2021 31st International Conference on Field-Programmable Logic and Applications (FPL) | 978-1-6654-3759-2/21/$31.00 ©2021 IEEE | DOI: 10.1109/FPL53798.2021.00060

Leveraging Fine-grained Structured Sparsity for CNN Inference on Systolic Array Architectures

Linqiao Liu
Department of ECE, University of Toronto
Toronto, Ontario, Canada
linqiao.liu@mail.utoronto.ca

Stephen Brown
Department of ECE, University of Toronto
Toronto, Ontario, Canada
brown@eecg.toronto.edu

***Abstract*—The high computational complexity of convolutional neural networks (CNNs) has motivated many studies of accelerating CNN inference on field-programmable gate arrays (FPGAs). Among these, designs that feature systolic arrays can effectively leverage the parallelism in CNNs while acheiving good placement and routing quality. Weight sparsity — the presence of zeros in CNN weights — can further reduce the number of necessary multiply-accumulate (MAC) operations in CNNs, but has yet resulted in performance gain on systolic arrays. In this work, we propose a novel fine-grained structured weight sparsity pattern, showcase a processing element (PE) design that leverages this sparsity pattern, and develop a systolic array CNN inference accelerator that targets an Intel Arria 10 GX1150 FPGA. When evaluated on ResNet-50 and VGG-16 that are trained and pruned on the ImageNet dataset, our accelerator achieves 2.26 TOPs/s and 1.21 TOPs/s, respectively, on the MAC operations, while keeping the top-1 accuracy degradation within 5%. These results translate to 2.86× and 1.75× speed-up compared to a dense systolic array baseline.**

***Index Terms*—Deep Learning, Convolution Neural Networks, Sparsity, Weight Pruning, Systolic Array, CNN Inference Acceleration, FPGA**

I. Introduction

Convolutional neural networks (CNNs) have demonstrated superior accuracy on various computer vision tasks such as image classification and object detection [1]–[3]. Nevertheless, they are computationally expensive and require large memory footprint, so it remains challenging to deploy CNNs for latency sensitive applications. Recent literature has revealed that many connections in CNNs are redundant, and the weights on the redundant connections can be set to zero via *pruning* without severe accuracy degradation [4]–[6]. Weight sparsity allows CNN accelerators to access weights in compressed formats and to skip ineffectual multiply-accumulate (MAC) operations, thus potentially achieving better performance.

However, exploiting weight sparsity requires non-trivial hardware support to reap gains in terms of speed and energy efficiency. Recent studies [7]–[9] reveal the two challenges facing implementing sparse neural networks on FPGAs. First, without any structural constraint, sparse weights are located randomly. As shown in Fig. 1, when pruned filters with uneven distribution of non-zero weights are mapped to different MAC units, the speed-up is limited by the filter that has the most weights. Second, expensive routing and staging logic is required to either gather the sparse weights and neurons that can be multiplied together, or to scatter the products resulted from multiplying sparse operands to the correct accumulators.

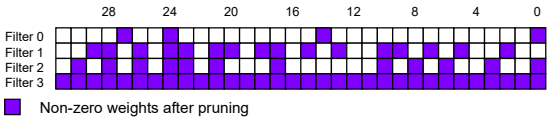


Fig. 1: A typical weight sparsity pattern produced by unstructured pruning.

In this work, we propose a fine-grained weight pruning structure, called Micro-range Cluster Bank-Balanced Sparsity (MCBBS), to lower the hardware cost of supporting weight sparsity on FPGAs, and to improve load balance across MAC units. Next, we design a processing element (PE) that supports MCBBS and parallel MAC operations. Finally, we implement an end-to-end sparse CNN inference accelerator that features a 2D systolic array of MCBBS PEs on an Arria 10 GX1150 FPGA. Thanks to the reduction in MAC operations and the dedicated hardware support for weight sparsity, our sparse CNN inference accelerator achieves 2.26 TOPs/s and 1.21 TOPs/s on the MAC operations in VGG-16 [1] and ResNet-50 v1.5 [2], respectively, while keeping the top-1 validation accuracy degradation on ImageNet [10] within 5%. The design files are available at [11] and [12].

