2019 IEEE International Conference on Edge Computing (EDGE)

Efficient Deep Neural Networks for Edge Computing

1st Mohammed Alnemari
Electrical and Computer Engineering
University of California, Irvine
Irvine, USA
malnemar@uci.edu

2nd Nader Bagherzadeh
Electrical and Computer Engineering
University of California, Irvine
Irvine, USA
nader@uci.edu

***Abstract*—Deep Neural Networks(DNNs) have demonstrated state-of-art performance on many problems in different fields such as computer vision, natural language processing, and others. Training of (DNNs) is computationally an expansive because of the backpropagation algorithm. In the inference stage, DNNs model requires an expansive computation and high storage cost, which makes them a challenging problem for devices that have constrained resources on the edge. In order to mitigate this issue, a plethora of methods has been applied and published in the literature in recent years. In this paper, FPTT is presented, a two-stage pipeline: filter pruning, and Tensor Train Decomposition, that work together to reduce the storage and computation requirements of DNNs in order to deploy them easily on the edge. Our work demonstrates the same accuracy or tiny degradation of accuracy with retraining after applying both stages. In particular, for the VGG-16 on CIFAR-10, CIFAR-100 and ImageNet we report compression reduction of 85%, 84.03%, and 93.69% respectively and for Alexnet on CIFAR-10, CIFAR-100 we report compression reduction of 96.5% and 94.5% , and for LeNet-5 on MNIST and CIFAR-10 we report compression rate of 93.2% and 98.71%.**

***Index Terms*—Deep Learning, Neural Networks, Model Compression, Edge Computing, Tensor Decomposition.**

I. Introduction

Deep Neural Networks(DNNs) have been used significantly in the last few years in many different fields, especially computer vision applications. This includes object classification, localization, segmentation, and tracking. The trend of DNNs in the past few years has increased the depth of neural networks by adding hidden layers. This depth has resulted in increased number of parameters which increases the storage and computation cost of these networks. The state-of-the-art in DNNs have presented very sophisticated powerful models in order to tackle the computer vision tasks. For example, VGG-16 model [2], which consists of 13 convolutional layers and 3 fully connected layers, requires 269.2Mb on CIFAR-10 after training for Keras model. Alexnet [3], which consists of 5 convolutional layers and 3 fully connected layers, requires 176.8 Mb on CIFAR-10 after training for Keras Model. LeNet-5, which consists of 3 convolutional layers and 2 fully connected layers, requires 14Mb after training on CIFAR-10 for Keras model.

DNNs on the edge will alleviate a several of the limitations brought by the cloud, such as privacy issues, network bandwidth and real-time processing. Though these models are very powerful, the large number of their parameters make them difficult to deploy on the edge. In addition, running high capacity networks on the edge will consume a lot of energy to fetch the weights and do the computation of dot product required by neural network layers.

There has been a lot of significant studies to reduce the storage and computation cost of the deep neural network by model compression [6] [7] [8]. These studies prune the parameters from the fully connected layers, which are responsible for only 1% of flops computation [9]. Other works [26] show the convolutional layers can be compressed but this requires a sparse BLAS library or even specialized hardware [6]. Other studies [9] [10] demonstrated pruning filters of convolutional layers instead of weight pruning, filter pruning removes the need of BLAS library or specialized hardware imposed by weight pruning. Some studies have suggested reducing the number of parameters of deep neural networks based on low-rank decomposition including Tucker Decompostion [12], Canonical Polyadic Decompostion [11] and Tensor Train Decomposition [1].

In this paper, FPTT is presented, a two-stage pipeline: filter pruning, and Tensor Train Decomposition. They work together to reduce the storage and computation requirements of DNNs in order to deploy them easily on the edge. In the first stage, we use filter pruning instead of weights pruning in order to alleviate the need for a BLAS library or specialized hardware. In the second stage, we used Tensor Train Decomposition, which is a form of nonrecursive tensor decomposition that not suffer from the curse of dimensionality and the parameters will be same as the canonical decomposition, but it is more stable and is based on a low-rank approximation of auxiliary unfolding matrices [13].

