Adaptive Tiling: Applying Fixed-size Systolic Arrays To Sparse Convolutional Neural Networks

H. T. Kung
Harvard University
Cambridge, MA, USA
Email: kung@harvard.edu

Bradley McDanel
Harvard University
Cambridge, MA, USA
Email: mcdanel@fas.harvard.edu

Sai Qian Zhang
Harvard University
Cambridge, MA, USA
Email: zhangs@g.harvard.edu

***Abstract*—We introduce *adaptive tiling*, a method of partitioning layers in a *sparse* convolutional neural network (CNN) into blocks of filters and channels, called *tiles*, each implementable with a fixed-size systolic array. By allowing a tile to adapt its size so that it can cover a large sparse area, we minimize the total number of tiles, or equivalently, the number of systolic array calls required to perform CNN inference. The proposed scheme resolves a challenge of applying systolic array architectures, traditionally designed for dense matrices, to sparse CNNs. To validate the approach, we construct a highly sparse *Lasso-Mobile* network by pruning MobileNet trained with an ℓ_1 regularization penalty, and demonstrate that adaptive tiling can lead to a 2-3x reduction in systolic array calls, on Lasso-Mobile, for several benchmark datasets.**

I. Introduction

The bulk of convolutional neural network (CNN) computation is matrix multiplication [7]. To speed up the operations of matrix multiplication, one may use systolic array multipliers [9, 10]. The systolic approach has an important advantage of minimizing the I/O cost by allowing a row or column of a matrix operand to enter the processor array only once for all of its dot product computations. For this reason, systolic arrays have been utilized in some recent processor implementations for CNNs (*e.g.,* [3, 4] and Google's Tensor Processing Unit (TPU) [7]).

However, systolic matrix multipliers are traditionally designed for dense matrices, while many efficient CNN architectures, such as those derived with weight pruning methods, consist of sparse filters where a high percentage of weights are zero. For example, pruning methods such as [5] and those based on Lasso (Regression Shrinkage and Selection via the Lasso) [14], significantly reduce the resource usage of a CNN, by removing filter weights from CNN layers. The remaining nonzero weights are distributed in an unstructured manner; this leads to suboptimal efficiency when conventional systolic arrays designed for dense matrices are used, as weights with zero values are still allocated in the systolic array.

In using a given fixed-size systolic array for a large CNN, we face another issue, namely, efficient partitioning of the CNN layers into small blocks so that each block is implementable with the systolic array. We want to use a small number of these blocks in order to minimize the number of subroutine calls to the systolic array.

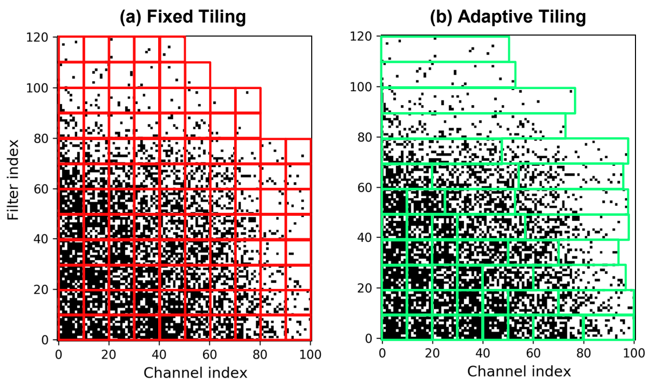


Fig. 1. A sparse filter matrix with the rows and columns sorted by the number of nonzero weights depicted under two tiling schemes. Each tile corresponds to a call to the systolic array. In (a), a 10×10 conventional systolic array can only cover 10×10 red tiles, resulting in 107 tiles required for the entire matrix. In (b), a 10×10 systolic array with adaptive tiling can cover green tiles which span a variable width based on the sparsity of the covered area, requiring only 45 tiles.

To address these challenges of implementing sparse CNN layers with a fixed-size systolic array, we propose *adaptive tiling* in this paper. We partition convolutional layer into blocks of filters and channels, called *tiles*, each implementable with a fixed-size systolic array. By allowing a tile to cover a larger area that exhibits a higher degree of sparsity, we minimize the total number of tiles, or equivalently, the number of systolic array calls. Adaptive tiling is able to cover wider sparse areas by combining multiple input data columns into a single column in the systolic array. At each cell in the systolic array, only one of the input data columns is utilized. In Section III-C, we show that the addition of multiple columns per cell adds a minor hardware cost compared to circuitry required for the Multiply-Accumulator (MAC) operations of matrix multiplication.

