FalCon: Fine-grained Feature Map Sparsity Computing with Decomposed Convolutions for Inference Optimization

Zirui Xu[1], Fuxun Yu[1], Chenxi Liu[1], Zhe Wu[2], Hongcheng Wang[2], Xiang Chen[1]
[1]George Mason University, [2]Comcast Applied AI Research
[1]{zxu21, fyu2, cliu26, xchen26@gmu.edu}, [2]{zhe_wu, hongcheng_wang@comcast.com}

Abstract

Many works focus on the model's static parameter optimization (e.g., filters and weights) for CNN inference acceleration. Compared to parameter sparsity, feature map sparsity is per-input related which has better adaptability. The practical sparsity patterns are non-structural and randomly located on feature maps with non-identical shapes. However, the existing feature map sparsity works take computing efficiency as the primary goal, thereby they can only remove structural sparsity and fail to match the above characteristics. In this paper, we develop a novel sparsity computing scheme called FalCon, which can well adapt to the practical sparsity patterns while still maintaining efficient computing. Specifically, we first propose a decomposed convolution design that enables a fine-grained computing unit for sparsity. Additionally, a decomposed convolution computing optimization paradigm is proposed to convert the sparse computing units to practical acceleration. Extensive experiments show that FalCon achieves at most 67.30% theoretical computation reduction with a neglected accuracy drop while accelerating CNN inference by 37%.

1. Introduction

Although Convolutional Neural Networks (CNNs) are widely applied in the image-related machine learning applications [1, 2, 43], it is still impractical to deploy them on the resource-limited embedded devices due to the bulky computation cost [20, 26]. To tackle this problem, many compression techniques have been devoted in eliminating CNN redundancy so as to reduce the computation workload [13, 17, 19, 30, 33, 48, 53–57].

As the most computation-intensive operation in CNN, convolution involves two basic components: model filters and feature maps (*a.k.a*, activation). Therefore, CNN redundancy can be eliminated from dual sides, *i.e.*, filter redundancy or feature map redundancy. Previously CNN optimization works mainly targets static model parameter redundancy removal including many pruning techniques. Compared to static model parameter sparsity, *dynamic feature map sparsity* is only proposed recently, which is less explored but has demonstrated its superior performance [10, 20, 23, 24, 30, 44, 51, 56, 57]. The feature map sparsity is adaptively identified for each individual input while model parameter sparsity is unified on the entire dataset. Therefore, feature map sparsity could generally yield higher sparsity ratios.

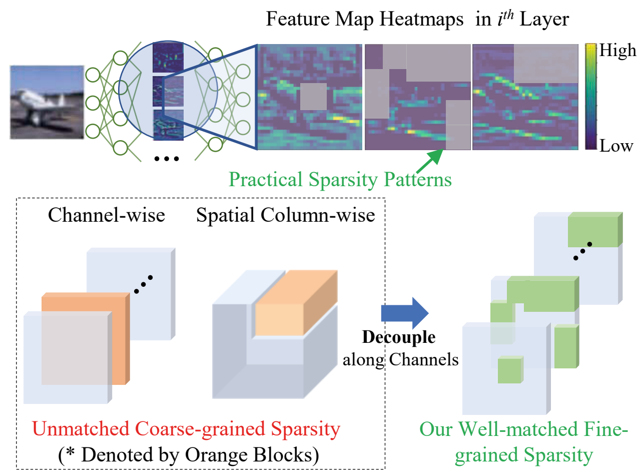


Figure 1: Overview of Different Feature Map Sparsity

However, the sparsity granularity on feature map still remain many performance potentials. As shown in Fig. 1, due to per-input adaptability, the practical sparsity patterns[1] such as background (sky and ground, indicated by grey patterns on the feature heatmaps) demonstrate three characteristics: 1) they are *non-structural* regarding input (*i.e.* various object shapes); 2) they are *randomly located*; 3) they are *non-identical* across channel due to the channel function variety. However, driven by computing efficiency, the performance gain of the current feature map sparsity works can only be achieved from structural feature sparsity removal. As shown in Fig. 1 as orange blocks, such structural computing schemes don't consider the above characteristics and

[1]Can be measured with various criteria such as activation-norm [56], similarity [19], and *etc.*

thereby fail to match the practical sparsity patterns, suffering from low sparsity removal performance [38]. This issue will be more serious for dynamic layer/block-dropping methods [8,47,49,52]. Therefore, *the essential challenge in our work is to develop a novel sparsity computing scheme that can adapt to the practical sparsity patterns while maintaining an efficient computing.*

In this paper, we first propose a convolution decomposition design to enable a *fine-grained computing unit* that can adapt to the randomness, non-structure, and non-identity of the practical sparsity patterns. Specifically, we decouple the traditional convolution operation along channel-wise, and thereby can obtain the single-channel computing freedom (*i.e.* each feature map can execute an individual convolution). Therefore, as shown in Fig. 1, the minimum computing unit for sparsity is refined to the kernel-size feature areas (green blocks) on each single feature map. Such kernel-wise sparse computing units can be further composed together to well match the practical sparsity patterns and achieves a fine-grained computing foundation.

When matching the practical sparsity patterns, our kernel-wise sparse computing units will introduce non-structural sparsity computing (they randomly distribute on each feature map with unbalanced sparsity ratios). In order to *convert computing units from non-structural to structural* for better computing efficiency, we propose a decompose convolution computing optimization paradigm. Specifically, through three technical steps, we can first eliminate the randomness and unbalance sparsity ratios of computing units and achieve structural sparsity computing. Furthermore, such structural sparsity can be efficiently removed from computing to achieve a practical speedup.

We further implement our decomposed convolution computing scheme on a sparsity pruning framework. The extensive experiments on multiple benchmarks demonstrate that our proposed method can achieve at most 67.30% computation workload reduction with neglected accuracy drop. In term of speedup, our computing scheme could also effectively translate the sparsity into run-time saving, accelerating CNN inference by 33.98% and 37.13% on GPU and CPU, respectively.

2. Background and Related Works

2.1. Convolution Decomposition

Assume the input feature maps and the filters on a traditional convolution layer are defined as: $\mathcal{IF}(w,h,c) \in \mathbb{R}^{w*h*C}$ and $\mathcal{W}(k,k,c,j) \in \mathbb{R}^{k*k*c*J}$ (here, w, h and C is feature map width, height, and the total channel number, k represents the kernel size and J is the filter number). The traditional convolution operation is filter-based, namely, each filter with size $k*k*c$ slides on all feature maps with full depth C and conducts inner-product. Finally, J filters will generate J output feature maps $\mathcal{OF}$. Since feature maps are bound together during computing, it is impossible to explore a fine-grained computing unit that can flexibly distribute on each feature map. Therefore, to explore a fine-grained sparsity on each single feature map, we need to decompose the traditional convolution process.

