Dynamic Block Sparse Reparameterization of Convolutional Neural Networks

Dharma Teja Vooturi, Girish Varma, Kishore Kothapalli
Center for Security Theory and Algorithmic Research
International Institute of Information Technology Hyderabad, India
dharmateja.vooturi@research.iiit.ac.in, girish.varma@iiit.ac.in, kkishore@iiit.ac.in

Abstract

Sparse neural networks are efficient in both memory and compute when compared to dense neural networks. But on parallel hardware such as GPU, sparse neural networks result in small or no runtime performance gains. On the other hand, structured sparsity patterns like filter, channel and block sparsity result in large performance gains due to regularity induced by structure. Among structured sparsities, block sparsity is a generic structured sparsity pattern with filter and channel sparsity being sub cases of block sparsity. In this work, we focus on block sparsity and generate efficient block sparse convolutional neural networks using our approach DBSR (Dynamic block sparse reparameterization). Our DBSR approach, when applied on image classification task over Imagenet dataset, decreases parameters and FLOPS of ResneXt50 by a factor of 2x with only increase of 0.48 in Top-1 error. And when extended to the task of semantic segmentation, our approach reduces parameters and FLOPS by 30% and 20% respectively with only 1% decrease in mIoU for ERFNet over Cityscapes dataset. To ease developments in this line of work, we open sourced our code on github (https://github.com/idharmateja/bsnn).

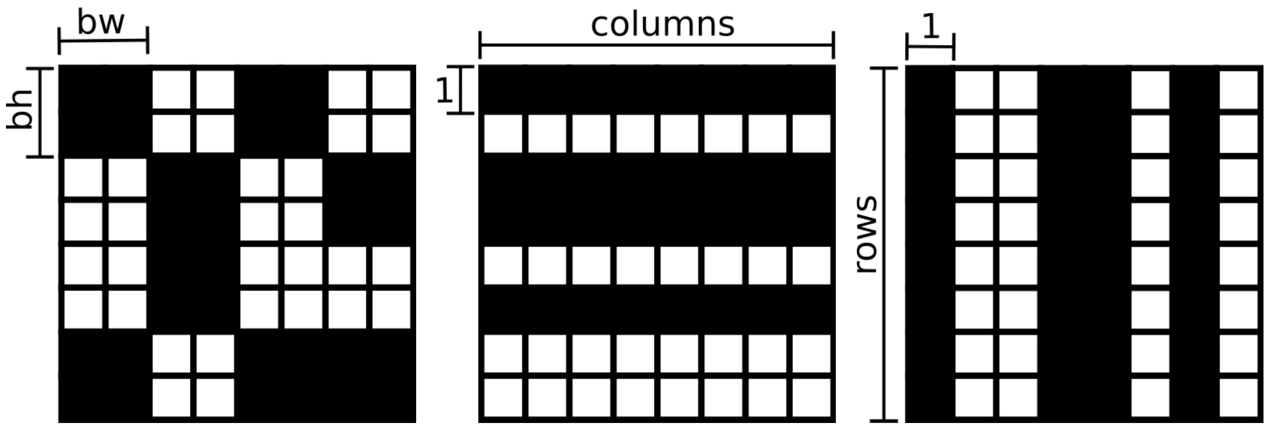
1. Introduction

The overall performance of a neural network can be characterized by three factors mainly: memory (number of parameters/weights), compute (number of FLOPS) and accuracy. An optimal neural network is the one which achieves maximum accuracy by using minimum amount of memory and compute. But there is no definite way of arriving at such a neural network and in practice there are trade offs between memory, compute, and accuracy. Sparsification of neural networks is an effective way to reduce memory and compute requirements with minimal tradeoff in accuracy.

Pruning technique [11] is one of the most widely used technique to generate sparse neural networks. In pruning, unimportant connections from a pre-trained dense neural network are removed and then the resultant sparse neural network is fine-tuned to maintain accuracy. Early works by Hassibi and Stork [6], and Le Cun *et al*. [11], have shown the effectiveness of pruning and in recent times, Han *et al*. [5], has used pruning technique to generate sparse convolutional neural networks that have significant gains in memory and compute. Apart from pruning technique, sparse neural networks can also be generated either by training them directly from scratch [24, 15] or during the training process [30, 16].