To illustrate the adaptive tiling approach, consider a *filter matrix* of a CNN layer, where each row is a filter in the layer and each column is a corresponding channel across all filters. Assume the CNN layer is pointwise convolution (*i.e.,* each filter is 1×1 instead of 3×3 as in standard convolution). As discussed earlier, these filters generally have a high percentage of zero weights, making the filter matrix highly sparse. Figure 1 depicts such a sparse filter matrix for the Lasso-Mobile network described in Section II-C, after the rows and columns have been sorted by the number of nonzero weights (black squares) present. The matrix multiplication for this CNN layer is the multiplication of this sparse filter matrix and an input data matrix (see Figure 4).

Partitioning the matrix multiplication for computation on a fixed-size systolic array amounts to tiling the sparse filter matrix into tiles which can each fit into the systolic array. The weights in each tile must be loaded into the systolic array and then the corresponding input data for every tile must be passed into the systolic array to perform matrix multiplication for the tile. The partial results generated by each tile are then added together to achieve the final result (the complete matrix multiplication between the filter matrix and data matrix). It is desirable to minimize the number of tiles, as this corresponds directly to the total execution time required to process the complete matrix multiplication.

Figure 1(a) illustrates that a conventional non-adaptive 10×10 systolic array can only cover 10×10 dense tiles, resulting in many tiles (107) to cover the entire matrix. In contrast, Figure 1(b) illustrates that a 10×10 systolic array with adaptive tiling can cover not only 10×10 dense tiles but also sparse tiles of larger sizes by increasing the tile width, resulting in fewer total tiles (45) for the sparse matrix. In practice, this would lead to over a 2x decrease in the runtime of the complete matrix multiplication, as it requires less than half the number of systolic array calls.

We summarize the contributions of this paper:

- An adaptive tiling method where tiles can expand to cover sparse areas of a filter matrix. As discussed in Section III, this is achieved using a systolic array design with multiple data columns per systolic cell.
- A network based on MobileNet that is trained with a simple ℓ_1 penalty, which we refer to as Lasso-Mobile. This achieves a highly sparse model compared to other approaches that aim for reduced model size.
- Analysis of the proposed adaptive tiling scheme compared against the conventional fixed tiling scheme.

II. Background

In this section, we first review Lasso-based regularization methods for reducing the number of weights in a CNN model. We then discuss systolic architectures as traditionally applied to dense matrix multiplication before describing our adaptive tiling approach to extend them to sparse filter matrices in Section III.

A. Lasso-based Methods to Reduce Model Size

Multiple papers in recent years have described methods for reducing the number of parameters in a learned CNN by applying Lasso penalties on the weights of the CNN. A simple form of this type of penalty is:

$$E(\mathbf{W}) = E_D(\mathbf{W}) + \lambda \cdot ||\mathbf{W}||_1 \qquad (1)$$

where $\mathbf{W}$ is the set of all weights in the CNN, $E(\mathbf{W})$ is the total loss, $E_D(\mathbf{W})$ is the loss with respect to the objective (*e.g.,* classification), $||\mathbf{W}||_1$ is the ℓ_1 penalty, and λ is the weight of the ℓ_1 penalty[1]. Larger values of λ lead to sparser $\mathbf{W}$. Equation 1 operates on all convolutional layers of a given CNN, denoted by $\mathbf{W}$, which could be MobileNet as describe in Section II-C. E_D is the loss over the training set. In Section IV, we use the original training set specified by each dataset.

By applying the Lasso penalty to each weight independently, the network may learn unstructured and irregular patterns of nonzero weights. While this model achieves considerable compression compared to the original CNN, the lack of structure in the nonzero weights makes it difficult to realize a corresponding speedup in efficiency on conventional hardware such as GPUs. This has lead to a multitude of approaches that aim to add some form of structure to the sparsity in order to achieve better GPU computational efficiency while still maintaining a good reduction in model size.