II. Related Works

DNNs, either convolutional or fully connected layers are designed to be overparameterized to generalize the deep neural networks models, resulting in many redundant parameters. Early works to tackle this problem were Optimal Brain Damage [14] and Optimal Brain Surgeon [15] in which second Taylor expansion was used to calculate the importance of the parameters. These methods pose a significant limitation

978-1-7281-2708-8/19/$31.00 ©2019 IEEE
DOI 10.1109/EDGE.2019.00014

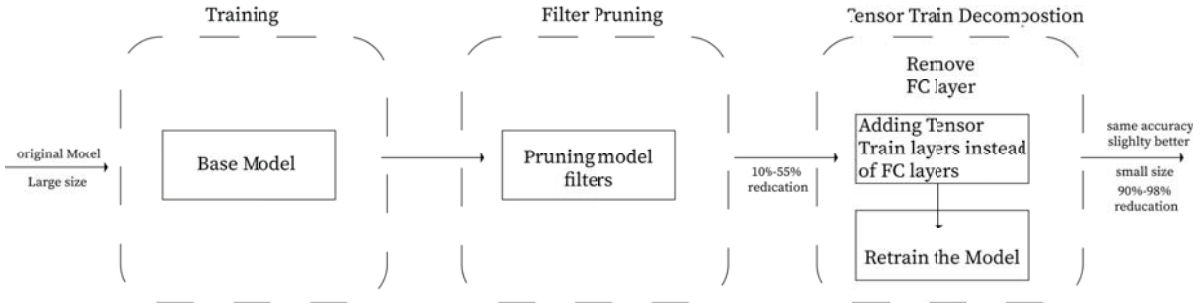

Figure 1. FPTT pipeline train the base model, apply filter pruning, apply Tensor Train decomposition,and finally retrain the model

on today's state-of-art deep neural networks models. They require costly calculations of second order derivatives. Another recent study is "Deep Compression" [6], which consists of three pipeline stages in order to compress the neural network: weights pruning, quantization, and Huffman coding. This suggested pruning the parameters by measuring the importance of these parameters by its absolute value in which deep neural network iteratively pruned the parameters below a certain threshold and fine-tune to acquire the original accuracy of the base models. Although, this approach is successful in compressing the neural network such as Alexnet [3], and VGG-16 [2], had significant limitations of producing sparse matrices that need BLAS library and also the compression mostly benefited the fully connected layers [9] while convolutional layers remained a challenge.

Filters pruning method was introduced by [9], which calculate the absolute sum of the kernel weights and iteratively removes unimportant filters with the smallest sum value. Filters pruning has an advantage compared to another approach presented by "Deep Compression" [6] in that the former will not result in sparse matrices that need a BLAS library or specialized hardware. Although this work was able to reduce the number of parameters by 55% on VGG-16, the overall number of the parameters remains a challenge. Our work will incorporate this work with one of the low-rank decomposition methods for the fully connected layer in order to reduce the computation and storage cost of these models further.

A lot of studies [17] show fully connected layers weights matrices is highly redundant, using low-rank decomposition is an approach to reduce the number of parameters by restricting its matrix rank without a significant drop in its predictive accuracy [18], [19]. A lot of methods introduced to use the low-rank approximation methods for fully connected layers and convolutional layers alike including Truncated Singular Value Decomposition [16], Tucker Decomposition [12], Canonical Polyadic Decomposition [11]. The limitation of these methods its heuristic searching for the rank of the matrices. Tensor Train Decomposition which is also a method of low-rank attempts to solve this issue by generalizing low-rank instead of searching heuristically of low-rank approximation of the weights. The weights will be treated as multi-dimensional-tensors and apply the Tensor Train Decomposition algorithm on it [1], [13].

Other works have been presented to reduce the computation and storage cost introduced by the state-of-art model such as quantization [6], binarization [20], ternarization [21], Huffman coding [6]. These methods can be incorporated with our work to further compression of the model. Although, binary neural networks which use +1, -1, and ternary neural network which was +1,0,-1 to represent the weight and activation reduce the number of parameters, still were not able to acquire the same performance of its full precision counterpart.

Other studies showing the use of reinforcement learning to generate neural network architecture [23] in which neural networks architecture description generated by Recurrent Neural Networks (RNNs) that trained via policy gradient, can be explored more possible architecture using Q-learning [24] [25]. Although, these models are very well optimized, but require a high amount of computation resources for training.

III. Proposed Method

In this section, the method is presented which is two pipeline stages: in the first stage, a filter pruning method on a model was applied, in the second stage Tensor Train Decomposition method was applied on the pruned model, and the model after that was retrain to acquire the same accuracy. The method is shown in Figure 1.

A. filter Pruning

Each convolutional layer in deep neural networks transforms an input feature map to output feature map which will be an input for the next convolutional layer. This is achieved by applying filters on the input channels which produce feature maps for each filter as shown in Figure 2.