Sparse models have reduced number of FLOPS when compared to dense models. Ideally, this reduction in FLOPS should translate into equivalent reduction in runtime of the model. But on parallel hardware like GPU, sparse models have poor or no runtime performance gains due to the irregularity in computation of sparse neural networks. The only way to realize the fruits of reduced number of FLOPS is to design specialized hardware like EIE, SCNN [3, 22] for accelerating sparse neural networks. But this requires a lot of work in terms of developing hardware, middleware and software stack. On top of that, the utility of such specialized hardware is limited to only sparse workloads. Whereas that is not the case with parallel hardware like GPU, where varied type of computationally intensive workloads can be accelerated. So the question one should ask is, "*Can we generate sparse neural networks whose compute is amenable to parallel hardware like GPU?*"

Computation in sparse operations is highly irregular in nature. Parallel hardware can be efficiently used only when the computation is regular in nature. So, one simple way to bring regularity in computation of a sparse neural network is by inducing structure in to sparsity. Several types of structured sparsity patterns (see Figure 1) like filter sparsity [17], channel sparsity [9] and block sparsity [29] were proposed. In filter/channel sparsity, the unit is a row/column which is either completely zero or non-zero. Whereas in block sparsity, the unit is a block of parameters with dimensions (bh, bw), where the parameters in a given block are either all zero or non-zero. Both filter and channel sparsity are special cases of block sparsity and it has been shown in [27], that the computation of block sparse operations can be efficiently processed on parallel hardware like GPU. This



(a) Block sparsity with block size (bh, bw), where bh is block height and bw is block width.

(b) Filter sparsity. Sub case of block sparsity with block size (1,columns)

(c) Channel sparsity.Subcase of block sparsity with block size (rows,1)

Figure 1. Block sparsity and it's subcases. For convolutional layer in a CNN, sparsity pattern in 4D parameter tensor W can be visualized as a matrix M, where M[i,j] is zero if all entries in 2D kernel W[i,j,:,:] are zeros.

is because the algorithms for processing block sparse operations heavily piggyback on the algorithms used for dense operations. Block sparsity is important also because, newer processor architectures has components like systolic array (TPU) and tensor core (V100 GPU), which process matrices in blocks. Hence block sparsity could be better exploited by these processor architectures. In this work, due to the generic nature of block sparsity pattern and it's good runtime performance benefits, we developed techniques to generate efficient block sparse networks. Following are our main contributions:

- Developed a simple, easy to use, and effective approach (DBSR) for generating structured sparse neural networks with most generic block sparsity pattern. When compared to dense training, our DBSR approach requires only one extra hyper parameter ζ, which controls the amount of sparsity.
- As part of DBSR approach, a block scaling operation and a block scaled convolution layer are proposed, which forms the basis for quickly exploring new techniques for generating structured sparse neural networks.
- We show the effectiveness of DBSR approach on important vision tasks like image classification and semantic segmentation over varied networks(VGG, Resnet20, Resnet50, ResneXt50, ERFNet) and datasets (CIFAR, Imagenet, Cityscapes).

In section 2, we go through the related work. Our approach and performance aspects of block sparsity are discussed in section 3. In section 4, we got through our experiments and finally in section 5, we conclude the paper and provide future work.

2. Related Work

Sparsification of neural networks [5, 4, 13, 23] is used as an effective technique to reduce number of parameters and FLOPS. But despite being an effective technique, it has been noted in [30, 20], that the sparse neural networks have poor runtime performance when compared to that of dense networks. In order to address the issue of runtime performance, the focus has shifted to structured sparsification [19, 12, 18, 33, 8].

Generation of structured sparse neural networks can be classified into two paradigms: invitro and invivo. In invitro paradigm, sparse model is generated from a pre-trained dense model. Where as in invivo paradigm, sparse model is learned during the training process. Common methodology underlying approaches based on invitro paradigm has two main steps. They are: 1) Prune structural units from a pre-trained model based on some criterion 2) Fine tune the pruned model to recover accuracy. Invitro approaches differ only in the chosen pruning criterion, and sparsity pattern. L_1 norm of the weights of a unit structure is chosen as the pruning criterion in [12, 18] and [29] for pruning filters and blocks respectively. In [17], filters of a current layer are pruned based on the channel weights of next layer. In [8], channel pruning is performed based on LASSO regression based channel selection method. In [33], filters are pruned across layers in a joint fashion by back propagating scores corresponding to the filters. In [19], filters are pruned using Taylor expansion based pruning criterion. Even though invitro approaches are less computationally intensive than a full training, there are many challenges like determining pruning percentages for layers, choice of hyper parameters for fine-tuning etc, which make it difficult to use invitro approaches in practice.

Our approach falls under invivo paradigm, where sparsity generation is tightly coupled with the training process. In our approach, we associate a scaling parameter for each structure, thus allowing our method to generate varied type of structured sparsity patterns. We focus on block sparsity pattern, which is the most generic structured sparsity pattern with filter and channel sparsity patterns being sub cases of block sparsity. Many of the previous approaches in invivo paradigm, focus only on filter or channel sparsity. In [14], filter sparsity is generated by regularizing scaling parameters already present in batch norm layers. To also support filter sparsity generation for layers with out batch norm, [9] introduced additional scaling parameter for each output activation channel. Both [9, 14] differ from our approach as they scale output activation channels rather than weights thus limiting to generation of only filter sparsity pattern. In [30], weights of a chosen structure are regularized by using group LASSO regularization. This differs from our approach, as we regularize scaling factors associated with the block of weights rather than weights themselves. The idea of using group LASSO regularization was adapted for recurrent neural networks to generate structured sparse RNNs with block sparsity pattern [21]. In [16], a binary variable is associated with each structure, and the binary value is stochastically learned during training. Our method is a non stochastic approach, where scaling variable is a real valued number that is trained along with weight parameters.