Among these approaches, Wen *et al.* [15] introduced the notion of structured sparsity by grouping the weights of each filter (filter-wise grouping) and the channels across all filters (channel-wise grouping) in a convolutional layer [15]. These groups are achieved by applying Group Lasso [17] to each filter or channel based on the desired sparsity structure. Group Lasso methods can lead to structures which are more accommodating to efficient implementations. However, as shown in the Table 2 of [12], without looking at computational efficiency, the sparsest model with best performance in terms of reducing the number of weights is achieved with a block size of 1×1, which is similar to Equation 1. Therefore, in this work we focus on an adaptive tiling method that is amenable to the type of sparsity seen when networks are trained with Equation 1, as they appear to achieve the best ratio between model accuracy and number of nonzero model weights.

B. Systolic Matrix Multiplier

A systolic array for matrix multiplication consists of a 2D array of systolic cells, as depicted in Figure 2 for a 3×3 array [9]. Each systolic cell is a MAC with two input and two output ports. In our design, a systolic cell has a pre-stored filter weight w. When a data item d arrives from the bottom input port, the cell computes the product $w{\cdot}d$ and adds it to an accumulating variable arriving from the left input port. After the MAC operation, the data item d enters the cell above.

Google developed the TPU in 2015 which utilizes a systolic architecture for CNN inference [7]. TPUs have been deployed in Google datacenters to perform efficient inference for CNN applications and significantly reduce the associated energy costs. While TPUs are more efficient than standard computing apparatuses, such as GPUs, they do not efficiently handle sparse filter matrices. In the worst case, each zero weight uses a cell in the systolic array (even though the MAC adds a zero to the partial result) in order to maintain synchronization of data. In Section III, we describe systolic arrays with adaptive tiling, which utilize multiple data columns per cell, in order to allow tiles to span more data columns for sparser regions of filter matrices.

[1]We use similar notation to that described in [15].

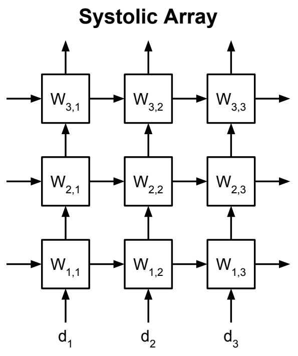


Fig. 2. A 3×3 systolic array. Each input, such as d_1, is utilized by all three cells in the corresponding column, *e.g.,* cells in the first column containing $W_{1,1}$, $W_{2,1}$, and $W_{3,1}$.

C. MobileNet and Lasso-Mobile

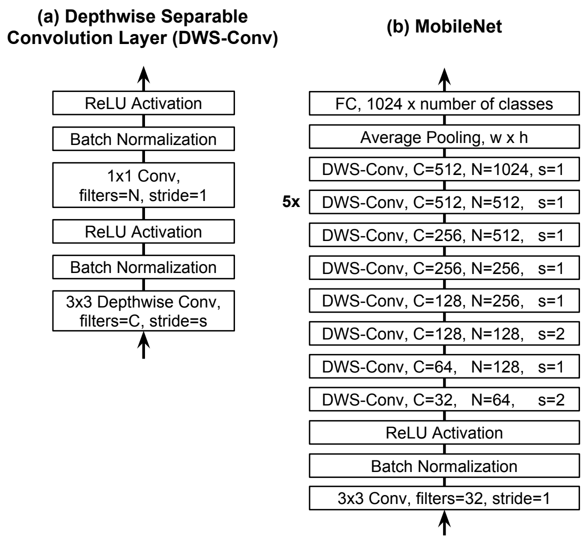
Based on depthwise separable convolution [1], MobileNet [6] has achieved state-of-the-art model compression results. Figure 3 shows the depthwise separable convolution layer in (a) and the MobileNet architecture in (b), as outlined in [6], which is used for all experiments in Section IV. In a MobileNet, computation at each convolutional layer is accomplished by applying depthwise 2D convolutions to each channel, followed by pointwise convolution over all channels. The pointwise convolution has a cost proportional to the number of filters in the layer, while depthwise 2D convolutions do not. Thus, pointwise convolution typically dominates the total cost (*e.g.,* 95% of total Multiply-Adds). For this reason, we focus on applying the proposed adaptive tiling scheme to pointwise convolution layers.

