# Pushing the Envelope of Dynamic Spatial Gating technologies

Xueqin Huang*
Texas A & M University
xueq13@tamu.edu

Urmish Thakker
Arm ML Research
uthakker@cs.wisc.edu

Dibakar Gope
Arm ML Research
dibakar.gope@arm.com

Jesse Beu
Arm ML Research
jesse.beu@arm.com

## ABSTRACT

There has been a recent surge in interest in dynamic inference technologies which can reduce the cost of inference, without sacrificing the accuracy of the model. These models are based on the assumption that not all parts of the output feature map (OFM) are equally important for all inputs. The parts of the output feature maps that are deemed unimportant for a certain input can be skipped entirely or computed at a lower precision, leading to reduced number of computation. In this paper we focus on one such technology that targets unimportant features in the spatial domain of OFM, called Precision Gating (PG). PG computes most features in low precision, to identify regions in the OFM where an object of interest is present, and computes high precision OFM for that region only. We show that PG leads to loss in accuracy when we push the MAC reduction achieved by a PG network. We identify orthogonal dynamic optimization opportunities not exploited by PG and show that the combined technologies can achieve far better results than their individual baseline. This Hybrid Model can achieve 1.92x computation savings on a CIFAR-10 model at an accuracy of 91.35%. At a similar computation savings, the PG model achieves an accuracy of 89.9%. Additionally, we show that PG leads to GEMM computations that are not hardware aware and propose a fix that makes PG technique CPU friendly without losing accuracy.

## CCS CONCEPTS

• **Computing methodologies → Neural networks**.

## KEYWORDS

neural networks, dynamic computation, IoT, efficient inference

**ACM Reference Format:**
Xueqin Huang, Urmish Thakker, Dibakar Gope, and Jesse Beu. 2020. Pushing the Envelope of Dynamic Spatial Gating technologies. In *The 2nd International Workshop on Challenges in Artificial Intelligence and Machine Learning for Internet of Things (AIChallengeIoT '20), November 16–19, 2020, Virtual Event, Japan.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3417313.3429380

*Work done while interning at Arm ML Research

## 1 INTRODUCTION

Neural networks are increasingly being deployed to enable IoT applications [1]. These applications need to operate in a fixed runtime budget and yet not sacrifice on accuracy. Traditionally, to meet these run-time budgets, neural networks were compressed before deployment using techniques like channel pruning, random pruning, structured matrices and quantization. These techniques make structural changes to a large model to make it smaller permanently. Generally, a large compression factor will also lead to a large reduction in inference run-time. However, large compression factors can lead to loss in accuracy [23, 25].

This has motivated the growth of research in the domain of dynamic techniques that do not change large models permanently and yet lead to faster inference. By not forcing the model to grow smaller in order to get faster, these models can achieve the dual objective of high accuracy and reduced computation, thus performing better than the standard compression techniques. Additionally, some of these techniques exploit optimizations that cannot be exploited statically, like determining the location of the object of interest in the input feature map (IFM) to limit the computation to only those sections of the IFM.

Dynamic techniques use an extremely small predictor network to determine saliency of the various components of the output feature map (OFM), given an IFM. Based on the saliency, they either completely skip the computation of certain parts of the network or calculate them at a lower precision. When an entire channel computation is deemed less important, the technique is referred to as Dynamic Channel Pruning (DCP) and when certain spatial locations of the OFM are determined to be less important, the corresponding techniques are called dynamic spatial gating (DSG). Two popular techniques in this domain are feature boosting and suppression (FBS) and precision gating (PG). FBS falls in the DCP category and PG falls in the DSG category.

Once trained, such networks may run on CPUs during inference. Generally, neural network inference on CPUs use vector compute units (VCU) to achieve higher throughput execution. A typical convolution operation is converted into a general matrix-matrix multiplication (GEMM) library call via an im2col operation. These GEMM libraries require dense computation in order to execute them on VCUs leading to a high-throughput and extremely efficient execution on a CPU. Introduction of sparsity can significantly impact the throughput of the GEMM operation because of poor utilization of these VCUs.

In this paper, we make two contributions:

- Firstly, we show that for large computation savings, PG leads to loss in accuracy. Further, in order to enable PG for applications that require large computation savings, we identify opportunities for optimization that are not exploited by PG and apply a method to exploit those opportunities. We show empirically that these opportunities are orthogonal. Further, by combining these methods we show we can achieve 1.45% more accuracy than a PG network for the same computation budget.
- Secondly, we show that while PG reduces the number of operations, it introduces sparsity that breaks common GEMM operations and leads to poor unitilization of HW vector units. Further, we show a vector-length blocking aware PG that can create block-based sparsity which can be executed using a block sparsity aware GEMM library.