Currently, there are some convolution decomposition works that leverage matrix factorization methods to decompose filters, such as Depth-wise Convolution [3], Network Decoupling [12], and DCFNet [42]. Since the goal of these works is computation reduction, the decomposed convolution is an approximation to the original ones, suffering from accuracy drop. Different with these previous works, we decompose convolution from the perspective of feature map for exploring a fine computing granularity. Therefore, the output feature maps in our proposed decomposed convolution are same as the original ones without any accuracy loss.

2.2. CNN Inference Speedup

Matching the practical sparsity patterns will introduce randomly distributed and unbalanced zero values during computing, generating nonconsecutive storage address during memory access. And it further causes memory bus and computing unit (*e.g.* GPU warps) under-utilization [21,36].

Currently, to tackle such data noncontiguous issue, most works propose specific hardware/compiler designs, *e.g.* sparse accelerators [14, 39]. However, they require heavy design efforts and cannot leverage the existing commodity CNN computing libraries such as cuBLAS and MKL. A few works solve the above issue from softwave-level: they leverage matrix transformation techniques to eliminate the zero values on the sparse weight matrix [4, 11, 21]. However, these works focus on weight sparsity and introduce an overhead: when applying transformation on weight matrix, they also need to reorganize feature map matrix to achieve real computation reduction.

Our work overcomes data noncontiguous issue also via software-level, but targets on feature maps. We propose a decomposed convolution computing paradigm to eliminate the randomly distributed zero values on each feature map matrix. Therefore, the sparse convolution still can be fully supported by the current General Matrix Multiplication (GEMM) acceleration libraries without any hardware/compiler-level modifications. Different from the previous works, our reorganization only involves feature maps. Filter matrices during our optimization doesn't need to be regulated, avoiding extra overheads.

3. Decomposed Convolution for Practical Sparsity Matching

In this section, we introduce a lossless decomposed convolution that achieves per-channel computing freedom. Consequently, a kernel-wise fine-grained computing unit is

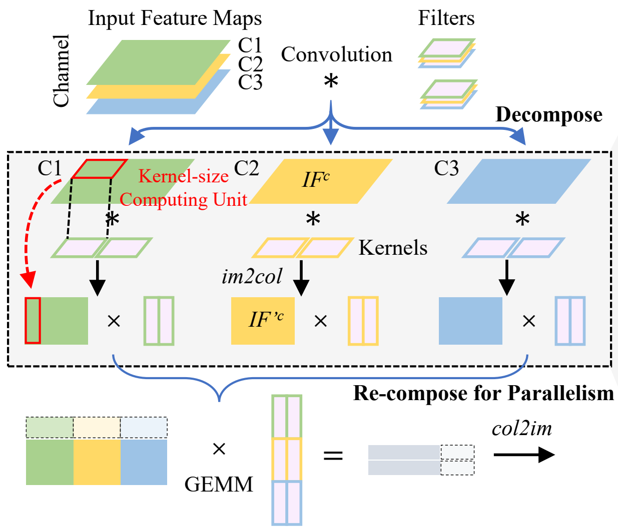


Figure 2: Decomposed Convolution with Kernel-wise Computing Unit

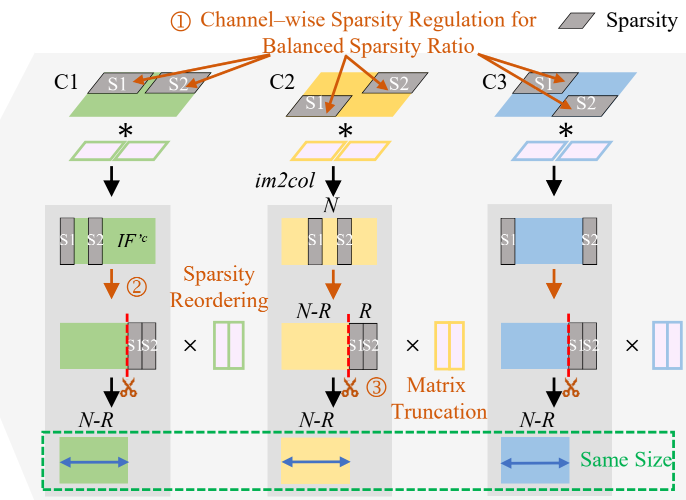


Figure 3: Sparse Decomposed Convolution Optimization Paradigm

identified. By combining multiple computing units, we can well match the practical input sparsity patterns.

3.1. Fine-grained Computing Unit with Decomposed Convolution Design

We decompose the traditional convolution operation along the channel dimension. As demonstrated in Fig. 2, we first split the input feature maps $\mathcal{IF}(w,h,c)$ (3D tensor) to C single feature map $\mathcal{IF}^c$ (2D matrix). Then, each $\mathcal{IF}^c$ only conducts convolution operation with kernels belong to channel c from all filters (represented by the specific color). In that case, the basic unit is changed from the single filter in the traditional convolution to the individual feature map, thereby realize per-channel computing freedom. Our decomposed convolution process can be formulated as:

$$\begin{aligned}\mathcal{OF} &= \sum_{c=1}^{C} \mathcal{IF}^c(w,h,\mathbf{1}) * \mathcal{W}_j^c(k,k,\mathbf{1}),\\ j &= 1,2,\cdots,J; \quad and \quad c = 1,2,\cdots,C.\end{aligned} \tag{1}$$

Here, $*$ represents convolution operation. $\mathcal{IF}^c(w,h,\mathbf{1})$ and $\mathcal{W}_j^c(k,k,\mathbf{1})$ indicates one feature map channel and the corresponding kernel of the filter. $\sum_{c=1}^{C}(\cdot)$ denotes the element-wise addition across C output feature maps.

The convolution operation in the traditional convolution is realized via a GEMM between feature map matrix and filter weight matrix, which is well supported by the compilers/hardware with high parallelism. After decomposition, each $\mathcal{IF}^c * \mathcal{W}_j^c$ generates an individual GEMM. Iteratively executing them in a loop mode will sacrifice the original parallelism. To tackle this issue, we further re-compose feature map together to maintain a complete GEMM (as shown in the bottom of Fig. 2). Therefore, element-wise addition in Eq. 1 will be replaced by a matrix concatenate operation.