In order to achieve a MobileNet with sparse pointwise filter matrices, we apply the ℓ_1 penalty as shown in Equation 1. The regularization leads to a high percentage of weights (*e.g.,* 80%) being set to zero. We refer to the resulting network as Lasso-Mobile. As we show in Section IV, Lasso-Mobile is able to significantly reduce the required number of weights when compared to MobileNet. This savings translates directly into fewer tiles when using the adaptive tiling method on the sparse filter matrices.

D. Systolic Array for Pointwise Convolution

As shown in Figure 4, we can express pointwise convolution in a depthwise separable layer as matrix multiplication between a filter matrix F and data matrix D, where each row in F is composed of weights of a filter over the channels, and each column of D is composed of results of depthwise 2D convolutions over the channels for a point in the feature map.

We use a 2D systolic array (Figure 2) for the matrix multiplication in the the pointwise convolution, where the array pre-stores the filter matrix F, with one weight in each systolic cell. As mentioned earlier, this systolic approach has an important advantage in that each column of the D matrix enters the systolic array only once for its dot product computations with all the filters in F, thereby minimizing the



(a) Depthwise Separable Convolution Layer (DWS-Conv)
ReLU Activation
Batch Normalization
1x1 Conv, filters=N, stride=1
ReLU Activation
Batch Normalization
3x3 Depthwise Conv, filters=C, stride=s
(b) MobileNet
FC, 1024 x number of classes
Average Pooling, w x h
DWS-Conv, C=512, N=1024, s=1
5x
DWS-Conv, C=512, N=512, s=1
DWS-Conv, C=256, N=512, s=1
DWS-Conv, C=256, N=256, s=1
DWS-Conv, C=128, N=256, s=1
DWS-Conv, C=128, N=128, s=2
DWS-Conv, C=64, N=128, s=1
DWS-Conv, C=32, N=64, s=2
ReLU Activation
Batch Normalization
3x3 Conv, filters=32, stride=1

Fig. 3. A Depthwise Separable Convolution layer (DWS-Conv), in (a), consists of a depthwise convolution with C 3×3 filters and a stride of s followed by standard convolution with N $1 \times 1 \times C$ filters, which is referred to as pointwise convolution. MobileNet, in (b), begins with a standard convolution layer followed by 12 depthwise separable convolution layers, average pooling across the spatial dimensions (width and height), and finally a Fully Connected (FC) layer. The **5x** denotes that the layer is repeated five times.

I/O cost. For filter matrices larger than the fixed size of the systolic array, we run the systolic array in multiple passes, one for each tile.

Note that the filter matrix F for Lasso-Mobile is sparse, as Figure 1 illustrates, with a high percentage of weights being zeros. As we will describe in the following section, the proposed adaptive tiling approach can adapt the width of each tile based on the sparsity, therefore requiring fewer tiles overall.

III. Overview of the Adaptive Tiling Approach

When a given fixed-size systolic array is too small to fit an entire filter matrix, such as the case for the pointwise layers in Lasso-Mobile, we use *adaptive tiling* to partition the matrix into tiles as illustrated by Figure 1. Tiling is adaptive in the sense that tiles may be of varying widths so that a wide tile can be used when covering a sparse area. The objective is to minimize the total number of tiles, as each tile requires a pass through the systolic array.

We achieve the tile adaptation by allocating multiple data columns, *i.e.,* the corresponding columns of the filter matrix (channels), to a single column of the systolic array. For a region in the filter matrix with a higher degree of sparsity, more data columns can be allocated to a single column, resulting in a wider tile. There are several considerations affecting the width of a tile, including the systolic array width (number of systolic cells in a row), the sparsity of the region in the filter matrix, the number of data columns allowable for each systolic cell, and the number of conflicts permitted between those columns

