% of the adversarial images created by itself. Even though MobileNets model attains a comparable accuracy level for clean images, it performs much worse when it is confronted with adversarial images. We believe that the low model capacity of MobileNets compared to Inception model as the factor contributing to higher vulnerability to adversarial attacks. Interestingly, the low-precision Int8 model performed almost similar to Inception-V3 model. Even though it is not possible to do a direct white-box attack for Int8 model, the adversarial images generated using Inception-V3 can reduce the accuracy significantly.

An important insight we can gain from this experiment is that the compression of pre-trained networks results in smaller networks with a relatively low impact on robustness compared to compact networks trained from the ground up. Investigating why compact models show higher vulnerability to white-box attacks is a research direction which may shed light on designing more robust compact networks.

D. Compression Rate

We use PAQ8 by [33] as a baseline. PAQ8 is agnostic to file structures and employs ensembles learning approaches for lossless compression. When compared to Deep Compression, the baseline compression reports a compression rate of 38× without any structural knowledge of the weight matrix. In INQ, [24] reports a compression rate up to 89× when represented in 3-bit storage format. In our work, we obtain a slightly higher compression rate when DEFLATE is used. We present our results as shown in Table IV.

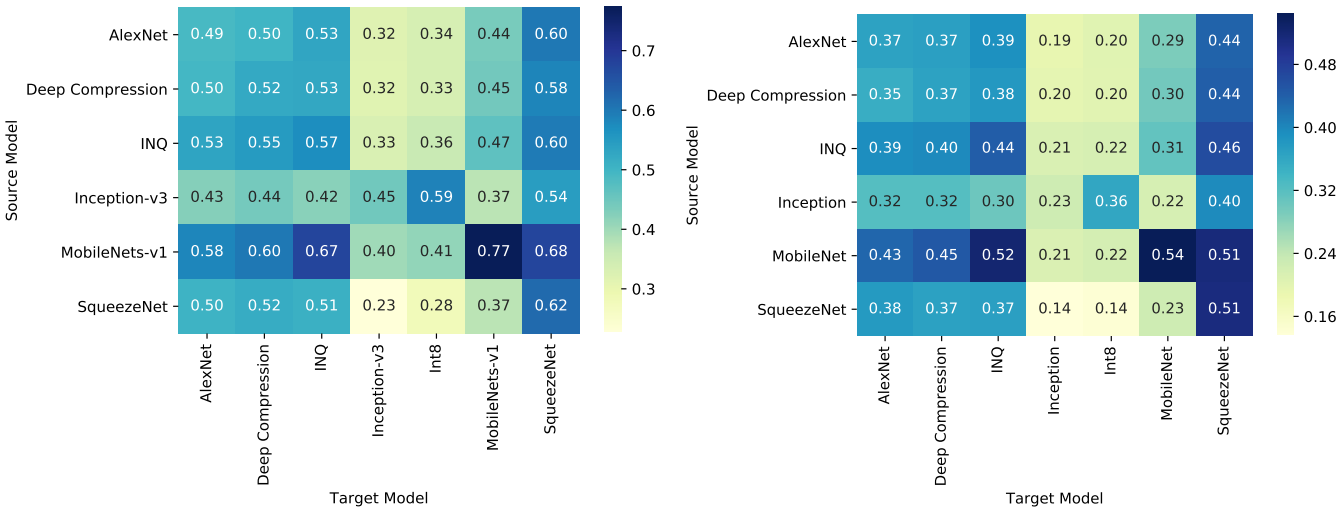
Overall, our method achieves an additional compression rate of 1× compared to the state-of-the-art. Furthermore, it gained an increased in performance of 1% for Top-5 accuracy and Top-1 accuracy. Given the fact that only us who use the adversarial training, we hypothesize that the increase in accuracy performance could be related to adversarial examples behaving as a regularizer.

E. Energy Consumption

We further investigate the energy efficiency of our proposal compared to AlexNet based models. We perform 10 evaluation runs on ImageNet's validation dataset (clean images). The evaluation settings are of default setting; 50 images per batch amounting to 1 epoch (1,000 iterations) on an Ubuntu