## 2 RELATED WORK

Design of compute efficient CNNs can be divided into two broad categories - static techniques and dynamic techniques.

## 2.1 Static Techniques

These techniques create efficient networks by making changes that are permanent, i.e., once deployed, the network does not change during inference. Some of the techniques in this domain are described in this section. Static techniques can be categorized into 4 broad domains - quantization, sparsity, tensor decomposition and efficient network design.

2.1.1 *Efficient network design.* One approach is hand-crafting efficient structures that require lesser number of computation and smaller storage requirement. The work in this domain include Squeezenet [14], MobileNets [10, 11] and Shufflenet [27]. Dynamic Techniques can be applied on top of these compression methods to get further gains in computation savings.

2.1.2 *Channel and Random Pruning.* This set of works deal with introducing sparsity into the neural network and removing weights that are deemed unnecessary. When weights are pruned without any structure, they might not translate to run-time benefits ([8]). A structure can also be imposed on the pruning mechanism, like pruning an entire channel of a CNN [9, 17]. Dynamic inference techniques have shown to achieve higher accuracy than these techniques.

2.1.3 *Tensor Factorization.* Tensor factorization is an approach that decomposes a single layer into smaller, more efficient layers [15, 21–25]. Such decomposition creates a smaller network. But often, these decomposition creates network that do not lead to faster execution on current hardware or lead to loss in accuracy.

2.1.4 *Quantization.* Quantization of a neural network implies running the network at a reduced precision [6, 7, 13]. Running at reduced precision allows for higher throughput execution on hardware. Dynamic inference techniques are orthogonal to optimizations in this domain and can further benefit from research in this domain.

## 2.2 Dynamic Techniques

While static technologies permanently alter the network architecture, dynamic technologies create efficient inference networks by selectively computing parts of the network that are needed for an input activation. Generally, these techniques have a saliency predictor that determines the importance of the output features and selectively computes the important features needed for the correct classification of the input. These techniques can be divided into three broad categories - dynamic channel pruning (DCP), dynamic layer skipping (DLS), dynamic spatial gating (DSG), Dynamic Layer Skipping (DLS) and Mixture of Experts (MoE).

2.2.1 *Dynamic Spatial Gating (DSG).* DSG techniques identify spatial regions in the OFM that are deemed important and focus their attention only on those parts of the OFM. Thus, if your OF is of size HxWxC_*out*, where H is height, W is the width and C_*out* is the number of output channels, DSG techniques focus only on the some elements of the spatial dimension (HxW) of each output channel, either by only computing those features or computing them at a higher precision than other features. Two recent papers in this domain are channel gating neural networks and precision gating networks. Channel gating neural networks identify regions in the features that contribute less to the output and skips the computation for those specific regions ([12]). **Precision Gating** is a quantization technique that computes most features using low precision and only a small subset of important features with higher precision to preserve accuracy ([28]).

2.2.2 *Dynamic Channel Pruning (DCP).* DCP techniques identify channels in the OFM that are deemed unimportant and skip computations for those channels. Batch-shaped channel gated networks use the concept called batch-shaping to enforce such sparsity, matching the marginal aggregate posterior of features in a network to a pre-specified prior beta distribution, to force the channel gating mechanism to be more conditional on data ([2]). Thus, based on data certain channel computation will be skipped, while others will be retained. **Feature boosting and suppression** (FBS) is a specific type of pruning strategy that dynamically prunes intermediate channels based on input features ([4]). It introduces an auxiliary connection, called the channel saliency predictor, to evaluate which channels have important information content. Based on the output of this predictor, a hyperparameter known as the gate density, *d*, top-k important channels are selected and computed.

2.2.3 *Dynamic Layer Skipping (DLS).* These techniques are more specific to ResNet style architecture with skip connections or RNN based models [20, 26]. These techniques use the saliency predictor network to determine what layers in the architecture need to be computed and what can be skipped.

2.2.4 *Mixture of Experts (MoE).* MoE is based on the idea that instead of using a single large neural network to process an input data, multiple domain experts can be used to process the input. Based on the input, the results from the domain experts can be given more weight. The routing to the experts is done via a predictor network. By gating experts that are deemed unimportant for the input, one can achieve faster computation [19]. DCP techniques can be viewed as a specific instantiation of the techniques in this domain.

