# Ternary Residual Networks

Extended Abstract

Abhisek Kundu, Kunal Banerjee, Naveen Mellempudi, Dheevatsa Mudigere, Dipankar Das,
Bharat Kaul, Pradeep Dubey*
{abhisek.kundu,kunal.banerjee,naveen.k.mellempudi,dheevatsa.mudigere,dipankar.das,bharat.kaul,pradeep.dubey}
@intel.com

## ABSTRACT

Sub-8-bit representation of DNNs incur some discernible loss of accuracy despite rigorous (re)training at low-precision. Such loss of accuracy essentially makes them equivalent to much shallower counterparts, diminishing the power of being deep networks. To address this problem of accuracy drop, we introduce the notion of *residual edges*, where we add more low-precision edges to sensitive branches of the sub-8-bit network to compensate for the lost accuracy (loss $\sim$ 1% of FP32 baseline). We present a perturbation theory to identify such sensitive edges. Moreover, for varying accuracy requirements in a dynamic environment, the deployed model can be upgraded/downgraded on-the-fly by partially enabling/disabling residual edges. Finally, all the ternary edges are sparse in nature, and the ternary residual conversion can be done in a resource-constraint setting with no low-precision (re)training. Experiments are shown on ResNet-101, GoogLeNet v3, and AlexNet pre-trained on ImageNet.

## 1 INTRODUCTION

There is a growing need to make deep neural networks (DNNs) resource-friendly via a compact and/or reduced-precision representation for both mobile devices and data servers. One approach is to reduce the number of parameters in the network (e.g., SqueezeNet [12], MobileNet [10], SEP-Nets [16]) to have a very small size ($\sim$ 5 MB) typically targeting mobile devices. However, it is not surprising that their overall accuracy is limited on complex dataset ImageNet [4], e.g., SEP-Net Top-1 accuracy is $\sim$ 10% off from that of ResNet-50. Deploying them on sensitive applications, e.g., autonomous cars and robots, might be impractical. The other approach is concerned with the reduction in size of parameter representation. Well-known methods of this kind are pruning [5, 23, 26], quantization [6, 11, 17, 19, 22, 27], binarization [3], ternarization [1, 8, 15, 18, 29], hashing [2], Huffman coding [7] and others [20, 28]. Such low-precision representation demands for specialized low-precision arithmetic [18, 24, 25] and hardware design. Google's TPU [13] sets a trend for a low-precision inference pipeline (8-bit activations, 8-bit weights). Here we are mainly focused on the trade-off between sub-8-bit representation ([11, 18, 20, 28]) and accuracy of deeper networks, keeping an eye on the power-performance factors. In reality, sub-8-bit models for deep networks incur some noticeable drop in accuracy, despite rigorous low-precision (re)-training . This loss severely undermines the purpose of deploying a deep (sub-8-bit) network, as we may find an equivalent shallower 8-bit network. We seek to answer: *Can a sub-8-bit model achieve similar accuracy as FP-32 model with better model size and power-performance numbers comparing to 8-8*?

*Pradeep Dubey is at Intel Parallel Computing Labs, Santa Clara, USA. Other authors are at Intel Parallel Computing Labs, Bangalore, India.

• **Contributions**: (1) We introduce the notion of *residual edges*, where we add more low-precision edges to sensitive branches of a DNN, and this is guided by a perturbation theory, (2) Unlike existing sub-8-bit models, we show how to achieve extremely high accuracy ($\sim$ 1% off from FP32), (3) Existing models cannot be altered once they are deployed ('fixed-accuracy-fixed-power' mode). However, our model can be upgraded/downgraded on-the-fly by partially enabling/disabling some of the residual edges in a dynamic environment, depending on the varying accuracy/power requirements. For example, when autonomous cars or robots are in a less eventful environment where less number of objects are involved, the classification problem becomes considerably simpler (sufficient to distinguish among distinct objects, such as humans, dogs, vehicles, trees, etc. rather than discriminating among multiple breeds of dogs), and by disabling many edges we can downgrade the model in terms of compute, power, etc., yet maintain very high accuracy for those (less number of) classes. Drawing an analogy between human attention and precision, and also an analogy between stress due to attention and power consumption, it is natural for us to be selectively attentive to certain tasks that requires processing more information. Such upgrade/downgrade of low-precision DNNs mimics a more real-world scenario that other existing models are unable to imitate.