TABLE III: Accuracy on adversarial images

Source Model		Target Model: Inception	AlexNet	DeepComp.	INQ	Sq.Net	MobileNets	Int8
Inception	Top-1	0.4279	0.3288	0.3132	0.3320	0.2624	0.4417	0.3176
	Top-5	0.7220	0.5432	0.5396	0.5604	0.4848	0.7006	0.5920
AlexNet	Top-1	0.5246	0.2912	0.2808	0.2704	0.2296	0.3960	0.5124
	Top-5	0.7543	0.5076	0.5012	0.4908	0.4528	0.6346	0.7360
DeepComp.	Top-1	0.5257	0.2888	0.2684	0.2724	0.2400	0.3910	0.5133
	Top-5	0.7493	0.5200	0.5040	0.4988	0.4488	0.6278	0.7368
INQ	Top-1	0.5163	0.2676	0.2532	0.2460	0.2320	0.3768	0.4947
	Top-5	0.7343	0.4868	0.4804	0.4496	0.4320	0.6162	0.7240
SqueezeNet	Top-1	0.5980	0.2836	0.2720	0.2812	0.2184	0.4410	0.5559
	Top-5	0.8088	0.4988	0.5028	0.5068	0.3928	0.6902	0.7932
MobileNets	Top-1	0.4630	0.2400	0.2252	0.1868	0.1864	0.1604	0.4527
	Top-5	0.7366	0.4556	0.4400	0.3848	0.3904	0.4084	0.7216



(a) Performance Drop in Top-1 Accuracy

(b) Performance Drop in Top-5 Accuracy

Fig. 2: Vulnerability on adversarial images generated using FGSM

TABLE IV: Compression on AlexNet based on various approaches. P denotes pruning, Q denotes Quantization, H denotes Huffman Encoding and D denotes DEFLATE.

Methods	Comp.	Top-1 Error %	Top-5 Error %	Size (MB)
Original	-	42.76	19.77	240
PAQ8 (P+H)	38x	42.70	19.67	6.3
DeepCompression (P+Q+H)	35x	43.76	20.32	6.9
INQ (P+Q)	89x	42.61	19.54	2.69
Proposed (P+Q+D)	**90x**	**41.82**	**18.65**	**2.64**

16.06 LTS PC with an Intel(R) Core(TM) i9-7900X CPU @ 3.30GHz CPU and NVIDIA GTX 1080 Ti SLI GPU. Table V shows the average value comparison of 10 runs in terms of inference time, average power and energy consumption. While all compression methods of AlexNet show better average power and energy consumption than original models, our method gained an increased energy efficiency.

VII. Conclusion

In this paper we studied the robustness to adversarial examples of compressed and compact CNN models trained on ImageNet. In our experiments we used two adversarial example generation techniques, FGSM and BIM. Using a variety of CNN models, we demonstrate that smaller models are as vulnerable as the state-of-the-art full sized models. One distinguishable observation we could make is that compact models such as SqueezeNet are more vulnerable to adversarial examples compared to models made smaller by compressing full sized pre-trained models. We believe that high redundancy of weights coupled with regularization techniques such as dropout makes the larger network architectures more robust to perturbations in input. Based on experimental results, we stress that robustness to adversarial attacks should be used as

TABLE V: Energy consumption on AlexNet based on various compression methods.

	Time (s)		Average Power (J)		Energy (J)	
Methods	Average	Std. Dev.	Average	Std. Dev.	Average	Std. Dev.
Original	28.684	0.110	227.433	4.205	6,523.977	137.654
DeepCompression	28.695	0.138	219.647	3.437	6,302.909	110.570
INQ	**28.680**	0.121	214.629	3.062	6,155.900	110.552
Proposed	28.686	0.124	**214.072**	3.492	**6,141.046**	111.892

a metric in addition to accuracy in any network compression process.

ACKNOWLEDGEMENT

This work was supported by JSPS Grant-in-Aid for Scientific Research (B)(Grant Number 17H01785) and JST CREST (Grant Number JPMJCR1687).

REFERENCES

[1] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *nature*, vol. 521, no. 7553, p. 436, 2015.

[2] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 1–9.

[3] S. Han, X. Liu, H. Mao, J. Pu, A. Pedram, M. A. Horowitz, and W. J. Dally, "EIE: Efficient inference engine on compressed deep neural network," in *Proceedings of the 43rd International Symposium on Computer Architecture*, IEEE Press, 2016, pp. 243–254.

[4] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," in *Advances in Neural Information Processing Systems 25*, Curran Associates, Inc., 2012, pp. 1097–1105.

[5] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *ICLR*, 2015.

[6] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," *ICLR*, 2016.

[7] K. Ullrich, E. Meeds, and M. Welling, "Soft Weight-Sharing for Neural Network Compression," *ICLR*, 2017.

[8] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, "Quantized neural networks: Training neural networks with low precision weights and activations," *The Journal of Machine Learning Research*, vol. 18, no. 1, pp. 6869–6898, 2017.