II. Related Work

Several FPGA-based inference accelerators for dense CNNs have been proposed, such as in [13] and [14]. When implemented on an Arria 10 GX1150 FPGA, the most performant accelerators from both works use more than 95% of the DSP blocks at high efficiency. To push beyond the performance limited by the number of DSPs, this work extends the 2D systolic array architecture in [13] by adding weight sparsity support.

Several works have also leveraged weight sparsity for neural network acceleration on FPGAs. ESE is an inference accelerator that targets weight sparsity in long short-term memory neural networks (LSTMs), which are dominated by matrix-vector multiplications [7]. Its sparse matrix-vector multiplication performance performance is dampened by workload imbalance across MAC units. To even the workload in sparse

978-1-6654-3759-2/21/$31.00 ©2021 IEEE
DOI 10.1109/FPL53798.2021.00060

LSTMs, Cao et. al. propose Bank-Balanced Sparsity (BBS), a fine-grained structured sparsity pattern [8]. In [9], the authors propose an accelerator for sparse CNNs. The design requires complex logic to support unstructured sparsity, leading to less than 50% DSP utilization even when all the logic elements have been consumed. In this work, we modify BBS in order to leverage it in a 2D systolic array of PEs for CNN inference acceleration.

III. Accelerator Design and Weight Sparsity Support

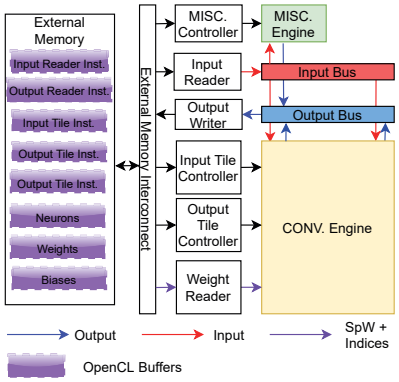


Fig. 2: CNN inference accelerator system architecture.

Our inference accelerator consists of *kernels* that are implemented using Intel FPGA SDK for OpenCL and are connected by FIFO channels, as shown in Fig. 2. Dedicated buffers for neurons, weights, and kernel instructions are placed in the accelerator's external DDR4 memory. A CPU host controls the accelerator via PCI/e bus.

The accelerator supports all the layers in VGG-16 and ResNet-50 in INT8 precision with quantization levels at integer powers of two. CONV and FC layers are mapped to the CONV Engine, and other layers are mapped to the MISC Engine. Batch-normalization layers are fused with preceding CONV layers, following the same procedure as in [15]. Average-pooling layers' normalization factors are constrained to be integer powers of two.

For a given CNN, the host prepares the kernel instructions and sparse weights offline, then streams them to the buffers. At the start of an inference call, the host transfers input neurons to the Neurons buffer, and launches the accelerator kernels. The input reader moves neurons from the Neurons buffer to the CONV Engine and the MISC Engine. The output writer streams output neurons back to the Neurons buffer. In order to synchronize the data accessed by input reader and output reader, the host waits for both kernels to finish executing the instructions for one layer before launching them for the next layer.

In the following sections, we focus on the acceleration of CONV and FC layers using weight sparsity.

A. CONV Engine Architecture

CONV and FC layers form the backbone of CNNs. Each CONV layer maps *IC* channels of input *feature maps* that has height and width (*M*, *N*) to *OC* channels of output feature maps that has height and width (*P*, *Q*). As shown in Equation 1, to compute an output neuron at coordinates (p, q) on output feature map f, we perform MAC operations between the filter $W^{f,c}$ — which has dimensions (K_H, K_W, IC) and is translated across the width and height of the input tensor at stride S — and the input neurons from the convolution window, $I^c_{(p-1)\cdot S+kh,(q-1)\cdot S+kw}$. Then, we add the bias b_f to the MAC result, and pass the sum through the activation function h. An FC layer can be regarded as a special case of CONV layer, where $K_H = P = M$, and $K_W = Q = N$.