We focus on ternary 8-2 DNNs due to their computational benefits (minimal number of multiplications and very small size). For a vector $\mathbf{x} \in \mathbb{R}^n$, we denote the (element-wise) Frobenius norm as $\|\mathbf{x}\|_F = \sqrt{\sum_{i=1}^{n} |\mathbf{x}_i|^2}$ (can be extended to matrices/tensors).

## 2 PERTURBATION IN PRE-TRAINED DNN

A DNN is a composition of parametric functions $f_i$ where (locally) optimal values of parameters $\mathbf{W}^{(i)}$ are achieved via network training. We interpret quantization and/or sparsification as adding a noise to $\mathbf{W}^{(i)}$ to produce sub-optimal $\tilde{\mathbf{W}}^{(i)}$. We want to quantify the effect of $\tilde{\mathbf{W}}^{(i)}$ on the final outcome, e.g., classification scores (Top-1 accuracy). For this, let us assume that our DNN has $\ell$ layers and let $\mathbf{y} \in \mathbb{R}^d$ ($d$ is the number of classes) be the output vector of layer $\ell$ such that its $i$-th component $\mathbf{y}_i$ contains the classification score for $i$-th class, for a given input $\mathbf{x}$. Let $\hat{\mathbf{y}} \in \mathbb{R}^d$ denote the perturbed vector $\mathbf{y}$ due to added noise. Then, the largest component of $\mathbf{y}$ should remain the largest in $\hat{\mathbf{y}}$ for no loss of accuracy. Practically, we want to first derive an upper bound on $\|\mathbf{y} - \hat{\mathbf{y}}\|_F / \|\mathbf{y}\|_F$ in terms of layer-wise perturbation of the pre-trained network, and then we want to control such perturbations to keep the final accumulated noise small. Let the set of functions be $f_{dnn}$ = { Convolution with Batch Normalization, Matrix Multiplication, ReLU, Pooling }. Functions in $f_{dnn}$ can be linear or non-linear, parametric or non-parametric, convex or non-convex, smooth or non-smooth. DNN: $f = f_\ell f_{\ell-1} ... f_1 : \mathcal{D}_0 \rightarrow \mathcal{D}_\ell$, where each $f_i \in f_{dnn}$, $f_i : \mathcal{D}_{i-1} \rightarrow \mathcal{D}_i$ with parameters $\mathbf{W}^{(i)}$, and each $\mathcal{D}_i$ is an arbitrary metric space where $\|\cdot\|$ denotes a distance metric on set $\mathcal{D}_i$ (for simplicity, we focus on normed space only). For all $\mathbf{X}^{(0)} \in \mathcal{D}_0$, we define $\mathbf{X}^{(i)} \in \mathcal{D}_i$ and $\tilde{\mathbf{X}}^{(i)} \in \mathcal{D}_i$ as follows. For $i = 1, ..., \ell$, $\mathbf{X}^{(i)} = f_i(\mathbf{X}^{(i-1)}; \mathbf{W}^{(i)}), \tilde{\mathbf{X}}^{(i)} = f_i(\tilde{\mathbf{X}}^{(i-1)}; \tilde{\mathbf{W}}^{(i)})$, where $\tilde{\mathbf{X}}^{(i)}$ and $\tilde{\mathbf{W}}^{(i)}$ are perturbed versions of $\mathbf{X}^{(i)}$ and $\mathbf{W}^{(i)}$, respectively. We want to measure how the outcome of $f$ gets perturbed in presence of $\tilde{\mathbf{X}}^{(i)}$ and $\tilde{\mathbf{W}}^{(i)}$, i.e., $\|\mathbf{X}^{(i)} - \tilde{\mathbf{X}}^{(i)}\|/\|\mathbf{X}^{(i)}\|$. We note that input to a layer might be perturbed due to perturbation in earlier layers and/or perturbation in the present layer (e.g., activation quantization) before it is applied to the layer function. For this we use separate notations as follows. For $i$-th layer, let $\tilde{\mathbf{X}}^{(i-1)}$ denote the perturbed input, $\hat{\mathbf{X}}^{(i-1)}$ denote the perturbed activation, and $\tilde{\mathbf{W}}^{(i)}$ denote the perturbed weights. Let us first define the following relative perturbations. $\Delta_i = \|\mathbf{X}^{(i)} - \tilde{\mathbf{X}}^{(i)}\|_F / \|\mathbf{X}^{(i)}\|_F, \gamma_i = \|\tilde{\mathbf{X}}^{(i)} - \hat{\mathbf{X}}^{(i)}\|_F / \|\mathbf{X}^{(i)}\|_F, \varepsilon_i = \|\mathbf{W}^{(i)} - \tilde{\mathbf{W}}^{(i)}\|_F / \|\mathbf{W}^{(i)}\|_F$. We derive the following to bound the relative change in the output of a layer.