[9] C. Zhu, S. Han, H. Mao, and W. J. Dally, "Trained ternary quantization," *ICLR*, 2017.

[10] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," in *NIPS Deep Learning and Representation Learning Workshop*, 2014.

[11] T. N. Sainath, B. Kingsbury, V. Sindhwani, E. Arisoy, and B. Ramabhadran, "Low-rank matrix factorization for deep neural network training with high-dimensional output targets," in *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference On*, IEEE, 2013, pp. 6655–6659.

[12] V. Sindhwani, T. Sainath, and S. Kumar, "Structured transforms for small-footprint deep learning," in *Advances in Neural Information Processing Systems*, 2015, pp. 3088–3096.

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

[14] I. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and Harnessing Adversarial Examples," *ICLR*, 2015.

[15] T. B. Brown, D. Mané, A. Roy, M. Abadi, and J. Gilmer, "Adversarial Patch," *arXiv preprint arXiv:1712.09665*, Dec. 27, 2017.

[16] D. Warde-Farley, I. Goodfellow, T. Hazan, G. Papandreou, and D. Tarlow, "Adversarial perturbations of deep neural networks. Perturbations," *Optimization, and Statistics*, vol. 2, 2016.

[17] A. Kurakin, I. Goodfellow, and S. Bengio, "Adversarial examples in the physical world," *ICLR Workshop*, 2017.

[18] N. Papernot, P. McDaniel, X. Wu, S. Jha, and A. Swami, "Distillation as a defense to adversarial perturbations against deep neural networks," in *2016 IEEE Symposium on Security and Privacy (SP)*, IEEE, 2016, pp. 582–597.

[19] W. Xu, D. Evans, and Y. Qi, "Feature squeezing: Detecting adversarial examples in deep neural networks," *arXiv preprint arXiv:1704.01155*, 2017.

[20] X. Yuan, P. He, Q. Zhu, R. R. Bhat, and X. Li, "Adversarial examples: Attacks and defenses for deep learning," *arXiv preprint arXiv:1712.07107*, 2017.

[21] Y. LeCun, J. S. Denker, and S. A. Solla, "Optimal Brain Damage," in *Advances in Neural Information Processing Systems 2*, Morgan-Kaufmann, 1990, pp. 598–605.

[22] S. Han, J. Pool, J. Tran, and W. Dally, "Learning both weights and connections for efficient neural network," in *Advances in Neural Information Processing Systems*, 2015, pp. 1135–1143.

[23] Y. Guo, A. Yao, and Y. Chen, "Dynamic network surgery for efficient dnns," in *Advances In Neural Information Processing Systems*, 2016, pp. 1379–1387.

[24] A. Zhou, A. Yao, Y. Guo, L. Xu, and Y. Chen, "Incremental network quantization: Towards lossless cnns with low-precision weights," *ICLR*, 2017.

[25] V. Vanhoucke, A. Senior, and M. Z. Mao, "Improving the speed of neural networks on CPUs," in *Proc. Deep Learning and Unsupervised Feature Learning NIPS Workshop*, vol. 1, Citeseer, 2011, p. 4.

[26] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer, "SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5MB model size," Feb. 23, 2016.

[27] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *arXiv preprint arXiv:1704.04861*, 2017.

[28] A. Kurakin, I. Goodfellow, and S. Bengio, "Adversarial Machine Learning at Scale," *ICLR*, 2017.

[29] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 2818–2826.

[30] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, and M. Bernstein, "Imagenet large scale visual recognition challenge," *International Journal of Computer Vision*, vol. 115, no. 3, pp. 211–252, 2015.

[31] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, "Caffe: Convolutional architecture for fast feature embedding," in *Proceedings of the 22nd ACM International Conference on Multimedia*, ACM, 2014, pp. 675–678.

[32] N. Papernot, F. Faghri, N. Carlini, I. Goodfellow, R. Feinman, A. Kurakin, C. Xie, Y. Sharma, T. Brown, A. Roy, A. Matyasko, V. Behzadan, K. Hambardzumyan, Z. Zhang, Y.-L. Juang, Z. Li, R. Sheatsley, A. Garg, J. Uesato, W. Gierke, Y. Dong, D. Berthelot, P. Hendricks, J. Rauber, R. Long, and P. McDaniel, "Technical Report on the CleverHans v2.1.0 Adversarial Examples Library," *arXiv preprint arXiv:1610.00768*, Oct. 3, 2016.

[33] B. Knoll and N. de Freitas, "A machine learning perspective on predictive coding with PAQ8," in *Data Compression Conference (DCC), 2012*, IEEE, 2012, pp. 377–386.