## 3 PRECISION GATING

Precision Gating (PG) is a type of DSG technology. It is an element-wise pruning strategy, which achieves acceleration by introducing sparsity in the spacial domain of the output features ([28]). In PG, the $B$-bit fixed-point format input features, $\mathbf{I}$, in each layer are split into the $B_{hb}$ most-significant bits, $\mathbf{I}_{hb}$, and the $B_{lb}$ least-significant bits, $\mathbf{I}_{lb}$. Here $B = B_{hb} + B_{lb}$. In the prediction phase, the partial product of most-significant bits, $\mathbf{I}_{hb}$ and the weights of the layer are used to obtain a partial output feature map (OFM), $\mathbf{O}_{hb}$. Based on the partial product $\mathbf{O}_{hb}$, a learnable gating threshold , $\Delta$, selects which features are important. That is, elements in a OFM that are larger than $\Delta$ are determined to be important, while elements that are smaller than $\Delta$ are deemed unimportant. This mechanism is used to generate an importance **mask**. In the update phase, PG computes the remaining less-significant bits partial product for these important features, $\mathbf{O}_{lb}$, based on $\mathbf{I}_{lb}$, and adds it to $\mathbf{O}_{hb}$. A hyperparameter called gating target, $\delta$, is introduced to control the pruning intensity in the loss function. During training $\Delta$ and $\delta$ are kept close to each other via L2-loss to ensure that the threshold values meet the required hyper-parameter target. The workflow of PG is illustrated in Figure 1.

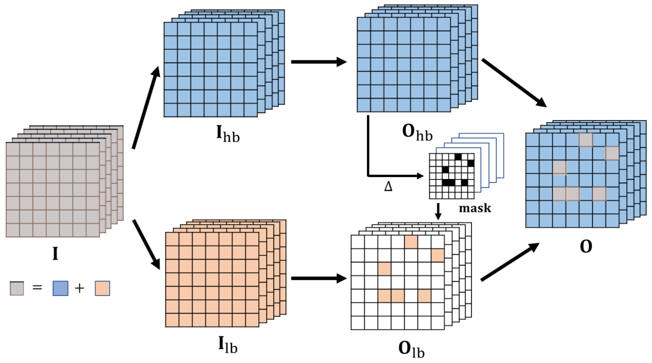


**Figure 1: The workflow for precision gating: The full precision input feature I is split into $\mathbf{I}_{hb}$ and $\mathbf{I}_{lb}$. In the predictor phase, $\mathbf{O}_{hb}$ is calculated to generate the decision map, *mask*, based on $\mathbf{I}_{hb}$. *mask* of a feature is assigned to 1, when $\mathbf{O}_{hb}$ of that feature excesses the threshold $\Delta$. $\mathbf{O}_{lb}$ is only computed on the important feature, where *mask* is equal to 1. The final output O is obtained by adding $\mathbf{O}_{hb}$ and $\mathbf{O}_{lb}$.**

In order to estimate the bit-level MAC savings of the network, we use the methodology described in [28]. In a PG network, most of the computation happens using the lower $B_{lb}$ bits of the activation and only a few computation happen using the higher order bits. Depending on the sparsity of the generated mask, s, we can estimate the average number of bits used in the calculation as -

$$B_{avg} = B_{lb} + (1 - s) * B_{hb} \tag{1}$$

And the corresponding Multiply-Accumulate (MAC) Reduction savings as -

$$MACSavings = (B_{lb} + B_{hb})/(B_{avg}) \tag{2}$$

Recent research in CPU instructions has shown how to translate this kind of bit-level MAC reduction for such computations into run-time benefits [5]. There is also a large body of research in hardware accelerators that has shown how to exploit such opportunities [16, 18].

## 4 PUSHING THE LIMITS OF PG

### 4.1 Loss of Accuracy with increasing sparsity

The amount of acceleration available under PG is proportional to the amount of spatial sparsity that is generated during training; more sparisty translates into increased performance benefit. Increasing sparsity is accomplished by increasing the threshold value in PG. We ran experiments on a baseline CIFAR-10 network, varying threshold values to achieve different levels of sparsity/performance and compare their results. Figure 2 plots accuracy vs MAC savings for PG for increasing threshold values and demonstrates how this leads to larger computation savings, but also a decrease in accuracy. This is because as the larger threshold values increases sparsity, more and more useful spacial information is being thrown out. However, if we stop threshold scaling before this accuracy-losing point and identify a dynamic computation technology that exploits other, orthoginal opportunities not exploited by PG, we may be able to find additional compute savings without sacrificing as much accuracy as continuing down the aggressive thresholding/sparsifying path.