$$Y^f_{p,q} = h(\sum_{kh}^{KH}\sum_{kw}^{KW}\sum_{c}^{IC} W^{f,c}_{kh,kw} \cdot I^c_{(p-1)\cdot S+kh,(q-1)\cdot S+kw} + b_f) \quad (1)$$

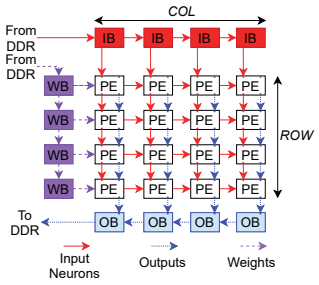
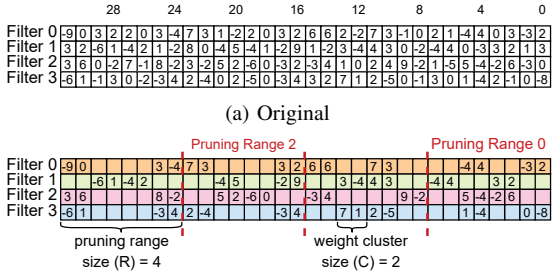


Fig. 3: Systolic array architecture.

To exploit the parallelism in CONV and FC layers, the CONV Engine features a 2D systolic array of processing elements (PEs) that are connected by FIFO channels, as shown in Fig. 3, similar to the architecture proposed in [13]. The regular 2D shape and the daisy-chain connections are amenable to a good-quality placement and routing solution on FPGAs. The weight buffers (*WBs*) and input neuron buffers (*IBs*) stream data to the PEs along rows and columns, effectively realizing data reuse. Each PE performs IC_{unroll} MACs in parallel and updates the partial sum stored in a private register. Meanwhile, the output depth and width dimensions are unrolled along the rows and columns of the systolic array, such that neurons from ROW output feature maps and COL output columns are processed concurrently. Each PE sends the partial sum along the drainage daisy-chain to the output neuron buffer (*OB*) on the same column, which applies the activation function to the MAC results before streaming them to the off-chip memory.

B. Micro-range Clustered Bank-Balanced Sparsity

To effectively leverage weight sparsity in the 2D systolic array of PEs, we propose **Micro-range Clustered Bank-Balanced Sparsity (MCBBS)**, a regularly structured weight sparsity pattern. Consider the dense filters with shape (K_H, K_W, IC) = (1, 1, 32), as shown in Fig. 4a. In order to apply



(a) Original

(b) MCBBS

Fig. 4: An example of pruning four filters with MCBBS.

MCBBS to the filters, in Fig. 4b we partition each filter into *pruning ranges* along its input depth. Then, we group adjacent elements in each pruning range into *clusters* of size C. The size of each pruning range is counted using the number of clusters it contains, R. Next, we rank the clusters in each pruning range according to their L_1 norms in descending order, and perform pruning at sparsity level S, leaving only the top $\lfloor S \cdot R \rfloor$ clusters in each pruning range, while setting the rest to zero. After applying MCBBS with settings ($C = 2$, $R = 4$) and sparsity level at 50%, we obtain the result shown in Fig. 4b.

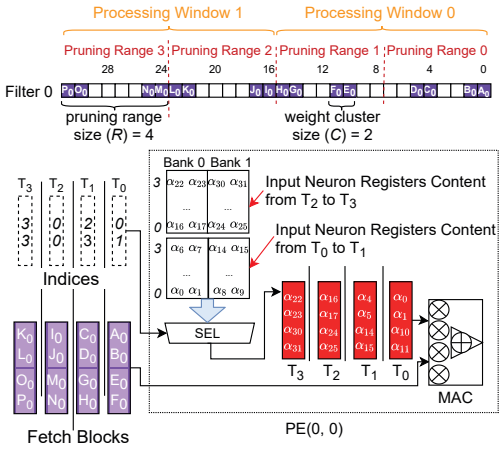


Fig. 5: Processing a filter pruned with MCBBS settings *(C=2, R=4)* in a PE with 4 multipliers. Pipeline registers and control logic are omitted.