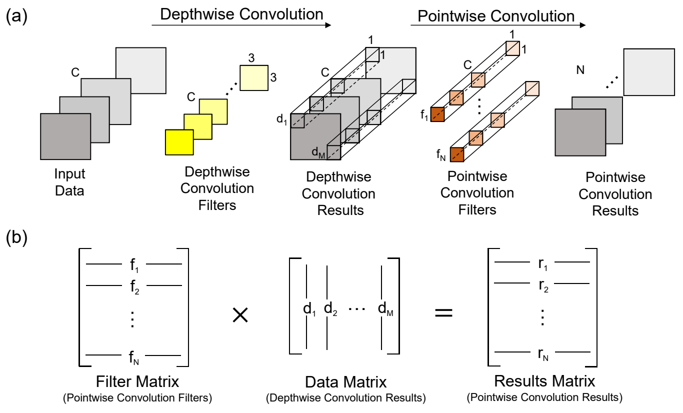

(a)
Depthwise Convolution
Pointwise Convolution
Input Data
Depthwise Convolution Filters
Depthwise Convolution Results
Pointwise Convolution Filters
Pointwise Convolution Results
(b)
Filter Matrix
(Pointwise Convolution Filters)
Data Matrix
(Depthwise Convolution Results)
Results Matrix
(Pointwise Convolution Results)

Fig. 4. Depthwise Separable Convolution consists of Depthwise Convolution with C 3×3 filters, followed by Pointwise Convolution with N $1 \times 1 \times C$ filters, as depicted in (a). Pointwise convolution between the depthwise convolution result and the pointwise convolution filters can be computed as a matrix multiplication between the filter matrix and the data matrix. In (b), each row f of the filter matrix F corresponds to a $1 \times 1 \times C$ pointwise convolution filter, and each column d of the data matrix D corresponds to the $1 \times 1 \times C$ vector formed by concatenating each element of depthwise convolution results across the channels.

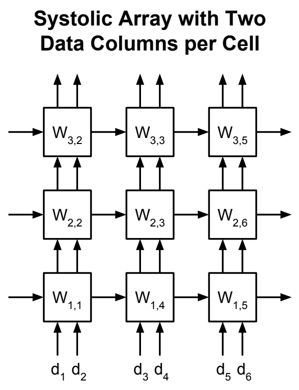


Fig. 5. A 3×3 systolic array with two data columns per cell. The first systolic array column takes data columns d_1 and d_2, and uses d_1 in row 1 and d_2 in rows 2 and 3. d_1 is not needed in rows 2 and 3 as the corresponding filter weights ($W_{2,1}$ and $W_{3,1}$) are zero.

of the filter matrix which share the same systolic array column. We elaborate on these issues in the following section.

A. *Adaptive Tiling via Multiple Columns per Cell*

Adaptive tiling enables a fixed-size systolic array to span more data columns than physical columns in the array by passing multiple data columns into a single systolic array column. When a filter matrix is sparse, only a subset of the data columns are typically used in each row. Therefore, in this case, multiple data columns can be combined into a single physical column in the systolic array as they are never used at the same time by any row (*i.e.,* only one weight is nonzero per row across the multiple data columns).

Figure 5 shows a 3×3 systolic array with two data columns per systolic array column. For example, for the first column which takes d_1 and d_2, the two data columns are passed into the lower left cell in the array simultaneously. However, only d_1 is used in the cell which stores $W_{1,1}$. The weight $W_{1,2}$ corresponding to d_2 is zero and therefore not required. Each additional cell in the systolic array column selects between d_1 and d_2 in this manner based on which corresponding weight is nonzero. Thus, when each systolic array column contains two data columns, the adaptive tiling scheme is able to double the width of the tile implementable by the fixed-size array (from 3 data columns to 6 data columns as depicted in Figure 5).

B. *Selecting Columns to Be Combined*

In order to combine two or more data columns into a single systolic array column, multiple data columns can never be required by the same row, as only a single MAC is implemented within each cell. If multiple data columns have nonzero weights in the same row, then all but one with the largest magnitude must be omitted from the result for the corresponding row. Therefore, ideally we aim to combine columns that do not conflict in any row in the sparse filter matrix.

However, in practice, we allow a small amount of overlapping use between data columns, such as 2-3 conflicts, in order to have wider tiles. The small amount of overlap does not significantly impact the result of CNN inference due to robustness of the network to small perturbations. In this case, a conflict can be viewed as a form of pruning, where the weight with the smaller absolute value is pruned. For the empirical results shown in Section IV, we allow for up to 3 conflicts between data columns which share a single column of the systolic array. If more than 3 conflicts exist, then these data columns are allocated into separate systolic array columns.