3. Approach

Our approach DBSR (Dynamic Block Sparse Reparameterization) generates block sparsity through the training process in a dynamic fashion. The main idea in DBSR approach is to use trainable scaling parameters for the blocks and generate block sparsity by pushing the values of scaling parameters to zero using L_1 regularization. DBSR approach makes use of two building blocks, namely block scaling operation and block scaled convolution layer. We first describe them in detail and later use them to formulate our approach. We also discuss about the performance aspects of block sparse networks.

Block Scaling Operation. Block scaling operation denoted by $*_b$, takes two input tensors S and W, and produces a single output tensor W_S. Input tensor S is a 2D tensor with dimensions (s_1, s_2) and W is at least a 2D tensor with dimensions $(w_1, w_2, ..w_n)$. Blocking is performed on outer two dimensions of W, with dimensions (bh, bw), where $bh = w_1/s_1$ and $bw = w_2/s_2$. In forward (resp backward) pass, each element in W (dW_S) is scaled by the scaling factor associated with it's block to generate $W_S(dW)$. In backward pass each entry in dS is calculated by taking dot product of the associated blocks in W and dW_S. Algorithms 1 and 2, details the computation in forward and backward pass of block scaling operation.

Algorithm 1 Forward pass for block scaling operation $W_S = S *_b W$.

```
1: $s_1, s_2 = S.shape$
2: $w_1, w_2.., w_n = W.shape$
3: $bh = w_1/s_1$                                ▷ Block height
4: $bw = w_2/s_2$                                ▷ Block width

5: for $i = 0 : w_1 - 1$ do
6:     for $j = 0 : w_2 - 1$ do
7:         $W_S[i, j] = S[i/bh, j/bw] * W[i, j]$
8:     end for
9: end for

10: return $W_S$
```

Algorithm 2 Backward pass for block scaling operation $W_S = S *_b W$.

```
1: for $i = 0 : s_1 - 1$ do
2:     for $j = 0 : s_2 - 1$ do
3:         $hr = i * bh : (i + 1) * bh - 1$
4:         $wr = j * bw : (j + 1) * bw - 1$
5:         $W^{bij} = W[hr, wr]$
6:         $dW_S^{bij} = dW_S[hr, wr]$
7:         $dS[i, j] = Sum(W^{bij} .* dW_S^{bij})$
8:         $dW[hr, wr] = S[i, j] * dW_S^{bij}$
9:     end for
10: end for
11: return $dW, dS$
```

Block scaled convolution layer. Convolution operation in a convolutional layer consists of producing an output activation tensor O, by applying convolution on an input activation tensor I. $O = conv(W, I)$, where W is a 4D parameter/filter tensor. Block scaled convolution layer is built on top of regular convolution layer with additional parameters and computation. In block scaled convolution layer, we store an additional 2D parameter tensor S along with W. Computation in a block scaled convolution layer is processed in two steps: 1) A block scaling operation is performed on S and W to generate W_S ($W_S = S *_b W$). 2) Using W_S, convolution operation is performed on input I to produce output O ($O = conv(W_S, I)$).

Formulation. Learning block sparsity in a neural network is jointly modelled with primary learning task as an optimization problem according to Equation 1, where (x_i, y_i) is an instance in dataset, M is a model, and L is a loss function. W and S correspond to weight and block scaling parameters respectively. $R_w(.)$ and $R_S(.)$ are regularizers used for W and S respectively.

$$\min_{W,S} \sum_{i=1}^{N} L\left(M(S,W,x_i),y_i\right)/N + R_w(W) + R_s(S) \quad (1)$$

Given the choice of block size, all convolutional layers in model M are converted into block scaled convolution layers. Before training, all parameters in S are given equal importance by initializing them to a value of 1. As training progresses, the values of parameters in S change dynamically due to the error propagation from loss function and regularizers. In order to generate sparsity, we push the values of scaling parameters S to zero by using L_1 regularization $(\zeta * |S|)$ for $R_s(.)$. Hyper parameter ζ controls the amount of sparsity in the model. Zero clipping is performed on S (where $S = \max(0, S)$) after every training iteration to ensure that the values in S remains positive. Once the model is trained, S can be discarded after updating blocks in W using a block scaling operation $(W = S *_b W)$.

In our approach, there is a small memory and compute overhead when compared to dense model training. Memory overhead is due to the need for storage of block scaling parameters S along with weight parameters W. Compute overhead comes from additional block scaling operation performed before convolution operation. These memory and compute overheads are limited only to training, and are not present in inference.