When pruning the filters, the output feature maps corresponding to the pruned filter can be pruned as well [9]. In this paper the same algorithm presented in [9] was applied, which uses L1-norm of each filter as pruning criteria. The algorithm is as follow: first, calculate the sum of its absolute kernel weight as shown in Equation (1). second, sort the filter by f_j and prune the filter with the smallest sum value, and their corresponding features map in which kernel related to the pruned features map can also be pruned. Although filters pruning was from a well-trained model as suggested by [9],

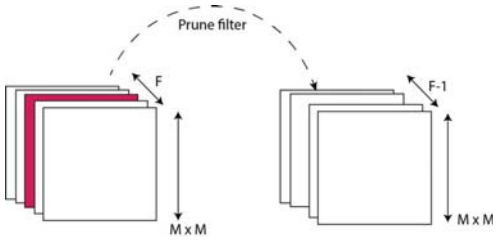

Figure 2. Filter Pruning

we had an issue while applying this method on models such as Alexnet and LeNet-5 where after applying it, we were not able to reacquire close or same accuracy as their base models. So, instead of applying the algorithm on a well-trained model, we applied it on untrained base model.

$$f_j = \sum_{n=1}^{n_i} = \sum \mid K_n \mid . \tag{1}$$

In order to prune the filters, the suggested method is used [9] which prune the filters across the network in a greedy fashion which result in high accuracy compared to independent approach or pruning filters layer by layer. Although the model produced after the filter pruning algorithm was small in size compared to the original model, it is not small enough to deploy on edge devices which have a constraint storage size and power consumption. In order to reduce this model further,the pruned model was passed to the second phase of our pipeline, which is Tensor Train Decomposition aimed to decompose the fully connected layers which account to decrease the model further.

B. Tensor Train Decomposition

Tensors are multidimensional generalization of the matrices. Tensor Train Decomposition is a form of nonrecursive tensor decomposition that not suffer from the curse of dimensionality. The parameters will be similar to Canonical Decomposition asymptotically, and its computation based on a low-rank approximation of auxiliary unfolding matrices [1] [13]. In this paper, Tensor Train layers used instead of fully connected layers. Tensor Train layers stored its weights matrix in Tensor Train format [1]. Tensor Train format, when applied on tensor $\mathbf{X}$, can be represent as shown in Equation (2) such that for each dimension $k = 1, ..., d$ and for each value of k-th dimension index $i_k = 1, ..., n_k$ there is a matrix exist $G_k[i_k]$ [1].

$$X(i_1, ..., i_d) = G_1[i_1]G_2[i_2]...G_d[i_d]. \tag{2}$$

All matrices $G_k[i_k]$ related to the same dimension K must be of the same size $r_{k-1} \times r_k$ and $r_0, r_d = 1$ in order to make matrix product Equation (2) of size 1×1. The sequence of $\{r_k\}_{k=0}^{d}$ is the rank of the Tensor Train format. The collection of the matrices $(G_k[j_k])_{i_k=1}^{n_k}$ are called cores. Equation (2) can be rewritten as shown in Equation (3) in which using the symbol $G_k[j_k](\alpha_{k-1}, \alpha_k)$ as elements of the matrices $G_k[j_k]$

$$X(i_1, ..., i_d) = \sum_{\alpha_0, ..., \alpha_d} G_1[i_1](\alpha_0, \alpha_1)...G_d[i_d](\alpha_{d-1}, \alpha_d). \tag{3}$$

In order to store Tensor $\mathbf{X}$ of all its elements requires $\prod_{k=1}^{d} n_k$ numbers compare to Tensor Train format which only requires $\sum_{k=1}^{d} n_k r_{k-1} r_k$ to store its elements, that shows Tensor Train format is very efficient in term of memory. $r_0 = r_d = 1$ can be represented geographically in linear tensor networks as shown in Figure (3). Ellipses represent indices of the original tensors i_k and indices of some of auxiliary tensors α_k, circles represent only the indices of the auxiliary tensors α_k and represent the link. If same auxiliary index is present in the two cores, we connect it. In order to evaluate entry of tensor , all tensor in ellipses has to be multiplied and we perform summation over all auxiliary indices [13].