C. *Cost of Multiple Data Columns per Cell*

Having given a high-level description of the systolic cells used in the adaptive tiling approach, we now provide a brief analysis on the cost of augmenting the MAC function of a systolic cell with multiple data columns. Suppose that the MAC is 16-bit and uses a bit-serial implementation. This MAC requires at least 16 1-bit full adders to implement the multiplication in a bit serial fashion. A multiplexer is required within each cell in order to select the data column that corresponds to the weight stored within the MAC. Compared to the MAC, the number of gates required to implement the multiplexer is substantially smaller (only 4 gates for a 2-to-1 multiplexer corresponding to two data columns per cell). For highly sparse filter matrices (*e.g.,* under 5% nonzeros), more than two data columns could be added into each cell to span the even sparser regions. Therefore, the adaptive tiling approach can efficiently accommodate highly sparse CNNs such as Lasso-Mobile, resulting from aggressive weight pruning with Lasso regularization.

IV. Evaluation

In this section, we analyze the performance of adaptive tiling over fixed tiling for sparse filter matrices in Lasso-

Mobile. First, we show that Lasso-Mobile achieves a much smaller number of nonzero weights, up to a 57x reduction, while still maintaining similar classification accuracy compared to standard MobileNet. Then, we examine how the unstructured sparse weights in the pointwise layers of Lasso-Mobile are amenable to the proposed adaptive tiling approach which provides a 2-3x reduction in required tiles, or systolic array passes, over a conventional fixed tiling approach.

A. Evaluation Setup and Datasets

We evaluate the performance of the MobileNet and Lasso-Mobile models on three datasets of varying difficulty. Fashion-MNIST [16] is a dataset that aims to provide an alternative to MNIST [11] as a baseline dataset for evaluation. It has the same image resolution of $1 \times 28 \times 28$ and the same number of samples as MNIST, but is significantly harder as the samples are clothing items instead of digits. CIFAR-10 [8] is a more challenging RGB image dataset with resolution of $3 \times 32 \times 32$. Tiny ImageNet [2] is a subset of the ILSVRC-2012 ImageNet [2] dataset, containing 200 of the 1000 classes, with 500 training samples per class, downsampled to a resolution of $3 \times 64 \times 64$. We use a Top-5 accuracy measure for Tiny ImageNet as is a common practice with ImageNet. MobileNet is trained using Stochastic Gradient Descent (SGD), with a learning rate of 0.02, and a Nesterov momentum of 0.9 [13].

Lasso-Mobile is trained in a similar fashion, but initialized with the learned weights of the MobileNet model. The ℓ_1 penalty uses λ of 0.0005 (see Equation 1) for all Lasso-Mobile networks. After training the Lasso-Mobile network, weights smaller than 0.002 are set to zero. If an entire filter or channel is zero it is pruned from the network before tiling occurs.

B. Impact of Model Size on Classification Accuracy

In the MobileNet paper, the authors mention that a simple method for reducing the number of model parameters is to decrease the number of channels in the pointwise layers, as they represent the majority of the model weights in the network. Figure 6 compares the MobileNet and Lasso-Mobile networks while varying number of channels. Each Lasso-Mobile network is trained using the corresponding reduced channel MobileNet network and achieves a significantly reduces model size while maintaining similar accuracy. The largest network (baseline MobileNet shown in Figure 3), contains over 2 million weights in the pointwise layers (the rightmost point on the red curve in Figure 6 (b)). The corresponding Lasso-Mobile network contains only 35K weights for the Tiny ImageNet dataset (the highest point on the blue curve in Figure 6 (b)), or a 57x reduction in nonzero weights compared to the MobileNet model. Note that the final two points on the Lasso-Mobile curve have a similar number of nonzero weights after pruning due to the Lasso penalty, even though the corresponding original MobileNet models on the red curve differ by a factor of 2x in number of nonzero weights.