Since each feature map conducts individual convolution after decomposition, the basic operation unit is refined to kernel-size feature areas ($k \times k$) on each single channel[2], shown as the red block in Fig. 2. Such kernel-size area will be further unfolded to a single column on the feature map matrix $\mathcal{IF}'^c$ during *im2col*, it can be identified as the minimum computing unit in the convolution process. Compared to the previous channel- or column-wise structure, our kernel-wise computing unit has much finer computing granularity. It should be noted that, although single activation on the feature map has the finest-granularity, it which will introduce extremely noncontiguous data structure and require specific hardware designs [21]. By contrast, our kernel-wise computing unit not only provides a fine computing granularity but also can be removed via the computing optimization scheme proposed in Section 4 to achieve inference speedup.

3.2. Matching Practical Sparsity Patterns

By decomposing convolution, a fine-grained computing unit is identified on each feature map which can be used to match practical sparsity patterns. During convolution computing, the practical sparsity patterns will be unfolded as random and non-structural zero value distribution on each feature map matrix $\mathcal{IF}'^c$ via *im2col*. Under such condition, our proposed kernel-wise computing units can well adapt to these practical sparsity patterns because of two main reasons: 1) Flexible location: since kernels are sliding on each feature map, our kernel-wise computing units can locate at any column of feature map matrix. 2) Small size: Most widely-used neural networks usually adapt a small kernel size (3×3 or even 1×1 for point-width convolution). On

[2]The computation unit in the traditional convolution is stacked kernel-size areas across C channels.

the contrary, for most image dataset such as *ILSVRC* [5] or *PASCAL* [7], the width/length of feature maps on most layers are much larger than kernel size (*i.e.* 224 or 112). Compared to the size of entire feature map matrix, our kernel-wise computing unit is fine-grained enough. Therefore, we can combine computing units on different location of feature map matrix together to approximate any zero distribution patterns that are generated by the practical sparsity.

4. Decomposed Convolution Computing Optimization for Speedup

The proposed decomposed convolution provides a fine-grained computing unit to match the practical sparsity patterns. However, by matching the randomness, non-structure, and non-identity of practical sparsity patterns, our kernel-wise sparse computing units will randomly distribute on each feature map with unbalanced sparsity ratios (indicated by the interleaved zero columns (grey) and non-zero columns (color) in Fig. 3), becoming non-structural and computing-inefficient. Therefore, in order to convert computing from non-structural to structural, we further propose a computing optimization paradigm to reorganize sparse computing unit distribution on feature maps, thereby can improve computing efficiency.

4.1. Computing Optimization Paradigm

Computing Flow Overview: The central goal of our computing optimization paradigm is to preserve the theoretical gains of sparsity while diminishing its randomness and unbalance on input feature matrix, thereby maintains the supposed parallelism on the decomposed convolution. Specifically, as shown in Fig. 3, to solve unbalanced sparsity ratios, we proposed *Channel-wise Sparsity Regulation* to enable each feature map has an identical sparsity ratio. To tackle the random distribution issue, we proposed *sparsity reordering* and *matrix truncation* to convert the random sparse feature map matrix to a smaller dense one, thereby decomposed convolution with sparsity can still be calculated by a dense GEMM and fully supported by hardware.

① ***Channel-wise Sparsity Regulation:*** In order to achieve re-composition, the sparsity ratio on each channel should be identical, which can be denoted as $\gamma^s = \frac{R}{N}$ (N and R is the total and sparse computing units number, respectively).

The sparsity regulation in the previous methods is usually realized by multiplying a zero-one masking matrix on each feature map $\mathcal{IF}^c$ and thus the sparse activation values will permanently become zeros [35, 48, 56]. Our sparse computing unit is kernel-size, zeroing an activation value inside it may affect other adjacent non-sparse computing unit (the kernel-size area that has partial overlapping with S1/S2), causing information loss. This is because that an activation value will be included into multiple surrounding kernel-size areas during kernel sliding.

Considering the fact that each column on the feature map matrix is exactly unfolded by one kernel-size area, we can regulate our sparsity ratio by applying masking operation on the unfolded feature map matrix instead of the original feature map. By doing that, assigning zeros to one kernel-size area will not affect its neighbours, thereby maintaining a better model accuracy. As shown in the bottom of Fig. 2, with our regulation, matrices have identical sizes.

② ***Sparsity Reordering:*** As shown in Fig. 3, the zero columns (represented by S1 and S2) are randomly distributed on each feature map matrix $\mathcal{IF'}^c$, introducing Sparse Matrix Multiplication (SpMM) during feature map computing, which is data non-continuous and not applicable for the realistic speedup [9]. To mitigate it, we eliminate randomness and non-continuity of these sparse columns through sparsity reordering, which is demonstrated in Fig. 3: assume the original column index on each $\mathcal{IF'}^c$ is $D^c = \{d_1, d_2, ..., d_N\}$, *e.g.* S1 on the first $\mathcal{IF'}^c$ is denoted as d_1. The total number of zero columns is R ($R = 2$ in our figure). During sparsity reordering, all the zero columns are shifted to the most right side of matrix $\mathcal{IF'}^c$. However, with iterative shifting-mode, index D^c needs to be updated K times, introducing large indexing overhead.

We optimize the shifting process with a parallelism mode: first, we extract all index d_i of zero columns; then we reorder all non-zero and zero columns from 0 to $N - R$ and from $N - R + 1$ to N, respectively. In that case, D^c only needs to be updated once, saving indexing cost.

③ ***Matrix Truncation:*** We further reduce the dimension of the feature map matrix $\mathcal{IF'}^c$ to decrease the computation workload during GEMM after re-composing, providing speedup potential. Specifically, as shown in Fig 3, we remove the entire zero blocks with size $R \times k^2$ on each channel. Thus, the size of the remained dense matrix is $(N - R) \times k^2$. By truncation operation, the sparse decomposed convolution computing is still realized by a dense GEMM with less computation workload. The overall computing optimization paradigm is summarized in Algorithm. 1 in Supplementary.

As Fig. 3 shows, after truncation, the feature map matrices $\mathcal{IF'}$ are smaller than the expected ones due to the exclusion of zero columns. Therefore, during *col2im* process, we need to further reshape them back to the original size. This process can be done by using index D^c and ${D_c}'$.