THEOREM 2.1. *Using the above notations, the relative change in output of i-th layer of DNN, can be bounded as*
$\Delta_i \le (\prod_{k=1}^{i} O(1+\varepsilon_k))\Delta_0 + \sum_{k=1}^{i}(\prod_{j=k+1 \le i}^{i} O(1+\varepsilon_j))\, O(\gamma_{k-1}) + \sum_{k=1}^{i}(\prod_{j=k+1 \le i}^{i}\ O(1+\varepsilon_j)) O(1+\gamma_{k-1}) O(\varepsilon_k)$.

The theorem can be proved using triangle inequality, recursion in $\Delta_i$, and with the assumption that locally optimal parameters are not orthogonal to the signal (inner product is not close to zero). The result suggests that at $i$-th layer of DNN, perturbations of parameters and activations of all the previous stages accumulate nonlinearly in a weighted manner (indicating higher sensitivity of earlier layers). We want this error to be small to keep the perturbed solution in a neighborhood of the local optima. Simplifying for small noise:

THEOREM 2.2. *Using the above notations, assuming* $\Delta_0 = 0$ *and* $\|\hat{\mathbf{X}}^{(i-1)}\|_F \le \tau_{i-1}\|\mathbf{X}^{(i-1)}\|_F$, *where* $\tau_{i-1} > 0$ *are constants, we derive the following for constants* $c_j > 0$.
$\Delta_i \le \sum_{k=1}^{i}(\prod_{j=k+1 \le i}^{i} c_j)(O(\gamma_{k-1}) + O(\varepsilon_k))$.

Keeping both $\gamma_i$ and $\varepsilon_i$ small implies overall small perturbation. Also, for earlier layers $\varepsilon_k$ should be kept much smaller than those in later layers to have an overall small perturbation. Our empirical evaluation on quantized DNNs corroborates this theory.

## 2.1 Low-Precision DNN

Constraining activations and weights to 8 bits appears to induce only small perturbation, resulting in typically $< 1\%$ loss in accuracy. This can be explained by the theoretical bounds presented above, where $\varepsilon_k$ and $\gamma_k$ are small. More challenging cases are sub-8-bit DNNs, e.g., 8-bit activations and ternary weights.

• **Ternary Conversion of Pre-trained DNN** : We convert FP32 weights $\mathbf{W}$ to ternary values $\{-\alpha, 0, +\alpha\}$, $\alpha \ge 0$, without re-training, via a simple threshold ($T > 0$) based approach similar to [15, 18]. Let $\hat{\mathbf{W}}$ denote a ternary weight, such that, $i$-th element $\hat{\mathbf{W}}_i = sign(\mathbf{W}_i)$, if $|\mathbf{W}_i| > T$, and 0 otherwise. The goal is to minimize element-wise error $E(\alpha, T) = \|\mathbf{W} - \alpha\hat{\mathbf{W}}\|_F^2$. For better accuracy, [18] introduced FGQ by dividing the weight tensors into disjoint sub-tensors of size $N$, and then ternarize such sub-tensors independently. Each sub-tensor requires only one multiplication during inference. Smaller $N$ leads to better accuracy but with more multiplications. $k$ disjoint ternary sub-tensors can represent $2k+1$ distinct values, i.e., model capacity increases linearly in number of such sub-tensors. Sub-8-bit representation of sensitive parameters may have a 'blurring' effect on later activations for very deep networks; consequently, extremely high accuracy results might be elusive in 8-2 model.