### 4.2 Dynamic Computation Optimization opportunity not exploited by PG

While PG exploits sparsity in the spatial domain, i.e., it reduces the required number of computation in regions of the image with lower relevant information content, it still computes all the channels of the OFM. As discussed in Section 2, there is a class of techniques that exploit dynamic sparsity in the channel domain. Our insight is that these orthogonal optimization techniques can be applied simultaneously to collaboratively improve compute performance while maintaining model accuracy. The remainder of this section will focus on discussing this contribution.

**Feature Boosting and Suppression:** Feature boosting and suppression (FBS) is a specific dynamic channel pruning strategy that adds channel-wise sparsity of the OFM based on the corresponding input channels ([4]). In the network, an auxiliary component is introduced to predict the importance of the each output channel. Based on the output of this auxiliary network, k-most important channels are selected and the rest of the channels are ignored. The number of the winner channel is determined by a hyperparameter called the gate density, $d$. Thus another way to view $d$ is that (100-$d$)% of channel in a OFM are not computed.

**Proving PG is orthogonal to FBS:** We ran experiments to confirm that FBS technique and PG technique do indeed extract orthogonal opportunities for optimization. In order to do this, we run PG at various precision values, for FBS with various gate density, $d$, values. We show that for the same precision, different gate density

| | % Channel Sparsity (1 - GateDensity) | $B/B_{hb}$ | % Spatial Sparsity |
|---|---|---|---|
| FBS+PG | 0% | 8/4 | 71.89% |
| | 10% | 8/4 | 71.48% |
| | 20% | 8/4 | 71.46% |
| | 30% | 8/4 | 70.46% |
| | | | |
| | 0% | 4/2 | 65.29% |
| | 10% | 4/2 | 64.92% |
| | 20% | 4/2 | 64.14% |
| | 30% | 4/2 | 64.95% |

**Table 1: Results indicating the orthogonality of FBS and PG techniques. The table shows the spatial sparsity of the FBS network at various channel sparsity when trained with PG for 2 different PG precision values ($B/B_{hb}$ values), where $B$ is the full precision bit and $B_{hb}$ is the most-significant bits. As shown in the table, for the same $B$ and $B_{hb}$ value, different gate density based FBS networks have similar spatial sparsity values i.e. the amount of opportunity that exists for PG to exploit remains consistent even when the channel sparsity increases in the FBS network. This indicates that FBS and PG extract different optimization opportunities.**

FBS networks have similar spatial sparsity. Table 1 shows the results for these experiments based on Cifar10 dataset. These results indicate that for FBS networks trained for different gate densities, i.e. for different channel sparsity, the spatial sparsity that can be exploited using PG remains the same.

## 4.3 Pushing the limits of acceleration using the hybrid model

By integrating PG and FBS, we can now accelerate a model at a higher rate without sacrificing on accuracy. FBS gains efficiency by reducing the FLOPs, while PG gains efficiency by reducing the average bit width. The total reduction in MACs is approximated by the following equation for the integrated model:

$$MAC\ Reduction = DCP\ MACs\ Reduction * B/B_{avg} \tag{3}$$

where $B$ is the full precision bit for the baseline. For the paper, we select $B/B_{hb}$ as 8/4. This MAC reduction value calculated by equation 3 is just a approximated value for the true MAC saving. The reduction of bitwidth gains efficiency for the network by reducing bit calculation cost per MAC rather than number of MACs.

Figure 2 shows the results of integrating these two techniques where the hybrid model consistently achieves higher accuracy for the same MAC reduction factor than any of the other models individually.

## 5 LACK OF HARDWARE AWARENESS IN DSG TECHNOLOGIES

PG creates sparse computation that can lead to inefficient execution on a CPU. This section highlights the issue with PG computation when running on a CPU and proposes a method to fix the issue.