4.2. Corner Case Discussion

Corner Case Definition: The proposed channel-wise sparsity regulation address unbalanced sparsity issue for matrix re-composition. However, when the unbalanced level is extremely high, as shown in Fig. 4: the first feature map matrix are entirely sparse (becoming channel-wise sparsity) while the other two feature map matrices only have 25% sparsity ratio (two zero columns S1 and S2). If we still

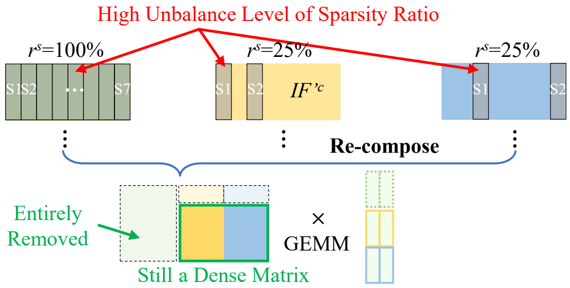


Figure 4: Corner Case and the Corresponding Computing Optimization

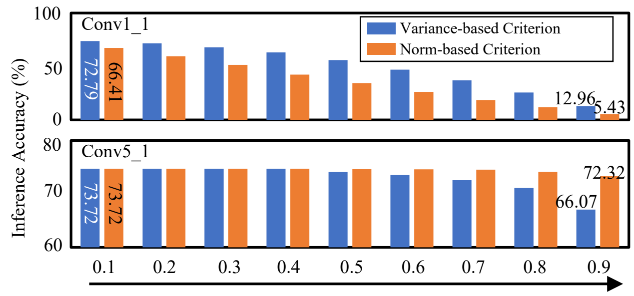


Figure 5: The pruning performance comparison between two metrics on *VGG16*.

regulate all feature maps with an unified sparsity ratio (*e.g.* 25%), such unbalanced situation will introduce a significant sparsity mismatching issue on the first feature map: a lot of potential sparsity cannot be correctly converted to zero columns, degrading the sparsity performance.

Computing Optimization for Corner Case: In order to tackle this issue, we slightly optimize the proposed sparse convolution computing paradigm. In the channel-wise sparsity regulation step, we adapt two different sparsity ratios: setting γ_1^s as 100% for those entirely sparse feature maps (*e.g.* green in Fig. 4) while still keeping the rest feature maps (yellow and blue) with an unified sparsity ratio γ_2^s. In the sparsity reordering and truncation steps, since entirely sparse feature maps will be converted as full-zero matrices, we can directly remove all of them. At last, as demonstrated in figure, the recomposed tensor is still a dense matrices, and can be calculated via dense GEMM. Therefore, our sparse convolution computing paradigm shows better generality that can well support channel-wise sparsity.

5. Decomposed Convolution Application on Sparsity Pruning

We apply our decomposed convolution on feature map sparsity pruning. Most existing pruning works are targeting proposing novel pruning criteria, while our proposed decomposed convolution computing scheme is orthogonal to them. By introducing our schemes, these works can be further boosted regarding pruning performance. In our paper, to show the performance gain of our proposed scheme, we only adopt several common-used pruning settings.

Pruning Criteria: Two simple pruning criteria (norm-based [28, 56] and variance-based [29]) are applied in our method implementation. They indicate the activation sum and distribution in a target kernel-size activation area, which can be formulated as: $\mathcal{IF}^c$ is defined as:

$$\begin{aligned}&\text{Norm}-\text{based: } A_{con}=\sum_{i}^{r^2} a_i^l,\\&\text{Variance}-\text{based: } A_{con}=\frac{1}{r^2}\sum_{i=1}^{r^2}(a_i^l-\mu),\quad \mu=\frac{1}{r^2}\sum_{i=1}^{r^2}a_i^l\end{aligned}\tag{2}$$

where a_i^l is the i^{th} feature activation value inside each kernel-size area and lower A_{con} value represents the corresponding area should be sparse. The only optimization of criterion selection here is that we apply variance-based criterion for lower-layers while applying norm-based criterion for higher-layers. The reason behind this is the layer function diversities [16]. Fig. 5 shows an example: on the low-level layer ($Conv1\_1$ in the figure), the first criterion always illustrates a better pruning performance than the second criterion regarding pruned model inference accuracy and vice versa for the high-level layer ($Conv5\_1$).

Sparsity Ratio and Training Strategy: In *FalCon*, since different layers/blocks of model can have various sparsity levels, they show distinct accuracy robustness *w.r.t* sparsity ratio γ^s. For a given model, we first analyze the layer/block robustness sensitivities and divide the entire model into several groups where each group shares the same γ^s. The influence of different group numbers regarding model accuracy will be evaluated in the ablation study and more details about sparsity ratio selection are provided in Supplementary. Although a higher pruning ratio can significantly benefit the CNN computing performance, directly applying our method on a pre-trained model in the CNN inference may introduce considerable accuracy loss. Therefore, we can either train the model from scratch or fine-tune a pre-trained model with the proposed sparse decomposed convolution computing scheme. During the training, network can gradually learn to focus on the important feature areas and neglect the non-important ones. We evaluate the effectiveness of our training-phase optimization in our ablation study and compare the performance of two training initialization strategies in Supplementary.

6. Experiment

In this section, we evaluate *FalCon* in three aspects: accuracy, theoretical FLOPs reduction, and realistic speedup.

6.1. Experiment Setup

Models and Datasets: We evaluate *FalCon* for single-branch networks (*VGGNet* [46] and *MobileNetV1* [22]) and multiple-branch networks (*ResNet* [15] and *MobileNetV2* [45]) on three benchmarks: *CIFAR-10*, *CIFAR-100* [27] and *ILSVRC-2012* [5].