• **Ternary Residual Edges:** Neither all the sub-tensors are approximated well in ternary, nor are they equally sensitive. For a poorly approximated sub-tensor, we ternarize the residual sub-tensor (difference between the FP32 one and the ternary one) and accumulate the output this ternary residual sub-tensor during inference. We might need multiple such residuals for a sub-tensor if all the earlier residuals are, as a whole, not good enough. Let the $j$-th layer weight tensor $\mathbf{W}^{(j)}$ be partitioned into $k$ disjoint sub-tensors $\mathbf{W}^{(j)}_{(i)}$ ($\tilde{\mathbf{W}}^{(j)}_{(i)}$ being the perturbed version), $i = 1, ..., k$. Then, sensitivity $\varepsilon^{(j)}_i$ of $i$-th sub-tensor in $j$-th layer is defined as follows. $\varepsilon^{(j)}_i = \|\mathbf{W}^{(j)}_{(i)} - \tilde{\mathbf{W}}^{(j)}_{(i)}\|_F / \|\mathbf{W}^{(j)}\|_F$. Note that, $\sum_{i=1}^{k}(\varepsilon^{(j)}_i)^2 = \varepsilon_j^2$ (defined earlier in Section 2. This suggests, for a given perturbation of weights of a layer, various sub-tensors may require different number of residuals. Let, $i$-th sub-tensor requires $r_i$ number of residuals. Then, there is total $\sum_{i=1}^{k}(r_i+1)$ multiplications, model capacity is $\sum_i 3^{(r_i+1)} - k + 1$, and model size (in bits) is $(8 + \frac{2n}{k})\sum_{i=1}^{k}(r_i+1)$, for $k$ number of sub-tensors and tensor size $n$.

• **Power-Performance Estimate:** Let $X$ be the power-performance gain for ternary 8-2 operations over 8-8. Then, power-performance for residual method with $C\times$ compute using FGQ sub-tensor size $N$ can be shown as $X/(C(X/N+1))$.

**Table 1: Ternary Residual Networks using FGQ. # sub-tensors and compute for FGQ ternary is 1×. Loss is w.r.t FP32.**

| $N = 64$ | Loss ∼ 1% | | Loss ∼ 2% | |
|---|---|---|---|---|
| | # sub-tensors | comp | # sub-tensors | comp |
| ResNet-101 | 2.3× | 2.5× | 2× | 2.2× |
| GoogLeNet v3 | 2.5× | 2.5× | 2.4× | 2.2× |
| AlexNet | 2.9× | 2.9× | 2.9× | 2.5× |

# 3 EXPERIMENTS

We use pre-trained FP-32 GoogleNetv3 [21], ResNet-101 [9], and AlexNet [14] models for ImageNet classification task. For 8-bit quantization, we have used the low-precision dynamic fixed point technique mentioned in [18]. Results of ternary residual networks are shown in Table 1. For power-performance gain, we estimate $X \sim 5.5$ for $N = 64$, which leads to an estimated power-performance gain of $\sim 2\times$ over 8-8 for accuracy 1% off from FP32 results. For comparison, we mention a few results of other sub-8-bit networks on ImageNet using AlexNet. (1) Binarized weights and activations of [20] incurred a loss of $\sim 12\%$, (2) the loss of binary weights and 2-bit activations of [28] was $\sim 6\%$ from FP-32, and (3) [11] with binary weight and 2-bit activations reduced the loss to $\sim 5.5\%$ from FP-32, (4) using FGQ with $N = 4$ (75% elimination of multiplications), [18] achieved $\sim 7.8\%$ loss from FP-32 using ternary weights and 4-bit activations without any low-precision re-training. Note, however, that (1) and (2) used FP-32 weights and activations for the first and last layers. A detailed analysis on power-performance for various methods is beyond the scope of this paper.

# REFERENCES


[1] Mitsuru Ambai, Takuya Matsumoto, Takayoshi Yamashita, and Hironobu Fujiyoshi. 2017. Ternary Weight Decomposition and Binary Activation Encoding for Fast and Compact Neural Networks. *https://openreview.net/pdf?id=ByOK0rwlx* (2017).

[2] Wenlin Chen, James T. Wilson, Stephen Tyree, Kilian Q. Weinberger, and Yixin Chen. 2015. Compressing neural networks with the hashing trick. In *ICML*.

[3] Matthieu Courbariaux, Itay Hubara, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. 2016. Binarized neural networks: Training deep neural networks with weights and activations constrained to + 1 or -1. *arXiv preprint arXiv:1602.02830* (2016).

[4] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. 2009. Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*. IEEE, 248–255.

[5] Yiwen Guo, Anbang Yao, and Yurong Chen. 2016. Dynamic Network Surgery for Efficient DNNs. In *Advances in Neural Information Processing Systems*.

