Robustness of tensor decomposition-based neural network compression

Théo Rudkiewicz, Mohamed-Oumar Ouerfelli, Riccardo Finotello, Zakariya Chaouai, Mohamed Tamaazousti

ROBUSTNESS OF TENSOR DECOMPOSITION-BASED NEURAL NETWORK COMPRESSION

Théo Rudkiewicz[⋆] *Mohamed Ouerfelli*[†] *Riccardo Finotello*[†]
Zakariya Chaouai[†] *Mohamed Tamaazousti*[†]

[⋆] Ecole Normale Supérieure Paris-Saclay
[†]Université Paris-Saclay, CEA, List, F-91120, Palaiseau, France

ABSTRACT

Neural networks (NN) often need to be lightweight and fast for practical deployment. Various NN compression techniques, including tensor decomposition, aim to achieve this goal. Tensor decomposition involves computing a decomposition of the weight tensor of a layer and replacing it with smaller layers using the decomposed weights. Subsequently, fine-tuning is performed to regain lost accuracy. However, while NNs are known to exhibit robustness issues, tensor decomposition for compression has primarily been evaluated on accuracy. In this study, we investigate the impact of tensor decomposition on the robustness of large CNN (Convolutional Neural Network) models. Through multiple experiments on different models trained on ImageNet, we demonstrate that tensor decomposition preserves model robustness. Furthermore, we observe that the choice of fine-tuning learning rate plays a crucial role in determining robustness. A high learning rate may enhance accuracy but significantly compromises robustness. Conversely, a low learning rate can effectively restore model robustness, albeit with a smaller accuracy improvement. These findings offer a practical approach to preserving model robustness without resorting to adversarial learning, thus eliminating the need for additional knowledge about the defense methods used in the original model.

***Index Terms*—** Neural network compression, tensor decomposition, robustness, adversarial attacks, trustworthy AI.

1. INTRODUCTION

For a decade, neural networks have been the best performing algorithms for a large variety of tasks. They first brought major breakthrough in visual tasks such as character recognition, image classification, object detection, image segmentation or depth estimation. Since then, their applications extended to various other areas where it made significant advancements such as Large Language models for Natural Language Processing. Those neural networks are characterized by an important number of parameters in contrast with traditional algorithms (such as nearest neighbors, binary tree…). For example, AlexNet , VGG-16 , ResNet or GoogleNet have tens of millions trainable parameters.

Compression. Although neural networks often benefit from a vast number of parameters during training, there may be potential to create a more compact representation of the trained model. In particular, this question is crucial because many applications demand efficient execution on embedded devices with limited computational resources. Notably, this is essential for tasks like image classification on smartphones and applications for autonomous vehicles. This led to the development of a rich line of research for developing techniques that aim to lower the number of parameters of neural networks with different techniques [1] such as quantization, pruning, knowledge distillation or low-rank factorization.

Quantization proposes to lower the numerical precision of weights during storage and computation. Pruning is suppressing some connections between layers, that can be arbitrary connections or blocks of connections to make the network effectively faster. Distillation generally relies on the teacher-student framework with the aim of training a smaller network with guidance from an already trained larger network. Low-rank factorization compresses the information of the weights tensors representing connections between layers by storing it in a low rank subspace. Those techniques can be either performed during or after training. In the latter case, this allows to compress already trained networks that were not specifically designed to be compressed. Furthermore, one should note that these different methods are in principle compatible for a simultaneous integration.

Adversarial attack and robustness. It has been showed that neural networks are vulnerable to adversarial examples [2]: inputs that have been slightly modified through small perturbations, can lead to incorrect predictions. The vulnerability to adversarial inputs can be problematic as it presents security risks and hinders the adoption of deep learning methods in safety-critical applications.

There are various attack techniques (see section 4) that aim to identify the most efficient perturbations to deceive the neural network. These attacks can be employed to assess and

improve the robustness of the neural network.

Contributions and outline. The Assessment List for Trustworthy Artificial Intelligence (AI) [3] outlines as key requirements both the crucial aspect of the environmental well-being (for example through smaller and less energy-intensive models) as well as the essentiality of the technical robustness of the models. It is thus natural to investigate the robustness of compression techniques, that we will assess through the vulnerability to various adversarial attacks. This question was already partially answered for quantization, pruning and knowledge distillation [4] but to the best of our knowledge there is no clear evaluation for the low-rank factorization technique. In this paper we propose to investigate the robustness of models compressed by low-rank factorization. In particular, we will focus on the two classical and popular tensor decomposition techniques, the Canonical polyadic (CP) decomposition and the Tucker decomposition. These two decompositions are specifically well suited for CNN largely used for visual data processing [5, 6].