Performance of block sparse CNNs. In popular deep learning libraries like cuDNN [1], convolution operation is performed by posing it as a matrix multiplication operation (GEMM) $(O = W * I_{mat})$, where each row in W matrix corresponds to a flattened 3D filter and I_{mat} is a lowered matrix of 3D input I. So in order to accelerate convolution operation, an efficient GEMM operation is required. In case of block sparse convolution, we need an efficient block sparse matrix matrix multiplication (BSMM) operation.

Efficient algorithms for BSMM operation can be adapted from fast dense matrix multiplication (GEMM) algorithms that are based on cache friendly blocking schemes. In blocking scheme, C matrix is divided into blocks, and a block in C is computed by iteratively loading corresponding blocks in A and B into a cache and then multiplying them. By keeping the blocks in the cache, memory accesses become more efficient and thus increases runtime performance. For BSMM, we can use the same blocking scheme based algorithms used for GEMM, except that computation corresponding to zero blocks are skipped. For example in Figure 2, we can see that a block C_{11} is computed in only two steps as there are only two nonzero blocks corresponding to C_{11} in A. Using the blocking scheme, efficient block sparse kernels for GPU are successfully developed in [27] to deliver ideal speedups.

4. Experiments

We evaluate our approach on Imagenet [26] and cityscapes [2] datasets for the task of image classification and semantic segmentation respectively. For image classification, we choose two state of the art networks Resnet50 [7], and ResneXt50 [31]. And for semantic segmentation, we choose realtime semantic segmentation network ERFNet [25]. In all our experiments, we use the same experimental setting used for dense model training

4.1. Effectiveness of DBSR on classification networks

Resnet50/Imagenet. In Resnet50, we chose 32x32 as the block size and trained multiple networks by varying ζ which controls the sparsity. From Table 1, we can see that with only an increase of 0.47 in Top-1 error, our model Resnet50-32x32-A decreases both parameters and FLOPS of Resnet50 by $\sim$30%. More compact models can be generated by increasing the value of ζ. For instance, Renset50-32x32-C decreases parameters by a factor of 2.5x, and FLOPS by a factor of 2x, with an increase of 2.66 in Top-1 error when compared with Resnet50. In [7], two small dense networks Resnet18 and Renset34 are carefully designed to have less memory and computational footprint. When compared with them, our models have more efficiency. Our model Resnet50-32x32-D has 1.47 less Top-1 error, and takes 30% less parameters and 20% less FLOPS when compared to Resnet18. Similarly when compared with Resnet34, our model Resnet50-32x32-B has 0.34 less Top-1 error, and takes $\sim$38% less parameters and FLOPS. From comparisons with small dense models such as Resnet18 and Resnet34, we can observe that big sparse models are more efficient in terms of both memory and FLOPS. This observation was also made by [20] in case of recurrent neural networks.

Model	Top-1 Error	#Params	#FLOPS
Resnet18	30.24	11.67M	1.8B
Resnet34	26.70	21.77M	3.6B
Resnet-50	24.61	25.5M	3.85B
Resnet50-32x32-A	25.08	17.89M	2.74B
Resnet50-32x32-B	26.36	13.36M	2.19B
Resnet50-32x32-C	27.27	10.81M	1.85B
Resnet50-32x32-D	28.77	8.41M	1.42B
ResneXt-50	23.35	24.96M	4.23B
ResneX50-32x32-A	23.83	12.72M	2.32B
ResneXt50-32x32-B	24.69	9.39M	1.83B
ResneXt50-32x32-C	25.36	8.03M	1.65B
ResneXt50-32x32-D	26.20	6.60M	1.59B

Table 1. Block sparse models generated using DBSR approach for the task of image classification on Imagenet ILSVRC2012 dataset. Centre crop is used for calculating error.