Table 1: Experiment Results on *CIFAR-10* Datasets

CNN Models	Pruning Methods	Baseline Accuracy(%)	Final FLOPs	FLOPs Reduction(%)	Final Accuracy(%)	Accuracy Drop(%)
VGG16	Taylor* [37]	93.30	1.75E+08	44.10	92.30	1.00
	GM* [19]	93.58	2.00E+08	35.90	93.23	0.35
	FO* [41]	93.40	1.75E+08	44.10	93.30	0.10
	TiNet** [34]	**93.99**	1.56E+08	50.00	93.85	0.14
	SFP** [17]	**93.99**	1.56E+08	50.00	93.85	0.14
	CP** [20]	**93.99**	1.56E+08	50.00	93.67	0.32
	DCP** [57]	**93.99**	1.56E+08	50.00	**94.16**	-0.17
	TS** [50]	93.44	1.56E+08	50.00	93.63($\pm$0.06)	-0.19
	LP [25]	92.77	1.07E+08	64.50	90.87	1.90
	HRank [31]	93.96	1.08E+08	65.30	92.34	1.62
	Ours1	93.32($\pm$0.09)	1.56E+08	50.00	93.63($\pm$0.07)	**-0.31**
	Ours2	93.32($\pm$0.09)	1.02E+08	**67.30**	91.92($\pm$0.05)	1.40
ResNet32	MIL† [6]	92.33	4.71E+07	31.20	90.74	1.59
	SFP† [17]	92.63($\pm$0.07)	4.03E+07	41.50	92.08($\pm$0.08)	0.55
	GM [19]	92.63($\pm$0.07)	3.23E+07	53.20	91.93($\pm$0.30)	0.70
	LFPC† [16]	92.63($\pm$0.07)	3.27E+07	52.60	92.12($\pm$0.32)	0.51
	DC [48]	**93.81**	3.43E+07	50.00	**92.50**	1.31
	Ours1	92.20($\pm$0.10)	3.23E+07	53.20	91.94($\pm$0.12)	**0.26**
	Ours2	92.20($\pm$0.10)	2.46E+07	**59.96**	91.40($\pm$0.08)	0.80
ResNet110	MIL† [6]	93.63	8.23E+07	34.20	93.44	0.19
	SFP† [17]	**93.68**($\pm$0.32)	7.50E+07	40.00	93.38($\pm$0.30)	0.30
	GM† [19]	**93.68**($\pm$0.32)	7.40E+07	40.80	93.74($\pm$0.10)	-0.06
	TS** [50]	93.49	7.50E+07	40.00	93.69($\pm$0.28)	**-0.2**
	LFPC† [16]	**93.68**($\pm$0.32)	4.96E+07	60.30	**93.79**($\pm$0.38)	-0.11
	Ours1	**93.68**($\pm$0.30)	4.96E+07	60.30	93.79($\pm$0.28)	-0.11
	Ours2	**93.68**($\pm$0.30)	4.17E+07	**62.25**	93.63($\pm$0.33)	0.05
MobileNetV1	WM⋆ [57]	**93.96**	2.62E+07	42.86	93.48	0.48
	Random DCP⋆ [57]	**93.96**	2.62E+07-	42.86	93.39	0.57
	DCP⋆ [57]	**93.96**	2.62E+07	42.86	**94.18**	-0.22
	Ours1	93.01($\pm$0.41)	2.62E+07	42.86	93.42($\pm$0.08)	**-0.41**
	Ours2	93.01($\pm$0.41)	2.36E+07	**45.11**	92.73($\pm$0.17)	0.68

*,**, †, and ⋆: the methods' performances are referred from [56], [50], [16], and [57], respectively.

Ours1: uses the FLOPs reduction ratio that is the highest one in the previous works. Ours2: uses the distribution median value of accuracy drop among the previous methods.

Training Setting: The entire *FalCon* is implemented on Pytorch1.4 [40]. On *CIFAR-10* and *CIFAR-100*, we use SGD optimizer and CosineAnnealing scheduler [32] with an initial learning rate of 0.1 and the training epoch is set as 200. The batch size is set as 256 for both training and inference. On *ILSVRC-2012*, the parameter setting and training schedule is the same as [19]. Specifically, the training epoch is set as 250. Moreover, the data argumentation strategies we use for *ILSVRC-2012* is the same as PyTorch official examples. The training strategy we used here is fine-tuning a pre-trained model.

Baselines: We compare our method with other existing state-of-the-art CNN inference optimization works, *e.g.* Taylor [37], GM [19], Antidote [56], TiNet [34], FO [41], SFP [17], CP [20], DCP [57], TS [50], MIL [6], LFPC [16], AMC [18], HRank [31], LP [25], and DC [48].

6.2. Evaluation on CIFAR Dataset

CIFAR-10: For *CIFAR-10* dataset, we test our *FalCon* on *VGG16*, *ResNet32*, *ResNet110*, and *MobileNetV1*. A smaller accuracy drop and a larger FLOPs reduction indicates a better optimization performance. We evaluate our method with two separate settings, which is shown in below of Table. 1.

As shown in Tab. 1, the experimental results clearly show the effectiveness of our proposed method. For example, for *VGG16*, in order to obtain an acceptable accuracy drop, most state-of-the-art methods can only realize the FLOPs reduction below 50%. When keeping the same 50% ratio, our method (ours1) even achieves 0.2% accuracy improvement, which outperforms other methods

Table 2: Experiment Results on *CIFAR-100* Datasets

CNN Models	Pruning Methods	Baseline Accuracy(%)	Final FLOPs	FLOPs Reduction(%)	Final Accuracy(%)	Accuracy Drop(%)
VGG16	Taylor* [37]	73.10	1.96E+8	37.30	72.50	0.60
	FO* [41]	73.10	1.96E+8	37.30	**73.20**	**-0.10**
	Antidote* [56]	73.10	1.72E+8	44.90	72.90	0.20
	Ours1	**73.12**($\pm$0.25)	1.72E+8	44.90	72.95($\pm$0.18)	0.17
	Ours2	**73.12**($\pm$0.25)	1.56E+8	**49.96**	72.79($\pm$0.33)	0.33
ResNet56	MIL† [6]	71.33	7.63E+7	39.30	68.37	2.96
	SFP† [17]	71.40	5.94E+7	52.60	68.79	2.61
	GM† [19]	71.41	5.94E+7	52.60	69.66	1.75
	LFPC† [16]	71.41	6.08E+7	51.60	70.83	0.58
	Ours1	**71.55**($\pm$0.07)	5.94E+7	52.60	**71.10**($\pm$0.09)	**0.45**
	Ours2	**71.55**($\pm$0.07)	5.24E+7	**58.28**	70.66($\pm$0.09)	0.89

* and † indicates the methods' performances are referred from [56] and [16], respectively.