We measure with extensive experiments on ImageNet the robustness of models obtained by compressing models with different degrees of robustness [1]. Our investigation highlights the fact that tensor decomposition-based compression maintains the robustness of the original model. Surprisingly, the results suggests that a small learning rate in the fine-tuning step is critical for the preservation of the robustness. Indeed, a large learning rate converges to a more accurate but less robust model. These results have significant implications for practical applications where the objective is to compress models while keeping their robustness. Indeed, existent methods necessitate adversarial fine-tuning to achieve a robust compressed model.

In section 2, we provide an overview of the current state of the art. In section 3 and section 4, we detail the methods utilized in our experiments and in particular the compression by tensor decomposition techniques as well as adversarial attacks. In section 5, we present our experiments and our results.

2. STATE OF THE ART

In this paper, we explore the intersection of the compression and robustness subjects.

For pruning, [7] concluded that pruning a standard model improves its robustness against Fast Gradient Signed Method (FGSM) and occlusion attacks, supposedly because of the regularization effect of pruning. For knowledge distillation, [8] shows that the student network does not inherit the robustness of the teacher model with standard knowledge distillation and propose a modification of the knowledge distillation protocol to partially preserve the robustness. In the domain of speech processing, [9] proposed another knowledge distillation technique focusing on robustness against noise. Fu *et al.* [10] showed that quantization improves robustness and proposed a new robustification technique based on quantization. Zhao *et al.* [11] studied when the attacks are transferable between compressed (quantized or pruned) and uncompressed models.

From a different angle, some other works try to use compression as a tool for better robustness. Ye *et al.* [12] uses pruning to create robust model that are not too big. Other works like [13] and [14] use decomposed tensors not directly to compress but as a better structure for robustness, they developed respectively defensive tensorization and tensor dropout. In addition to compressing models it is possible to decompose the input tensor data to remove noise and perturbations to enhance the robustness as noted by [15].

Concerning the combination of low rank factorization and robustness, Gui *et al.* [16] developed a framework to compress models without hurting the robustness of the model. It uses an optimization algorithm that integrates the adversarial fine-tuning with the constraints of sparsity and pruning. This optimization algorithm can be performed on a model that has been compressed through low-rank factorization. Thus, their focus lies in exploring the combination of adversarial training with low-rank factorization, whereas our approach probes a more fundamental question: whether tensor decomposition inherently preserves robustness.

Our experiments led to a surprising and interesting observation, a standard fine-tuning with a small enough learning rate is able to converge to a minimum that restores most of the robustness of the original model. Additionally, we observed that the restored robustness is comparable to what could have been achieved through adversarial fine-tuning as we compared the two methods. These new observations offer a different method for compressing robust models with tensor decomposition. It is simpler, more generalizable and does not require supplementary expertise or knowledge on the defense method used for the original model.

3. COMPRESSION BY DECOMPOSITION

The compression by tensor decomposition is done by replacing layer of the network with a new smaller layer or sequence of layers. For our experiments, we investigated the important task of image classification with convolution networks, we therefore focus on the compression of linear layers (also called fully connected layer) and convolution layers.

3.1. Linear layer compression

A linear layer is a function $f_M : \mathbb{R}^n \to \mathbb{R}^m$ defined by $f_M(x) = Mx$ where $M \in \mathbb{R}^{m \times n}$. M can be approximated by a product $M \approx UV^\top$ with $U \in \mathbb{R}^{m \times r}$, $V \in \mathbb{R}^{n \times r}$ and $r \in \mathbb{N}$ a parameter that controls how hard the approxima-

[1] The results and the compressed models are available here: `https://github.com/TheoRudkiewicz/ICIP2024-robustness`

tion is. For a fixed rank r, the best approximation can be computed using the singular value decomposition (SVD). Using this decomposition we have $f_M(x) = Mx \approx UV^\top x = (f_U \circ f_{V^\top})(x)$, hence we can replace the linear layer f_M with two successive linear layers f_U and $f_{V^\top}$.

Now we compare the storage and computational cost of the original layer and the new layers. In the first case there is mn parameters and mn additions-multiplications. On the other hand, the two layers have $mr + nr$ parameters and require $mr+nr$ additions-multiplications. As a consequence, if r is small enough this process lead to compressing the linear layer.

