Towards Robust Compressed Convolutional Neural Networks

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

Jun Jin Choong *
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*—Recent studies on robustness of Convolutional Neural Network (CNN) shows that CNNs are highly vulnerable towards adversarial attacks. Meanwhile, smaller sized CNN models with no significant accuracy loss are being introduced to mobile devices. However, only the accuracy on standard datasets is reported along with such research. The wide deployment of smaller models on millions of mobile devices stresses importance of their robustness. In this research, we study how robust such models are with respect to state-of-the-art compression techniques such as quantization. Our contributions include: (1) insights to achieve smaller models and robust models (2) a compression framework which is adversarial-aware. In the former, we discovered 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

Machine learning algorithms has helped us overcome many struggles in our every day life. From language translation, discovering new drugs and domination in games such as Go, the performance of machine learning algorithms today is unquestionably highly favored by recent breakthrough in training Deep Neural Networks (DNN) which is now called Deep Learning [1]. One of the primary ingredient to the successes of Deep Learning models is due to the success of highly complicated models such as the Convolutional Neural Network (CNN) architectures itself [2]. These complicated model resulted in higher performance gain, with the consequences of gaining larger model. As a result, the model is bigger in file size, higher memory consumption and slower inferencing speed [3]. Although negligible for larger devices such as personal computers, it remains a fundamentally challenged problem for smaller devices especially smart phones and devices on the edge (embedded devices). These devices are typically build to be energy efficient and has much smaller computational power. State-of-the-art DNN models such as AlexNet Caffemodel [4] and VGG-16 Caffemodel [5] easily outsize these requirements; with the former model being 200MB and the latter 500MB. Such large models are difficult to fit on smaller devices with low footprint. To ensure state-of-the-art performances in such scenario, several methods has been proposed to shrink CNN models [6–9]. These methods are specifically tuned to handle compression without compromising the accuracy of existing models. Furthermore, they are architecture agnostic and does not require models to be rebuild. On the contrary, approaches like distillation [10] and low-rank factorization [11, 12] require models to be rebuild. Although various approaches exist, a crucial question remains to be unaddressed, "How far can compressed Convolutional Neural Networks be compressed without compromising accuracy and robustness?".

* These authors contribute equally

To this end, we attempt to address the above question. We show experimentally show that compressed and compact CNNs are equally vulnerable to adversarial examples; a sample of input data which has been perturbed to encourage the classifier to misclassify. In attempt to study 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. We summarize 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 experimentally show that adversarial training via quantization can improve accuracy of compressed networks whilst also being smaller. Our proposed framework achieved better compression rate (90× on AlexNet) than the state-of-the-art method and is 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. The remaining sections discuss about our experimental setup and findings.

II. Generating Adversarial Examples

Szegedy, et al. [13] reported an intriguing property of machine learning models including neural networks; their vulnerability to adversarial examples. One can simply trick a neural network model to misclassify data with a high confidence by slightly perturbing input images [14]. On the contrary, a bolder technique to adversarial examples includes the introduction of an adversarial patch (sticker) in the training image [15]. By including this sticker on various images, Brown *et al.* is able to prove that a highly confident CNN classifier performs otherwise. The nonlinearity property of CNNs has been previously hypothesized as a possible cause of such tricks (attacks). Later finding [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 attacks across various models can be explained as a result of adversarial perturbations being highly aligned with the weight vectors of a model, and different CNN models learning similar functions when trained to perform the same task.

Two types of adversarial attacks can be performed depending on how the adversarial images are generated. The first type of attacks, *white-box attacks* are constructed for a particular Machine Learning (ML) model utilizing knowledge of its parameters. White-box attacks on an ML model can be mitigated by restricting the access to the particular model. *Black-box attacks* exploits *transferability* property, an inherent aspect of machine learning 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.

Without special attention, adversarial examples can lead to disastrous accidents especially in fields such as autonomous vehicles (self-driving cars). For instance, 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 techniques used for generating adversarial images. FGSM is a "one-shot" technique for efficiently generating adversarial examples using a fixed l_∞ norm. Given x as an input image and x^{adv} be the generated adversarial example. Let us 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:

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

being ϵ a hyper-parameter to be selected for some $\epsilon > 0$ that controls the perturbation magnitude.