Fig. 5 presents an MCBBS PE, and demonstrates the computation of an output neuron using a 50% sparse MCBBS filter over four clock cycles. The PE consists of a front-end that selects dense neurons and matches them with sparse weights, and a back-end that performs 4 MAC operations on the operands gathered by the front-end in one cycle. We rearrange the weights in the sparse filter so that all 4 multiplier lanes can be kept busy. Specifically, every $P = 2$ adjacent pruning ranges are grouped into *processing windows*. In general, P is chosen such that $P \cdot C$ equals the number of multiplier lanes. Next, we package weight clusters into *fetch blocks*. The first fetch block consists of the first non-zero weight clusters from the pruning ranges in the first processing window, and the second fetch block consists of the second non-zero weight clusters from the same pruning ranges. We iterate through all processing windows in this order to construct the fetch blocks as shown in the lower left of Fig. 5.

To help the PE to fetch matching neurons for each non-zero weight cluster, we index each retained cluster using its position in the pruning range that it is extracted from. The number of bits required for each index is $\lceil \log_2 R \rceil$. By increasing the cluster size C, the routing and memory overhead required for the indices can be modestly reduced.

In each PE, input neurons from the same processing window are buffered in registers that are arranged in P banks. Each bank consists of R physical addresses and is C-values wide. Neurons for each cluster of C multiplier lanes are selected from one register bank via a R:1 multiplexer. The select signals are the weight indices. As the first weight fetch block of a processing window arrives, all the neurons from the same processing window are loaded into the registers, such as in cycles T_0 and T_2.

MCBBS is equivalent to BBS when $C = 1$, and $R = \lceil \frac{IC}{P} \rceil$, but it differs from BBS in two important ways. First, due to weight clustering, the number of indices in each filter pruned with MCBBS is $\frac{1}{C}$ of the number of indices required by BBS. Second, BBS requires each PE to buffer as many neurons as the depth of the convolution window, thus consuming a large amount of logic elements. We do not consider implementing the input buffers using hardened memory blocks (M20Ks), since they are reserved for the other on-chip buffers. To scale up the computation ceiling of an array of BBS PE, the only viable way is to increase the intra-neuron parallelism per PE, IC_{unroll}. In contrast, each MCBBS PE only needs to buffer the input neurons from the same processing window, lowering the FPGA resource consumption per PE. Thus, we gain the flexibility to adjust the systolic array's 2D topology.

C. PE Coalescing

To further save logic elements, we group sparse PEs in the same systolic array columns into groups of size G, and factor out the input neuron registers, control logic, and drainage logic from each PE, as shown in Fig. 6. The reduction in logic element consumption improves the scalability of the systolic array. By carefully choosing the group size G, we can limit the fan-out while achieving logic elements saving.

IV. Experimental Results

A. CNN Pruning and Quantization

To investigate the trade-off between inference accuracy and arithmetic operations reduction due to pruning, in PyTorch v1.6 [16] we apply pruning followed by quantization-aware training to VGG-16 and ResNet-50 v1.5[1] with two different

[1]There are two sets of differences between our model and ResNet-50B in [2]. First, some residual blocks in ResNet-50B have 1 × 1 convolution with stride 2, but in ResNet-50 v1.5 the stride of 3 × 3 CONV layers in these blocks are set to 2, and the 1 × 1 convolution strides are set to 1. Second, we set the denominator of the average pooling layer to 64.

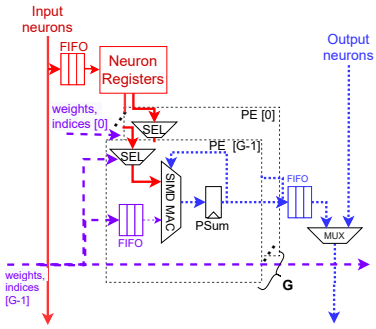
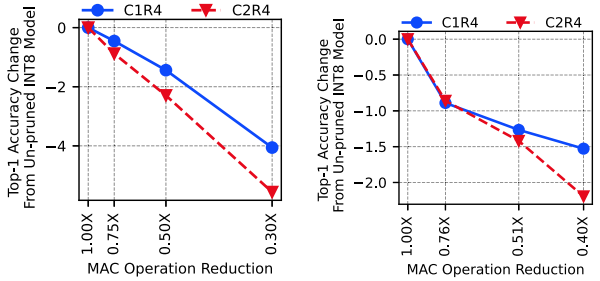


Fig. 6: Coalescing PEs in the same systolic array columns into groups. Pipeline registers are omitted.



