Bulletin of the *Transilvania* University of Braşov • Vol. 13 (62) No. 2 - 2020
Series I: Engineering Sciences
https://doi.org/10.31926/but.ens.2020.13.62.1.10

COMPRESSION AND OPTIMIZATION OF NEURAL NETWORKS BY TERNARIZATION METHODS

A.C. TĂNASĂ[1] **D. TĂNASĂ**[1]

Abstract: *Current deep neural networks are achieving higher and higher performances, but this comes to a great computational cost which creates a difficulty to use them on embedded platforms like smartphones, tables, autonomous cars. Fortunately, this can be addressed using quantization techniques, one of which is the ternarization when only 3 bits are used.*

Key words: *neural networks, ternarization, threshold, sparsity.*

1. Introduction

Deep neural networks are becoming the alternative approach to many well-known applications in the field of machine learning. However, as networks get deeper, it becomes increasingly difficult to integrate a network with a large number of parameters on an embedded device of reduced size and performance.

This status quo represents an aspect of interest through the work done by many institutions and research teams. From using the half-precision standard [1] to aggressively use quantized weights [8], activations [9] and gradients [2] from 8 to 2 bits, researchers and engineers try to further reduce the number and difficulty of calculations during training. Although a binary neural network can be even 32x smaller than a regular model, this extreme compression rate comes along with a sharp drop in accuracy. Courbariaux [2] and Zhu [3] proposed ternary neural networks as a fair trade-off between the accuracy and size of architecture.

2. Previous Work

The research for this paper followed the Trained Ternary Quantization method [11] using two scaling coefficients Wlp, Wln for each layer l and quantized learnable weights (or parameters) {- Wln, 0, Wlp } instead of the traditional method {-1, 0, 1} or {-E, 0, E} where E represents the absolute average value of the layer, which is not learnable. Also, weights are both used ternarized and floating-point during training, but only the ternarized version is used during testing/validation. All of this makes it possible to adjust which of the three values {- Wln, 0, Wlp } is assigned to a weight.

[1] Dept. of Electronics and Computers, *Transilvania* University of Braşov, Romania.

Their paper argues that their ternarized methods achieve higher accuracy on the CIFAR-10 [11] with AlexNet architecture than the full-precision version. Within the current paper, we wanted to study if their claims also apply to other architectures: ResNet architectures [3] and also we wanted to study in detail the benefit achieved for the hardware and time consumption.

3. Methods

The studied ternarization algorithm is supposed to be a flexible algorithm, applicable to each architecture, regardless of deep neural network type (convolutional, generative, residual etc.). As it is described in Figure 1, the input of any neural network during the training process consists of two types of data: RGB/grayscale images and a label for each image, in this case, a class label. The learning process is supervised and once the number of iterations is chosen, the quality of the network is evaluated through learning: we provide an image at the entrance and we expect the prediction to be as close to the image class as possible.

The ternarization algorithm is used as an additional step in the training process just before the gradient is calculated and the weights are optimized.

The input to the compression algorithm consists in full-precision weights and a coefficient or threshold (chosen before training). The output consists in the ternary weights corresponding to each layer in the network. Because it is used as an extra step in the learning process, the method has the advantage of not being dependent on databases or architectures which creates the flexibility advantage.