Basic Iterative Method (BIM) [17] is a straightforward extension of FGSM technique. BIM applies FGSM iteration multiple times with small step size and clip pixel values of intermediate results after each step. This iteration process provides a guarantee that the adversarial images are in an ϵ-neighborhood of the original images. Let $\text{Clip}_{x,\epsilon}\{x'\}$ denotes the function which performs per-pixel clipping of x' image ensuring the result in L_∞ ϵ-neighborhood of the original input x. The generated adversarial BIM 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

CNN models are often over parametrized. For that reason, a fraction of weights (parameters) in a CNN model can be removed or pruned without reducing the original accuracy. The requirement for compression of deep neural networks is driven by the restrictions on the aimed hardware platform. Compressing model size as well as reducing energy consumption are common objectives in obtaining smaller sized neural networks. In this work, we loosely define all such techniques as network compression. The study on network compression can be classifed into following categories:

- *Compressing pre-trained full size networks.* This includes pruning methods, reducing the precision of weights, non-linear quantization and soft weight sharing.
- *Constructing and training compact CNN models from scratch.*

The former compression class are network agnostic while the latter requires re-engineering of the network architecture. In this work, our focus is the former compression class.

A. Compressing Pre-Trained Networks

Convolutional neural networks often contain redundant weights provide us a large set of weights that can be pruned away (i.e., set to zero). Network pruning technique, called as optimal brain damage was first introduced by LeCun *et al.* [21] as an attempt to avoid overfitting and reduce

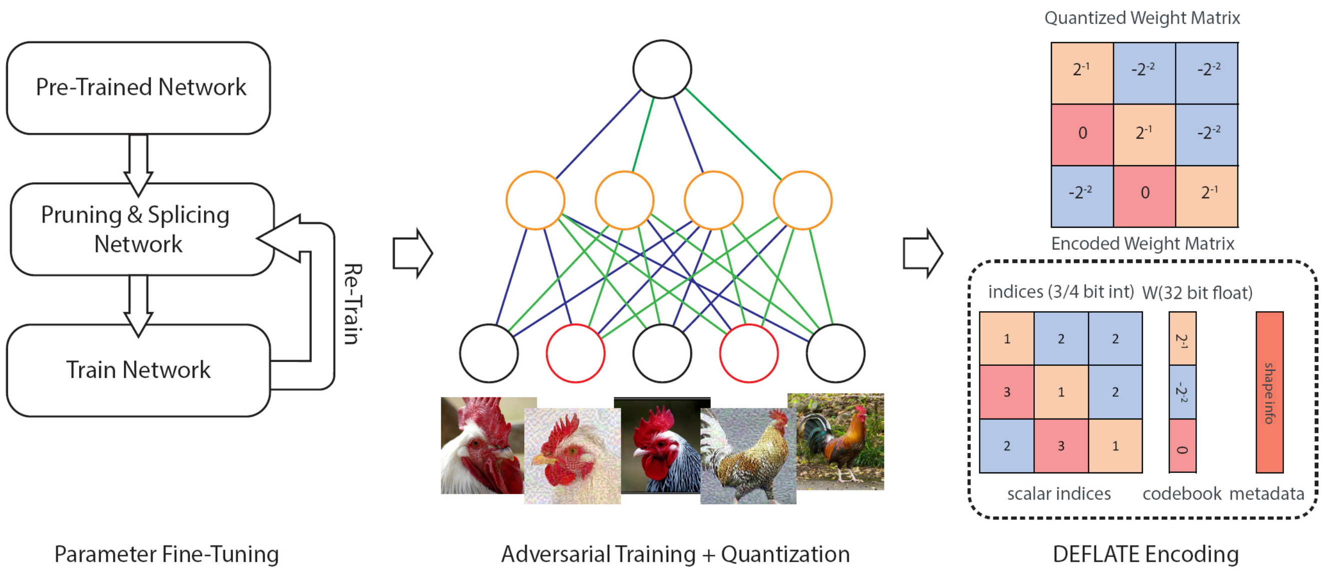


Fig. 1: Proposed pipeline of CNNs compression to increase robustness and compression rate. Firstly, a pre-trained full size model is fine-tuned and pruned unnecessary weights. Secondly, a combination of adversarial training with original image datasets are introduced during network quantization phase. Finally, codebook is produced based on finalized network 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.