Fully connected layers can be represented using Equation (4) in which $W \in \mathbb{R}^{m \times n}$ is weight matrix and $b \in \mathbb{R}^m$ is the bias vector.

$$y = W_x + b \tag{4}$$

Figure 3. Tensor Train Networks

Tensor Train layers store its weight matrix in Tensor Train format. Using Tensor Train format allows using hundreds and even thousands of hidden units with the number of parameters. The number of these parameters can be controlled either by using different number of units or change the Tensor Train rank. Tenor Train layers can be represented as shown in Equation (5)

$$y(i_1, ..., i_d) = \sum_{j_1, ..., j_d} G_1[i_1, j_1]...G_d[i_d]X(j_1, ..., jd)+B(i_1, ..., i_d). \tag{5}$$

Tensor Train format replaces a tensor $\mathbf{X}$ with its approximation tensor $\mathbf{Y}$ in which $||\mathbf{X} - \mathbf{Y}|| \leq accuracy||\mathbf{Y}||$. This format will reduce storage while maintaining the same accuracy. In order to add the Tensor Train layers to the model, the fully connected layers of the model after the filter pruning in the first stage has to be removed. The Tensor Train layers added to the top of the model results in an immense reduction of the DNNs models. DNNs usually trained using stochastic gradient descent algorithm in which using the backpropagation procedure to compute the gradient. The backpropagation procedure computes the gradient of the loss function of last layer and propagate through layers in reverse order. In order to use Tensor Train layers, we convert the gradient matrix to Tensor Train format, and add them to the estimation of the weight matrices. This method requires $\mathcal{O}(MN)$ memory which is not acceptable for the constrained device on the edge. So, In order to tackle this issue the same algorithm presented in [1]was applied where the gradient of the loss function computed directly to the core of the Tensor Train format of

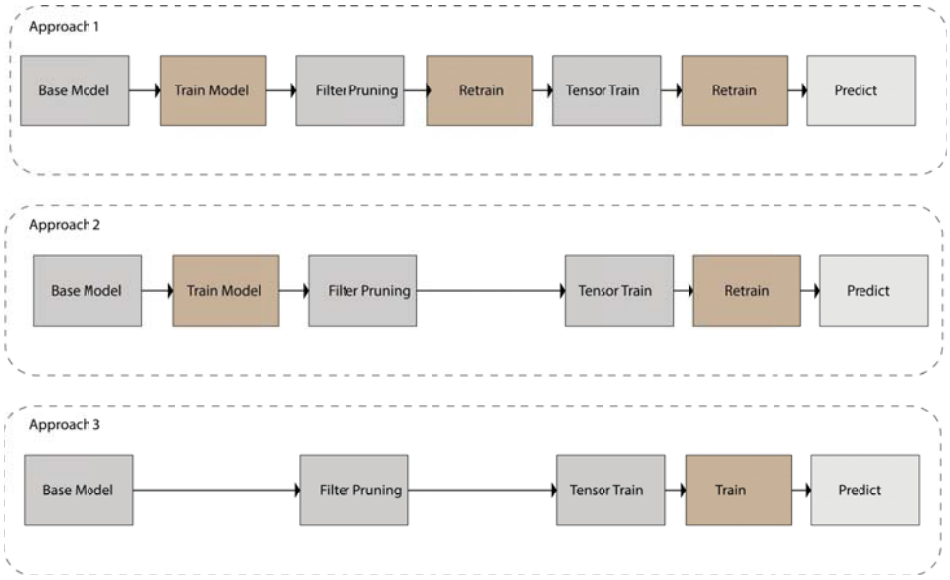

Figure 4. Three Different approach applied on the experiment, approach one train base model, apply filter pruning, retrain the model, apply Tensor Train, retrain and finally predict. Second approach train base model, apply filter pruning, apply Tensor Train, retrain, and finally predict. Third approach, apply filter pruning on the base model, apply Tensor Train, train, and finally predict

the weights in Tensor Train layers that requires $\mathcal{O}(M_{r_{k-1}r_k})$. See [1] for details description of all the equations of the loss function and modification of the gradients.

IV. Experiments

In our experiment, three different approaches have been performed. In the first approach, train the base model, prune filters, retain, apply Tensor Train Decomposition, retain, and finally predict. In the second approach, prune an untrained base model, train, Tensor Train Decomposition, retrain and finally predict. In the third approach, prune the untrained base model, apply Tensor Train Decomposition, train the model, and finally predict. The third method founded to work better with shallow or medium DNNs models resulting in tiny degradation compared to the base model. Also, the second approach founded to increase the accuracy by a small magnitude and works for all of the state-of-art deep neural networks models either deep or shallow. The first approach works well when the network is deep like VGG-16 and VGG-19, but not when networks that are not as deep as VGG-16 such as Alexnet and LeNet-5. The three approaches are shown in Figure (4). In our experiment, three different models used which are VGG-16, Alexnet, and LeNet-5. The results showing in the paper approach 2 for VGG-16 and approach 3 for Alexnet and LeNet-5.