3.2. Choice of the rank - VBMF

Choosing the right r is difficult as it should be small enough to compress the layer but should also lead to a small enough approximation error. In the literature, different proposed procedure [17, 18] use the VBMF (Variational Bayesian Matrix Factorization) algorithm. The idea of this algorithm it to estimate the rank of a matrix M with access to a noised version of the matrix: $M + \varepsilon$ where ε is a centered gaussian noise. In our procedure, we use a compromise between $\mathrm{R}_{\max} = \min\{m, n\}$ as the theoretical maximum rank of the matrix and $\mathrm{R}_{\mathrm{VBMF}}$ the rank given by VBMF. We parameterize this rank by α (α will be precised in the experiments and called VBMF ratio) and it is given by:

$$\mathrm{R}_\alpha = \mathrm{R}_{\mathrm{VBMF}} + (1-\alpha)(\mathrm{R}_{\max} - \mathrm{R}_{\mathrm{VBMF}}) \qquad (1)$$

3.3. Convolution layer compression

The convolution layer is one of the most common layer in neural network especially for computer vision. According to [1], they represent more than 90% of the computation time in AlexNet, VGG or ResNet.

Convolution layer. A convolution $\mathfrak{C}_\mathcal{K}$ transforms a tensor $\mathcal{X} \in \mathbb{R}^{S\times W\times H}$ (for example an image of size $H \times W$ with $S = 3$ colors) into an other tensor $\mathcal{Y} \in \mathbb{R}^{T\times W'\times H'}$ and is parameterized by a kernel tensor $\mathcal{K} \in \mathbb{R}^{T\times S\times W_d\times H_d}$. (For convenience we also note w_d and h_d such that $2w_d+1 = W_d$ and $2h_d+1 = H_d$.) $\mathcal{Y}$ is computed according to the following formula [2]:

$$\mathcal{Y}[t, y, x] = \sum_{s=1}^{S} \sum_{h=-h_d}^{h_d} \sum_{w=-w_d}^{w_d} \mathcal{K}[t, s, h, w]\mathcal{X}[s, y+h, x+w]$$

As for the linear layer, to compress a convolution layer we can decompose the parameter $\mathcal{K}$ and replace the layer $\mathfrak{C}_\mathcal{K}$ with a succession of new layers. The original tensor has TSW_dH_d parameters and the associated convolution require as much additions-multiplications. We will present the two most natural generalisation of the SVD decomposition, both already used with success for neural network compression: the CP decomposition [19] and Tucker decomposition [18].

CP decomposition. The CP-decomposition of $\mathcal{K}$ is:

$$\mathcal{K} = \sum_{r=1}^{R} \lambda_r U_r^{(T)} \otimes U_r^{(S)} \otimes U_r^{(W_d)} \otimes U_r^{(H_d)} \qquad (2)$$

Where $\lambda_i \in \mathbb{R}$ is the equivalent of singular value, $U_i^{(j)} \in \mathbb{R}^{n_i}$ is the equivalent of a singular vector and R is the rank of the decomposition. $\otimes$ is the outer product which is the generalization to tensors of the product of a column vector by a line vector.

The choice of R is the same dilemma as for the SVD. As suggested by [17], it is possible to unfold the tensor of order 4 into different matrix and estimate the rank of each matrix and then take the maximum of all the different folding as $\mathrm{R}_{\mathrm{VBMF}}$. We can also compute a maximal possible rank with $\mathrm{R}_{\max} = (TSW_dH_d)/\max\{T, S, W_d, H_d\}$. Like for linear layer we use a combination of $\mathrm{R}_{\max}$ and $\mathrm{R}_{\mathrm{VBMF}}$ given by Eq. (1). This decomposition can be computed efficiently with an alternating least square algorithm. (We used the Tensoly [20] library.)

We now compute the output with the decomposed version of $\mathcal{K}$:

$$\sum_{r=1}^{R} \sum_{w=-w_d}^{w_d} \sum_{h=-h_d}^{h_d} \sum_{s=1}^{S} U_r^{(T)}[t] U_r^{(W)}[w] U_r^{(H)}[h] U_r^{(S)}[s] \mathcal{X}[s, y+h, x+w]$$

This expression leads to a new sequence of 4 convolution layers (1×1 parameterized by $U^{(S)}$, $H_d \times 1$ parameterized by $U^{(H)}$, $1\times W_d$ parameterized by $U^{(W)}$ and 1×1 parameterized by $U^{(T)}$) which has $R(T + S + H_d + W_d)$ parameters and as much additions-multiplications. If we compare to the original convolution if R is small enough there is a compression of the layer.