Table 3: Experiment Results on *ILSVRC-2012* Datasets

CNN Models	Pruning Methods	Baseline Accuracy(%) Top-1 (Top-5)	Final FLOPs	FLOPs Reduction(%)	Final Accuracy(%) Top-1 (Top-5)	Accuracy Drop(%) Top-1 (Top5)
ResNet50	SFP† [17]	**76.15 (92.87)**	2.03E+9	41.80	62.14 (84.60)	14.01 (8.27)
	GM† [19]	**76.15 (92.87)**	1.62E+9	53.50	**74.83** (92.32)	1.32 (0.55)
	TS [50]	76.10 (-)	1.75E+9	50.00	72.80 (-)	3.30 (-)
	Ours1	75.83 (92.78)	2.03E+9	53.50	74.59 (92.51)	**1.24 (0.27)**
	Ours2	75.83 (92.78)	1.28E+9	**63.38**	73.55 (91.99)	2.28 (0.79)
MobileNetV2	TiNet [34]	70.11 (-)	1.96E+8	**44.70**	63.71 (-)	6.40 (-)
	DCP [57]	70.11 (-)	1.96E+8	**44.70**	64.22 (-)	5.89 (-)
	AMC [18]	71.80 (-)	2.49E+8	30.00	70.80 (-)	**1.00 (-)**
	Ours1	71.60 (90.41)	1.96E+8	**44.70**	69.45 (89.31)	2.15 (1.10)

† indicates the methods' performance is referred from [16].

Table 4: Realistic Acceleration Evaluation

Model (Dataset)	Computing Unit	Baseline Latency (ms)	Optimized Latency (ms)	Realistic Acceleration (%)
MobileNetV1 (CIFAR-10)	Titan XP	10.32	7.11	31.10
	i7-6700K	65.62	41.25	37.13
ResNet32 (CIFAR-10)	Titan XP	18.51	12.22	33.98
	i7-6700K	36.81	24.23	34.14
MobileNetV2 (ILSVRC-2012)	Titan XP	45.51	34.12	25.03
	i7-6700K	152.10	113.84	25.15

regarding accuracy drop. Furthermore, our method with the second setting (ours2) can aggressively achieve 67.30% FLOPs reduction with 1.40% accuracy drop while HRank and LP show 1.90% and 1.60% accuracy loss. As for *ResNet32*, GM [19] shows the best computation reduction performance (53.20%) among several baselines. However , our method with the same ratio (53.20%) has a much lower accuracy drop (0.26%). Also, with only 0.70% accuracy loss, ours2 can achieve the highest FLOPs reduction performance (59.96%). For *MobileNetV1* which is designed to be lightweight, our method shows the best accuracy performance. In terms of computation efficiency, our method achieves around 2.25% more FLOPs reduction ratio compared to other methods. These results validate the effectiveness of our proposed method that can accurately identify and remove the redundancy in model inference.

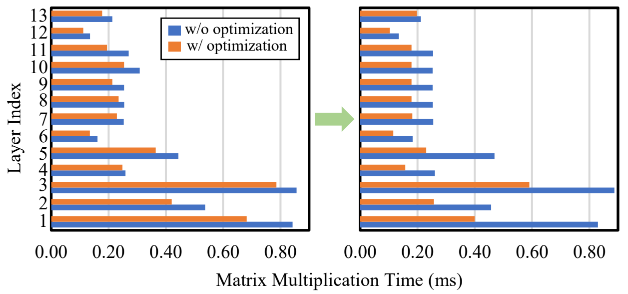
CIFAR-100: We also test our method with *VGG16* and *ResNet56* on *CIFAR-100*. From the Tab. 2 can find that our method achieve around 5%$\sim$12% and 6%$\sim$20% more FLOPs reduction compared to three state-of-the-art methods with negligible accuracy loss, which also demonstrates our method's performance effectiveness.

We also observed the distinct FLOPs reduction ratio among different models. This is because that for a given dataset, each model has a specific redundancy and our sparsity can well identify such redundancy.

6.3. Evaluation on ILSVRC-2012 Dataset

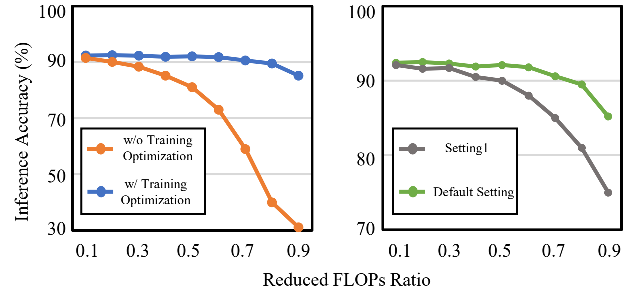
For *ILSVRC-2012*, we test *FalCon* on two models: *ResNet50* and *MobileNetV2*. we leverage layer sensitivity analysis discussed in Section 5.2 to select proper sparsity ratio for each block on the two models.

Tab. 3 summarizes the evaluation results. On *ResNet50*,



(a) Reduced FLOPs Ratio = 0.1 (b) Reduced FLOPs Ratio = 0.5

Figure 6: Layer-wise Matrix Multiplication Acceleration Evaluation for *MobileNetV1*



(a) Training Optimization Performance (b) Layer Ratio Influence

Figure 7: Accuracy of *ResNet32* on *CIFAR-10* regarding Different Hyper-parameters

with same computation reduction ratio (53.50%), our method can outperform SFP [17], GM [19], TS [50] in terms of accuracy drop. When keeping a median accuracy loss, the FLOPs reduction gain of our method is the largest compared to other candidate algorithms. This is because *FalCon* can accurately identify each redundant convolution operation with the finest granularity. On *MobileNetV2*, compared to TiNet [34] and DCP [57], our method can reduce the similar computation workload while maintaining a higher accuracy level. AMC [18] has a similar accuracy drop, but it can only achieve the lowest FLOPs reduction.

We can find the input size is another key factor for our sparsity performance: since the proposed granularity is kernel-size, increasing input size will generate a larger feature map, thereby our granularity can match the ideal sparsity pattern better.

6.4. Speedup Evaluation

Layer-wise Matrix Multiplication Acceleration: We first evaluate the speedup performance of the proposed sparsity reordering and matrix truncation inside the CNN model inference. Specifically, we compare each layer's matrix multiplication latency of *MobileNetV1* on *CIFAR-10* with/without our computing optimization, and the results are shown in Fig. 6. When FLOPs reduction ratio increases from 0.1 to 0.5, the original SpMM without optimization shows negligible speedup. On the contrary, our method brings significantly acceleration for most layers (*e.g.* 42% reduction in layer 1), thereby proves its efficiency. Layer 13 has less computation load, thereby its potential acceleration margin is relatively lower when considering the computing overhead (reordering, truncation, etc.).