A. VGG-16

It is a high capacity network designed for ImageNet dataset. The base model was used in this experiment that provided by Keras framework which consists of 13 convolutional layers and 3 fully connected layers with no dropout or batch normalization layers [2]. Model VGG-16 was trained on 3 different dataset which are CIFAR-10, CIFAR-100, and ImageNet.

1) CIFAR-10: It is a widely used dataset with 50,000 RGB images of 32×32 pixels for training and 10,000 for testing. In this experiment, horizontal flip, random shift and a random rotate for data augmentation were applied. The network is trained from scratch for 100 epochs with 256 batch size and 0.05 learning rate and eventually achieves 9.35% error rate. The base model after training has 3.36×10^7 parameters.

After applying our pipeline on the base model, which is

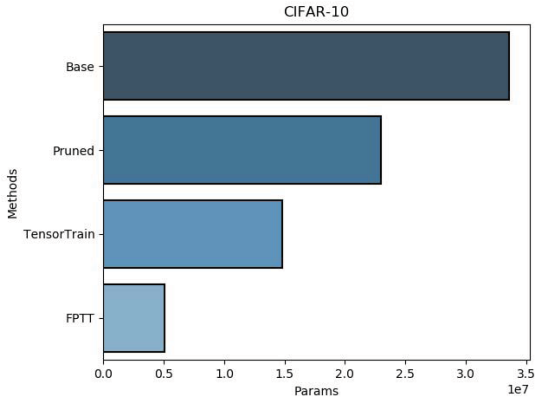

Figure 5. VGG-16 Model on CIFAR-10 show number of the parameters with base model, filter pruned model, Tensor Train decomposed model and FPTT model

filter pruning followed by tensor decomposition as mentioned in Section 3 and train our model with 100 epoch and 0.05 learning rate; eventually the model achieves a slightly high error rate of 9.01%. The pruned decomposed model has 5.04×10^6 parameters. The model achieves 85% reduction

compared to the base model. Also, this model is better than the model in [9] which achieved a reduction of 64% of their base model. Also, this model is better than the model in [1] which achieves 41% of reduction of their base model.

2) CIFAR-100: It is a widely used dataset with 50,000 RGB images of 32×32 pixels for training and 10,000 for testing. In this experiment, horizontal flip, random shift and a random rotate for data augmentation were applied. The network is trained from scratch for 100 epochs with 256 batch size and 0.05 learning rate and eventually achieves 37.1% error rate. The base model after training has 3.4×10^7 parameters.

After applying our pipeline on the base model untrained,

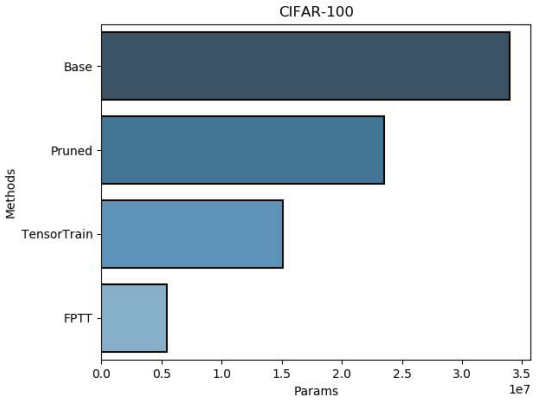

Figure 6. VGG-16 Model on CIFAR-10 show number of the parameters with base model, filter pruned model, Tensor Train decomposed model and FPTT model

and train the model with same hyperparameter and the same data augmentation, the model achieves 35.93% error rate. The pruned decomposed model has 5.43×10^6 parameters. This model achieves 84.03% reduction of the base model.

3) ImageNet: It widely used dataset with 1.2 million RGB images of 224×224 pixels for training and 50 000 for testing. In our experiment, the model used was trained by Keras framework, which has an error rate 28.9% and 1.38×10^8 parameters.

After applying the pipeline on the model,the model retrained with 100 epochs, 32 batch size and 0.01 learning rate using stochastic gradient descent with 0.9 momentum. The model achieves an error rate 26.6% and the number of parameters is 8.71×10^6.