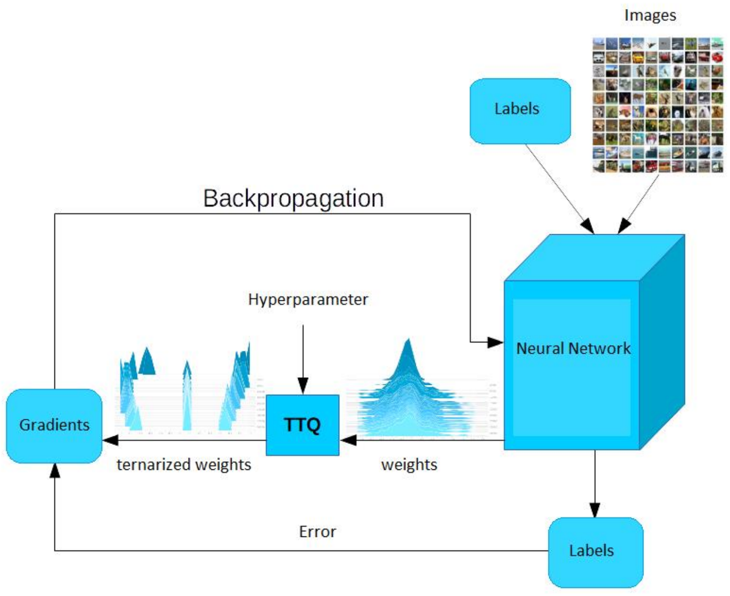

Fig. 1. *The environment outside the method*

The process begins with initialising the full-precision weights with a pseudo-random distribution as it can be seen in Figure 3. Then, they need to be normalized in the range [-1, 1] by dividing each weight by the largest value. Once the distribution is within a known interval, the intermediate values which were previously calculated based on either a threshold or a sparsity coefficient (chosen before training) are quantized. Everything which is less than the value -threshold selected becomes -1, any value in the range (-threshold, threshold) becomes 0 and any value greater than the threshold becomes 1. The threshold or sparsity coefficient is the same regardless of the layer to reduce the search space. Limiting the algorithm to these ternary weights it is observed that the learning process is very troublesome because the weights are not allowed to change as they need to, causing large losses in accuracy. The other algorithm uses two floating-point values that are multiplied with the ternary values, the weights, can be changed when training. In essence, the values in the set {-1, 0, 1}, are expressed in the set {-Wn, 0, Wp} which are the final ternary values. This change, although it involves the introduction of 2 coefficients in floating point for each layer, makes the accuracy similar to that of the original network, yet giving us a compression rate of approximately 16 times.

The algorithms vary depending on the type of ternarization: based on a threshold (TW TTQ) as seen in upside of Figure 2 or based on a sparsity coefficient (P TTQ) as seen in downside of Figure 2. Both of them follow the same steps: normalization, quantization, multiplication with ternarization factors.