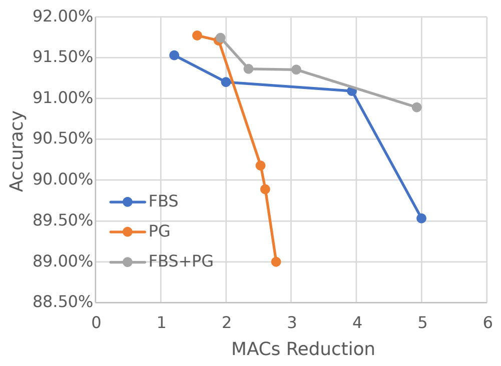
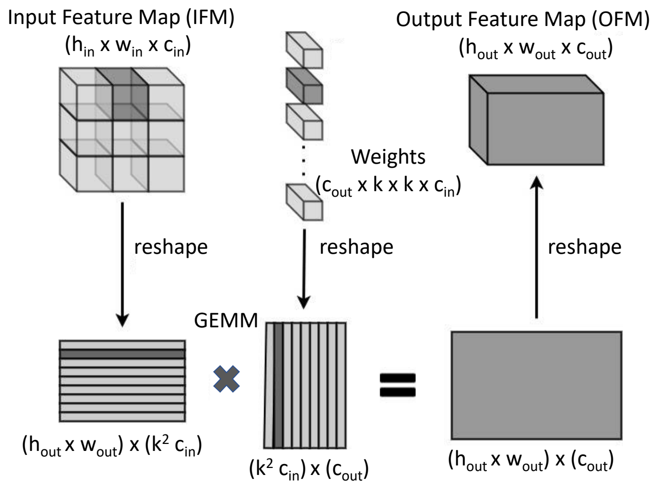


**Figure 2: Accuracy/MACs trade-off for vanilla FBS, vanilla PG, and the FBS-PG integrated model. FBS is trained for different spatial sparsity values leading to varying MAC reduction and accuracy values. PG is trained at different threshold values leading to different amount of spatial sparsity been exploited and hence different MAC and accuracy points.**

## 5.1 Background on inference on CPUs

Typically, convolutions are run using GEMM operations on a CPU. This is done via the im2col operation, that converts convolution operations into GEMM (Figure 3).



Input Feature Map (IFM)
$(h_{in} \times w_{in} \times c_{in})$
Output Feature Map (OFM)
$(h_{out} \times w_{out} \times c_{out})$
Weights
$(c_{out} \times k \times k \times c_{in})$
reshape
reshape
reshape
GEMM
$(h_{out} \times w_{out}) \times (k^2\ c_{in})$
$(k^2\ c_{in}) \times (c_{out})$
$(h_{out} \times w_{out}) \times (c_{out})$

**Figure 3: im2col operations on CPUs to convert convolution operation into GEMM**

Figure 4 shows an example of implementing the GEMM kernel using a vector compute unit (VCU) aware algorithm [3, 5] . A typical source of performance improvement for GEMM calls is matrix element reuse. Tiling (blocking) is a popular technique to increase such reuse. Tiling is also used at an instruction granularity [3, 5] using a VCU aware matrix multiplication instruction (Figure 4). This is done via packing 2D matrices in vector registers. The CPU register file is loaded with a large blocks of data elements spanning multiple registers that are processed repeatedly by a sequence of

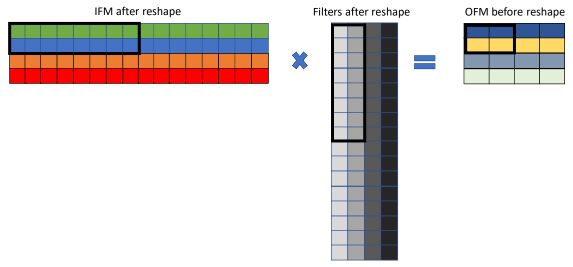


**Figure 4: The 2 × 2 output cells are partially computed through reuse via $A_{top}$ x $B_{top}$, $A_{top}$ x $B_{bottom}$, $A_{bottom}$ x $B_{top}$, $A_{bottom}$ x $B_{bottom}$, improving the overall load-to-compute ratio. In this example, 8 steps are required to fully compute C.**

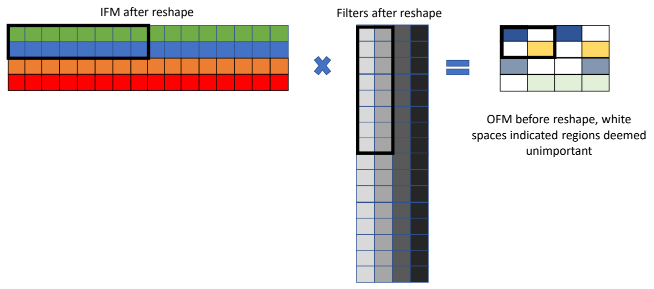

IFM after reshape
Filters after reshape
OFM before reshape, white spaces indicated regions deemed unimportant