Table I
VGG-16

Model	Parameters	Error(%)	Compression(%)
Cifar-10-base	3.36×10^7	10.1%	-
Cifar-100-base	3.4×10^7	37.1%	-
Imagenet-base	1.38×10^8	28.9%	-
Cifar-10-FPTT	5.04×10^6	9.01%	85%
Cifar-100-FPTT	5.43×10^6	35.93%	84.03%
Imagenet-FPTT	8.71×10^6	26.6%	93.69%

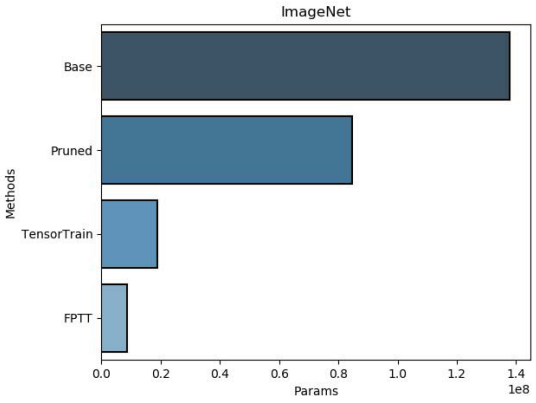

Figure 7. VGG-16 Model on ImageNet show number of the parameters with base model, filter pruned model, Tensor Train decomposed model and FPTT Model

B. Alexnet

It is a high capacity network designed for ImageNet dataset. In this experiment the model built [3] using Keras framework. The model consists of 5 convolutional layers and 3 fully connected layers with no dropout or batch normalization layers. Alexnet trained on CIFAR-10, CIFAR-100.

1) CIFAR-10: In our experiment, the same data augmentation for VGG-16 model were used. The network trained from scratch for 250 epoch with 128 batch size and 0.05 learning rate and eventually the model achieves 13.05% error rate and 2.21×10^7 parameters. After applying the pipeline on the base model and train the model with the same hyperparameter; the model achieves slightly lower error rate with 14.07%. The pruned decomposed model has 7.67×10^5 parameter which is 96.5% reduction of the base model.

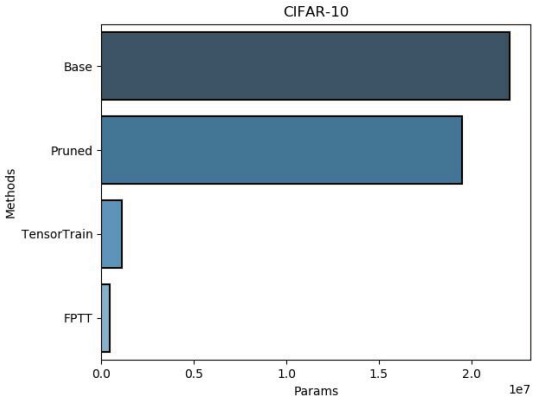

Figure 8. Alexnet Model on CIFAR-10 show number of the parameters with base model, filter pruned model, Tensor Train decomposed model and FPTT model

2) CIFAR-100: In our experiment we use the same data augmentation and the same hyperparameter we applied on Alexnet for CIFAR-10. We achieve an error rate of 43.7%

with 2.25×10^7 parameters. After applying our pipeline we achieve an error rate of 44.44% with 1.1×10^6 parameters which is 94.5% reduction of the base model.

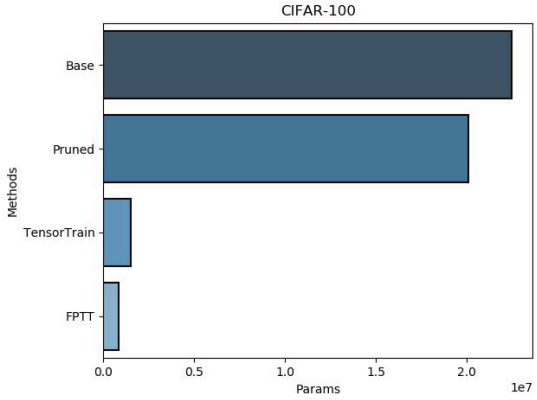

Figure 9. Alexnet Model on CIFAR-100 show number of the parameters with base model, filter pruned model, Tensor Train decomposed model and FPTT model

Table II
ALEXNET

Model	Parameters	Error(%)	Compression(%)
Cifar-10-base	2.21×10^7	13.05%	-
Cifar-100-base	2.25×10^7	43.7%	-
Cifar-10-FPTT	7.67×10^5	14.07%	96.5%
Cifar-100-FPTT	1.1×10^6	44.44%	94.5%

C. LeNet-5

It a small neural network architecture designed for handwriting recognition. In this experiment, the model built [4] using Keras framework. The model consists of 3 convolutional layers and 2 fully connected layers. The LeNet-5 model trained on CIFAR-10 and MNIST.