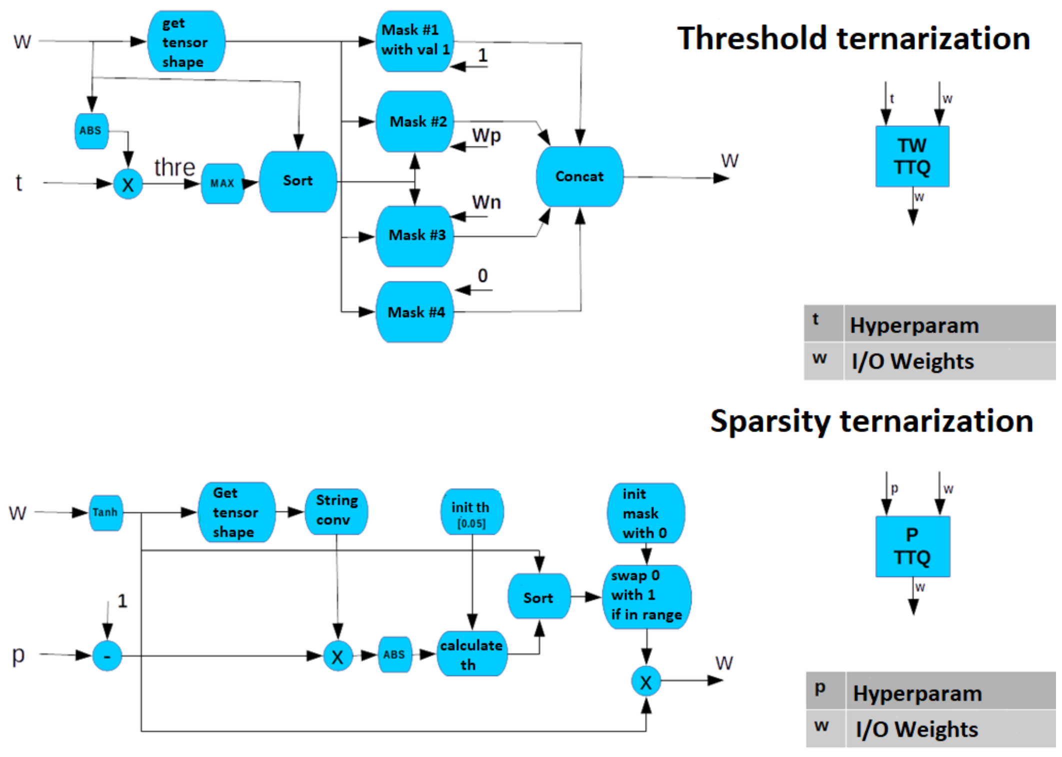

Fig. 2. *Ternarization algorithms*

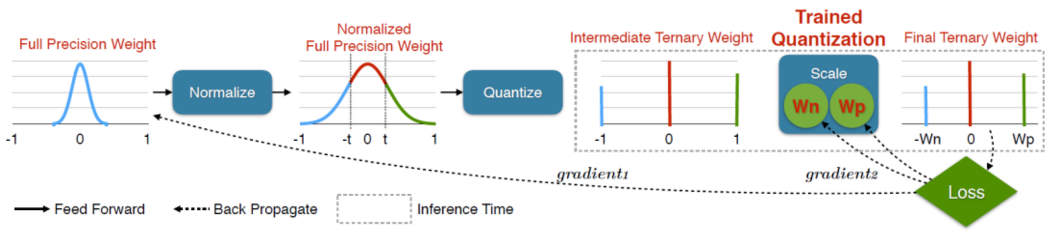

Fig. 3. *Processing ternarization algorithm. Image taken from the article* [10]

4. Experiments

The experiments were based on the Cifar10 database and on multiple residual networks [5] due to the substantial impact in the community of machine learning. Success is based on an observation made by the authors that the gradients vanish from the last layer to the first one [12], so they implemented skipped-connections between layers. This is an undesirable effect during training and is one of the main reasons why deep networks are reluctant. As the network gets deeper the harder the weights will change compared to the initialization, which leads to the conclusion that the network does not learn the basic features. In the residual networks, the gradient propagates conveniently and makes it possible to train deeper networks with hundreds of convolutional layers.

The architectures used for the experiments are the second generation of residual neural networks [4]. The CIFAR-10 dataset is a collection of images containing 60,000 32x32 color images in 10 different classes (airplanes, cars, birds, cats, deer, dogs, frogs, horses, ships, and trucks). Each class is represented by 6,000 samples.

In all our experiments the same hyper-parameters (learning rate, batch size, number of epochs and iterations, optimizer) were maintained for a proper comparison. The learning rate was programmed in the form of intervals that can be expressed both by the number of epochs or iterations. An iteration represents the inference over a subset of images and an epoch represents the inference over the entire data set (several epochs, implicitly representing the inference over the data set several times). After each iteration, the optimizer starts the back-propagation process where all the weights are modified to minimize the error between the model prediction and the label. The optimizer chosen was the Momentum Optimizer [7], an extension of the Stochastic Gradient Descent Optimizer. For integrity, the results were followed both during training and validation process.

In the training process, we followed the ability of the architecture to learn the dataset features. The validation process followed the ability of the architecture to generalize and classify new images using the learnt features.

The original architecture with weights expressed in floating-point on 32b (shortened Original), the ternarized architecture based on a sparsity coefficient (abbreviated P TTQ) and the ternarized architecture based on a threshold (abbreviated T TTQ) are presented in the following tables.

All architectures were trained from scratch, with the same hyper-parameters, on the same hardware. The chosen sparsity coefficient is 0.5, which means that 50% of the weights should be found in 0, and the ternarization threshold chosen is 0.05.

Table 1 shows the classification error obtained after training. Based on the value, the model performances to classify and generalize can be tracked. The error in training and validation stages need to be as small as possible in order not to encounter the phenomenon of overfitting, which means that the model is not able to generalize.