**Figure 5: Sparsity introduced by GEMM call in the OFM. White spaces in OFM represent features in the spatial domain of each channel that are deemed unimportant**

instructions so that each register load instruction can be reused across multiple compute instructions. The vector register size of the activation and weight matrices processed by the instructions are indicated using the black bounding box in Figure 4. In Figure 4, 2 x 8 blocks of elements of the 4 x 16-element matrix structure are loaded into registers and the matrix multiplication instruction generates a 2 × 2 output cell of the output matrix C. The output of one instance of the matrix multiplication instruction generates a partial result of the output cell. In this case the multiplication of $A_{top}$ and $B_{top}$ as shown in Figure 4 generates a partial value of the 2x2 matrix in C. The final value for the output cell is computed across multiple MAC instructions by adding the results of corresponding elements derived from matrix multiplications of $A_{top}$ x $B_{top}$, $A_{top}$ x $B_{bottom}$, $A_{bottom}$ x $B_{top}$ and $A_{bottom}$ x $B_{bottom}$. The other output cells within the output matrix C can then be performed through similar calculations using different pairs of rows and columns from the loaded activation and weight matrix structures. By reusing the same set of inputs for multiple instructions, **one can improve the overall load-to-compute ratio** compared to an approach where separate load operations are required to load the operands for each individual instruction.

## 5.2 Why DSG techniques are not CPU friendly

DSG techniques like PG introduce sparsity in the OFM. For example, PG breaks the GEMM computation into two phases. In the phase 1, the upper few bits of the matrix are used to determine what parts of the OFM are important for each output channel ($\mathbf{O}_{hb}$). In phase 2, only the regions that are deemed important for each output channel are calculated, skipping everything else ($\mathbf{O}_{lb}$). This introduces fine-grained sparsity in the C matrix as shown in Figure 5. Further, each channel has a different **mask**, leading to different sparsity locations for different channel. As a result, this breaks the vectorization of the GEMM operation as discussed above. This leads to reduced reuse during GEMM kernel execution, poor utilization of VCUs and hence poor performance. Infact, the GEMM kernel reduces to execution of dot-product instruction that, while vectorizable, has poor element reuse.

## 5.3 Hardware Aware PG

In order to make PG hardware aware, we try two different strategies. PG breaks GEMM execution on CPUs because it creates a different mask for each output channel. In order to change that, we experiment with two different output mask creation strategies - a single mask for all the channels (M1) and blocking aware channel creation (M2).

The workflows for two 2D masks strategies are shown in Figure 6. M1 generates 2D mask, $mask_{2D}$, based on the reduce mean calculation over the output of the most-significant bits, $O_{hb}$, along the channel direction. The tile operation is performed on $mask_{2D}$ to generate the final 3D mask, $mask_{3D}$. Comparing with M1, M2 brings more flexibility into the model by splitting $O_{hb}$ into several groups $O_{hb\_sub}$ with the same channel number, $k$, and generates 3D masks of the corresponding subgroup, $mask_{3D\_sub}$, with the strategy M1. The final 3D mask, $mask_{3D}$, is formed by merging $mask_{3D\_sub}$ together. The M1 strategy leads to creation of a 2D-mask imposing a stringent constraint on a PG network. M2 creates a collection of VCU aware 2D-masks. For example, in the Figure 5, we will create a single mask for each block of 2 output channels. In general, if the blocking in the OFM for GEMM computation is of size $k$, we will create blocks of size $k$. Table 2 shows the accuracy of CIFAR-10 PG networks trained using vanilla PG with 3D masks, PG using 2D Masks (M1) and PG using blocking aware masks (M2). To train the CIFAR-10 network, we use the network as shown in [4] and train using cutout, random-flip and random-crop augmentation. We train the network for 200 Epochs using an inital learning rate of 0.3 and decreasing it by a factor of 10 after every 50 epochs. As shown in the Table, M1 strategy leads to a loss in accuracy, M2 strategy can preserve most of the accuracy of the vanilla PG network while also being significantly hardware performance friendly.