Tucker decomposition. The Tucker decomposition decomposes $\mathcal{K}$ as $\mathcal{K} = \mathcal{G} \times_1 U^{(T)} \times_2 U^{(S)} \times_3 U^{(H_d)} \times_4 U^{(W_d)}$. Where $\mathcal{G} \in \mathbb{R}^{R_T\times R_S\times R_{H_d}\times R_{W_d}}$ is a core tensor, $\left(U^{(T)}, U^{(S)}, U^{(H_d)}, U^{(W_d)}\right)$ a set of factor matrices with $U^{(A)} \in \mathbb{R}^{R_A\times A}$ and $\times_i$ is the mode product along mode i.

Contrary to the CP decomposition, to be able to use the decomposition in a new sequence of layer, its better to only decompose along axis S and T leading to what is called Tucker-2 decomposition: $\mathcal{K} = \mathcal{G} \times_1 U^{(T)} \times_2 U^{(S)}$. This decomposition can be computed efficiently with an alternating least square algorithm [20]. For one element we get this expression:

$$\mathcal{K}[t, s, h, w] = \sum_{r_s=1}^{R_S} \sum_{r_t=1}^{R_T} \mathcal{G}[r_t, r_s, h, w] U^{(T)}[r_t, t] U^{(S)}[r_s, s]$$

Like for CP we can replace $\mathcal{K}$ by its decomposition in the expression of the convolution to get the new sequence of layer. We then get three new layer: first a linear layer $f_S(\mathcal{X}) = \mathcal{X}_1[r_s, y, x] = U^{(S)}[:, r_s] \cdot \mathcal{X}[:, y, x]$, then a smaller convolution $\mathfrak{C}_\mathcal{G}$ parameterized by $\mathcal{G}$ and another linear layer

[2] Here we avoid the boundary case and assume that we can access elements of $\mathcal{K}$ with negative indexes.

$f_T(\mathcal{X}) = U^{(T)}[t,:] \cdot \mathcal{X}_2[:,y,x]$. In other word we have $\mathfrak{C}_\mathcal{K} = f_T \circ \mathfrak{C}_\mathcal{G} \circ f_S$.

Like for CP it is possible to use VBMF to find the ranks of the decomposition [18]. The rank R_{S-VBMF} is given by VBMF on the unfolding of size (S, TH_dW_d) and the one for R_{T-VBMF} by VBMF on the unfolding of size (T, SH_dW_d). Then we get the rank R_S and R_T by applying Eq. (1). As the new sequence has $R_S S + R_S R_T H_d W_d + R_T T$ parameters and require as much addition-multiplication with a rank small enough it leads to a reduction of the number of parameters and addition-multiplication.

3.4. Fine-tuning

Once the original layers have been replaced by their compressed version, we get a fully functional neural network that can be used to perform prediction and can be trained. As the decompositions are computed to minimize the reconstruction error of each tensor and not the global loss of accuracy, it is possible to train the decomposed network to improve the accuracy. (In most of the case this lead to almost recover the original accuracy.)

The fine-tuning was done with the Adam optimizer with $\beta = (0.9, 0.999)$ and a weigh-decay of 0. For a starting learning rate μ we perform 5 epoch at μ, 5 at $\mu/10$ and 5 at $\mu/100$. The learning rate depends on the decomposition, we used 10^{-7} for CP and 10^{-6} for Tucker-2.

4. ADVERSARIAL ATTACKS AND ROBUSTNESS

4.1. Adversarial attacks

An adversarial example can be represented as follows: given an original input x, x_{adv} is an adversarial example of x if $x_{\text{adv}} = x + \delta$ (δ is a small perturbation) such that the prediction for x_{adv} by a given model differs from the prediction for x.

Among all the different attacks we will use white-box non-target attacks (the attacker has full reading access to the network) that are gradient based. Those attacks are very efficient and can mislead a normal network with very small perturbations.

We used three different attacks to check the robustness of the network : the Fast Gradient Sign Method (FGSM) [2], the Projected Gradient Descent (PGD) [21] and the Carlini and Wagner (CW) attack [22].