The error obtained on residual architectures on the CIFAR10 data set Table 1

Type	**Arhitecture**	**ResNET18**	**ResNET30**	**ResNET54**	**ResNET102**
Training	Original	0.00297	0.00089	0.000554	7.4E-05
	P TTQ	0.0843	0.07651	0.07499	0.07704
	T TTQ	0.04771	0.01371	0.00168	0.0037175
Validation	Original	0.0826	0.0727	0.0653	0.0626
	P TTQ	0.138	0.1372	0.1366	0.1366
	T TTQ	0.1127	0.0951	0.0823	0.0888

We can express the accuracy, which is the number of correct predictions over the entire datatset, as a percentage based on the classification error. This is observable in Table 2.

The accuracy obtained on residual architectures on the CIFAR10 dataset Table 2

Architecture	**Validation**			**Training**		
	Original	**P TTQ**	**T TTQ**	**Original**	**P TTQ**	**T TTQ**
ResNET18	91.74%	86.20%	88.73%	99.70%	91.57%	95.23%
ResNET30	92.73%	86.28%	90.49%	99.91%	92.35%	98.63%
ResNET54	93.47%	86.34%	91.77%	99.94%	92.50%	99.83%
ResNET102	93.74%	86.34%	91.12%	99.99%	92.30%	99.63%

For this set of experiments, the original model has better accuracy, but this aspect could be improved by optimizing the hyper-parameters for PTTQ and TTTQ with different values than the original.

The differences show that the ternarized models with the threshold method converge better and have higher accuracy than the ternarized ones using the sparsity coefficient. In the case of threshold ternarization, the accuracy does not differ by more than 3% in the case of validation.

Figures 4 and 5, emphasize multiple conclusions:

- there are similarities between training and validation stages to all the 12 architectures;
- there is a tendency of training depending on the method of ternarization chosen;
- the original model converges faster;
- ternarized networks with a threshold (also known as the T TTQ attribute) tend to converge faster, but the slope is smoother compared to the non-ternarized networks;
- ternarized networks with sparsity coefficient tend to converge slower and more poorly.