This reduction in weights for the Lasso-Mobile networks translates directly into a reduction in the number of systolic

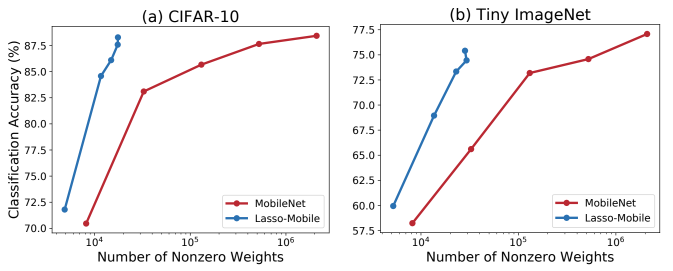


Fig. 6. Comparing the performance of MobileNet under different numbers of channels (as suggested in [6] as a way of reducing model size) to Lasso-Mobile for the CIFAR-10 and Tiny ImageNet datasets. The x-axis is the total number of nonzero weights in all pointwise layers in log scale. Each point on the MobileNet curve is used as the initial model to train a corresponding point on the Lasso-Mobile curve.

array passes required to implement Lasso-Mobile over MobileNet. This shows the benefit of using Lasso even when the starting model, MobileNet, is already considered to have a small number of weights. Additionally, comparing the lowest points for both the MobileNet and Lasso-Mobile curves in Figure 6 (b), the Lasso-Mobile network achieves a 1.4x decrease in number of nonzero weights, from 8.2K to 5.8K, while maintaining similar accuracy. This suggests that even smaller pointwise filter matrices have unnecessary weights, as they can be pruned without a loss of accuracy.

C. Number of Nonzero Weights per Pointwise Layer

Figure 7 shows the number of nonzero weights in each pointwise layer of Lasso-Mobile for all the datasets. Even though all three networks start with the same number of parameters, over 2 million total weights in the pointwise layers, they each achieve a different total number of nonzeros. Tiny ImageNet uses substantially more nonzero weights in the last layers, which is most likely due to the much larger number of classes (200 versus 10 for Fashion-MNIST and CIFAR-10). Additionally, the distribution of nonzeros is different for each network. For instance, the CIFAR-10 model has significantly more nonzeros in the 5th and 6th pointwise layer than the Tiny ImageNet model, and significantly less nonzeros in the last layers. This indicates that the optimal number of filters per layer is dependent on the dataset, and would generally require extensive hyper-parameter search to find.

D. Efficiency Gains of Adaptive Tiling

Figure 8 shows the efficiency gains of using adaptive tiling over fixed tiling for each pointwise layer for the Lasso-Mobile network trained for CIFAR-10 in (a) and Tiny-ImageNet in (b). By using adaptive tiling, we are able to make fewer systolic array calls by creating tiles which cover wider sparse regions as compared to fixed tiling. The gain in efficiency is measured by the number of tiles in adaptive tiling over that for fixed tiling.

We see that the adaptive tiling method substantially outperforms the fixed tiling method by a factor of 2x to 3x for all

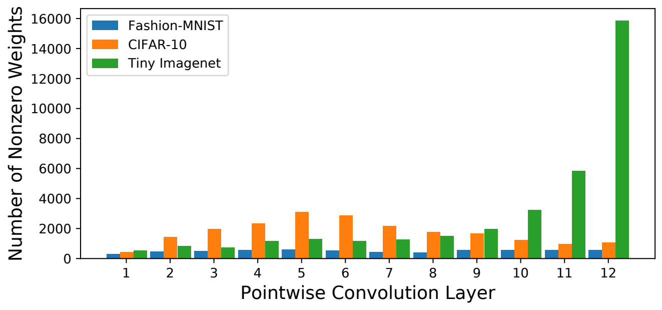


Fig. 7. The number of nonzero weights in each pointwise convolution layer for the three evaluation datasets using Lasso-Mobile. The total number of nonzeros for all pointwise convolution layers is 6K for Fashion-MNIST, 21K for CIFAR-10, and 35K for Tiny ImageNet.

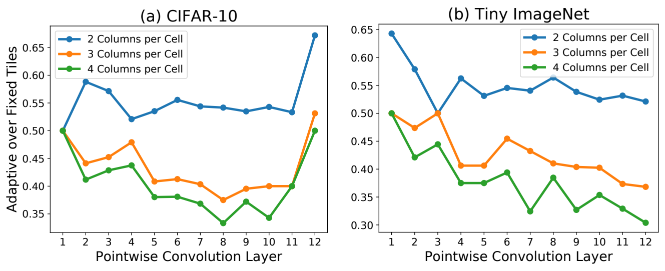


Fig. 8. The ratio of required systolic array calls for the adaptive tiling scheme over the fixed tiling scheme for each pointwise layer in CIFAR-10 Lasso-Mobile model (a) and Tiny-ImageNet Lasso-Mobile model (b). The lines denote different number of data columns per cell (2, 3, or 4) within each systolic cell using adaptive tiling. For all settings, a systolic array size of 10×10 is used, and three conflicts are allowed among data columns sharing a systolic array column, as discussed in Section III-B.