4.2. Adversarial training

This sensitivity to adversarial perturbation requires to introduce methods to prevent this effect. Until now, two methods exist to harden neural networks against adversarial perturbations. The first one is regularization methods that penalize noise expansion throughout the network [23]. The second method, the most popular, is the adversarial training [2, 21] which consists of using adversarial examples generated from the training data set to increase robustness locally around the training samples. In this paper, we use this method to create a robust model.

Adversarial learning consists of learning on the adversarial examples computed through the adversarial perturbation. This learning procedure is typically presented as a robust min-max optimization problem, given by Eq. (3).

$$\theta^*_{\text{adv}} = \underset{\theta\in\Theta}{\operatorname{argmin}} \frac{1}{N} \sum_{(x^{(i)},y^{(i)})\in\mathcal{D}} \max_{\|\delta\|\leq\epsilon} \mathcal{L}(x^{(i)}+\delta, y^{(i)}; \theta) \quad (3)$$

where $\mathcal{L}(x, y; \theta)$ is the loss function for each example (x, y) of the data set $\mathcal{D}$ containing N examples.

The training is usually processed using an optimization algorithm based on gradient descent on mini-batches. It is important to note that at each iteration of the optimization process, the neural network parameters are updated, and it is necessary to compute the adversarial perturbations with respect to these new parameters at each iteration. This step represents an additional computation time compared to classical learning.

5. EXPERIMENTS

In this section, we investigate the robustness of tensor-decomposition based compression by studying the effect of diverse adversarial attacks on different architectures with different robustness properties. All the experiments were conducted on the challenging ImageNet dataset. The models we investigate are provided by [24]. They consist in various robustness levels $\varepsilon_{\text{train}}$ of the architectures ResNet18, ResNet50, WideResNet-50-2 and WideResNet-50-4, pretained on ImageNet.

The robustness levels we considered for these models and for the norm 2 constraint are $\varepsilon_{\text{train}} = 0$ (non robust model), $\varepsilon_{\text{train}} = 0.1$ (slightly robust model), $\varepsilon_{\text{train}} = 1$ and $\varepsilon_{\text{train}} = 5$ (robust models). We also investigated robustified models WideResNet-50-2 with respect to the infinite norm for $\varepsilon_{\text{train}} = 0.5$, $\varepsilon_{\text{train}} = 2$ and $\varepsilon_{\text{train}} = 8$.

Table 1 reports the results obtained for the well-known ResNet50 architecture, for a PGD adversarial attack. Similar tables for the other architectures and FGSM adversarial attack have been obtained with a qualitative consistency across the results. They are shared in the GitHub repository (see footnote1).

In the first part of this section we compressed models with different robustness with the standard procedure as described in section 3. After observing that the standard fine-tuning procedure suppress the robustness of largely robust method, we showed in the second part, that this could be overcome by using a small learning rate. In the third part, we compared with a baseline consisting in adversarial fine-tuning.

Table 1. Top 1 accuracy under PGD adversarial attacks for models of Resnet-50 architecture with different robustness degrees (parameterized by $\varepsilon_{\text{train}}$). We indicate the compression ratio (of the number of parameters) of each compressed model and the nature of the fine-tuning (F.T.) we used.

				Top 1 Accuracy (%)							
				Attack strength ε							
$\varepsilon_{\text{train}}$	**Model**	**Comp. ratio**	**F.T.**	0.001	0.0025	0.005	0.0075	0.01	0.03	0.1	1
0	**Original**			63.7	47.1	25.4	10.6	4.8	0.2	0.1	0.0
	CP	1.554	∅	58.3	44.9	25.2	11.9	5.7	0.2	0.1	0.0
			Standard	66.3	53.9	33.3	16.8	8.3	0.4	0.1	0.0
	Tucker2	1.433	∅	55.5	38.9	19.4	8.2	3.7	0.1	0.0	0.0
			Standard	64.7	48.6	26.5	12.2	6.4	0.2	0.0	0.0
0.1	**Original**			72.6	69.2	62.7	55.5	48.3	11.9	5.5	0.1
	CP	1.556	∅	68.2	64.8	58.5	51.3	44.5	11.4	5.6	0.1
			Standard	71.7	67.7	60.0	52.1	43.9	9.0	4.2	0.0
	Tucker2	1.430	∅	62.1	58.0	51.4	44.2	37.3	8.0	3.6	0.0
			Standard	71.0	66.2	57.2	47.7	38.3	6.0	2.7	0.0
5	**Original**			55.7	55.3	54.6	53.8	53.1	47.1	42.3	7.5
	CP	1.505	∅	44.2	43.8	43.0	42.2	41.4	35.5	31.0	3.6
			Standard	55.1	54.4	53.1	51.8	50.4	39.3	32.1	2.3
	Tucker2	1.505	∅	31.9	31.4	30.7	29.9	29.1	23.2	19.1	1.4
			Standard	48.9	48.2	47.1	46.1	44.9	36.7	30.8	3.1