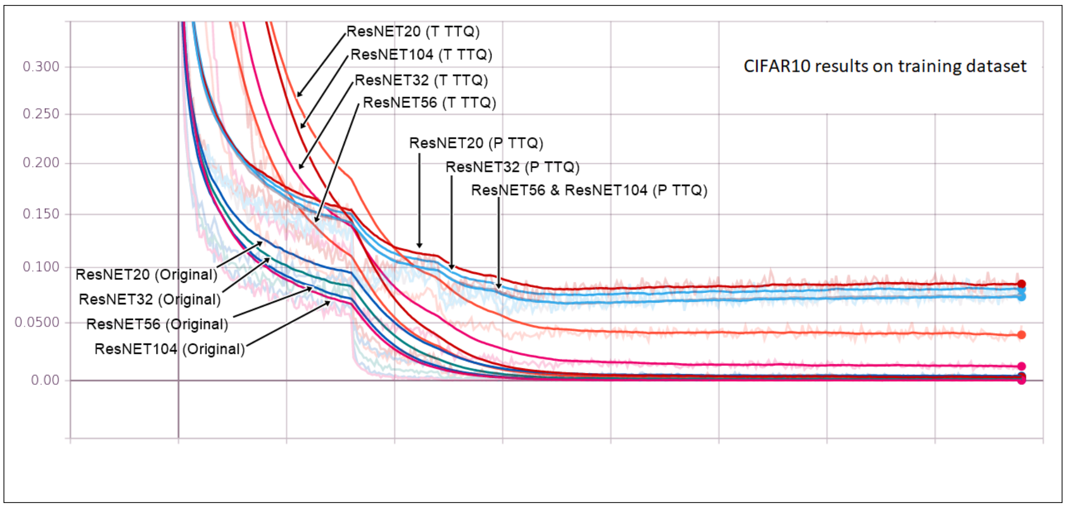

Fig. 4. *Convergence trends of architectures in training*

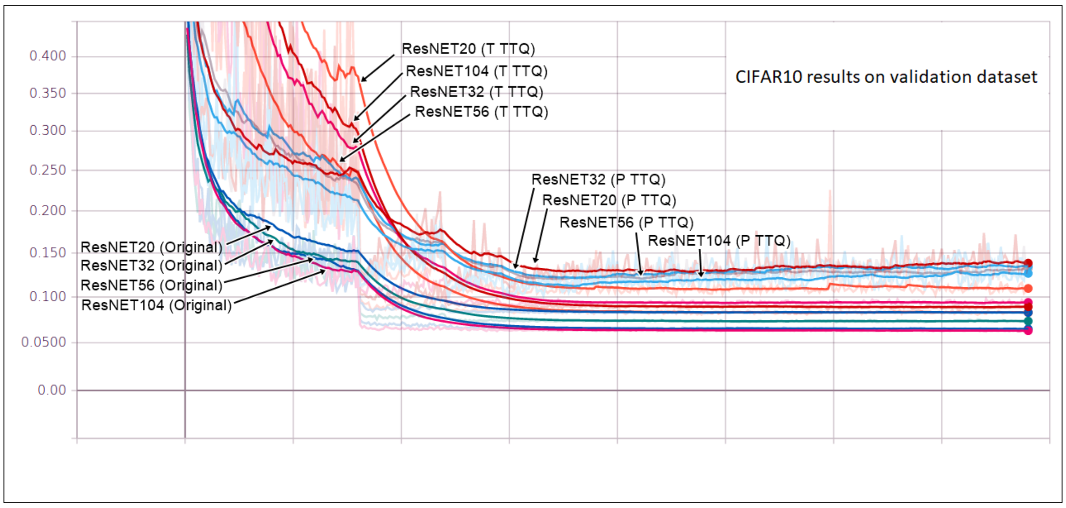

Fig. 5. *The convergence trends of architectures for validation*

Another important feature is network compression. The outstanding compression rates are presented in Table 3.

Memory compression Table 3

Architecture	Number of parameters	Number of layers	Memory in full – precision [MB]	Compressed memory [MB]	Compression rate
ResNET18	269722	59	1.078888	0.0681385	15.83
ResNET30	464154	95	1.856616	0.1171785	15.84
ResNET54	853018	167	3.412072	0.2152585	15.85
ResNET102	1630746	311	6.522984	0.4114185	15.85

As last metric, an interpretative analysis of the method was used by visual comparison of the histograms from the input and output of the ternarization method. In Figure 6, it is shown in blue the weight distribution before the ternarization method and after with a sparsity coefficient of 50%. It was considered that the evaluation of a single layer (the first convolutional layer from the first residual block) would be sufficient to avoid the redundancy caused by the pseudo-random initialization of the weights on each layer.

In this three-dimensional histogram, the Ox axis represents the value of the weights (in floating-point), the Oy axis represents the number of weights of that value, and the Oz axis represents the number of the iteration (the front one representing the newest iteration).

These distributions were chosen because in Figure 6 the distribution appears to be Gaussian and the values were distributed according to the requirement given by the sparsity coefficient. Visually, one can conclude that over 50% of the weights are centered in 0. In Figure 6 the orange distribution seems much narrower and by choosing the 0.05 threshold we were able to divide the distribution into three distinct areas, highlighting the performance of the method. After the experiments performed on the CIFAR10 database, it is observed that the ternarization method presents both advantages and disadvantages. The degradation in accuracy of the architectures can also be seen as unfinished training because once we introduced the ternarization algorithm no changes were made to hyperparameters that have a direct implication in the evaluation metric. The compression is particularly good and the rate is the same as the one presented in the article.