model complexity. In most cases, existing methods on network pruning has proposed to trade accuracy for size and performance. However, a recent finding by Han *et al.* [22] alleviated this issue by coupling pruning with fine-tuning. In a three stage compression pipeline of network pruning with quantization and Huffman encoding, Han *et al.* [6] introduced Deep Compression which obtained a better compression ratio. Notably, AlexNet was compressed up to 35× and VGG-16 up to 49× without loss of accuracy. This approach, however, is computationally slow and inefficient. Iterative pruning itself needs huge amount of training iteration and time to guarantee no accuracy loss. Furthermore, 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 treating it as a problem of mixture model. Even though the compression ratio is significantly higher, this approach is not scalable for larger models like VGG-16. More recently, Zhou *et al.* [24] suggest to quantize network weights in powers of twos or zeroes. Termed as Incremental Network Quantization (INQ), the proposed method obtained an impressive result with up to 89× compression ratio on AlexNet when coupled with DNS and 3-bit compression. Because the value of quantized weights are constrained in powers of two or zeroes, it has the potential to give a fast inferencing speed without requirement of specialized hardware. This approach is promising to be implemented with bit-shifting operations which are widely available on CPU and GPU.

One important disadvantage of the compression techniques discussed above is that they need customized hardware or improvements to existing CNN frameworks to gain the full potential of smaller model size. Reducing the precision of weights and operations of pre-trained networks [25] is another technique used to obtain better speed and reduced model size. Even though low precision network models can be applied on commodity hardware easily, the size of such models are often bigger than the ones obtained through pruning and quantization.

B. *Designing Compact Neural Network Models*

Besides compression, another approach to achieve smaller network models with comparable performance to the larger models is constructing a compact network architectures. SqueezeNet [26] and MobileNets [27] are examples of such compact architectures designed to be used on hardware platforms with resource limitations. Even though such models exhibit accuracy performance only slightly worse than the state-of-art, they are significantly smaller than the latter. In our investigation, we use both SqueezeNet and MobileNets models due to their widely known application.

IV. Adversarial-aware Compression Framework

One distinct contribution to existing compression approaches which design models without a robustness objective, we include training with adversarial examples taking advantage of three interdependent techniques: Dynamic Network Surgery (DNS) on-the-fly network weight pruning, Incremental Network Quantization (INQ) low bit-width layer precision weights and DEFLATE encoding. The outcome of this pipeline is a small yet robust network. It has been hypothesized that adversarial training of CNNs can act as a model regularizer [28]. Hence, this training makes the network much more robust against adversarial attacks and gain a better performance.

A. Network Pruning with Dynamic Network Surgery

We utilize an efficient pruning algorithm, Dynamic network surgery (DNS) [23] to reduce the redundancy of network weights. DNS consists of two operations, pruning and splicing. *Pruning* operation reduces the number of weights (parameters) of CNN model by removing unimportant parameters. This operation is similar to a well proven pruning method of [6]. *Splicing* overcomes the problem of incurring accuracy loss resulted from incorrect pruning or over-pruning by restoring some of the previously pruned weights based on their present importance value. Parameter importance is decided by the combined application of pruning and splicing. Moreover, DNS obtained 7× speed-up (approximately 140 epochs) during re-training of AlexNet model and it achieved better compression rate of pruning from 9× to 17.7×.

Specifically, the objective of DNS is to optimize the given l^{th} layer of the network using the following loss function:

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

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:

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

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

We conduct adversarial training phase to minimize an upper bound on the expected cost over noisy examples by including noise to the original inputs.

We utilize a proven efficient network quantization technique, called INQ [24] to convert pre-trained full-precision network into a low-precision version constrained weights in either powers of two or zero.

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

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, the weight set is updated using 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.

The proposed adversarial training combined with DNS, INQ and DEFLATE is presented in Algorithm 1. Several hyperparameters in Algorithm 1 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 work, 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

Han, et al. [6] introduced Huffman encoding 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

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

Algorithm 1 Proposed Compression Method (DNS + INQ + DEFLATE) with Adversarial Training

```
Input: training data with adversarial examples $\mathbf{X}$, pre-trained full-precision CNN model $\{\mathbf{W}_l : 1 \leq l \leq L\}$
Output: quantized weights $\{\widehat{\mathbf{W}}_l : 1 \leq l \leq L\}$ 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 $1 \leq l \leq L$
  for $n = 1, ..., N$ do
      Reset base learning rate and the learning policy
      Based on $\sigma_n$, 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 $\mathbf{A}_l^{(2)} : 1 \leq l \leq L$
      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($\widehat{\mathbf{W}}, \mathbf{h}$)
```

approach, we use the DEFLATE (LZ77 + Huffman) method for compression instead of standard Huffman encoding. The DEFLATE method is a widely supported method for lossless compression. In practice, this method has widely been used by standard consumer hardwares such as System on Chips (SoC) for fast compression and decompression.