5.1. Standard procedure

We begin by implementing the standard procedure for a tensor decomposition based compression of neural networks. It consists in choosing an appropriate VBMF ratio α for each architecture. The parameters chosen are $\alpha_{\text{CP}} = 0.8, \alpha_{\text{Tucker2}} = 0.8$ for ResNet18, $\alpha_{\text{CP}} = 0.9, \alpha_{\text{Tucker2}} = 0.8$ for ResNet50 , $\alpha_{\text{CP}} = 0.8, \alpha_{\text{Tucker2}} = 0.8$ for WideResNet-50-2, $\alpha_{\text{CP}} = 0.8, \alpha_{\text{Tucker2}} = 0.9$ for WideResNet-50-4. On each of these compressed models, we perform the standard fine-tuning that we described in the previous sections. This procedure allows us to investigate the robustness of the compressed models.

We perform the FGSM, PGD attacks on these different models and CW attack on the very robust models. We observe in all the architectures that the robustness is well conserved (and even improved if the model is not robust).

5.2. Importance of a small learning rate

For robust models, when the learning rate is adjusted based on accuracy improvement, the robustness of the models is compromised. This could intuitively be understood by observing that largely robust models have significantly less accuracy than their non robust counterparts (for example, Resnet50 of $\varepsilon_{\text{train}} = 0$ has a top 1 accuracy of 75.8 on ImageNet while the robust version with $\varepsilon_{\text{train}} = 5$ has a top 1 accuracy of only 56.13, as reported in [24]). This implies that performing standard fine-tuning after compression of a largely robust model will very likely converges to models with better accuracies and thus loosing of robustness preserved by the tensor decomposition of the layers.

To retain this robustness, we investigated the use of small learning rates. We aim to examine whether there exists a minimum nearby our compressed model that can recover the robustness of the original model. We performed 30 epochs with a starting learning rate of 10^{-6} for Tucker2 and 10^{-7} for CP.

Figure 1 illustrate the difference between a large and a small learning rate for the architecture WideResNet50-2 with a robustness $\varepsilon_{\text{train}} = 5$ with the constraint of norm 2. We clearly observe that the compressed model obtained through large learning rate fine-tuning has a better accuracy for small attacks but worst robustness than the original model (as it converged to a more accurate but less robust model). For the compressed model obtained through small learning rate fine-tuning, the robustness and accuracy are in bulk conserved (with a slight decrease due to the compression).

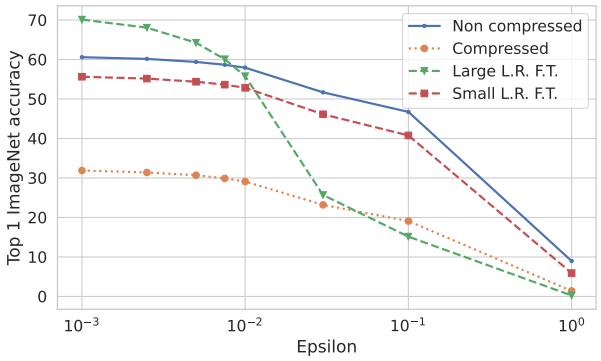

Fig. 1. Comparison of the top 1 accuracy on ImageNet for a robust WideResnet50-2 model (with $\varepsilon_{\text{train}} = 5$ for a nomn 2 constraint) between the original version, the compressed version with small learning rate fine-tuning (10^{-6}) and the compressed model obtained with large learning rate fine-tuning (10^{-4}).