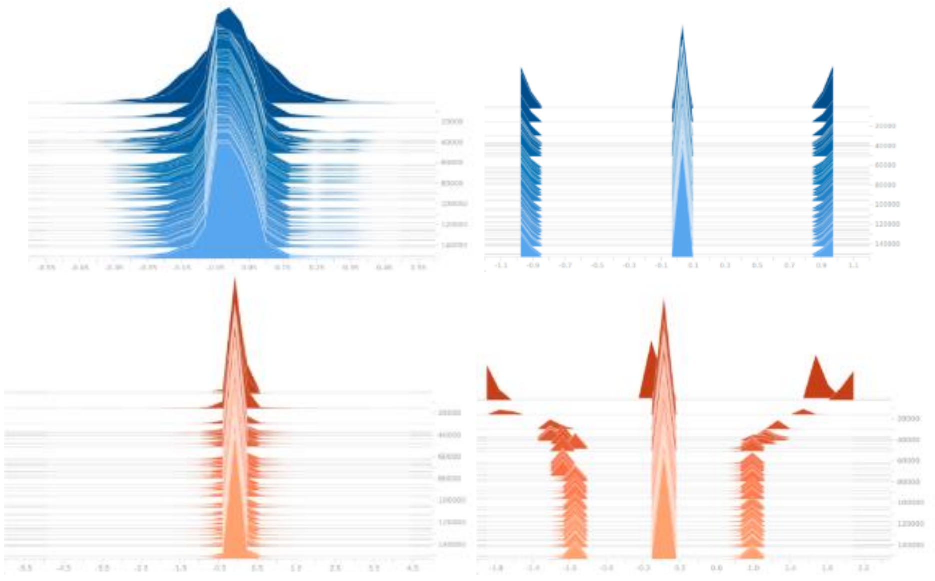

Fig. 6. *Distribution of weights before (left blue) and after (right blue) the ternarization method according to the sparsity coefficient and the distribution of weights before (left orange) and after (right orange) the ternarization method according to the threshold*

5. Conclusions

Starting from the need given by the increasingly complex systems and their integration into progressively superior, iterative products, computer vision reaches a peak of technique, algorithms and hardware. The last decade has been marked by the transition to such solutions, related machine learning, which aim to solve a high number of previously unresolvable problems. The necessity of the product on the market attracts progressively more restrictive, more spectacular, more "friendly" specifications with the hardware field. The mobile and automotive domain attracts the need for low power consumption, the application domain raises the problem of performance, the embedded domain requires small and achievable methods. We can consider, at the end of this study, that ternarization can be a response to the requests of the domains.

As detailed in this paper, the applicability, compression performance and classification performance were tracked. From all these points of view, the TTQ method can be considered a worthwhile solution to the deployment, compression and classification performance of deep learning architectures.

References

1. Amodei, D., Anubhai, R., et al.: *Deep Speech 2: End-to-end Speech Recognition in English and Mandarin*. In: arXiv preprint arXiv: 1512.02595, 2015.
2. Courbariaux, M.: *Binarized Neural Networks: Training Deep Neural Networks with Weights and Activations Constrained to +1 or -1*. In: arXiv preprint arXiv: 1602.02830, 2016.
3. Fengfu, L., Bin, L.: *Ternary Weight Networks*. In: arXiv preprint arXiv: 1605.04711, 2016.
4. He, K., Zhang, X., et al.: *Identity Mappings in Deep Residual Networks*. In: arXiv preprint arXiv: 1603.05027, 2016.
5. He, K., Zhang, X., et al.: *Deep Residual Learning for Image Recognition*. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2016, p. 770-778.
6. Itay, H., Courbariaux, M., et al.: *Quantized Neural Networks: Training Neural Networks with Low Precision Weights and Activations*. In: arXiv preprint arXiv: 1609.07061, 2016.
7. Qian, N.: *On the Momentum Term in Gradient Descent Learning Algorithms*. In: Neural Networks: The Official Journal of the International Neural Network Society **12**(**1**) (1999), p. 145-151.
8. Rastegari, M., Ordonez, V., et al.: *Xnor-net: Imagenet Classification Using Binary Convolutional Neural Networks*. In: arXiv preprint arXiv: 1603.05279, 2016.
9. Shuchang, Z., Zekun, N., et al.: *Dorefa-net: Training Low Bitwidth Convolutional Neural Networks With Low Bitwidth Gradients*. In: arXiv preprint arXiv: 1606.06160, 2016.
10. Zhu, C., Song, H., et al.: *Trained Ternary Quantization*. In: arXiv preprint arXiv: 1612.01064, 2016.
11. https://en.wikipedia.org/wiki/Vanishing_gradient_problem. Accessed: 15.01.2019.
12. https://en.wikipedia.org/wiki/CIFAR-10. Accessed: 08.02.2019.