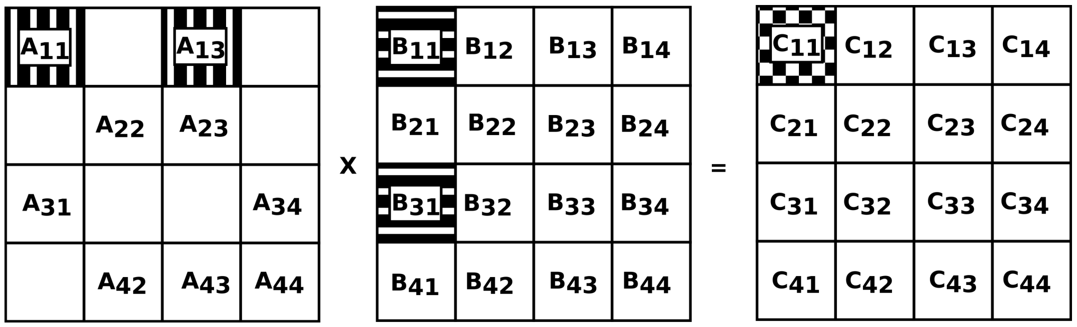
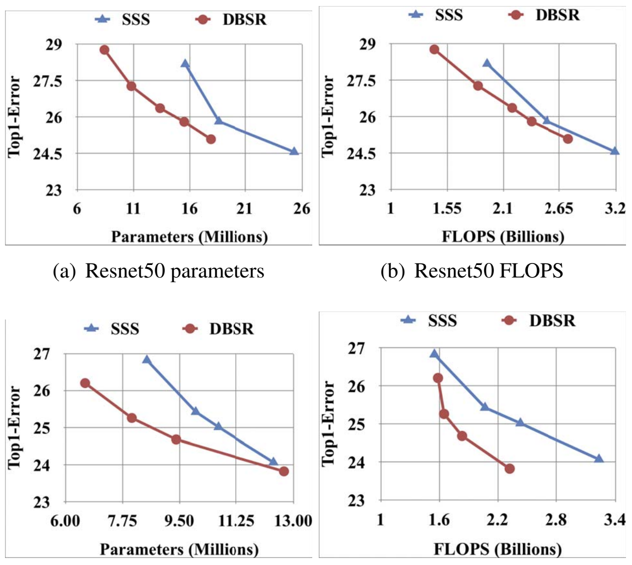


Figure 2. Cache friendly blocking scheme for efficiently processing BSMM (block sparse matrix multiplication) operation.

ResneXt50/Imagenet. In ResneXt50, grouped convolutions are introduced to improve accuracy and reduce memory and computational requirements. While applying DBSR, we do not replace grouped convolutional layers with block scaled convolution layers. The reason for that is a grouped convolution layer has inherent block sparsity pattern with blocks strictly lying on the main diagonal. From Table 1, we can see that with only an increase of 0.48 in Top1-error, our model ResneXt50-32x32-A decreases parameters and FLOPS of ResneXt50 by 49% and 45% respectively. While model ResneXt50-32x32-C with higher ζ values decreases parameters by a factor of $\sim$3.1x and FLOPS by a factor of $\sim$2.6x with only 2.01 increase in Top-1 error.

Comparison with SSS [9]. SSS(sparse structure selction) is a recent work, where structured sparse neural networks are generated from dense networks (Resnet50, ResneXt50) through the training process. Similar to SSS, our DBSR approach generates structured sparse networks with block sparsity pattern through the training process. From Figure 3, we can see that when compared to models generated using SSS, DBSR models are more efficient in memory, compute, and error. For both Resnet50 and ResneXt50, we can see from Figures 3(a) & 3(c), that DBSR approach becomes even more effective as the number of parameters in the model decreases. This is especially useful for effectively running low capacity models on resource limited embedded GPUs like Jetson TK2.

Comparison with structured pruning methods. We compare our DBSR approach with other structured pruning based methods like filter pruning [12], channel pruning [8] and ThinNet [17]. From Table 2, we can see that our model Resnet50-32x32-A has 0.75-1.6 less Top-1 error when compared to other pruning based approaches while still taking less number of parameters/FLOPS. When compared with pruned Resnet-101 from [32], our model has 0.36 less Top-1 error and has 26% less FLOPS for similar number of parameters. It should be noted that pruning approach is orthogonal



(a) Resnet50 parameters (b) Resnet50 FLOPS

(c) ResneXt50 parameters (d) ResneXt50 FLOPS

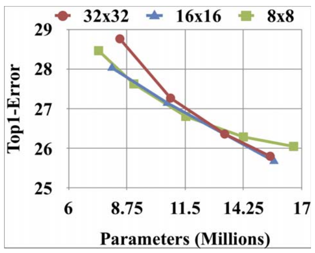
Figure 3. Comparison of block sparse models from DBSR approach with structured sparse models from (SSS)[9] for the task of image classification on Imagenet dataset.

to DBSR approach and can be applied on DBSR models to further decrease memory and computational requirements.

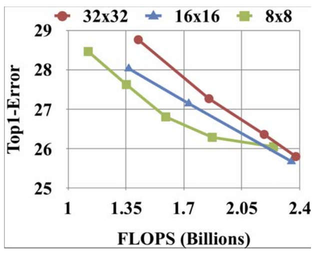
Varying block size. In this experiment, we would like to see the effect of block size on model efficiency. We train multiple block sparse models with block sizes 8,16 and 32. From Figure 4, we can see that the models with smaller block sizes are more effective when the model capacity(number of parameters/FLOPS) is less. This might be because when number of parameters are less, flexibility offered by smaller block sizes helps in improving connectivity. But as the model capacity increases, the effectiveness gap among block sizes decreases. From Figure 4, we can see that the block size 16 has more efficiency than block size 8 for models with higher capacity. This might be because blocking may have a regularization effect and is higher for larger block sizes.