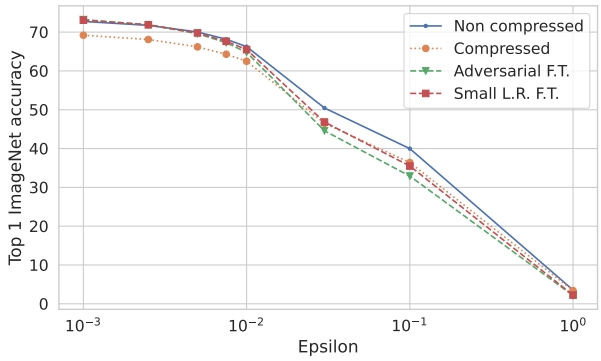

Fig. 2. Comparison of the top 1 accuracy on ImageNet for a robust WideResnet50-2 model (with $\varepsilon_{\text{train}} = 1$) between the original version, the compressed version, without fine-tuning, with small learning rate fine-tuning (10^{-6}) and with adversarial fine-tuning.

5.3. Comparison with adversarial fine-tuning

As a baseline comparison, we investigated the replacement of the standard fine-tuning by an adversarial fine-tuning. It is the existent method in the literature proposed by Gui *et al.* [16] (if we disregard the sparsity and quantization constraints). For this, we used the robustness package [25]. We performed 30 epochs with a decrease of the learning rate every 10 epochs by a factor 10. $\varepsilon_{\text{train}}$ is the same as the robustness degree of the original model unless precised otherwise. We used multiple learning rates each time and reported the result of the empirically optimal ones. The remaining parameters are the default ones in the robustness package.

We illustrate the similarity between the standard and the adversarial procedure by considering the architecture WideResNet50-2 with a robustness $\varepsilon_{\text{train}} = 1$ with the constraint of norm 2. In Figure 2, we plot the top 1 accuracy on ImageNet for the original model, the accuracy of the compressed model (using CP decomposition) without fine-tuning as well as the accuracy of the compressed model after fine-tuning with the small learning rate, and a compressed model obtained by replacing the standard fine-tuning by adversarial fine-tuning. Similar results are obtained for different architectures and robustness levels.

6. CONCLUSION

We provided strong evidence that compression by tensor decomposition is able to preserve the robustness of the original model, a crucial property for the security, reliability, and ethical integrity of AI systems in various domains. To maintain high robustness in models, it is essential to employ a small learning rate during fine-tuning, even though larger learning rates yield higher accuracy. The achieved robustness is comparable to that obtained by using adversarial fine-tuning, yet it does not necessitate expertise in defense methods. We believe it is an important step forward for a wider use of compression by decomposition.

We focused our experiments on image classification, but it would be interesting to conduct similar experimentation and determine if these conclusions generalize to other domains. As tensor decomposition leads to compressed but valid neural networks, it is possible to use tensor decomposition as a first step before performing pruning or quantization. Even if separated results on pruning and quantization on one side and our results on the other side suggest that combining different techniques should not cause robustness issues, it would be important to properly confirm it.

7. ACKNOWLEDGEMENTS

This publication was made possible by the use of the FactoryIA supercomputer, financially supported by the Ile-de-France Regional Council. This work is also supported by the PEPR-IA : ANR-23-PEIA-0010 and DeepGreen : ANR-23-DEGR-0001.

8. REFERENCES

[1] Jian Cheng, Pei-song Wang, Gang Li, Qing-hao Hu, and Han-qing Lu, "Recent advances in efficient computation of deep convolutional neural networks," *Frontiers of Information Technology & Electronic Engineering*, vol. 19, pp. 64–77, 2018.

[2] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy, "Explaining and harnessing adversarial examples," *arXiv preprint arXiv:1412.6572*, 2014.

[3] Pekka Ala-Pietilä, Yann Bonnet, Urs Bergmann, Maria Bielikova, Cecilia Bonefeld-Dahl, Wilhelm Bauer, Loubna Bouarfa, Raja Chatila, Mark Coeckelbergh, Virginia Dignum, et al., *The assessment list for trustworthy artificial intelligence (ALTAI)*, European Commission, 2020.

[4] Arie Wahyu Wijayanto, Jun Jin Choong, Kaushalya Madhawa, and Tsuyoshi Murata, "Towards robust compressed convolutional neural networks," in *2019 IEEE International Conference on Big Data and Smart Computing (BigComp)*. IEEE, 2019, pp. 1–8.

[5] Yuwang Ji and Qiang Wang, "Fast cp-compression layer: Tensor cp-decomposition to compress layers in deep learning," *IET Image Processing*, vol. 16, no. 9, pp. 2535–2543, 2022.

[6] Ye Liu and Michael K Ng, "Deep neural network compression by tucker decomposition with nonlinear response," *Knowledge-Based Systems*, vol. 241, pp. 108171, 2022.