The DEFLATE method consists of two phases: duplicate string removal and bit reduction. Duplicated series of bytes spotted are back-referenced, linking the current series of bytes to the previous location. This operation is performed using a sliding window. Huffman coding is then used for bit reduction. Essentially, a series of bytes which appear more often are represented by the shorter sequence of bits. In our case, the higher the frequency weights are represented by a shorter bit length. This representation are composed using a Huffman tree which predetermines the sequence of codes when reconstructed.

V. Experimental Setup

In this section, the network models and datasets that we use in our experiments are explained. ImageNet Scale Visual Recognition Challenge 2012 (ILSVRC 2012) [30] is used as the dataset for training and validating all of CNN models. Currently this is one of the largest publicly available dataset consists of 1.2 million training images, 1,000 object classes, 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 Incremental Network Quantization method (INQ)
- Compact models: SqueezeNet and MobileNets-V1
- Low precision model: Inception-V3, both weights and computations converted to 8-bit integer values

We use open source frameworks of Caffe [31] and Tensorflow in all experiments. We use Caffe models obtained from “Caffe model-zoo”[1]. A Tensorflow-based library for adversarial attack ’Cleverhans’ [32] is used to generate all adversarial images in robustness evaluations. For CNN models which have no publicly available pre-trained Tensorflow models (e.g., AlexNet), the model weights are converted into Tensorflow from Caffe model using a free and open-source library [2]. We verified the converted Tensorflow models prior using in experiments with ImageNet validation dataset. All the results are reported in two standard metrics, namely top-1 and top-5 accuracy. We conduct our experiments on an Ubuntu 16.06 LTS machine powered by an Intel(R) Core(TM) i9-7900X CPU @ 3.30GHz CPU and NVIDIA GTX 1080 Ti SLI GPU.

VI. Experimental Results and Discussion

In this section, the performance of different network models against adversarial images are demonstrated. All models in the experiments are pre-trained with ImageNet dataset. We report the accuracy with respect to the ImageNet validation dataset consisting of 50,000 images. The ϵ value of FGSM method is set to 2. One of an interesting issue in generating the adversarial examples is known as *label leaking effect* [28].

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

An adversarially trained CNN model can learn to exploit regularities in the adversarial example generation process if the true labels of images are used in the adversarial image generation using a single-step attack technique such as FGSM. This gives us incentives to construct adversarial examples without using ground truth labels. To handle this issue, we utilize the class corresponding to the maximum prediction probability instead.

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

A. White-box attacks

White-box attacks is an evaluation where the attacked CNN model is also used to generate the adversarial examples. For example, to evaluate the accuracy of an Inception-v3 model on ImageNet, we generate adversarial images with ImageNet validation dataset using the same Inception-v3 model. Table II shows the accuracy of each model against white-box attacks.

Among all AlexNet-based model, Table II shows that our proposed model is more robust against white-box adversarial attacks. It obtains higher accuracy on clean images as well as adversarial examples generated using FGSM and BIM techniques compared to the original and other AlexNet-base compressed models.

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
Ours	Top-1	**0.5818**	**0.3364**	**0.3620**
	Top-5	**0.8135**	**0.5648**	**0.6008**

B. Black-box attacks

In black-box attacks, we generate the adversarial examples using one model to attack all other models using the generated adversarial images. Akin to white-box attacks, FGSM and BIM methods are used to generate the adversarial examples. However, we observed that the effectiveness of BIM method for black-box attacks is low as reported by [28]. Thus, we limited our investigation on transferability exclusively to FGSM. In low precision model such as Int8, because both matrix multiplications and network weights of Inception-V3 are converted to 8-bit integers, it is not possible to calculate gradients of the Int8 model. Hence, we only evaluate the robustness of low precision Int8 model on black-box attacks.

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

Table III shows the accuracy on adversarial images generated using FGSM method. We can observe that all CNNs models show high transferability of adversarial images. One disctinct 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 measure the vulnerability of a model using the following formula.

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

The vulnerability of each model as heat-maps are shown in Figures 2a and 2b. Each diagonal cell represents the model accuracy on adversarial images generated by the same model, hence the white-box attacks. Rest of the cells show the result of accuracy on images under black-box attacks.

MobileNets and SqueezeNet models exhibit the maximum deviation of accuracy performance for white-box attacks. According to the results, MobileNets model can not correctly label 84% of the adversarial images generated by its own model parameters. Even though MobileNets model achieves a comparable accuracy performance for clean images, it performs much worse when it is attacked by adversarial images. We believe that the low model capacity of MobileNets network compared to Inception-V3 as the factor contributing to higher vulnerability to adversarial attacks. Interestingly, the low-precision Int8 model performed almost similar to its full-precision Inception-V3. 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.