## 6 CONCLUSION

In this paper we show two optimizations to improve a popular Dynamic Spatial Gating technology called Precision Gating. We showed that PG leads to loss in accuracy in scenarios that require larger computation savings. We propose a method to fix this problem by identifying opportunities for optimization not explored by PG. The resultant hybrid model achieves 1.4% more accuracy for

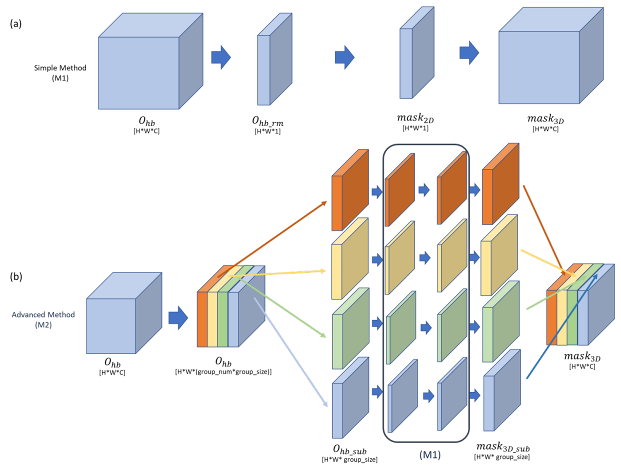


**Figure 6: The workflows for two 2D mask generators: (a) single mask for all the channels (M1) and (b) blocking aware channel creation(M2) strategies. M1 simply generates 2D mask based on the reduced mean of the most-significant bits output, $O_{hb\_rm}$, while M2 splits $O_{hb}$ into k-channel groups and applies M1 on each group to generate corresponding $mask_{2D\_sub}$.**

| Model | $\Delta$ | $k$ | $B/B_{hb}$ | sparsity | $B_{avg}$ | Accuracy |
|---|---|---|---|---|---|---|
| baseline | 0 | | 8/2 | 62.83% | 4.23 | 91.57% |
| M1 | 0 | | 8/2 | 12.23% | 7.2662 | 91.55% |
| M2 | 0 | 8 | 8/2 | 45.71% | 5.2574 | 91.83% |
| M2 | 0 | 16 | 8/2 | 38.68% | 5.6792 | 91.46% |
| M2 | 0 | 32 | 8/2 | 25.08% | 6.4952 | 91.66% |

**Table 2: Comparison of vanilla PG with 3D masks, PG using 2D Masks (M1) and PG using blocking aware masks (M2).**

the same amount of computation savings for a CIFAR-10 model. Additionally, we show that PG creates sparse computation that are not CPU friendly. We show a vector-length aware blocking technique based PG that is CPU friendly and can use SIMD based vector compute units in an embedded CPU.