[7] Artur Jordão and Hélio Pedrini, "On the Effect of Pruning on Adversarial Robustness," in *ICCV*, 2021.

[8] Rulin Shao, Jinfeng Yi, Pin-Yu Chen, and Cho-Jui Hsieh, "How and when adversarial robustness transfers in knowledge distillation?," *arXiv:2110.12072*, 2021.

[9] Heitor R. Guimarães, Arthur Pimentel, Anderson R. Avila, Mehdi Rezagholizadeh, Boxing Chen, and Tiago H. Falk, "Robustdistiller: Compressing Universal Speech Representations for Enhanced Environment Robustness," in *ICASSP*, June 2023, pp. 1–5.

[10] Yonggan Fu, Qixuan Yu, Meng Li, Vikas Chandra, and Yingyan Lin, "Double-Win Quant: Aggressively Winning Robustness of Quantized Deep Neural Networks via Random Precision Training and Inference," in *ICML*, July 2021, pp. 3492–3504.

[11] Yiren Zhao, Ilia Shumailov, Robert Mullins, and Ross Anderson, "To compress or not to compress: Understanding the Interactions between Adversarial Attacks and Neural Network Compression," Apr. 2020.

[12] Shaokai Ye, Kaidi Xu, Sijia Liu, Hao Cheng, Jan-Henrik Lambrechts, Huan Zhang, Aojun Zhou, Kaisheng Ma, Yanzhi Wang, and Xue Lin, "Adversarial Robustness vs. Model Compression, or Both?," in *ICCV*, 2019.

[13] Adrian Bulat, Jean Kossaifi, Sourav Bhattacharya, Yannis Panagakis, Timothy Hospedales, Georgios Tzimiropoulos, Nicholas D Lane, and Maja Pantic, "Defensive tensorization," *arXiv:2110.13859*, 2021.

[14] Arinbjörn Kolbeinsson, Jean Kossaifi, Yannis Panagakis, Adrian Bulat, Animashree Anandkumar, Ioanna Tzoulaki, and Paul M. Matthews, "Tensor dropout for robust learning," *IEEE Journal of Selected Topics in Signal Processing*, vol. 15, no. 3, pp. 630–640, 2021.

[15] Seungju Cho, Tae Joon Jun, Mingu Kang, and Daeyoung Kim, "Applying Tensor Decomposition to image for Robustness against Adversarial Attack," Mar. 2020.

[16] Shupeng Gui, Haotao Wang, Haichuan Yang, Chen Yu, Zhangyang Wang, and Ji Liu, "Model Compression with Adversarial Robustness: A Unified Optimization Framework," in *NeurIPS*, 2019, vol. 32.

[17] M. Astrid, S.-I. Lee, and B.-S. Seo, "Rank selection of CP-decomposed convolutional layers with variational Bayesian matrix factorization," in *ICACT*, 2018.

[18] Yong-Deok Kim, Eunhyeok Park, Sungjoo Yoo, Taelim Choi, Lu Yang, and Dongjun Shin, "Compression of Deep Convolutional Neural Networks for Fast and Low Power Mobile Applications," Feb. 2016.

[19] Vadim Lebedev, Yaroslav Ganin, Maksim Rakhuba, Ivan Oseledets, and Victor Lempitsky, "Speeding-up convolutional neural networks using fine-tuned cp-decomposition," *ICLR*, 2015.

[20] Jean Kossaifi, Yannis Panagakis, Anima Anandkumar, and Maja Pantic, "Tensorly: Tensor learning in python," *Journal of Machine Learning Research (JMLR)*, vol. 20, no. 26, 2019.

[21] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu, "Towards deep learning models resistant to adversarial attacks," *ICLR*, 2018.

[22] Nicholas Carlini and David Wagner, "Towards Evaluating the Robustness of Neural Networks," Mar. 2017.

[23] Daniel Jakubovitz and Raja Giryes, "Improving dnn robustness to adversarial attacks using jacobian regularization," in *ECCV*, 2018, pp. 514–529.

[24] Hadi Salman, Andrew Ilyas, Logan Engstrom, Ashish Kapoor, and Aleksander Madry, "Do adversarially robust imagenet models transfer better?," *NeurIPS*, vol. 33, pp. 3533–3545, 2020.

[25] Logan Engstrom, Andrew Ilyas, Hadi Salman, Shibani Santurkar, and Dimitris Tsipras, "Robustness (python library)," 2019.