pointwise layers. The benefit is slightly reduced in the earlier layers, which are considerably smaller and denser than the latter layers. The latter layers see the largest benefit, as they are larger and significantly sparser. Increasing the number of data columns per cell from 2 to 4 reduces the number of tiles from a factor of 2x to 3x. This suggests that allocating more columns in each systolic cell, while making the implementation slightly more expensive, leads to adaptive tiling which can fit much larger sparse areas, and therefore require fewer array calls overall.

V. Conclusion

In this paper we consider the problem of using a given fixed-size systolic array for inference on large sparse CNNs. To minimize the number of systolic array calls, we take a two-step approach. First, we use Lasso with a ℓ_1 penalty to prune an exiting network such as MobileNet. Second, we use adaptive tiling to partition the resulting network such as Lasso-Mobile for implementation with the fixed-size systolic array.

We demonstrate that the first step can lead to a significant, up to 57x, decrease in the number of weights, and thus calls to the systolic array. However, the remaining weights are distributed in an unstructured manner, making efficient inference with conventional systolic arrays designed for dense matrices (fixed tiling) difficult. The second step addresses this issue by using adaptive tiling, which enables a fixed-size systolic array to span a variable width based on the sparsity of the tile. This is enabled by modifying each cell in a systolic array to take two or more data columns. We argue that the additional hardware required to implement multiple data columns per cell adds a relatively small cost compared to the MACs. We demonstrate that adaptive tiling leads to a 2-3x decrease in the number of required systolic array calls compared to fixed tiling.

With these results, we conclude that it is feasible to achieve high efficiency when applying fixed-size systolic arrays to sparse CNNs derived from aggressive weight pruning.

Acknowledgment

This research is supported in part by a gift from MediaTek USA, Inc.

References

[1] F. Chollet. Xception: Deep learning with depthwise separable convolutions. *arXiv preprint arXiv:1610.02357*, 2016.
[2] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 248–255. IEEE, 2009.
[3] Z. Du, R. Fasthuber, T. Chen, P. Ienne, L. Li, T. Luo, X. Feng, Y. Chen, and O. Temam. Shidiannao: Shifting vision processing closer to the sensor. In *ACM SIGARCH Computer Architecture News*, volume 43, pages 92–104. ACM, 2015.
[4] C. Farabet, B. Martini, B. Corda, P. Akselrod, E. Culurciello, and Y. LeCun. Neuflow: A runtime reconfigurable dataflow processor for vision. In *Computer Vision and Pattern Recognition Workshops (CVPRW), 2011 IEEE Computer Society Conference on*, pages 109–116. IEEE, 2011.
[5] 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.
[6] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.
[7] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers, et al. In-datacenter performance analysis of a tensor processing unit. In *Proceedings of the 44th Annual International Symposium on Computer Architecture*, pages 1–12. ACM, 2017.
[8] A. Krizhevsky, V. Nair, and G. Hinton. The cifar-10 dataset, 2014.
[9] H. T. Kung. Why systolic architectures? *IEEE Computer*, 15:37–46, 1982.
[10] H. T. Kung and C. E. Leiserson. Systolic arrays (for vlsi). In *Sparse Matrix Proceedings 1978*, pages 256–282. Society for Industrial and Applied Mathematics, 1979.
[11] Y. LeCun, C. Cortes, and C. J. Burges. The mnist database of handwritten digits, 1998.
[12] S. Narang, E. Undersander, and G. F. Diamos. Block-sparse recurrent neural networks. *CoRR*, abs/1711.02782, 2017. URL http://arxiv.org/abs/1711.02782.
[13] S. Ruder. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 2016.
[14] R. Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 267–288, 1996.
[15] 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.
[16] H. Xiao, K. Rasul, and R. Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms, 2017.

[17] M. Yuan and Y. Lin. Model selection and estimation in regression with grouped variables. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 68(1):49–67, 2006.