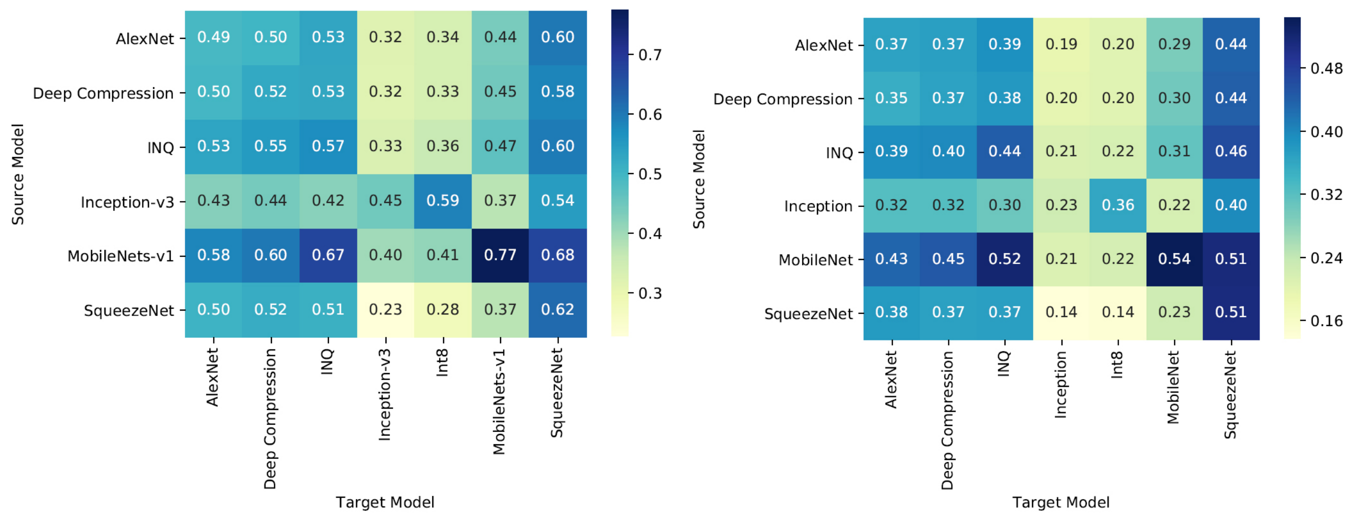
An essential insight we can obtain from this results is that the compression of pre-trained full size networks gives us smaller networks with a relatively low impact on robustness compared to compact models trained from scratch. Investigating why compact CNN models show higher vulnerability to white-box attacks is an interesting research direction which may shed light on how to design more robust compact models.

D. Compression Rate

We compare our proposed compression algorithm against PAQ8[33], Deep Compression and INQ[24] as baseline compression methods. PAQ8 is agnostic to file structures and

TABLE III: Accuracy on adversarial images

		Target Model						
Source 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	Compression Rate	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
Ours (P+Q+D)	**90x**	**41.82**	**18.65**	**2.64**

employs ensembles learning approaches for lossless compression [33]. When compared to Deep Compression, the baseline compression reports a compression rate of 38$\times$ without any structural knowledge of the weight matrix. In INQ, [24] reports a compression rate up to 89$\times$ when represented in 3-bit storage format. Our compression method obtains a slightly better compression rate when DEFLATE is used. We show the full results as presented in Table IV.

Overall, our method obtains an improved compression rate of 1$\times$ compared to the state-of-the-art. Furthermore, it achieved 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 training behaving as a model regularizer.

VII. Conclusion

In this work we investigated the robustness to adversarial attacks of compressed and compact Convolutional Neural Networks (CNNs) models trained on ImageNet. Two popular methods of adversarial image generation called FGSM and BIM, are used in our robustness evaluation. From the experimental result on various CNN models, we observe that smaller models (compressed and compact CNNs) are as vulnerable as their state-of-the-art full sized counterparts. One important insight we found from this experiment is that compact models trained from scratch such as SqueezeNet are more vulnerable to adversarial attacks compared to compressed version of full sized pre-trained models. We believe that the high redundancy of network parameters coupled with regularization methods such as dropout makes the larger CNNs architectures more robust to input perturbations. Furthermore, based on experimental results, we highlight that the robustness to adversarial attacks should be used as an evaluation metric in addition to accuracy in any deep network compression approaches.

VIII. Acknowledgements

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. Manfffdfffdfffdfffd, 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.