Model	Top-1 Error	Top-5 Error	Params	#FLOPs
ResNet-34-pruned [12]	27.44	-	19.9M	3.08B
Resnet-50-pruned [12] (From [9])	27.12	8.95	-	3.07B
Resnet-50-pruned(2x) [8]	27.70	9.20	-	2.73B
Resnet50-pruned (ThinNet-30) [17]	27.96	9.33	16.94M	2.44B
Resnet-50-32x32-B (Ours)	**26.36**	**8.23**	**13.36M**	**2.19B**
Resnet-101-pruned [32]	25.44	-	17.30M	3.69B
Resnet50-32x32-A (Ours)	**25.08**	**7.73**	**17.89M**	**2.74B**

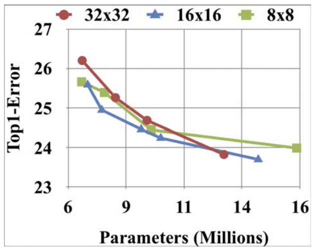
Table 2. Comparison of block sparse models from DBSR approach with models from other structured pruning methods for the task of image classification over Imagenet dataset. We can see that DBSR models are more efficient in parameters and FLOPS when compared to SSS models.



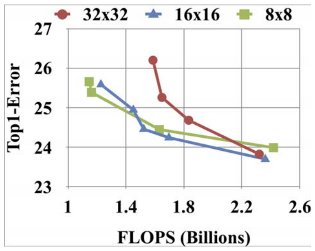
(a) Resnet-50 parameters



(b) Resnet-50 FLOPS



(c) ResneXt-50 parameters



(d) ResneXt-50 FLOPS

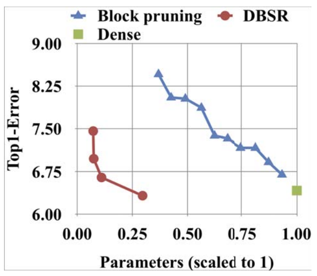
Figure 4. Effect of varying block size on model efficiency.

4.2. Comparison with block pruning

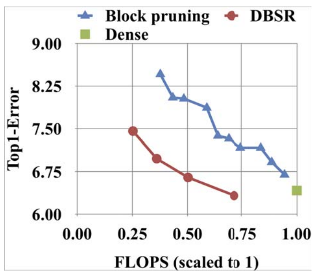
Our DBSR approach generates structured sparse neural networks with block sparsity pattern. Block sparse neural networks can also be generated by adapting pruning approach [7] to prune blocks instead of individual elements. We compare our DBSR approach with block pruning approach on CIFAR datasets CIFAR10 & CIFAR100 [10] over VGG and ResNet-20 [7] networks. Our VGG network for CIFAR is adapted from VGG16 network [28], with fc6 and fc7 replaced by a single fc layer of size 512, and batch normalization layer is added to all convolution and fc layers.

In DBSR, we train the model for 240 epochs with base learning rate set to 0.1. Where as for block pruning, we finetune the pruned model for 24 epochs with base learning rate set to 0.01. For both DBSR and block pruning, the learning rate is decreased by a factor of 10 at 3/6, 4/6 and 5/6 th of the training/finetuning cycle. We use SGD with nestrov momentum optimization with momentum and weight decay values set to 0.9 and 1e-4 respectively

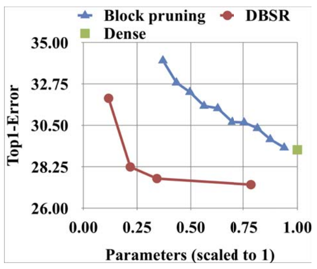
VGG/CIFAR. For VGG, we set block size to 32x32 and generate block sparse models with different model capacities. From Figure 5, we can see that models generated from our DBSR approach are more efficient that of block pruning approach. Block pruning has an effect on error even for small amount of pruning. Where as that is not the case for DBSR models. For CIFAR10, upto 90% parameter reduction and 50% FLOPS reduction can be achieved while maintaining accuracy comparable to that of dense. Similarly for CIFAR100, upto 80% parameter reduction and 40% FLOPS reduction can be achieved with comparable accuracies to that of dense.