Model-wise End-to-End Acceleration: We further evaluate the proposed method's realistic acceleration performance on GPU (Titan XP) and CPU (I7-6700K), respectively. The results are shown in Tab. 4 with *MobileNetV1*, *ResNet32*, and *MobileNetV2* on both *CIFAR-10* and *ILSVRC-2012*. Baseline is the time cost without applying our sparse convolution computing paradigm. We can find, after applying *FalCon*, the end-to-end inference time will achieve 31.10% $\sim$ 37.13%, 33.98% $\sim$ 34.14%, and 25.03% $\sim$ 25.15% acceleration on *MobileNetV1*, *ResNet32*, and *MobileNetV2*. As discussed in [19], the acceleration gap between theoretical FLOPs reduction and realistic acceleration is caused by the overhead of sparsity reordering and re-composition, and the limitation of IO delay as well.

6.5. Ablation Study

In ablation study, we will further explore the performance of our training-phase optimization and layer sparsity ratio setting discussed in Section 5.

Influence of Training Optimization: Fig. 7 shows the the influence of our propose training optimization scheme in terms of inference accuracy. If pruning the CNN inference directly from a pre-trained model, the inference accuracy will drop dramatically when reduced FLOPs ratio larger than 0.3 due to the pre-trained model cannot fit the sparsity. On the contrary, by applying our training optimization, the network can gradually learn to focus on the important kernel-size areas and neglect the non-important ones.

Influence of Layer Sparsity Ratio: As aforementioned in Section 5, we let a certain number of layers (residual blocks) in the plain (branch) networks as a group and assign each group a certain γ^s. To further investigate the influence of group number size, we increase the layer/residual block number in each group, represented as setting 1 and compare it with our default setting. Fig. 5 shows the comparison results. We can easily find that keeping more blocks with the same pruning ratio will degrade the sparsity identification performance, thereby introduce a larger accuracy drop.

7. Conclusion

In this paper, we proposed *FalCon*, a sparsity computing scheme for CNN inference speedup. By decomposing the traditional convolution from channel-wise, we identified a fine computing granularity that can well match the practical sparsity patterns. We further proposed a decomposed convolution computing optimization paradigm to enable our sparse computing units can bring realistic acceleration. Experiments demonstrate that the proposed *FalCon* achieves superior performance regarding model accuracy, theoretical FLOPs reduction, and inference speedup.

References

[1] Vijay Badrinarayanan, Alex Kendall, and Roberto Cipolla. Segnet: A deep convolutional encoder-decoder architecture for image segmentation. *IEEE transactions on pattern analysis and machine intelligence*, 39(12):2481–2495, 2017.

[2] Guoguo Chen, Carolina Parada, and Georg Heigold. Small-footprint keyword spotting using deep neural networks. In *2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 4087–4091. IEEE, 2014.

[3] François Chollet. Xception: Deep learning with depthwise separable convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1251–1258, 2017.

[4] Chunhua Deng, Siyu Liao, Yi Xie, Keshab K Parhi, Xuehai Qian, and Bo Yuan. Permdnn: Efficient compressed dnn architecture with permuted diagonal matrices. In *2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 189–202. IEEE, 2018.

[5] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[6] Xuanyi Dong, Junshi Huang, Yi Yang, and Shuicheng Yan. More is less: A more complicated network with less inference complexity. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5840–5848, 2017.

[7] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The pascal visual object classes (voc) challenge. *International Journal of Computer Vision*, 88(2):303–338, June 2010.

[8] Angela Fan, Edouard Grave, and Armand Joulin. Reducing transformer depth on demand with structured dropout. *arXiv preprint arXiv:1909.11556*, 2019.

[9] Alexander Frickenstein, Manoj Rohit Vemparala, Christian Unger, Fatih Ayar, and Walter Stechele. Dsc: Dense-sparse convolution for vectorized inference of convolutional neural networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, pages 0–0, 2019.

[10] Xitong Gao, Yiren Zhao, Łukasz Dudziak, Robert Mullins, and Cheng-zhong Xu. Dynamic channel pruning: Feature boosting and suppression. *arXiv preprint arXiv:1810.05331*, 2018.

[11] Cong Guo, Bo Yang Hsueh, Jingwen Leng, Yuxian Qiu, Yue Guan, Zehuan Wang, Xiaoying Jia, Xipeng Li, Minyi Guo, and Yuhao Zhu. Accelerating sparse dnn models without hardware-support via tile-wise sparsity. *arXiv preprint arXiv:2008.13006*, 2020.

[12] Jianbo Guo, Yuxi Li, Weiyao Lin, Yurong Chen, and Jianguo Li. Network decoupling: From regular to depthwise separable convolutions. *arXiv preprint arXiv:1808.05517*, 2018.

[13] Shaopeng Guo, Yujie Wang, Quanquan Li, and Junjie Yan. Dmcp: Differentiable markov channel pruning for neural networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1539–1547, 2020.

[14] Song Han, Xingyu Liu, Huizi Mao, Jing Pu, Ardavan Pedram, Mark A Horowitz, and William J Dally. Eie: Efficient inference engine on compressed deep neural network. *ACM SIGARCH Computer Architecture News*, 44(3):243–254, 2016.

[15] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[16] Yang He, Yuhang Ding, Ping Liu, Linchao Zhu, Hanwang Zhang, and Yi Yang. Learning filter pruning criteria for deep convolutional neural networks acceleration. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2009–2018, 2020.

[17] Yang He, Guoliang Kang, Xuanyi Dong, Yanwei Fu, and Yi Yang. Soft filter pruning for accelerating deep convolutional neural networks. *arXiv preprint arXiv:1808.06866*, 2018.

[18] Yihui He, Ji Lin, Zhijian Liu, Hanrui Wang, Li-Jia Li, and Song Han. Amc: Automl for model compression and acceleration on mobile devices. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 784–800, 2018.

[19] Yang He, Ping Liu, Ziwei Wang, Zhilan Hu, and Yi Yang. Filter pruning via geometric median for deep convolutional neural networks acceleration. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4340–4349, 2019.

[20] Yihui He, Xiangyu Zhang, and Jian Sun. Channel pruning for accelerating very deep neural networks. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1389–1397, 2017.

[21] Parker Hill, Animesh Jain, Mason Hill, Babak Zamirai, Chang-Hong Hsu, Michael A Laurenzano, Scott Mahlke, Lingjia Tang, and Jason Mars. Deftnn: Addressing bottlenecks for dnn execution on gpus via synapse vector elimination and near-compute data fission. In *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 786–799, 2017.

[22] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

[23] Jie Hu, Li Shen, and Gang Sun. Squeeze-and-excitation networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7132–7141, 2018.

[24] Weizhe Hua, Yuan Zhou, Christopher De Sa, Zhiru Zhang, and G Edward Suh. Boosting the performance of cnn accelerators with dynamic fine-grained channel gating. In *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, pages 139–150, 2019.