## REFERENCES

[1] Colby R. Banbury, V. Reddi, M. Lam, W. Fu, A. Fazel, J. Holleman, X. Huang, R. Hurtado, D. Kanter, Anton Lokhmotov, D. Patterson, D. Pau, Jae sun Seo, Jeff Sieracki, Urmish Thakker, Marian Verhelst, and Poonam Yadav. 2020. Benchmarking TinyML Systems: Challenges and Direction. *ArXiv* abs/2003.04821 (2020).
[2] Babak Ehteshami Bejnordi, Tijmen Blankevoort, and Max Welling. 2019. Batch-Shaping for Learning Conditional Channel Gated Networks. arXiv:1907.06627 [cs.LG]
[3] Francesco Petrogalli Dan Andrei Iliescu. 2018 (accessed September 3, 2020). *Arm Scalable Vector Extensions and application to Machine Learning*. https://developer.arm.com/solutions/hpc/resources/hpc-white-papers/arm-scalable-vector-extensions-and-application-to-machine-learning
[4] Xitong Gao, Yiren Zhao, Łukasz Dudziak, Robert Mullins, and Cheng zhong Xu. 2018. Dynamic Channel Pruning: Feature Boosting and Suppression. arXiv:1810.05331 [cs.CV]
[5] Dibakar Gope, Jesse Beu, and Matthew Mattina. 2020. High Throughput Matrix-Matrix Multiplication between Asymmetric Bit-Width Operands. arXiv:2008.00638 [cs.LG]
[6] Dibakar Gope, Jesse G. Beu, Urmish Thakker, and Matthew Mattina. 2020. Ternary MobileNets via Per-Layer Hybrid Filter Banks. *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)* (2020), 3036–3046.
[7] Dibakar Gope, Ganesh Dasika, and Matthew Mattina. 2019. Ternary Hybrid Neural-Tree Networks for Highly Constrained IoT Applications. In *Proceedings of Machine Learning and Systems 2019*. 190–200.
[8] Song Han, Huizi Mao, and William J. Dally. 2015. Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding. arXiv:1510.00149 [cs.CV]
[9] Yihui He, Xiangyu Zhang, and Jian Sun. 2017. Channel Pruning for Accelerating Very Deep Neural Networks. arXiv:1707.06168 [cs.CV]
[10] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. 2017. MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. arXiv:1704.04861 [cs.CV]
[11] Jie Hu, Li Shen, Samuel Albanie, Gang Sun, and Enhua Wu. 2017. Squeeze-and-Excitation Networks. arXiv:1709.01507 [cs.CV]
[12] Weizhe Hua, Yuan Zhou, Christopher De Sa, Zhiru Zhang, and G. Edward Suh. 2018. Channel Gating Neural Networks. arXiv:1805.12549 [cs.LG]
[13] 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. *CoRR* abs/1609.07061 (2016). arXiv:1609.07061 http://arxiv.org/abs/1609.07061
[14] 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 <0.5MB model size. arXiv:1602.07360 [cs.CV]
[15] Max Jaderberg, Andrea Vedaldi, and Andrew Zisserman. 2014. Speeding up Convolutional Neural Networks with Low Rank Expansions. arXiv:1405.3866 [cs.CV]
[16] P. Judd, J. Albericio, T. Hetherington, T. M. Aamodt, and A. Moshovos. 2016. Stripes: Bit-serial deep neural network computing. In *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 1–12.
[17] Pavlo Molchanov, Stephen Tyree, Tero Karras, Timo Aila, and Jan Kautz. 2016. Pruning Convolutional Neural Networks for Resource Efficient Inference. arXiv:1611.06440 [cs.LG]
[18] H. Sharma, J. Park, N. Suda, L. Lai, B. Chau, V. Chandra, and H. Esmaeilzadeh. 2018. Bit Fusion: Bit-Level Dynamically Composable Architecture for Accelerating Deep Neural Network. In *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*. 764–775.
[19] Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. 2017. Outrageously Large Neural Networks: The Sparsely-Gated Mixture-of-Experts Layer. arXiv:1701.06538 [cs.LG]
[20] Jin Tao, Urmish Thakker, Ganesh Dasika, and Jesse Beu. 2019. Skipping RNN State Updates without Retraining the Original Model *(SenSys-ML 2019)*. Association for Computing Machinery, New York, NY, USA, 31–36. https://doi.org/10.1145/3362743.3362965
[21] U. Thakker, J. Beu, D. Gope, G. Dasika, and M. Mattina. 2019. Run-Time Efficient RNN Compression for Inference on Edge Devices. In *2019 2nd Workshop on Energy Efficient Machine Learning and Cognitive Computing for Embedded Applications (EMC2)*. 26–30.
[22] Urmish Thakker, Jesse Beu, Dibakar Gope, Ganesh Dasika, and Matthew Mattina. 2020. Rank and run-time aware compression of NLP Applications. arXiv:2010.03193 [cs.CL]
[23] Urmish Thakker, Jesse G. Beu, Dibakar Gope, Chu Zhou, Igor Fedorov, Ganesh Dasika, and Matthew Mattina. 2019. Compressing RNNs for IoT devices by 15-38x using Kronecker Products. *CoRR* abs/1906.02876 (2019). arXiv:1906.02876 http://arxiv.org/abs/1906.02876
[24] Urmish Thakker, Igor Fedorov, Jesse Beu, Dibakar Gope, Chu Zhou, Ganesh Dasika, and Matthew Mattina. 2019. Pushing the limits of RNN Compression. arXiv:1910.02558 [cs.LG]
[25] Urmish Thakker, Paul Whatmough, Matthew Mattina, and Jesse Beu. 2020. Compressing Language Models using Doped Kronecker Products. arXiv:2001.08896 [cs.LG]
[26] Xin Wang, Fisher Yu, Zi-Yi Dou, Trevor Darrell, and Joseph E. Gonzalez. 2018. SkipNet: Learning Dynamic Routing in Convolutional Networks. In *Proceedings of the European Conference on Computer Vision (ECCV)*.
[27] Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun. 2017. ShuffleNet: An Extremely Efficient Convolutional Neural Network for Mobile Devices. arXiv:1707.01083 [cs.CV]
[28] Yichi Zhang, Ritchie Zhao, Weizhe Hua, Nayun Xu, G. Edward Suh, and Zhiru Zhang. 2020. Precision Gating: Improving Neural Network Efficiency with Dynamic Dual-Precision Activations. arXiv:2002.07136 [cs.CV]