1) CIFAR-10: In the experiment, horizontal flip, random shift and random rotation for data augmentation were used.The network is trained from scratch for 100 epochs with 256 batch size and 0.05 learning rate and eventually the model achieves 24.96% error rate. The base model after training has 1.74×10^6 parameters.

After applying the pipeline on the untrained base model, and train the model with same hyperparmeter and the same data augmentation; the model achieves 26.23% error rate with 8.52×10^4 parameters which is 95.1% reduction of the base model.

2) MNIST: With this dataset, we use the same data augmentation and hyperparameter settings we used for CIFAR-10. We train the network from scratch and we achieve 13.61% with 1.31×10^6.

After applying the pipeline the model achieves 10.01% error rate with 8.86×10^5 parameters. This is a reduction of 93.2% of the base model.

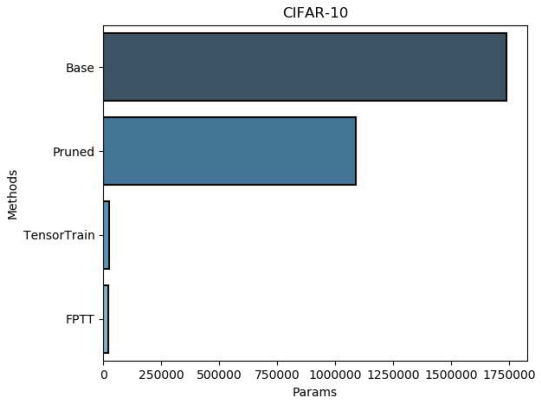

Figure 10. LeNet-5 Model on CIFAR-10 show number of the parameters with base model, filter pruned model, Tensor Train decomposed model and FPTT model

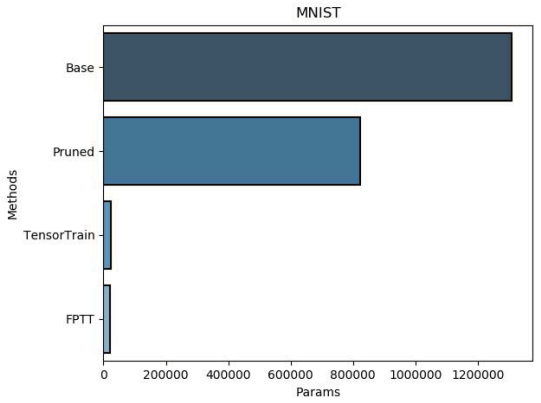

Figure 11. LeNet-5 Model on MNIST show number of the parameters with base model, filter pruned model, Tensor Train decomposed model and FPTT model

V. CONCLUSION

Recent studies show many of parameters in the neural networks are redundant, and the state-of-art neural network architectures are overparameterized. To address this issue a pipeline presented with two stages in which filter pruning applied first, followed by Tensor Train Decomposition. This pipeline outperform these two methods when they applied individually. In the Table IV there is a comparison in the size after applied the pipeline on model with CIFAR-10.

Table III
LENET-5

Model	Parameters	Error(%)	Compression(%)
Cifar-10-base	1.74×10^6	24.96%	-
Mnsit-base	1.31×10^6	13.61%	-
Cifar-10-FPTT	2.24×10^4	26.23%	98.71%
Mnsit-FPTT	8.86×10^4	10.01%	93.2%

Table IV
SIZE OF MODEL ON CIFAR-10

Model	Base size	FPTT size	(Factor)
VGG-16	269.2MB	37.6MB	7.1x
Alexnet	176.8MB	6.2MB	28.5x
LeNet-5	14MB	455.6KB	30.7x

REFERENCES

[1] Novikov, A., Podoprikhin, D., Osokin, A., Vetrov, D. P. (2015). Tensorizing neural networks. In Advances in Neural Information Processing Systems (pp. 442-450).

[2] Simonyan, K., Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556.

[3] Krizhevsky, A., Sutskever, I., Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In Advances in neural information processing systems (pp. 1097-1105).