[25] Qiangui Huang, Kevin Zhou, Suya You, and Ulrich Neumann. Learning to prune filters in convolutional neural networks. In *2018 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 709–718. IEEE, 2018.

[26] Forrest N Iandola, Song Han, Matthew W Moskewicz, Khalid Ashraf, William J Dally, and Kurt Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and¡ 0.5 mb model size. *arXiv preprint arXiv:1602.07360*, 2016.

[27] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[28] Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning filters for efficient convnets. *arXiv preprint arXiv:1608.08710*, 2016.

[29] Hang Li, Chen Ma, Wei Xu, and Xue Liu. Feature statistics guided efficient filter pruning. *arXiv preprint arXiv:2005.12193*, 2020.

[30] Tailin Liang, Lei Wang, Shaobo Shi, and John Glossner. Dynamic runtime feature map pruning. *arXiv preprint arXiv:1812.09922*, 2018.

[31] Mingbao Lin, Rongrong Ji, Yan Wang, Yichen Zhang, Baochang Zhang, Yonghong Tian, and Ling Shao. Hrank: Filter pruning using high-rank feature map. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1529–1538, 2020.

[32] Ilya Loshchilov and Frank Hutter. Sgdr: Stochastic gradient descent with warm restarts. *arXiv preprint arXiv:1608.03983*, 2016.

[33] Jian-Hao Luo and Jianxin Wu. Neural network pruning with residual-connections and limited-data. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1458–1467, 2020.

[34] Jian-Hao Luo, Jianxin Wu, and Weiyao Lin. Thinet: A filter level pruning method for deep neural network compression. In *Proceedings of the IEEE international conference on computer vision*, pages 5058–5066, 2017.

[35] Huizi Mao, Song Han, Jeff Pool, Wenshuo Li, Xingyu Liu, Yu Wang, and William J Dally. Exploring the regularity of sparse structure in convolutional neural networks. *arXiv preprint arXiv:1705.08922*, 2017.

[36] Atefeh Mehrabi, Donghyuk Lee, Niladrish Chatterjee, Daniel J Sorin, Benjamin C Lee, and Mike O'Connor. Learning sparse matrix row permutations for efficient spmm on gpu architectures.

[37] Pavlo Molchanov, Stephen Tyree, Tero Karras, Timo Aila, and Jan Kautz. Pruning convolutional neural networks for resource efficient inference. *arXiv preprint arXiv:1611.06440*, 2016.

[38] Wei Niu, Xiaolong Ma, Sheng Lin, Shihao Wang, Xuehai Qian, Xue Lin, Yanzhi Wang, and Bin Ren. Patdnn: Achieving real-time dnn execution on mobile devices with pattern-based weight pruning. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 907–922, 2020.

[39] Angshuman Parashar, Minsoo Rhu, Anurag Mukkara, Antonio Puglielli, Rangharajan Venkatesan, Brucek Khailany, Joel Emer, Stephen W Keckler, and William J Dally. Scnn: An accelerator for compressed-sparse convolutional neural networks. *ACM SIGARCH Computer Architecture News*, 45(2):27–40, 2017.

[40] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *arXiv preprint arXiv:1912.01703*, 2019.

[41] Zhuwei Qin, Fuxun Yu, Chenchen Liu, and Xiang Chen. Functionality-oriented convolutional filter pruning. *arXiv preprint arXiv:1810.07322*, 2018.

[42] Qiang Qiu, Xiuyuan Cheng, Guillermo Sapiro, et al. Dcfnet: Deep neural network with decomposed convolutional filters. In *International Conference on Machine Learning*, pages 4198–4207. PMLR, 2018.

[43] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788, 2016.

[44] Mengye Ren, Andrei Pokrovsky, Bin Yang, and Raquel Urtasun. Sbnet: Sparse blocks network for fast inference. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 8711–8720, 2018.

[45] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4510–4520, 2018.

[46] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[47] Andreas Veit and Serge Belongie. Convolutional networks with adaptive inference graphs. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 3–18, 2018.

[48] Thomas Verelst and Tinne Tuytelaars. Dynamic convolutions: Exploiting spatial sparsity for faster inference. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2320–2329, 2020.

[49] Xin Wang, Fisher Yu, Zi-Yi Dou, Trevor Darrell, and Joseph E Gonzalez. Skipnet: Learning dynamic routing in convolutional networks. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 409–424, 2018.

[50] Yulong Wang, Xiaolu Zhang, Lingxi Xie, Jun Zhou, Hang Su, Bo Zhang, and Xiaolin Hu. Pruning from scratch. *arXiv preprint arXiv:1909.12579*, 2019.

[51] Bob Wei, Mengye Ren, Wenyuan Zeng, Ming Liang, Bin Yang, and Raquel Urtasun. Perceive, attend, and drive: Learning spatial attention for safe self-driving. *arXiv preprint arXiv:2011.01153*, 2020.

[52] Zuxuan Wu, Tushar Nagarajan, Abhishek Kumar, Steven Rennie, Larry S Davis, Kristen Grauman, and Rogerio Feris. Blockdrop: Dynamic inference paths in residual networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 8817–8826, 2018.

[53] Zirui Xu, Zhuwei Qin, Fuxun Yu, Chenchen Liu, and Xiang Chen. Direct: Resource-aware dynamic model reconfiguration for convolutional neural network in mobile systems. In *Proceedings of the International Symposium on Low Power Electronics and Design*, pages 1–6, 2018.

[54] Zirui Xu, Fuxun Yu, Chenchen Liu, and Xiang Chen. Reform: Static and dynamic resource-aware dnn reconfiguration framework for mobile device. In *Proceedings of the 56th Annual Design Automation Conference 2019*, pages 1–6, 2019.

[55] Zirui Xu, Fuxun Yu, Zhuwei Qin, Chenchen Liu, and Xiang Chen. Directx: Dynamic resource-aware cnn reconfiguration framework for real-time mobile applications. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 40(2):246–259, 2020.

[56] Fuxun Yu, Chenchen Liu, Di Wang, Yanzhi Wang, and Xiang Chen. Antidote: attention-based dynamic optimization for neural network runtime efficiency. In *2020 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 951–956. IEEE, 2020.

[57] Zhuangwei Zhuang, Mingkui Tan, Bohan Zhuang, Jing Liu, Yong Guo, Qingyao Wu, Junzhou Huang, and Jinhui Zhu. Discrimination-aware channel pruning for deep neural networks. In *Advances in Neural Information Processing Systems*, pages 875–886, 2018.