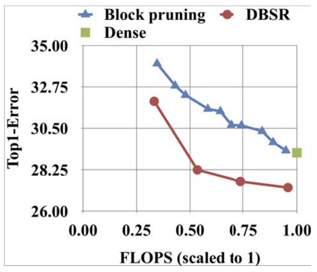
(a) CIFAR10 parameters



(b) CIFAR10 FLOPS



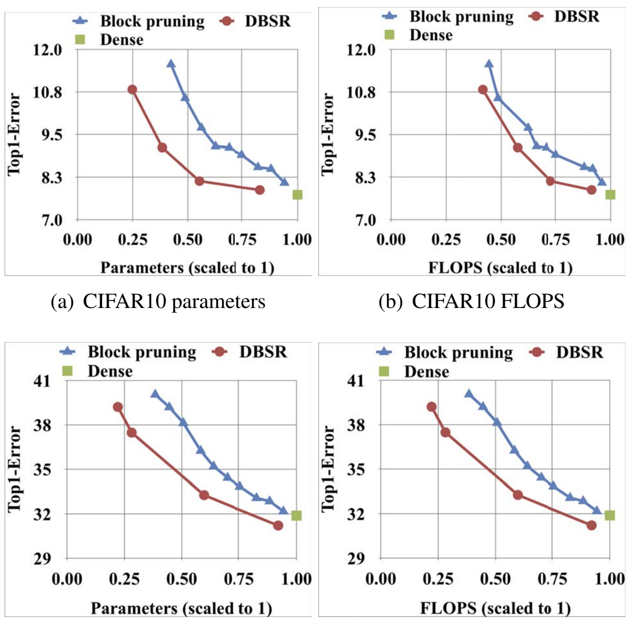
(c) CIFAR100 parameters



(d) CIFAR100 FLOPS

Figure 5. Comparison of DBSR approach with block pruning approach on image classification task over CIFAR datasets for VGG network with block size 32x32

Resnet20/CIFAR. In Resnet20, number of channels in convolutional layers are less and range only from 16 to 64. Hence we chose 8x8 as the block size for Resnet20. From Figure 6, we can see that our approach performs better than block pruning. Effectiveness of DBSR for Resnet20 is less when compared with that of VGG. This might be due to the residual connections in Resnet20, which make it more robust to block pruning approach. With little loss in accuracy, upto 50% parameters and 30% FLOPS can be reduced for CIFAR10, and upto 30% parameters and FLOPS can be reduced for CIFAR100.



(a) CIFAR10 parameters (b) CIFAR10 FLOPS

(c) CIFAR100 parameters (d) CIFAR100 FLOPS

Figure 6. Comparison of DBSR approach with block pruning approach on image classification task over CIFAR datasets for Resnet20 network with block size 8x8

4.3. Semantic Segmentation

We extend our DBSR approach on to the task of semantic segmentation and evaluate on cityscapes dataset [2] using ERFNet [25] network. The choice of ERFNet is due to its low capacity and ability to do real time semantic segmentation, which is critical for many applications. ERFNet follows an encoder-decoder architecture with factorized convolutions. In our experiments, we choose 16x16 as the block size and convert all convolution layers into block scaled convolution layers. For training, we use the same setup used for dense training. Table 3, shows our results on ERFNet over cityscapes dataset. With only a loss of 1% in mIoU, our model ERFNet-16x16-A decreases parameters by 30% and FLOPS by 20% when compared with dense ERFNet. More compact models can be found in Table 3 by increasing the value of ζ.

5. Conclusion

In this work, we developed a simple and effective technique called DBSR (Dynamic Block Sparse Reparameterization) for generating efficient block sparse neural networks. In DBSR, block sparsity is generated by using a set of trainable scaling parameters for the blocks and pushing them to zero during training using L_1 regularization. Unlike structured pruning methods, where sparsity and structure are induced post training, our DBSR approach tightly integrates structured sparsity generation with the training process and thus produces more efficient models. We have shown the effectiveness of our approach on standard vision tasks like image classification and semantic segmentation, and have obtained efficient models when compared to state of the art structured sparsity approaches.

Model	mIoU	#Params	#FLOPS
ERFNet	67.75	2.05M	29.83B
ERFNet-16x16-A	66.74	1.47M	23.82B
ERFNet-16x16-B	66.30	1.23M	21.92B
ERFNet-16x16-C	65.37	1.02M	19.47B
ERFNet-16x16-D	63.61	0.75M	15.87B

Table 3. Block sparse ERFNet models with 16x16 block size for the task of semantic segmentation over cityscapes dataset.

For future work, we would like to extend our DBSR approach to other vision tasks like object detection and instance segmentation. And also to further increase model efficiencies, we would like to explore layer specific schemes.

References

[1] S. Chetlur, C. Woolley, P. Vandermersch, J. Cohen, J. Tran, B. Catanzaro, and E. Shelhamer. cudnn: Efficient primitives for deep learning. *arXiv preprint arXiv:1410.0759*, 2014. 4

[2] M. Cordts, M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth, and B. Schiele. The cityscapes dataset for semantic urban scene understanding. In *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016. 4, 7

[3] S. Han, X. Liu, H. Mao, J. Pu, A. Pedram, M. A. Horowitz, and W. J. Dally. Eie: Efficient inference engine on compressed deep neural network. *SIGARCH Comput. Archit. News*, 44(3):243–254, June 2016. 1

[4] S. Han, H. Mao, and W. J. Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015. 2

[5] S. Han, J. Pool, J. Tran, and W. Dally. Learning both weights and connections for efficient neural network. In *Advances in neural information processing systems*, pages 1135–1143, 2015. 1, 2

[6] B. Hassibi and D. G. Stork. Second order derivatives for network pruning: Optimal brain surgeon. In *Advances in neural information processing systems*, pages 164–171, 1993. 1