[4] LeCun, Y., Bottou, L., Bengio, Y., Haffner, P. (1998). Gradient-based learning applied to document recognition. Proceedings of the IEEE, 86(11), 2278-2324.

[5] Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K. Li, F.-F. (2009). ImageNet: A large-scale hierarchical image database.. CVPR (p./pp. 248-255), : IEEE Computer Society. ISBN: 978-1-4244-3992-8

[6] Han, S., Mao, H., Dally, W. J. (2015). Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. arXiv preprint arXiv:1510.00149.

[7] Han, S., Liu, X., Mao, H., Pu, J., Pedram, A., Horowitz, M. A., Dally, W. J. (2016, June). EIE: efficient inference engine on compressed deep neural network. In 2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA) (pp. 243-254). IEEE.

[8] Han, S., Pool, J., Tran, J., Dally, W. (2015). Learning both weights and connections for efficient neural network. In Advances in neural information processing systems (pp. 1135-1143).

[9] Li, H., Kadav, A., Durdanovic, I., Samet, H., Graf, H. P. (2016). Pruning filters for efficient convnets. arXiv preprint arXiv:1608.08710.

[10] Molchanov, P., Tyree, S., Karras, T., Aila, T., Kautz, J. (2016). Pruning convolutional neural networks for resource efficient inference. arXiv preprint arXiv:1611.06440.

[11] Lebedev, V., Ganin, Y., Rakhuba, M., Oseledets, I., Lempitsky, V. (2014). Speeding-up convolutional neural networks using fine-tuned cp-decomposition. arXiv preprint arXiv:1412.6553.

[12] Kim, Y. D., Park, E., Yoo, S., Choi, T., Yang, L., Shin, D. (2015). Compression of deep convolutional neural networks for fast and low power mobile applications. arXiv preprint arXiv:1511.06530.

[13] Oseledets, I. V. (2011). Tensor-train decomposition. SIAM Journal on Scientific Computing, 33(5), 2295-2317.

[14] Hassibi, B., Stork, D. G. (1993). Second order derivatives for network pruning: Optimal brain surgeon. In Advances in neural information processing systems (pp. 164-171).

[15] LeCun, Y., Denker, J. S., Solla, S. A. (1990). Optimal brain damage. In Advances in neural information processing systems (pp. 598-605).

[16] Girshick, R. (2015). Fast r-cnn. In Proceedings of the IEEE international conference on computer vision (pp. 1440-1448).

[17] Denil, M., Shakibi, B., Dinh, L., De Freitas, N. (2013). Predicting parameters in deep learning. In Advances in neural information processing systems (pp. 2148-2156).

[18] Sainath, T. N., Kingsbury, B., Sindhwani, V., Arisoy, E., Ramabhadran, B. (2013, May). Low-rank matrix factorization for deep neural network training with high-dimensional output targets. In 2013 IEEE international conference on acoustics, speech and signal processing (pp. 6655-6659). IEEE.

[19] Xue, J., Li, J., Gong, Y. (2013, August). Restructuring of deep neural network acoustic models with singular value decomposition. In Interspeech (pp. 2365-2369).

[20] Hubara, I., Courbariaux, M., Soudry, D., El-Yaniv, R., Bengio, Y. (2016). Binarized neural networks. In Advances in neural information processing systems (pp. 4107-4115).

[21] Alemdar, H., Leroy, V., Prost-Boucle, A., Ptrot, F. (2017, May). Ternary neural networks for resource-efficient AI applications. In 2017 International Joint Conference on Neural Networks (IJCNN) (pp. 2547-2554). IEEE.

[22] Huang, Q., Zhou, K., You, S., Neumann, U. (2018, March). Learning to prune filters in convolutional neural networks. In 2018 IEEE Winter Conference on Applications of Computer Vision (WACV) (pp. 709-718). IEEE.

[23] Zoph, B., Le, Q. V. (2016). Neural architecture search with reinforcement learning. arXiv preprint arXiv:1611.01578.

[24] Baker, B., Gupta, O., Naik, N., Raskar, R. (2016). Designing neural network architectures using reinforcement learning. arXiv preprint arXiv:1611.02167.

[25] Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., ... Dieleman, S. (2016). Mastering the game of Go with deep neural networks and tree search. nature, 529(7587), 484.

[26] Iandola, F. N., Han, S., Moskewicz, M. W., Ashraf, K., Dally, W. J., Keutzer, K. (2016). SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and¡ 0.5 MB model size. arXiv preprint arXiv:1602.07360.