[6] Suyog Gupta, Ankur Agrawal, Kailash Gopalakrishnan, and Pritish Narayanan. 2015. Deep Learning with Limited Numerical Precision. In *ICML*. 1737–1746.

[7] Song Han, Huizi Mao, and William J. Dally. 2016. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. In *ICLR*.

[8] Song Han, Jeff Pool, John Tran, and William J. Dally. 2015. Learning both weights and connections for efficient neural network. In *Advances in Neural Information Processing Systems*. 1135–1143.

[9] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 770–778.

[10] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwid Adam. 2017. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861* (2017).

[11] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. 2016. Quantized neural networks: Training neural networks with low precision weights and activations. *arXiv preprint arXiv:1609.07061* (2016).

[12] Forrest N. Iandola, Song Han, Matthew W. Moskewicz, Khalid Ashraf, William. J. Dally, and Kurt Keutzer. 2016. SqueezeNet: Alexnet-level accuracy with 50x fewer parameters and 0.5 mb model size. *arXiv preprint arXiv:1602.07360* (2016).

[13] Norm Jouppi. 2016. Google supercharges machine learning tasks with TPU custom chip. *Google Blog, May* 18 (2016).

[14] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*. 1097–1105.

[15] Fengfu Li, Bo Zhang, and Bin Liu. 2016. Ternary weight networks. *arXiv preprint arXiv:1605.04711* (2016).

[16] Zhe Li, Xiaoyu Wang, Xutao Lv, and Tianbao Yang. 2017. SEP-Nets: Small and Effective Pattern Networks. *arXiv preprint arXiv:1706.03912* (2017).

[17] Zhouhan Lin, Matthieu Courbariaux, Roland Memisevic, and Yoshua Bengio. 2016. Neural Networks with Few Multiplications. *arXiv preprint arXiv:1609.07061* (2016).

[18] Naveen Mellempudi, Abhisek Kundu, Dheevatsa Mudigere, Dipankar Das, Bharat Kaul, and Pradeep Dubey. 2017. Ternary Neural Networks with Fine-Grained Quantization. *https://arxiv.org/abs/1705.01462* (2017).

[19] Daisuke Miyashita, Edward H Lee, and Boris Murmann. 2016. Convolutional neural networks using logarithmic data representation. *arXiv preprint arXiv:1603.01025* (2016).

[20] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. 2016. XNOR-Net: ImageNet Classification Using Binary Convolutional Neural Networks. In *ECCV*.

[21] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. 2015. Rethinking the Inception Architecture for Computer Vision. *arXiv preprint arXiv:1512.00567* (2015).

[22] Yaman Umuroglu, Nicholas J Fraser, Giulio Gambardella, Michaela Blott, Philip Leong, Magnus Jahre, and Kees Vissers. 2016. FINN: A Framework for Fast, Scalable Binarized Neural Network Inference. *arXiv preprint arXiv:1612.07119* (2016).

[23] Wei Wen, Chunpeng Wu, Yandan Wang, Yiran Chen, and Li Hai. 2016. Learning Structured Sparsity in Deep Neural Networks. In *Advances in Neural Information Processing Systems*.

[24] Darrell Williamson. 1991. Dynamically scaled fixed point arithmetic. In *Communications, Computers and Signal Processing, 1991., IEEE Pacific Rim Conference on*. IEEE, 315–318.

[25] D Williamson, S Sridharan, and P McCrea. 1985. A new approach for block floating-point arithmetic in recursive filters. *IEEE transactions on circuits and systems* 32, 7 (1985), 719–722.

[26] Tien-Ju Yang, Yu-Hsin Chen, and Vivienne Sze. 2017. Designing Energy-Efficient Convolutional Neural Networks using Energy-aware Pruning. In *CVPR*.

[27] Aojun Zhou, Anbang Yao, Yiwen Guo, Lin Xu, and Yurong Chen. 2017. Incremental Network Quantization: Towards Lossless CNNs with Low-Precision Weights. *poster at International Conference on Learning Representations* (2017).

[28] Shuchang Zhou, Yuxin Wu, Zekun Ni, Xinyu Zhou, He Wen, and Yuheng Zou. 2016. DoReFa-Net: Training low bitwidth convolutional neural networks with low bitwidth gradients. *arXiv preprint arXiv:1606.06160* (2016).

[29] Chenzhuo Zhu, Song Han, Huizi Mao, and William J Dally. 2016. Trained Ternary Quantization. *arXiv preprint arXiv:1612.01064* (2016).