(a) VGG-16 (b) ResNet-50 v1.5

Fig. 7: Trade-off between top-1 classification accuracy and MAC operations reduction achieved by pruning the CNNs with MCBBS at configurations *(C=1, R=4)* and *(C=2, R=4)*.

MCBBS configurations on ImageNet: *(C, R)* = (2, 4) and *(C, R)* = (1, 4). Specifically, we iteratively prune the CNNs at 25%, 50%, and 75% layer-wise sparsity with fine-tuning. Inspired by the observation that the accuracy of ResNet-34 on ImageNet is sensitive to CONV layers that are at the beginning and the end of each residual block [17], we cap the weight sparsity to 50% on CONV layers where the number of input channels differ from the number of output channels. Moreover, we do not prune any input layer. Finally, we obtain the top-1 validation accuracy of CNNs by performing inference on the accelerator. As shown in Fig. 7, weight clustering (*C=2*) yields slightly worse accuracy than treating each weight individually during pruning (*C=1*). Moreover, ResNet-50 v1.5 is more robust to MCBBS pruning than VGG-16 at the same operation reduction level. We suspect that the *short-cut connections* in ResNet-like CNNs help fine-tuning to recover accuracy after pruning.

B. Inference Accelerator Performance

We evaluate the sparse CNN inference accelerator architecture on an Arria 10 FPGA Development Kit, which has an Intel Arria 10 GX1150 FPGA and one bank of 2 GB DDR4 memory with a peak throughput of 17 GB/s. We implement the inference accelerator using Intel FPGA SDK for OpenCL Pro 19.3.0.

In Table I, we compare three inference accelerators. We implement *Dense* as a baseline without sparsity support, similar to the design in [13]. We also implement two sparse accelerators with different PE configurations, *(C=2, R=4, P=8)* and *(C=1, R=4, P=16)*, respectively. All three accelerators have the same number of MAC units distributed across the same systolic array topology: *(ROW, COL, G) = (24, 7, 8)*. In the sparse accelerators, MCBBS support incurs moderate costs in terms of ALM ($< 15\%$) and M20K ($< 10\%$).

We evaluate the accelerators' performance on VGG-16 and ResNet-50 v1.5 that are pruned with 75% layer-wise sparsity according to IV-A. Since the accelerators are optimized for CONV and FC layers, we detail the performance on these layers in addition to their end-to-end performance. Compared to the dense baseline, the sparse accelerator with C1R4P16 PEs achieves $2.86\times$ and $1.75\times$ on the MAC operations in VGG-16 and ResNet-50 v1.5, respectively. The remaining portion in the end-to-end latency values are due to 1) memory-bound transfers for pooling and element-wise add operations, and 2) host synchronization overhead, which amounts to about 30 μs to 50 μs per layer.

TABLE I: COMPARISON OF CNN INFERENCE ACCELERATOR IMPLEMENTATIONS

	Dense	Sparse (C2R4P8)	**Sparse (C1R4P16)**
fMAX (MHz)	220	226	242
Logic (ALM)	288K (67%)	322K (76%)	336K (79%)
RAM (M20K)	1604 (59%)	1716 (63%)	1785 (66%)
DSP	1352 (89%)	1352 (89%)	1352 (89%)
V16[1] Top-1 Accuracy (%)	68.90	63.34	64.84
V16 Board Peak Power (W)	23.5	26.1	27.8
V16 Latency (ms) (CONV+FC Only)	44.71 (39.09)	19.25 (14.23)	18.62 (13.69)
V16 Throughput (GOPs/s) (CONV+FC Only)	692 (791)	1607 (2174)	1662 (2260)
R50[2] Top-1 Accuracy (%)	73.03	70.84	71.50
R50 Board Peak Power (W)	21.1	21.6	22.6
R50 Latency (ms) (CONV+FC Only)	20.62 (11.60)	17.13 (7.10)	16.53 (6.75)
R50 Throughput (GOPs/s) (CONF+FC Only)	397 (693)	478 (1152)	495 (1210)