[7] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016. 4, 6

[8] Y. He, X. Zhang, and J. Sun. Channel pruning for accelerating very deep neural networks. In *The IEEE International Conference on Computer Vision (ICCV)*, Oct 2017. 2, 5, 6

[9] Z. Huang and N. Wang. Data-driven sparse structure selection for deep neural networks. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 304–320, 2018. 1, 3, 5, 6

[10] A. Krizhevsky and G. Hinton. Learning multiple layers of features from tiny images. Technical report, Citeseer, 2009. 6

[11] Y. LeCun, J. S. Denker, and S. A. Solla. Optimal brain damage. In *Advances in neural information processing systems*, pages 598–605, 1990. 1

[12] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf. Pruning filters for efficient convnets. *ICLR*, 2016. 2, 5, 6

[13] B. Liu, M. Wang, H. Foroosh, M. Tappen, and M. Pensky. Sparse convolutional neural networks. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015. 2

[14] Z. Liu, J. Li, Z. Shen, G. Huang, S. Yan, and C. Zhang. Learning efficient convolutional networks through network slimming. In *The IEEE International Conference on Computer Vision (ICCV)*, Oct 2017. 3

[15] Z. Liu, M. Sun, T. Zhou, G. Huang, and T. Darrell. Rethinking the value of network pruning. *arXiv preprint arXiv:1810.05270*, 2018. 1

[16] C. Louizos, M. Welling, and D. P. Kingma. Learning sparse neural networks through l_0 regularization. *arXiv preprint arXiv:1712.01312*, 2017. 1, 3

[17] J.-H. Luo, J. Wu, and W. Lin. Thinet: A filter level pruning method for deep neural network compression. In *The IEEE International Conference on Computer Vision (ICCV)*, Oct 2017. 1, 2, 5, 6

[18] H. Mao, S. Han, J. Pool, W. Li, X. Liu, Y. Wang, and W. J. Dally. Exploring the regularity of sparse structure in convolutional neural networks. *arXiv preprint arXiv:1705.08922*, 2017. 2

[19] P. Molchanov, S. Tyree, T. Karras, T. Aila, and J. Kautz. Pruning convolutional neural networks for resource efficient inference. *ICLR*, 2016. 2

[20] S. Narang, E. Elsen, G. Diamos, and S. Sengupta. Exploring sparsity in recurrent neural networks. *arXiv preprint arXiv:1704.05119*, 2017. 2, 4

[21] S. Narang, E. Undersander, and G. Diamos. Block-sparse recurrent neural networks. *arXiv preprint arXiv:1711.02782*, 2017. 3

[22] A. Parashar, M. Rhu, A. Mukkara, A. Puglielli, R. Venkatesan, B. Khailany, J. S. Emer, S. W. Keckler, and W. J. Dally. SCNN: an accelerator for compressed-sparse convolutional neural networks. *CoRR*, abs/1708.04485, 2017. 1

[23] J. Park, S. Li, W. Wen, P. T. P. Tang, H. Li, Y. Chen, and P. Dubey. Faster cnns with direct sparse convolutions and guided pruning. *arXiv preprint arXiv:1608.01409*, 2016. 2

[24] A. Prabhu, G. Varma, and A. Namboodiri. Deep expander networks: Efficient deep networks from graph theory. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 20–35, 2018. 1

[25] E. Romera, J. M. Alvarez, L. M. Bergasa, and R. Arroyo. Erfnet: Efficient residual factorized convnet for real-time semantic segmentation. *IEEE Transactions on Intelligent Transportation Systems*, 19(1):263–272, 2018. 4, 7

[26] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015. 4

[27] A. R. Scott Gray and D. P. Kingma. Gpu kernels for block-sparse weights. 2017. 1, 4

[28] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. In *International Conference on Learning Representations*, 2015. 6

[29] D. T. Vooturi, D. Mudigree, and S. Avancha. Hierarchical block sparse neural networks. *arXiv preprint arXiv:1808.03420*, 2018. 1, 2

[30] W. Wen, C. Wu, Y. Wang, Y. Chen, and H. Li. Learning structured sparsity in deep neural networks. In *Advances in neural information processing systems*, pages 2074–2082, 2016. 1, 2, 3

[31] S. Xie, R. Girshick, P. Dollár, Z. Tu, and K. He. Aggregated residual transformations for deep neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1492–1500, 2017. 4

[32] J. Ye, X. Lu, Z. Lin, and J. Z. Wang. Rethinking the smaller-norm-less-informative assumption in channel pruning of convolution layers. *arXiv preprint arXiv:1802.00124*, 2018. 5, 6

[33] R. Yu, A. Li, C.-F. Chen, J.-H. Lai, V. I. Morariu, X. Han, M. Gao, C.-Y. Lin, and L. S. Davis. Nisp: Pruning networks using neuron importance score propagation. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018. 2