[1] VGG-16 [2] Custom ResNet-50 v1.5

V. CONCLUSION

This works presents an FPGA-based CNN inference accelerator that combines the benefits of a novel structured weight sparsity, MCBBS, and systolic array. When implemented on an Intel Arria 10 GX1150 FPGA, the proposed accelerator achieves 2260 GOPs/s and 1210 GOPs/s on the MAC operations in VGG-16 and ResNet-50 v1.5 pruned with MCBBS, respectively, while degrading the top-1 validation accuracy on ImageNet by less than 5%.

Acknowledgment: In this work, CNN training and FPGA compilation are performed on the compute nodes generously provided by the Intel Labs Academic Compute Environment.

REFERENCES

[1] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

[2] K. He, X. Zhang *et al.*, "Deep residual learning for image recognition," in *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*. IEEE Computer Society, 2016, pp. 770–778.

[3] J. Redmon and A. Farhadi, "Yolov3: An incremental improvement," *CoRR*, vol. abs/1804.02767, 2018.

[4] Y. LeCun, J. S. Denker, and S. A. Solla, "Optimal brain damage," in *Advances in Neural Information Processing Systems 2, [NIPS Conference, Denver, Colorado, USA, November 27-30, 1989]*. Morgan Kaufmann, 1989, pp. 598–605.

[5] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding," in *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, 2016.

[6] J. Frankle and M. Carbin, "The lottery ticket hypothesis: Finding sparse, trainable neural networks," in *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019.

[7] S. Han, J. Kang *et al.*, "ESE: efficient speech recognition engine with sparse LSTM on FPGA," in *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, FPGA 2017, Monterey, CA, USA, February 22-24, 2017*. ACM, 2017, pp. 75–84.

[8] S. Cao, C. Zhang *et al.*, "Efficient and effective sparse LSTM on FPGA with bank-balanced sparsity," in *Proceedings of the 2019 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, FPGA 2019, Seaside, CA, USA, February 24-26, 2019*. ACM, 2019, pp. 63–72.

[9] L. Lu, J. Xie *et al.*, "An efficient hardware accelerator for sparse convolutional neural networks on fpgas," in *27th IEEE Annual International Symposium on Field-Programmable Custom Computing Machines, FCCM 2019, San Diego, CA, USA, April 28 - May 1, 2019*. IEEE, 2019, pp. 17–25.

[10] J. Deng, W. Dong *et al.*, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2009), 20-25 June 2009, Miami, Florida, USA*. IEEE Computer Society, 2009, pp. 248–255.

[11] https://github.com/mustard-seed/SparseDNNAccelerator

[12] https://github.com/mustard-seed/SparseNN_training

[13] X. Wei, C. H. Yu *et al.*, "Automated systolic array architecture synthesis for high throughput CNN inference on fpgas," in *Proceedings of the 54th Annual Design Automation Conference, DAC 2017, Austin, TX, USA, June 18-22, 2017*. ACM, 2017, pp. 29:1–29:6.

[14] Y. Ma, Y. Cao *et al.*, "An automatic RTL compiler for high-throughput FPGA implementation of diverse deep convolutional neural networks," in *27th International Conference on Field Programmable Logic and Applications, FPL 2017, Ghent, Belgium, September 4-8, 2017*. IEEE, 2017, pp. 1–8.

[15] D. Wang, K. Xu *et al.*, "Abm-spconv: A novel approach to fpga-based acceleration of convolutional neural network inference," in *Proceedings of the 56th Annual Design Automation Conference 2019, DAC 2019, Las Vegas, NV, USA, June 02-06, 2019*. ACM, 2019, p. 87.

[16] A. Paszke, S. Gross *et al.*, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32*. Curran Associates, Inc., 2019, pp. 8024–8035. http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf

[17] H. Li, A. Kadav *et al.*, "Pruning filters for efficient convnets," *CoRR*, vol. abs/1608.08710, 2016.